Virtual Data Center Networks Embedding Through Software Defined Networking

Raphael Vicente Rosa, Christian Esteve Rothenberg and Edmundo Madeira
State University of Campinas (UNICAMP), Sao Paulo, Brazil
Email: raphaelvrosa@lrc.ic.unicamp.br, chesteve@dca.fee.unicamp.br, edmundo@ic.unicamp.br

***Abstract*—Software Defined Networking (SDN) has entered the networking scene opening new ways to design, deploy, and operate networks by introducing new abstractions and programmability of the network control and data planes. In this paper, we take an SDN approach to embed virtual data center networks supporting a Network-as-a-Service model. The proposed architecture is built around a number of abstractions to create virtual topologies using convenient BGP configurations that allow an efficient mapping to a physical network of OpenFlow 1.3 switches. At the heart of the proposal is an algorithm designed to perform efficient allocation of resources to virtual paths based on network state data, such as allocated virtual networks and resource utilization metrics. Requirements such as bandwidth and resilience are used to define the tenants policies and construct the virtual topology graphs. The experimental evaluation on an emulated testbed shows that the proposed algorithm performs efficient load balancing and altogether yields better utilization of the physical resources under different tenant traffic patterns.**

I. Introduction

By abstraction the underlying physical network away, networking virtualization has become a key enabling technology of multiple-tenant cloud data center (DC). Server virtualization enables physical server partitions into multiple isolated virtual machines (VMs). Likewise, in the form of network virtualization, the allocation and efficient use of network resources (*e.g.,* bandwidth, switches, addresses), have been widely implemented in DCNs [1]. A virtualized data center provides computational and network resources allowing tenants to apply their own policies, define their address spaces, manage their pool of VMs independently, and so on.

The effective management of virtual data centers (VDCs) is a challenge in itself [2]. VDC tenants often have heterogeneous network requirements such as performance isolation, flexible traffic allocations, fault tolerance, load balancing, deployment of new applications, and network innovation support [2]. Many efforts are being devoted to deliver efficient solutions to the problem of multi-objective resource optimization in VDCs (e.g., [3], [4]) where the virtual network embedding task becomes a challenging algorithmic issue [5]. These challenges become less tractable in dynamic environments and often involve vast amount of real-time data to perform centralized analytics processing in order to keep the resource utilization as high as possible without sacrificing SLAs (cf. [6].

In this paper, we explore the concept of Network-as-a-Service (NaaS) [7] to build virtual data center networks following a Software Defined Networking (SDN) approach. The proposed architecture allows dynamic allocation of virtual networks in data centers accordingly to bandwidth and resilience requirements. We consider a scenario where a data center Infrastructure Provider (InP) allocates physical resources to multiple tenants which act as Service Providers (SPs) and have their demands for VMs allocations well defined. In this way, any addressing scheme, traffic communication pattern and resilience parameter is used to embed virtual networks.

Our prototyping efforts leverage the RouteFlow platform [8], which allows the execution of IP routing protocol stacks over an arbitrary network of OpenFlow switches. We define a virtual plane using the BGP routing protocol with multipath support, following the premises of a recent proposal [9] to operate with a folded-Clos network topology. In the data plane, we use a physical infrastructure that supports OpenFlow protocol version 1.3 [10]. In the virtualized control plane of the platform, we build routing services that aggregate information from the physical and virtual planes, and carry the task mapping the virtual networks to the available resources. We propose an allocation algorithm with the main task to allocate bandwidth from the physical topology by mapping the routes of the per-tenant virtual topology. Policies and accounting demands of tenants define the bandwidth and resilience requirements, which are used by the mapping algorithm to build virtual networks and configure the data plane routes to provide resource isolation by features introduced in OpenFlow 1.3, such as *group* and *metering* tables.

The core contribution of this paper is the definition of virtual network graphs as a service based on data-centric abstractions to allocate network resources efficiently. We evaluate our virtual networks embedding approach in terms of link stress and utilization and compares it with existing proposals in the literature. In this sense, this work is distinguished from others by the following aspects: (*i*) proposes a SDN approach to deliver virtual networks using the BGP protocol as an operator-friendly means well-suited to data center networks based on folded-Clos topologies; (*ii*) extends the RouteFlow platform to support OpenFlow 1.3 and offer applications with northbound APIs to express network policies such as reservation of bandwidth and multi-path routing; and (*iii*) define an efficient algorithm for mapping virtual network that provides load balance attached to resilience and bandwidth guarantees.

The structure of this paper is as follows. Section 2 introduces relevant background. Section 3 presents the proposed architecture leaving to Section 4 the details on proposed algorithm to allocate virtual networks. Section 5 presents our evaluation work. Section 6 discusses the results and relates them to the existing literature and avenues for future work. Finally, Section 7 concludes the paper .

II. Background

This section describes the main components in a virtualized data center, related proposals in the literature, as well as the RouteFlow software-defined IP routing platform, a key component of the proposed architecture.

A. Data Center Network Virtualization

A data center is made up of servers, network equipment (e.g., switches/routers, cables) and power distribution and cooling systems. The data center network is defined by the network topology (*e.g.,* BCube, Clos) and network protocols used for communication between its components (*e.g.,* Ethernet, IPv6). A DCN is usually defined by the following configuration. Servers are aggregated into racks and connected to a Top-of-Rack (ToR) switch. This element connects the End-of-Row switches (EoR), which are used as intermediates towards Core switches. A VDC is defined as a set of virtualized resources (*e.g.,* VMs, virtual switches/routers) interconnected through virtual networks that share the same physical network substrate and are independently deployed and managed [2].

Vast amount of recent work on data center network virtualization is being devote to address challenges such as: virtualization mechanisms; cost-efficient topologies; performance isolation; scalability; fault tolerance; packet forwarding techniques, and so on. Basically, DCN resources are used in two forms, (*i*) competition or (*ii*) allocation. In the first case, we highlight Seawall [11], Netshare [12] and FairCloud [13], which propose a fair share of network resources by statistical multiplexing and minimum bandwidth requirements to VMs —but no deterministic guarantees of network resources (e.g., latency, bandwidth). In the second case, remarkable proposals including Gatekeeper [14], SecondNet [15], Oktopus [6], Proteus [16] and ElasticSwitch [3], perform the allocation of minimum guaranteed bandwidth to sets of VMs (tenants) in different ways, such as heuristics, network distributed flow control and VMs temporal patterns communication analysis.

B. RouteFlow

RouteFlow [8] is an SDN routing platform that logically centralizes network control, unifying the network information state and decouple the logical routing from the configuration of network equipment. The RouteFlow platform provides IP routing protocol stacks defined in a virtual plane mapped to the resources of a physical network infrastructure with OpenFlow support services. This feature enables the provision of a platform as a service concept to the networking [7] with flexible resource mapping of a virtual topology on a physical network infrastructure which can be distributed or shared. Consisting of three planes, virtual, physical and control, RouteFlow has respectively in each of these planes its main components: `rfclient`, `rfproxy` and `rfserver`.

In the virtual plane, virtual routers, defined via Linux operating system level of virtualization (Linux Containers - LXCs), are interconnected by an OpenFlow switch. Inside the virtualized Linux routers, the `rfclient` application captures routes computed by a routing engine (*e.g.,* Quagga, XORP, BIRD) and sends them to the control plane. The data plane is formed by OpenFlow switches connected to network controllers running the `rfproxy` application. In the control plane,

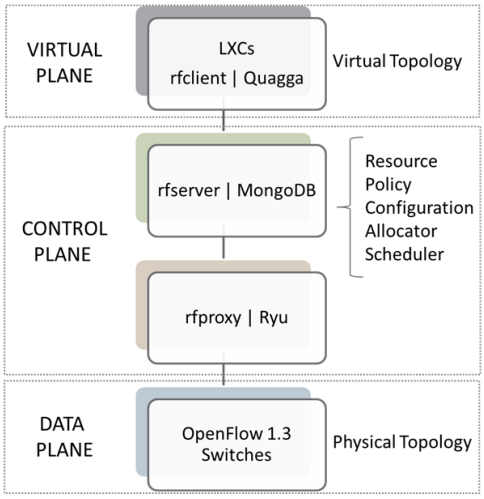


Fig. 1: Architecture

the `rfserver` application stores all the configuration state and defines the mapping between physical and virtual networks. It also manages the state of the virtual/physical mapping maintained in a database (*e.g.,* MongoDB) and performs the formatting and exchange of messages between `rfclient` and `rfproxy` applications such as IP routes and ARP tables computed in the virtual plane.

III. Architecture

The proposed architecture (see Fig. 1) results of putting together two recent pieces of work. On one side, RouteFlow [8], as a means to create and execute virtual data center networks based on programmable control of the network, the unification of the information state from virtual and data planes, and decoupling of logical routing and configuration from network equipment. On the other side, the BGP-centric DC design by Lapuhhov *et al.* [9] exploits basic configuration features of the BGP protocol for intra-AS routing in DCNs. The basic idea consists of employing a folded-Clos DCN topology upon which BGP with multipath support is configured. Advantages of this proposal include: practical routing design for large DCs; simple protocol with low code complexity and easy operational support; minimizing equipment and routing protocol failures; and operating and capital costs reduction.

We leverage the RouteFlow platform to build a DCN architecture where the virtual topology is defined in terms of the BGP routing protocol, and the data plane is based on OpenFlow 1.3 devices. The direct control capabilities of both the virtualized control plane topologies and programmable data plane are the enabling features to support the virtual networks mapping algorithm proposed in this paper. To exercise the outcomes from the mapping algorithm, we employ features of OpenFlow 1.3 (*e.g., group*, *metering*, and multiple tables) as commanded by the enhanced `rfproxy` application.

The following subsections, organized according to the architecture planes (data, virtual and control), describe the proposed approach to support the DCN requirements starting by mapping the physical network infrastructure to aggregated virtual topologies.

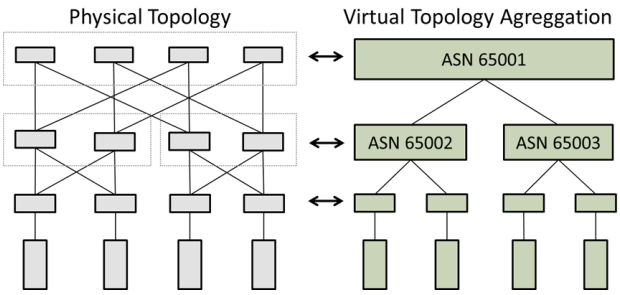


Fig. 2: Virtual and Physical Topology Aggregation Mapping

A. Data Plane

The data plane is based on folded-Clos topology of OpenFlow 1.3 switches managed by OpenFlow controllers running `rfproxy` application. Aided by a topology discovery application, `rfproxy` captures addition and removal events of switches and links in the physical topology sending them to the control plane to construct the physical topology available for mappings. OpenFlow 1.3 functionalities are used to support the mapping of virtual networks, such as the use of *group* forwarding tables similar to Equal-Cost Multipath (ECMP) mechanism, bandwidth control implemented with *metering* tables, L2 addresses rewriting, MPLS encapsulation and tagging, as arranged in a suitable multiple table pipeline.

Flow rules programmed in the physical topology use MPLS tags as unique identifiers of a tenant virtual network. In Core and EoR switches, traffic is forwarded only by matches on these tags. At ToRs switches, four tables of the OpenFlow 1.3 pipeline are programmed. Table 0 matches MPLS tags with traffic destined inside the rack and takes the actions to withdraw the MPLS shim layer and go to table 1, where the match occurs on IP network addresses, and have the actions of rewriting MAC addresses and forwarding to the next table. In Table 2, matches on previously rewritten servers MAC addresses generates the action of traffic forwarding to their proper connected ports. Table 3 handles traffic to be sent out of the rack by actions of adding a shim MPLS layer and tagging the traffic, defined on the route that was set to a particular tenant in the control plane. All flow rules are installed with hard timeouts to define the permanency of a virtual topology in the data plane. Bandwidth limitation rules are programmed in ToRs, delimiting rack traffic in and out by *metering* tables associated respectively to the flow rules in tables 2 and 3.

B. Virtual Plane

The `rfclient` application In the virtual control plane has been improved to support the detection of multipath routes. The Quagga routing engine running the BGP protocol is configured with the advantages for intra-AS DCNs in folded-Clos topology as per [9]. As shown in Fig. 2, we mapped the base topologies, physical and virtual, representing the data and virtual planes by aggregating elements containing the same ASN of a folded-Clos topology layer (*e.g.,* Core, EoR). Since elements of the same folded-Clos topology layer are not interconnected, they have equidistant routes to all other elements of the topology, and routes with the same AS-PATH can be computed to all network destinations. Initially, all control messages transferred between the virtual and data planes are passed via the network controller. To avoid overloading the network controller, after mapping the base topologies, flow entries are installed to keep all control messages in the real of the virtual plane switch.

C. Control Plane

The control plane is formed by the `rfserver` application, the database with the mapping state of the physical and virtual planes, and the main components developed to perform the virtual network embedding (see Fig. 3): **Resource**, **Policy**, **Configuration**, **Allocator**, and **Scheduler**.

Resource performs the storage and management of information obtained from the virtual (*e.g.,* LXCs, interfaces, routes) and data (*e.g.,* switches, links) planes with the creation of the *PhysicalTopology* and *VirtualTopology* classes along their representative attributes. These two classes are inherited from a *Topology* class which builds graphs for abstraction purposes. This component also defines the *Topologies* class that has as the required functions to instantiate physical and virtual topologies, manage their mapping, configure their settings (*e.g.,* routes, links, switches, LXCs), and analyze the configuration and state of the physical and virtual topologies. In addition, the *Servers* class represents all VMs in servers and their correspondent racks.

Policy is responsible for managing requests for virtual network allocations and include requirements such as bandwidth, resilience, IP and MAC addresses of hosts that constitute the virtual topology to be established, or any related feature that constitute the requirements for any data transfer property between VMs/servers or applications. When a policy is constructed and the mapping is performed, the virtual network topology is stored and associated with a specific identifier which is programmed in flow rules by MPLS tags in the data plane. Within this component, a class named *Policies* performs registration, management and storage of policies created with their respective built and mapped virtual topologies.

Configuration embodies the mapping algorithms that perform both the association between topologies as the routes mapping from the virtual base topology to create virtual topologies. Moreover, within this component are algorithms that assist in major mapping functions, which deal with the changes of virtual and physical topologies resources to better meet the policy mapping requests. Basically the algorithms defined in this component comply the policies to determine routes, links and bandwidth allocations in resources of the base physical topology defined by the Resource component. The outcome from the configuration algorithm is a virtual topology graph annotated with the compliant policy attributes.

Allocator performs all configuration interface between virtual topologies and the data plane. It is through the allocation process that virtual topologies are configured, mappings are carried based on the Configuration component, and the actual mapping to the physical topology is executed by commands sent to `rfproxy` application. Hence, the main function of this component is the translation of abstracted virtual topology graphs properties into OpenFlow-like data plane messages. In addition, it also performs the configuration of virtual switch routes in the virtual plane.

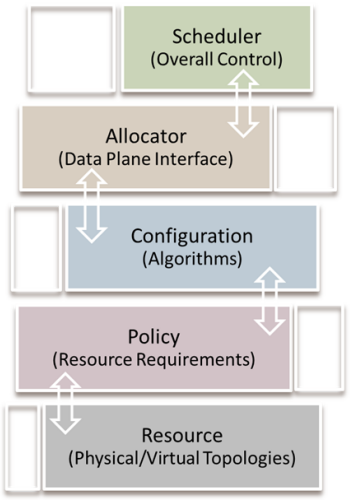


Fig. 3: Control Plane Components

Scheduler is the component responsible for managing the overall operation of the above components. Through it, policies are created, established and configured and virtual and physical mapping topologies are defined as policy constraints to be dynamically allocated and deallocated. That is, this component performs all the interface communication with the `rfserver` application and handles the main configuration features and orchestrates the operations of all components of the architecture .

D. Proof of Concept Execution

Next we illustrate the operation of the proposed architecture elements by initializing the components in the following order: LXCs and `rfclient` applications; virtual plane switch; database; `rfserver` application; network controller and `rfproxy` application; physical topology switches. The `rfserver` application initiates all components aforementioned as RouteFlow platform services.

After physical topology discovery and virtual plane convergence, the scheduler component creates a default policy which defines the mapping of the base, physical and virtual, topologies. At first, the objects representing these topologies are instantiated to be used in subsequent mappings. Furthermore, the object that represents rack servers is created for policies instantiation. Then, three *threads* are executed to perform the tasks of: (*i*) creating mapping demands and policies (*e.g.,* triggered by OpenStack, Hadoop, and the like applications); (*ii*) allocation of policies to create virtual topology graphs; and (*iii*) policy deallocation as mapping duration times expires.

The first *thread*, based on a traffic matrix between VMs, servers or applications, performs the representation of these communication pattern in servers using the object *Servers* and creates a traffic matrix between racks. This information is aggregated and defined into a policy, which is placed in a queue to be allocated. Any VM allocation or application abstraction technique (*e.g.,* [17], [18]) can be programmed in this thread to produce a data communication pattern. The allocator checks this queue and performs the allocation of policies on the base physical topology via the proposed mapping algorithm. Depending on the allocation time set for each policy, the deallocation *thread* removes virtual topologies from the base physical topology.

IV. Algorithms

The *Configuration* component has two main algorithms: (*i*) the *resource bookkepping algorithm* manages the information state of the physical topology resources to the (*ii*) *mapping algorithm* build and allocate virtual topology graphs.

A. Resource Bookkepping Algorithm

In each switch of the base physical topology object of the control plane, two attributes stand out. The first defines a *bw_port_table* containing the available bandwidth percentage on each switch port/link. Each time a link has its resource properties changed, such as a virtual route allocation, the percentage of available bandwidth on this link is computed and stored in *bw_port_tables* of its adjacent switches in the form {*adjacent link port : percentage of available link bandwidth*}. The other attribute concerns the *bw_table*, which defines the percentage of available bandwidth to destination addresses of switch routes. Each time a route is mapped to a link, its adjacent switches update their *bw_tables* as the destination address of the mapped route, providing inputs in the form {*destination address : [port of destination route address : percentage of bandwidth available for the route]*}.

Algorithm 1 (*Resource Bookkepping Algorithm*): Updates switches *bw_tables* of base physical topology

Require: Information of base physical topology resources state (topo_phy_base)
Ensure: All switches *bw_tables* updated from (topo_phy_base)
1: **for all** switch ToR in topo_phy_base **do**
2: SwitchesQueue.addItem(ToR, ToRNetworkAddressPrefix, ToR.bw_table, port=1)
3: **end for**
4: **while** SwitchesQueue not empty **do**
5: (switch, NetworkAddressPrefix, switch._bw_table, port) ← SwitchesQueue.popItem()
6: **if** (switch, NetworkAddressPrefix, port) in QueuedSwitches **then**
7: switch._bw_table ← QueuedSwitches(switch, NetworkAddressPrefix, port)
8: QueuedSwitches.removeItem(switch, NetworkAddressPrefix, port)
9: **end if**
10: VisitedSwitches.addItem(switch, NetworkAddressPrefix, port)
11: bw_usage_list ← list of all bandwidth values in switch.bw_table addressed to NetworkAddressPrefix
12: bw_usage_mean ← Higher value to be equally allocated in all bw_usage_list entries of NetworkAddressPrefix
13: **if** port in links to neighbor switches **then**
14: switch_bw_table[NetworkAddressPrefix][port] ← min(bw_usage_mean, switch._bw_port_table[port])
15: **end if**
16: **for all** ports not in adjacent switch links **do**
17: **if** (link.neighborSwitch, NetworkAddressPrefix, link.neighborDestinationPort) not in QueuedSwitches and VisitedSwitches **then**
18: SwitchesQueue.addItem(link.neighborSwitch, NetworkAddressPrefix, switch_bw_table, link.neighborDestinationPort)
19: **end if**
20: **if** (link.neighborSwitch, NetworkAddressPrefix, link.neighborDestinationPort) in QueuedSwitches **then**
21: QueuedSwitches(switch, NetworkAddressPrefix, port) ← switch_bw_table
22: **end if**
23: **end for**
24: **end while**

It is important to note that a *bw_table* comprises the end-to-end available route bandwidth, while the *bw_port_table* defines the local available bandwidth, i.e., only in adjacent switch links. Also, ToRs have in their *bw_table* registries of available bandwidth of its interconnected servers, thus defining all the bandwidth available in racks. Based on these attributes,

the key point of the Algorithm 1 (*Resource Bookkepping*) is the update of *bw_tables* of all the base physical topology switches, so that they are consistent with the state of the mapped policies and switches *bw_port_tables*, and can be used in the mapping algorithm. This update occurs only on the base physical topology, every time a policy is created, allocated and deallocated. There are no restrictions to extend this algorithm to server virtual switches, since it is based on the a Breadth First Search over the base physical topology graph.

B. Mapping Algorithm

This algorithm is also defined by a Breadth First Search over the base physical topology graph. Likewise, ToR switches are the input nodes, representing a policy with a traffic matrix between them. The output of the algorithm is a virtual topology built on route information of the base virtual topology and available bandwidth of the base physical topology. The mapping occurs by bandwidth annotations, defined by policies unique identifiers, made on base physical topology links. These markings are performed according to the selected routes of the base virtual topology and two requirements, bandwidth and resilience, defined in the policies created.

Algorithm 2 (*Mapping Algorithm*): Maps virtual topologies in base physical topology

Require: base physical topology (topo_phy_base), base virtual topology (topo_virt_base), policy
Ensure: virtual topology mapped
1: **for all** ToR switches in policy traffic matrix **do**
2: **for all** NetworkAddressPrefix in policy traffic matrix $\neq$ network address range of ToR switch **do**
3: lxc $\leftarrow$ lxc of topo_virt_base mapped to ToR
4: SwitchesQueue.addItem(lxc, ToR, NetworkAddressPrefix)
5: SwitchesFeatures(lxc, ToR, NetworkAddressPrefix) = ToR bandwitdh to NetworkAddressPrefix in policy traffic matrix
6: **end for**
7: **end for**
8: **while** SwitchesQueue not empty **do**
9: (lxc, switch, NetworkAddressPrefix) = SwitchesQueue.popItem()
10: **if** NetworkAddressPrefix not in QueuedSwitches **then**
11: VisitedSwitches.addItem(lxc, switch, NetworkAddressPrefix)
12: **end if**
13: RequestedBandwidth = SwitchesFeatures(lxc, switch, NetworkAddressPrefix)
14: switch_selected_routes $\leftarrow$ SelectRoutes(switch, lxc, NetworkAddressPrefix, RequestedBandwidth)
15: RoutesBandwidth $\leftarrow$ RequestedBandwidth equally divided between switch_selected_routes
16: **for all** route in switch_selected_routes **do**
17: **if** RoutesBandwidth[route] allocated in topo_phy_base link defined by route **then**
18: Adds switch, route in switch and link in VirtualTopology
19: Defines DestinationLXC and DestinationSwitch as lxc and switch associated in link defined by route
20: SwitchesQueue.addItem(DestinationLXC, DestinationSwitch, NetworkAddressPrefix)
21: SwitchesFeatures(DestinationLXC, DestinationSwitch, NetworkAddressPrefix) $\leftarrow$ RoutesBandwidth[route]
22: **else**
23: Mapping $\leftarrow$ False
24: Stop execution loops
25: **end if**
26: **end for**
27: **end while**
28: **if** Mapping $\neq$ True **then**
29: Undo all policy mappings done so far in topo_phy_base
30: **end if**

In option "SelecRoutes Agreg" in Algorithm 3, the resilience policies criteria are defined by percentages, which express the amount of routes that will be selected for the evaluation of all available mapping paths. Furthermore, in this option, the use of route selection, average minus twice the standard deviation allows the selected routes within two percentiles of the average values of *bw_port_tables*. This parameter ensures that there is load balancing on the ports on a switch, unlike the option "SelecRoutes Traditional" where only one route with less available bandwidth is selected and no criterion for switch ports load balancing is employed.

Algorithm 3 (*Select Routes*): Select routes to be mapped

Require: (switch, lxc, NetworkAddressPrefix, RequestedBandwidth)
Ensure: Selected routes for mapping
1: switch_routes $\leftarrow$ lxc.get_routes(NetworkAddressPrefix)
2: Select switch_routes with higher switch.bw_port_table capacity that satisfy resilience policy requirements
3: **if** Option *SelectRoutes Agreg* **then**
4: Calculate mean, standard deviation, higher and smaller values from switch.bw_port_table with ports defined by switch_routes
5: Select combination of routes from switch_routes which satisfy RequestedBandwidth divided between them **and** that have switch._bw_port_table higher or equal to the mean less two times standard deviation of switch.bw_port_table
6: From previously selected routes, select those with less difference between the higher and smaller values in case of their selection and definition in switch.bw_port_table
7: Return routes previously selected
8: **end if**
9: **if** Option *SelectRoutes Traditional* **then**
10: Return route from switch_routes that satisfy RequestedBandwidth of NetworkAddressPrefix in switch.bw_table
11: **end if**

V. Evaluation

The experimental testbed was assembled using RouteFlow code base with the aforementioned modifications. The `rfproxy` application was built using the Ryu controller,[1] with OpenFlow 1.3 upgrades.[2] We use an OpenFlow 1.3 software switch (`ofsoftswitch13`)[3] both in the data and virtual planes and defined 12 and 48 switches folded-Clos topologies using Mininet [19]. We consider the physical topology and the control plane connection between each switches to have 10,000 units of bandwidth and 1,000 between servers and switches. Furthermore, we assume 20 and 40 servers per rack, in the 12- and 48- switch topologies, respectively, and each server hosting up to 20 VMs. All experiments were used with two virtual machines, one running all control and virtual planes (6 Cores and 12 GB RAM), and another executing the data plane (2 Cores and 4 GB RAM).

A. Analytical Evaluation

We start our evaluation by analyzing the architecture constructed using BGP virtual routers defined by the aggregation mapping of the physical topology. We seek to observe a comparison between the folded-Clos topologies with and without aggregation, respectively referenced as Clos and ClosAgreg. As shown in Table I, we can observe that as the number of physical switches increase, the virtual topology aggregation requires fewer connections between virtual routers, smaller amount of TCP/BGP connections, and a lower number of control messages exchanged on each BGP update event.

B. Experimental Evaluation

1) Time Performance of Mapping Operations: In order to get performance criteria of the proposed architecture and

[1] https://github.com/osrg/ryu
[2] https://github.com/routeflow/ryu-rfproxy
[3] https://github.com/CPqD/ofsoftswitch13

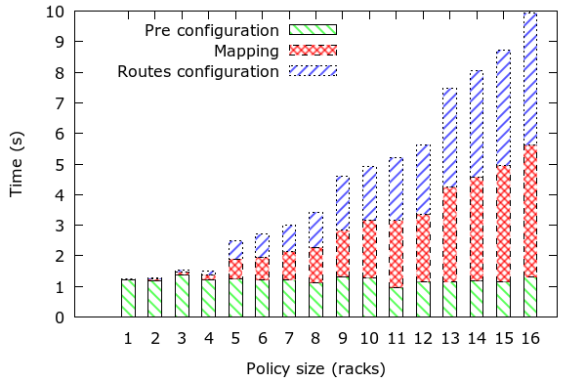


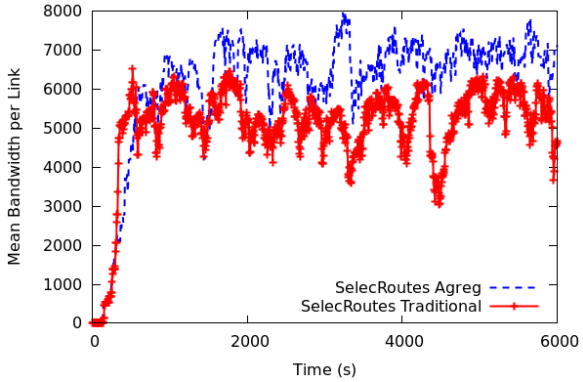
Fig. 4: Mapping timers

algorithm, we conducted experiments to evaluate the mapping time as the size of virtual network requests grows up. We discriminate the following operation time windows: (*i*) checking of pre-configuration and topologies; (*ii*) virtual network mapping and graph building; and (*iii*) configuration of routes in the data plane. We performed experiments varying the size of the mapping policies from 1 to 16 racks in 48 switches folded-Clos topology. As we can observe in Fig. 4, the pre-configuration time is practically constant and does not depend on the policies size. The mapping and configuration time increase as the policies size because the amount of routes to be analyzed, selected, mapped and configured in ToRs. In the case of the analyzed topology, we note that there are small jumps of time as a policy uses racks non-interconnected by the same EoR switches, which is due to the need of routes mapping in Core switches.

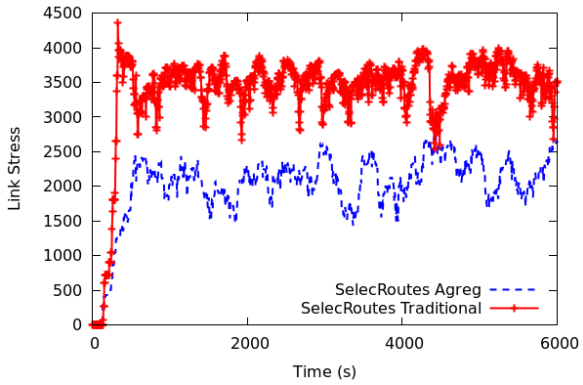
2) Bandwith Gains and Link Stress: Next we evaluate the Algorithm 2 and compare the performance of *SelecRoutes Traditional* and *SelecRoutes Agreg* strategies of Algorithm 3. We perform the creation of mapping demands in a Poisson process with VMs being allocated orderly as the availability of servers bandwidth. We established the following parameters for this experiment: arrival of demands by a Poisson process with an average of 30 requests per minute. Each request contains, respectively, for topologies with 12 and 48 switches: number of virtual machines uniformly distributed between 10 and 30 and between 30 and 70; traffic demand between VMs uniformly distributed between 1 and 10 units of bandwidth, defined by all-to-all traffic pattern; mapped virtual networks with residence network time uniformly distributed between

TABLE I: Comparison of base and aggregated topologies.

Topology	Switches Data Plane	Virtual Routers	Router Sessions	Control Messages
Clos	12	12	16	32
ClosAgreg	12	7	6	12
Clos	48	48	64	128
ClosAgreg	48	9	8	16
Clos	96	96	256	512
ClosAgreg	96	11	34	68
Clos	128	128	1024	2048
ClosAgreg	128	35	68	136



(a) Bandwidth used per link



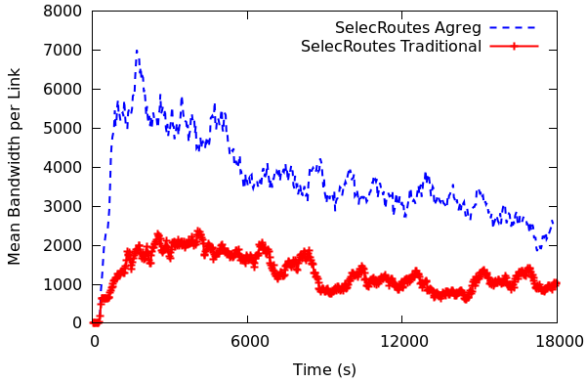
(b) Link stress

Fig. 5: 12 switches folded-Clos topology

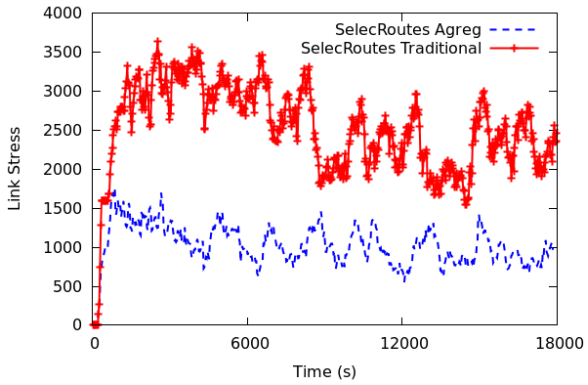
180 and 300 and between 540 and 660 seconds; and total experiment time set in 6,000 and 18,000 seconds.

We evaluate the proposed algorithms in terms of average bandwidth and its variation (*link stress*) in physical network links to understand the load balancing behaviour on data plane topologies with 12 (Fig. 5) and 48 (Fig. 6) switches. The results obtained show that independent from the number of switches in the physical topologies, the *SelecRoutes Agreg* option excels with a minor link stress and using a higher mean bandwidth per link, which can be explained due to the network load balancing capabilities of the proposed algorithms.

3) Performance for Varying Traffic Patterns: In order to obtain a sensitivity analysis on different traffic patterns, we carry an experiment in the 48-switch folded-Clos topology with the same parameters initially used, but following different patterns of communication between VMs: *all-to-one* and *one-to-all*. Again we observe in Fig. 7 the same behavior showed in all-to-all traffic communication pattern (Fig. 5). In both experiments, *all-to-one* (Fig. 7(a)) and *one-to-all* (Fig. 7(b)), we obtain a lower link stress using the (*SelecRoutes Agreg*) strategy. In all experiments, Figs. 5, 6 and 7, we obtain the same average number of policies allocated for both algorithms, and zero ratio of denied mapping policies.

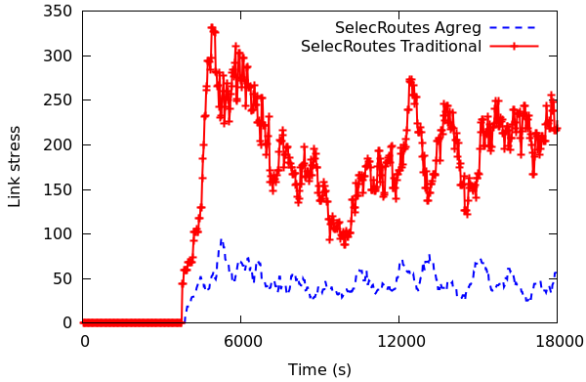


(a) Bandwidth used per link

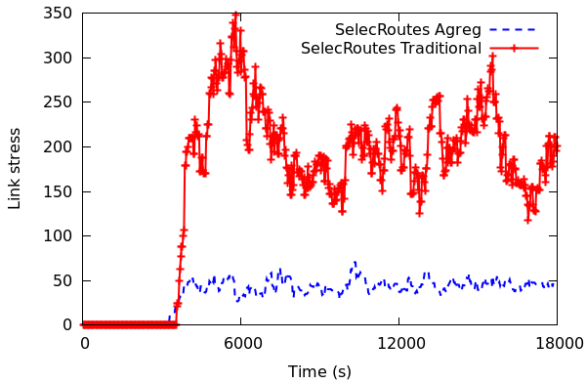


(b) Link stress: All-to-all traffic communication pattern

Fig. 6: 48-switch folded-Clos topology



(a) Link stress: All-to-one traffic communication pattern



(b) Link stress: One-to-all traffic communication pattern

Fig. 7: Link stress for varying traffic patterns.

VI. Discussion and Future Work

The high degree of freedom in creating virtual data center networks by aggregating information from both physical and virtual planes provides an insight yet unexplored in related work. Firstly, high-level goals can be easily implemented in terms of policies for virtual networks allocation demands both in the data and control planes, either by programming rules into the routes selection process or by setting BGP configuration knobs as proposed by [20]. Secondly, the information aggregated from the physical and virtual topology graphs into the logically centralized control plane enables a variety of opportunities to network control and virtual networks mappings. These features foster the main goals of this work, because they propose new data discrimination operations, Fundamentally, the proposed architecture allows meeting different policy requirements in support of efficient DCN sharing objectives. Next, we review a selection of topics sparing discussions on related and future work.

Data-centric policy abstractions: As noted in the experimental results, the proposed algorithm performs efficient load balancing in the network when compared to the algorithm *SelecRoutes Traditional* option, commonly seen in the literature (*e.g.,* [6] and [16]). The focus in data-centric policy features brings out the main aspects of load balancing of the algorithm proposed. The network knowledge base established in the *Resource Bookkeeping Algorithm* brings a topology graph perspective with a consistent view of physical network resources to be used by the *Mapping Algorithm*. Building upon this feature, the traffic granularity available in the *SelecRoutes Agreg* option could be dynamically adjusted to efficiently select routes for different applications, based on their traffic characteristics and priority (*e.g.,* Hadoop, backup traffic). These properties define an important field of future work where online data measured statistics and energy consumption related features are correlated to the degree of DCN load balancing.

Hose model: So far, related work (*e.g.,* [3], [6], [11]–[16]) explicitly deals with the allocation/competition bandwidth trade-off and its bandwidth isolation and queue sharing requirements. The bandwidth allocation scheme used in this paper handles different traffic patterns being adequate to support data center load balancing for different application communication patterns which can be implemented as policy definitions. The *Policy* component does not forbid statistical multiplexing in traffic allocation analysis to be also built into the control plane of the architecture, to yield, e.g., work conserving [3] traffic allocations. [21] include discernments that meet the traffic communication abstractions implemented in our work. For example, policy definitions can easily be implemented according to the VOC [6] or TAG [18] models for traffic applications abstractions and their attributes (*e.g.,* bandwidth, addresses, resilience, priorities).

Limitations: Centralized solutions often involve scalability concerns. As in our case, the aggregated state information in a centralized control plane is subject to known performance

issues –a matter of ongoing research within the SDN community. In a different perspective, we focused in the virtual mapping aggregation of physical folded-Clos topologies with the advantages identified by [9] using RouteFlow to orchestrate the virtual control plane. An interesting research topic would be whether any multi-path topology could be used even with non-uniform load balancing across paths (*e.g.,* Jellyfish). In this case, for example, a controller-based routing engine could be used instead of a virtual topology routing plane. And finally, fault tolerance inserted in virtual networks reconfiguration is a work possibility already observed in WANs that can be a simple extension of the proposed mapping algorithm.

OpenFlow: Unlike existing proposals in the literature, our architecture and proof of concept prototype is built using OpenFlow version 1.3 and handles load balancing requirements through multi-path traffic management and fine granularity. OpenFlow 1.3 features allow all necessary properties to program virtual topology graphs in different ways, for example by setting the filtering of data path virtual traffic only by matching on TCP/IP ports/protocols (e.g., NVGRE, VXLAN). Likewise, according to recent results on packet spraying techniques [22], OpenFlow 1.3 queue definitions could also be used to prioritize various traffic communication aspects, such as isolation and QoS. The evaluation criteria of performance and scalability in DCNs using new translation addressing schemes and other OpenFlow 1.3 features, are research topics that are already underway and emerge as a prominent SDN research topic.

VII. Conclusion

In this paper, we sought to evaluate the allocation of virtual networks in data centers using the NaaS model with the application of SDN concepts. We built an architecture using the RouteFlow SDN control platform to shape the entire virtualized data center network environment. We explore the mapping to the physical data center network infrastructure from a virtual topology where routes from the BGP protocol –configured specifically for this environment– are obtained upon the allocation of virtual data center topologies. Through algorithms implemented on the control plane of the RouteFlow platform, we were able to show efficient allocation of virtual networks considering bandwidth, load balancing and routing protocol overhead. Therefore, issues related to both forms of addressing, network flexibility, development and creation of new network applications, among others, can be made with total freedom on the proposed architecture. This fact allows to address several DCNs problems by the extension of this work, considering different operational and practical requirements for these type of networks.

Acknowledgment

We would like to thank CNPq for the financial support.

References

[1] M. Rabbani, R. Pereira Esteves, M. Podlesny, G. Simon, L. Zambenedetti Granville, and R. Boutaba, "On tackling virtual data center embedding problem," in *IFIP/IEEE IM 2013*, 2013, pp. 177–184.

[2] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *Commun. Surveys Tuts., IEEE*, vol. 15, no. 2, pp. 909–928, 2013.

[3] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing," in *Proc. of the ACM SIGCOMM '2013*. New York, NY, USA: ACM, 2013, pp. 351–362.

[4] R. Niranjan Mysore, G. Porter, and A. Vahdat, "Fastrak: Enabling express lanes in multi-tenant data centers," in *Proc. of the CoNEXT '13*. New York, NY, USA: ACM, 2013, pp. 139–150.

[5] A. Fischer, J. Botero, M. Till Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Commun. Surveys Tuts., IEEE*, vol. 15, no. 4, pp. 1888–1906, 2013.

[6] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. of the ACM SIGCOMM 2011*. New York, NY, USA: ACM, 2011, pp. 242–253.

[7] E. Keller and J. Rexford, "The "platform as a service" model for networking," in *Proceedings of INM/WREN'10*. Berkeley, CA, USA: USENIX Association, 2010, pp. 4–4.

[8] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *Proc. of HotSDN '12*. New York, NY, USA: ACM, 2012, pp. 13–18.

[9] P. Lapukhov, A. Premji, and E. J. Mitchell, "Use of bgp for routing in large-scale data centers," Working Draft, IETF Secretariat, Internet-Draft draft-lapukhov-bgp-routing-large-dc-06.txt, Aug. 2013.

[10] Open Network Foundation, "Openflow switch specification version 1.3.0 (wire protocol 0x04)," ONF, Tech. Rep., 2012.

[11] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: Performance isolation for cloud datacenter networks," in *Proc. of the HotCloud '10*. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–1.

[12] V. T. Lam, S. Radhakrishnan, R. Pan, A. Vahdat, and G. Varghese, "Netshare and stochastic netshare: Predictable bandwidth allocation for data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 3, pp. 5–11, Jun. 2012.

[13] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: Sharing the network in cloud computing," in *Proc. of the ACM HotNets '2011*. New York, NY, USA: ACM, 2011, pp. 22:1–22:6.

[14] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proceedings of WIOV'11*. Berkeley, CA, USA: USENIX Association, 2011, pp. 6–6.

[15] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: A data center network virtualization architecture with bandwidth guarantees," in *Proceedings of Co-NEXT '10*. New York, NY, USA: ACM, 2010, pp. 15:1–15:12.

[16] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proceedings of SIGCOMM '12*. New York, NY, USA: ACM, 2012, pp. 199–210.

[17] C. Fuerst, S. Schmid, and A. Feldmann, "Virtual network embedding with collocation: Benefits and limitations of pre-clustering," in *CloudNet '2013, IEEE*, Nov 2013, pp. 91–98.

[18] J. Lee, M. Lee, L. Popa, Y. Turner, P. Sharma, and B. Stephenson, "Cloudmirror: Application-aware bandwidth reservations in the cloud," in *HotCloud'13*, 2013.

[19] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc. of the CoNEXT '12*. New York, NY, USA: ACM, 2012, pp. 253–264.

[20] P. Lapukhov and E. Nkposong, "Centralized routing control in bgp networks using link-state abstraction," Working Draft, IETF Secretariat, Internet-Draft draft-lapukhov-bgp-sdn-00.txt, 2013.

[21] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network support for resource disaggregation in next-generation datacenters," in *Proc. of the HotNets '2013*. New York, NY, USA: ACM, 2013, pp. 10:1–10:7.

[22] A. Dixit, P. Prakash, Y. Hu, and R. Kompella, "On the impact of packet spraying in data center networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 2130–2138.