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 opened new ways to design, deploy, and operate networks with new abstractions and programmability at network control and data planes. In this paper, we approach SDN to embed virtual data center networks employing the Network-as-a-Service model. The proposed architecture is built upon abstractions to create a virtual topology using BGP configurations that allow an efficient mapping to a physical network of OpenFlow 1.3 switches. In the control plane, an algorithm is designed to perform efficient allocation of network resources to virtual paths based on the data plane state, 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 graph mappings. 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

In the form of network virtualization, the allocation and efficient use of network resources (*e.g.,* bandwidth, switches, addresses), have been widely implemented in data center networks (DCNs) [1]. A virtualized data center provides computational and network resources allowing tenants to apply their own policies, define addresses spaces, manage their pool of VMs independently, and so on. Virtual data center (VDC) tenants often have heterogeneous network requirements such as performance isolation, flexible traffic allocations, fault tolerance and load balancing [2]. Many efforts are being devoted to deliver efficient solutions to solve multi-objective resource optimizations in VDCs (e.g., [3], [4]) where the virtual network embedding task becomes a challenging algorithmic issue [5].

In this paper, we explore the concept of Network-as-a-Service (NaaS) [6] 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. Our prototyping efforts leverage the RouteFlow platform [7] to define a virtual plane using the BGP routing protocol with multipath support, following the premises of a recent proposal [8] to operate with a folded-Clos topology. In the data plane, we use a physical infrastructure supporting OpenFlow protocol version 1.3 [9]. And in the control plane, we build services and algorithms that aggregate information from the physical and virtual planes, carry the task of mapping network requests into virtual data center topology graphs. Therefore, allocating physical topology resources in order to satisfy policies accounting demands of tenants, with their bandwidth and resilience requirements.

978-1-4799-0913-1/14/$31.00

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.

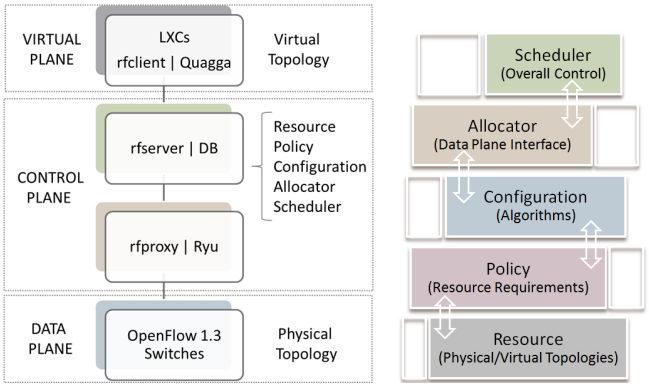
The structure of this paper is as follows. Section 2 introduces relevant background. Section 3 presents the proposed architecture leaving to Section 4 the details on proposed algorithm to allocate virtual networks. Section 5 presents our evaluation work. Section 6 discusses the results and relates them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

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 [2]. Basically, DCN resources are used in two forms, (*i*) competition or (*ii*) allocation. In the first case, we highlight Seawall [10], Netshare [11] and FairCloud [12], 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 [13], SecondNet [14], Oktopus [15], 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.



(a) Architecture (b) Components

Fig. 1: Architecture and Components

B. RouteFlow

RouteFlow [7] is an SDN routing platform that logically centralizes network control, unifying the network information state, and decouples the logical routing from the configuration of network equipment. It provides IP routing protocol stacks defined in a virtual plane mapped to the resources of a data plane with OpenFlow support services. In the control plane, the provision of a platform as a service concept [6] is performed with flexible resource mapping which can be distributed or shared. In the virtual plane, routers, defined via Linux operating system level of virtualization (Linux Containers - LXCs), are interconnected by an OpenFlow switch. Inside these routers, the `rfclient` application captures routes computed by a routing engine (*e.g.,* Quagga, XORP) and sends them to the control plane. The data plane contains OpenFlow switches connected to network controllers running the `rfproxy` application. In the control plane, the `rfserver` application stores all the configuration that defines the mapping between physical and virtual planes, maintained in a database (*e.g.,* MongoDB), and performs the formatting and exchange of messages between `rfclient` and `rfproxy` such as IP routes and ARP tables computed in the virtual plane.

III. Architecture

The proposed architecture in Fig. 1(a) results of putting together two recent pieces of work. On one side, RouteFlow [7], supporting virtual data center networks based on programmable control of the network with the unified information state from virtual and data planes and the decoupling of logical routing and configuration from network equipment. On the other side, the DCN design by Lapuhhov *et al.* [8] that exploits configuration features of the BGP protocol for intra-AS routing in folded-Clos topology, benefiting: a practical routing design for large data centers (DCs); simple protocol use with low code complexity and easy operational support; minimum equipment and routing protocol failures; and operating and capital costs reduction. 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.

A. Data Plane

The data plane is based on a folded-Clos topology of OpenFlow 1.3 switches managed by OpenFlow controllers running `rfproxy` application, which captures topology discovery events and sends them to the control plane to construct the physical topology resources available for mappings. Flow rules programmed in the physical topology use MPLS tags to uniquely identify each tenant virtual network. In Core and EoR switches, traffic is forwarded only by matches on these tags. At ToRs switches, 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 in ToRs delimit rack traffic in and out by *metering* tables associated respectively to the flow rules in tables 2 and 3, and the use of *group* forwarding tables spreads traffic in all data plane switches ports as ECMP behavior.

B. Virtual Plane

The `rfclient` application supports the detection of multipath routes computed by the Quagga routing engine running the BGP protocol, configured with the advantages for intra-AS DCNs in folded-Clos topology [8]. We mapped the base topologies, physical and virtual, representing the DCN data and virtual planes by aggregating elements containing the same ASN of a folded-Clos topology layer. 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 virtual plane switch.

C. Control Plane

The control plane is formed by the `rfserver` application and database with the mapping state of the physical and virtual planes, and the main components developed to perform the virtual network embedding (see Fig. 1(b)). **Resource** stores and manages graphs representing the information obtained from the virtual (*e.g.,* LXCs, interfaces, routes) and data (*e.g.,* switches, links) planes along their attributes. This component also instantiate physical and virtual topologies, manage and configure their mappings and settings (*e.g.,* routes, links), and analyze their configuration and state. A **Policy** is created when there is a virtual network allocation request containing requirements (*e.g.,* addresses schemes, bandwidth, resilience) between tenant components (e.g., servers, VMs, applications). The virtual network topology graph created by the policy mapping is stored and associated with a unique identifier used in flow rules programmed in the data plane.

Configuration embodies mapping and auxiliary algorithms that perform the association between topologies, update their resources, and maps the routes from the base virtual topology to create virtual topologies graphs annotated with compliant attributes requested by policies. The **Allocator** performs the control-data plane communication sending commands to the `rfproxy` application translating the abstracted virtual topology graphs properties into OpenFlow-like data plane messages. **Scheduler** performs all the interface communication with the `rfserver` application, handles the main architecure setting and orchestrates its components operations. Through it, policies are created, established and configured, and virtual/physical topology mappings are defined as policy constraints are dynamically allocated and deallocated.

According to the architecture initialization, the `rfserver` application executes all components aforementioned as RouteFlow platform services. In **Scheduler** three *threads* are executed. The first, based on a mapping request (*e.g.,* triggered by OpenStack, Hadoop, and the like applications), performs the representation of the tenant communication pattern creating a traffic matrix between racks. A policy aggregating this information with other requested network features (*e.g.,* bandwidth, resilience, addresses) is created and placed in a queue to be allocated. Any VM allocation or application abstraction technique (*e.g.,* [17], [18]) can be programmed in this first thread to produce a tenant traffic matrix. The second *thread* 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 of the physical topology resources to the (*ii*) *Mapping Algorithm* build and allocate virtual topology graphs over it. In each switch of the base physical topology object in the control plane, two attributes stand out. The first defines a *bw_port_table*, in the form {*adjacent link port : percentage of available link bandwidth*}, which is updated each time a link has its resource properties changed (*e.g.,* bandwidth). The second concerns the *bw_table*, in the form {*destination address : [port of destination route address : percentage of bandwidth available for the route]*}, that 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 address of the route.

Bw_tables comprise the end-to-end available route bandwidth, while the *bw_port_table* defines the local available bandwidth only in adjacent switch links. Also, ToRs have in their *bw_table* registries of available bandwidth of their interconnected servers and racks. The key point of the *resource bookkepping algorithm* is a Breadth First Search (BFS) with all ToRs as input nodes to walk all available paths to the other ToRs, updating the *bw_tables* of all the base physical topology switches with consistent state of the mapped policies and switches *bw_port_tables*. This update occurs only on the base physical topology, every time a policy is created, allocated and deallocated.

Algorithm 1 (*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 ≠ network address range of ToR switch **do**
3: lxc ← 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 ← SelectRoutes(switch, lxc, NetworkAddressPrefix, RequestedBandwidth)
15: RoutesBandwidth ← 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) ← RoutesBandwidth[route]
22: **else**
23: Mapping ← False
24: Stop execution loops
25: **end if**
26: **end for**
27: **end while**
28: **if** Mapping ≠ True **then**
29: Undo all policy mappings done so far in topo_phy_base
30: **end if**

The algorithm 1 outputs a virtual topology graph built with BGP route informations of the base virtual topology fitting the available bandwidth of the base physical topology. Performing a BFS, the mapping occurs by bandwidth annotations of policies unique identifiers made on base physical topology links that are performed according to the selected routes of the base virtual topology and two policies requirements, bandwidth and resilience. In Algorithm 2, the option "SelecRoutes Agreg" has resilience policies criteria defined by percentages of the amount of routes that will be selected of all available mapping paths. Furthermore, 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*, ensuring load balancing on the switch ports.

V. Evaluation

The experimental testbed was assembled using RouteFlow base code with the aforementioned modifications. The `rfproxy` application, with OpenFlow 1.3 upgrades, was built using the Ryu controller[1]. We use an OpenFlow 1.3 software switch[2] both in the data and virtual planes and defined a 48 switches folded-Clos topology 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]https://github.com/osrg/ryu
[2]https://github.com/CPqD/ofsoftswitch13

Algorithm 2 (*Select Routes*): Select routes to be mapped

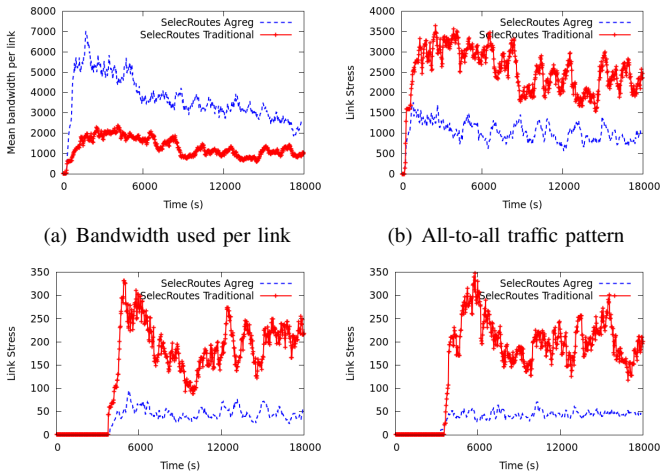
Require: (switch, lxc, NetworkAddressPrefix, RequestedBandwidth)
Ensure: Selected routes for mapping
1: switch_routes ← lxc.get_routes(NetworkAddressPrefix)
2: Select switch_routes with higher switch.bw_port_table capacity that satisfy resilience policy requirements
3: **if** Option *SelectRoutes Agreg* **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**

1,000 between servers and switches. Furthermore, we assume 40 servers per rack 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).

We evaluate the Algorithm 1 and compare the performance of *SelecRoutes Traditional* and *SelecRoutes Agreg* strategies of Algorithm 2. We perform the creation of mapping demands in a Poisson process with VMs being allocated orderly as the availability of servers bandwidth, and so building policies as ToRs traffic matrices were defined by these allocations. 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: number of virtual machines uniformly distributed between 30 and 70; traffic demand between VMs uniformly distributed between 1 and 10 units of bandwidth, defined by all-to-all, all-to-one and one-to-all traffic patterns; mapped virtual networks with residence network time uniformly distributed between 540 and 660 seconds; and total experiment time set in 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 behavior on data plane topologies (Fig. 2) switches. The results obtained show that 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. It is important to mention we obtained the same average of policies allocated for both algorithms, and zero ratio of denied mapping policies in all experiments.

VI. Discussion and Future Work

The load balancing technique implemented is dynamic and proactive as the mapping requests arrive and depart to the *Resource Bookeeping Algorithm* execution in parallel with Algorithm 1. As noted in the experimental results in Figure 2, the proposed algorithm performs efficient load balancing in the network for different traffic communication patterns when compared to the algorithm *SelecRoutes Traditional* option, commonly seen in the literature (*e.g.,* [15] and [16]). We think 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



(a) Bandwidth used per link (b) All-to-all traffic pattern

(c) All-to-one traffic pattern (d) One-to-all traffic pattern

Fig. 2: Link Stress for different traffic patterns

allocations. [20] include discernments that meet the traffic communication abstractions implemented in our work that, for example, can define policies according to the VOC [15] or TAG [18] models for traffic applications abstractions.

Besides the centralized SDN scalability limitations, we have the logical centralized flexibility to easily program mapping rules into the routes selection process by matching on different TCP/IP ports/protocols (e.g., NVGRE, VXLAN) and setting BGP configuration knobs as proposed by [21]. As future work, an interesting topic would be whether any multipath 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. Also, fault tolerance inserted in virtual networks reconfiguration is a work possibility already observed in WANs that can be a simple extension of the proposed architecture and mapping algorithm.

VII. Conclusion

In this paper, we sought to evaluate virtual data center networks mapping 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 explored the mapping of the physical DCN infrastructure to a virtual aggregated topology where routes from the BGP protocol – configured specifically for this environment – are obtained to allocate virtual topology graphs. Through algorithms implemented on the RouteFlow control plane, we were able to show efficient mapping of virtual networks considering bandwidth, load balancing and routing protocol overhead. Finally, we think topics related to the policies abstractions proposed allows to address several DCNs problems by the extension of this work, considering different operational and practical requirements for these 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] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[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] 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.

[21] 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.