A Scalable Container-based Virtualized Data Center Emulation Framework

Gaurav Gautam, Sandhya Rathee, Preetam Patil, and Parimal Parag
Centre for Networked Intelligence, RBCCPS and EECS Division, Indian Institute of Science, Bengaluru, India
Email:{gauravgautam, sandhyar, preetampatil, parimal}@iisc.ac.in.

2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS) | 978-1-6654-2104-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/COMSNETS53615.2022.9668443

***Abstract*—Data Center Networks (DCNs) powering cloud services as well as social networks pose unique network design/optimization challenges. In response, numerous solutions have been proposed utilizing SDN, programmable data plane, and telemetry. However, testbeds supporting the experimental features while imitating DCN traffic—for implementing and analyzing such solutions—are not trivial to build. We present a data center emulation framework for development and validation of data center algorithms and protocols using lightweight virtualized components such as containers and virtual switches. The framework facilitates automated creation of DCN topologies, telemetry, SDN controller, and data center workload generation at scale. The framework also enables combining diverse virtualized elements and hardware elements. This demo exhibits our in-progress framework implementation that is capable of instantiating a leaf-spine topology with P4-capable switches, gNMI telemetry, BGP routing and end-hosts.**

Keywords*—*Data Center, Containers, Mininet, Testbed.

I. Introduction

Cloud service providers delivering on-demand compute, storage, and networking services use massively scalable data centers. So do the social media platforms powering the data-hungry social networking apps. Data center networks (DCNs) consist of a large number of computing and storage servers connected by switches and routers in a well-managed architecture. To satisfy on-demand services and to drive operational efficiency, resources are provisioned dynamically within the data centers. The dynamic nature and massive scale results in increased complexity in configuring and managing DCNs. Owing to the preference towards using Ethernet as a converged infrastructure, DCNs carry traffic with diverse throughput and latency demands. Multi-tenancy in public cloud imposes service level objectives that add further to the complexity. DCNs are shown to suffer from unique problems such as incast [1] and degradation of mice flow performance due to elephant flows [2].

Software-Defined Networking (SDN) proposes to address some of the challenges in DCNs. In SDN enabled networks, a centralized controller—aided by telemetry information—can program the data plane for enhanced control over routing and load balancing [3]. Recent enhancements such as data plane programmability through P4 [4] and in-band network telemetry (INT) [5] have enabled specific solutions for TCP congestion control [6]. Implementing and experimentally validating such proposals requires hardware/virtual devices supporting experimental features such as P4. Further, proposed solutions need to be validated against realistic workloads. To the best of our knowledge, there are no standardised benchmarks to generate traffic faithfully representing represent DCN network workloads. Therefore, we propose to develop an SDN enabled DCN emulation framework that supports customizable topology generation, telemetry, and workload generation.

Mininet [7], [8] is commonly utilised for emulating SDNs. The major limitation of Mininet is that it is not scalable. A mininet instance runs within a single host (shared kernel space). Integrating multiple mininet instances (running on different servers) in non-trivial. The ngSDN [9] platform is useful for experimenting with P4 based network. But ngSDN also uses Mininet to emulate the network topology. These platforms are more general, they are not experiment ready. You have to add your telemetry, routing, and traffic support. We present a scalable container-based virtualized platform for programmable DCN. We provide a scalable topology where you just need to update the script for number of leaf, spine, and host. It uses containers to instantiate the topology both switches and hosts. Thus, it does not require installation of switch binaries in your local system. Also, using the proposed platform the user is given the freedom to choose different programmable switches.

Our proposed solution is not limited to one host. The testbed can be spread over a cluster running over multiple hosts. Our goal to make it a hybrid testbed where we can have real systems, hardware switches, software switch, and containers—all working together seamlessly. To execute any application in mininet, like traffic generation applications, it needs to be installed in the system whereas that is not the case with containers. Containers are useful when we need to run different workloads such as web applications, iperf flows, distributed storage nodes, etc.

II. Desired Features of a DCN testbed

Emulating realistic hyperscale data center behaviour while consuming proportionally miniscule computation and storage resources is a challenge. Our testbed targets the following goals

- Support for different virtualization technologies such as VMs, containers, and microVMs.
- Ability to plug in hardware instances of switches and end-hosts to make it a hybrid (software and bare-metal) test-bed.
- Scale-out architecture: the emulated topology should at least emulate a few repeatable data center design elements such as pods faithfully. Since achieving it—albeit at reduced scale—in a single host is unlikely, the

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on January 18,2022 at 17:42:42 UTC from IEEE Xplore. Restrictions apply.

architecture should support spanning multiple physical servers.

- Workload traffic generation, with customizable traffic profiles that represent data center traffic (such as streaming, distributed databases, map-reduce, distributed storage, and HTTP microservices).
- Support for Network Operating System (such as ONOS).
- Automated creation of data center typologies, e.g., fat-tree, D-cube, Jellyfish, etc.
- Ability to implement and test algorithms and protocols for path optimization, congestion control, etc.
- Border gateways to emulate WAN and inter Data center networking will be an added advantage.

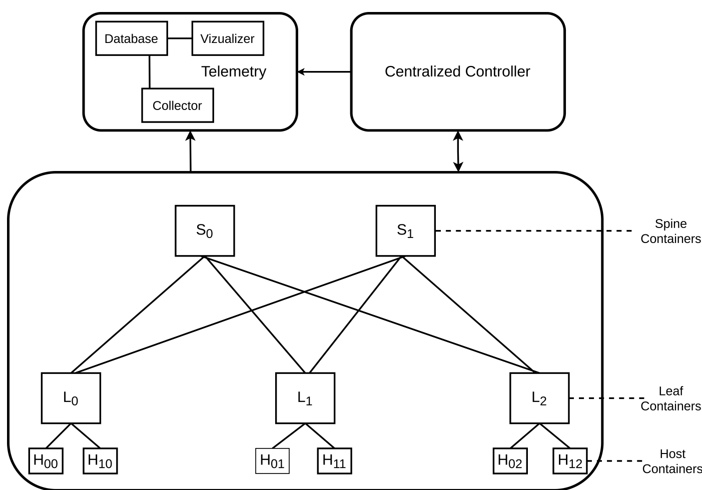

Fig. 1. Proposed architecture

III. Architecture

Figure 1 presents the architecture of our proposed platform. It has four modules:

1) Network topology
2) Device configuration
3) Telemetry collector
4) Controller

The underlying network topology consists of network switches (conventional/programmable) and end-hosts. Network topology module provides choice to the user to choose among different type of topologies—such as fat-tree topology or leaf-spine topology—with a configurable number of switches and hosts as per user requirement. The user can choose type of switch (bmv2, openVswitch, or conventional switch) as well. Once the topology is ready, the devices need to be configured depending on the network policies. The device configuration module translates the network polices into forwarding rules. The controller requires knowledge about the current state of the network for its decision making. To achieve this, the testbed is accompanied with a central telemetry collector. Telemetry collector has three sub-modules: collector, database, and visualizer. It is the responsibility of the collector to fetch telemetry data from underlying network and store it in a database. We use influxdb to store the collected telemetry. For visualization we use Grafana [10] that accesses telemetry information from influxdb [11]. The user needs to specify the parameter to the visualizer module and it will produce the requested graphical representation. Using a centralized controller, we control all these components and write experiment scripts. The centralized controller can be seen as a combination of set of utilities to control containers, routing, and telemetry. The proposed architecture provides flexibility to add individual iperf [12] flows and traffic profile for mixed traffic. One can change host container image to add traffic of one's own choice. The workflow of our experiment is given in figure 2.

Python3 is being used as the primary language to write libraries and experiments. Docker instances are controlled by docker-py [13] API and system commands. One can send commands to control container using docker-py or ssh. An additional server is running on each container instance using which important data can be pulled by socket programming. Containers are used to compile and push P4 code to data plane, and these containers are built upon containers and libraries provided by P4 tutorial [14], and ngSDN [9].

IV. Demo Results

In this demo, we demonstrate the capabilities of our container-based DCN testbed to (a) create and start a virtualized topology, (b) configure the network using libraries, (c) run traffic profiles, and (d) collect telemetry data. We also present results from preliminary experiments using the testbed to share some insights about the scalability aspects and demonstrate the completed features.

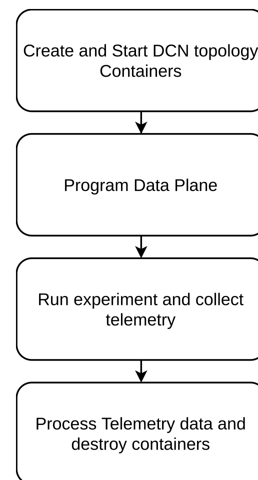

Fig. 2. DCN experiment workflow

Currently, the implementation supports scalable leaf-spine topology. A simple container acts as a switch with BGP routing using FRR[15]. It also has a capability to use P4-programmable bmv2[16] switch. Telemetry data from the switches can be collected over the gNMI protocol .

One such spine-leaf topology is illustrated in Figure 3. Each spine switch is connected to every leaf switch and every leaf switch is connected to two hosts. The user can vary the

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on January 18,2022 at 17:42:42 UTC from IEEE Xplore. Restrictions apply.

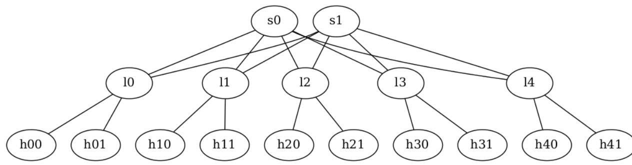

Fig. 3. Topology visualization generated by DCN framework

number of spine switches, leaf switches, and hosts through a configuration file.

We conducted the following experiments using a VM having 32 CPUs and 62GB RAM for results here. Figure 4 demonstrates the time taken to create and start the topology consisting of containers and links. Here, the number of spine switches and hosts per leaf are constant – 2 spine switches and 8 hosts per leaf switch. The number of links vary w.r.t to the number of leaf switches, there will be ten links per leaf node (i.e., two links for spine switches, and eight links for hosts). Time taken to create and start the containers and links increases w.r.t to increase in the number of leaf switches or links in the network.

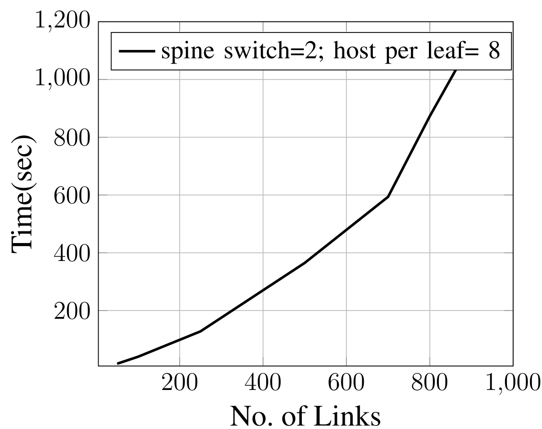

Fig. 4. Number of links vs. time to create and start topology for leaf-spine topology

Figure 5 illustrates the bandwidth utilization of a flow. The chart on top shows the bandwidth utilization of the link connecting spine switch and leaf switch and the bottom chart is showing the bandwidth utilization of the link connecting the leaf switch and the host. The charts are automatically updated every 5 seconds.

The iperf measurements on the VM host with no containers running yielded an aggregate 149 Gbps throughput. When we started our topology with 2 spines, 60 leaves and 2 hosts/leaf (without any workloads), the iperf throughput on VM host dropped to 100 Gbps. Next, we started iperf workloads in the container topology. When traffic was taking one hop route(host->leaf->host), we found throughput to be saturating around 35 Gbps. We observed that increasing the number of hops results in reduced throughput. This indicates that the network stack processing overheads are significant in the container environment. We are currently investigating the bottlenecks in our framework further.

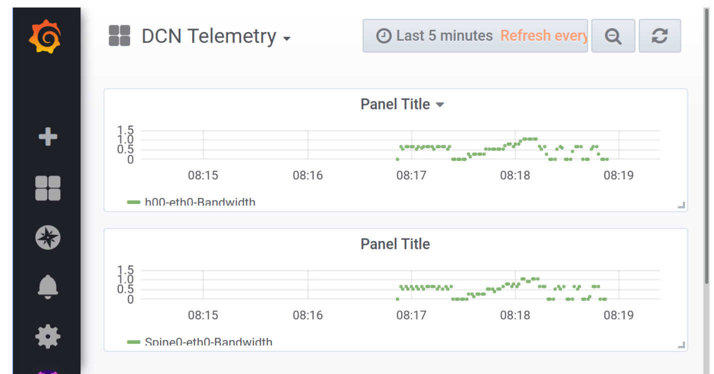

Fig. 5. Example screenshot of DCN telemetry using grafana

To demonstrate data plane programmability features of our testbed, we also conducted experiments on INT [5] wherein we were able to trace the path and measure queue latency of packets (results omitted due to lack of space).

V. Acknowledgment

This work is supported by the Centre for Networked Intelligence (a Cisco CSR initiative) at the Indian Institute of Science, Bengaluru.

References

[1] W. Chen, F. Ren, J. Xie, C. Lin, K. Yin, and F. Baker, "Comprehensive understanding of tcp incast problem," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 1688–1696, IEEE, 2015.

[2] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, (New York, NY, USA), p. 267–280, Association for Computing Machinery, 2010.

[3] Y.-C. Wang and S.-Y. You, "An efficient route management framework for load balance and overhead reduction in sdn-based data center networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1422–1434, 2018.

[4] "P4 open source programming language." https://p4.org/. Accessed: 21-11-2021.

[5] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, vol. 15, 2015.

[6] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, *et al.*, "Hpcc: High precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 44–58, 2019.

[7] "Mininet project." http://mininet.org/. Accessed: 18-11-2021.

[8] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM Workshop on Hot Topics in Networks*, p. 19, ACM, 2010.

[9] "Next-gen sdn tutorial (advanced)." https://github.com/opennetworkinglab/ngsdn-tutorial. Accessed: 18-11-2021.

[10] "Grafana dashboard." https://grafana.com/. Accessed: 18-11-2021.

[11] "Influxdb." https://www.influxdata.com/. Accessed: 18-11-2021.

[12] "iperf tool." https://iperf.fr/. Accessed: 18-11-2021.

[13] "Docker sdk for python." https://github.com/docker/docker-py. Accessed: 18-11-2021.

[14] "P4 tutorial." https://github.com/p4lang/tutorials. Accessed: 18-11-2021.

[15] "Frrouting project." https://frrouting.org/. Accessed: 18-11-2021.

[16] "Behavioral model (bmv2, software p4 reference switch)." http://bmv2.org/. Accessed: 18-11-2021.

Authorized licensed use limited to: J.R.D. Tata Memorial Library Indian Institute of Science Bengaluru. Downloaded on January 18,2022 at 17:42:42 UTC from IEEE Xplore. Restrictions apply.