DEMO: An Open Research Framework for Optical Data Center Networks

Yiming Lei
Max Planck Institute for Informatics

Federico De Marchi
Max Planck Institute for Informatics

Raj Joshi
Harvard University

Jialong Li
Max Planck Institute for Informatics

Balakrishnan Chandrasekaran
Vrije Universiteit Amsterdam

Yiting Xia
Max Planck Institute for Informatics

ABSTRACT

Optical data center networks (DCNs) have emerged as a promising design for cloud network infrastructure. However, research in this field is constrained by the requirement for specialized hardware and software stacks and the high engineering barriers of building optical testbeds. To address these challenges, we present an experimental platform that can realize diverse optical DCN architectures in a plug-and-play manner. We demonstrate the ease of realizing different architectures with Python scripts and show performance comparisons of running end-to-end cloud applications.

CCS CONCEPTS

• **Networks** → **Data center networks**; **Programmable networks**; **Network experimentation**;

KEYWORDS

Data Center Networks, Optical Networks

ACM Reference Format:
Yiming Lei, Federico De Marchi, Raj Joshi, Jialong Li, Balakrishnan Chandrasekaran, and Yiting Xia. 2024. DEMO: An Open Research Framework for Optical Data Center Networks. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM Posters and Demos '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3672202.3673712

1 INTRODUCTION

Optical data center networks (DCNs) have arisen as a promising cloud network infrastructure design in the post Moore's law era for merchant silicon. Over the years, numerous optical DCN architectures using different optical switching hardware have been proposed [4, 6–11, 15–17, 19–22, 24–26], particularly the recent trend of traffic-oblivious optical DCNs that use high-speed optical circuit switches (OCSes) to rotate circuit connections between Top-of-Rack switches (ToRs) according to a predefined topology schedule [4, 5, 16, 17].

However, network and system research for optical DCNs is fundamentally constrained by the underlying optical hardware. For one thing, each optical DCN architecture requires specialized hardware and software stacks, obscuring innovation across domains. For another, due to the high engineering barrier of building optical testbeds, evaluation of architecture-specific software solutions has to rely on home-grown simulators [4, 5, 16] or limited host-based emulation without an actual network fabric [18].

This work aims to bridge the gap between the diverse optical DCNs architectures and the lack of a unified experimental platform. We present a general framework that enables the plug-and-play realization of different optical architectures. Towards that, we abstract the fundamental building blocks for optical DCNs, including a programmable ToR system, a connection toolbox, and a network management plane with configuration and monitoring APIs. We demonstrate the simplicity of realizing different optical architectures with customized topology and routing using Python scripts of approximately 50 lines of code. We also show side-by-side performance comparisons of software solutions across hardware architectures running end-to-end cloud applications.

Table 1: Framework User APIs.

Category	APIs	
Topology	**connect**(ToR1, port1, ToR2, port2, slice)	
	round_robin(#ToRs, #ports)	
	round_robin_offset(#ToRs, #ports)	
	round_robin_dimension(#ToRs, #ports, h)	
Routing	**routing**(**routing_fn**)	**fn_direct**(src, dst, slice)
	neighbors(ToR, slice)	**fn_vlb**(src, dst, slice)
	earliest_path(src, dst, slice, max_hop)	**fn_ebs**(src, dst, slice)
		fn_opera(src, dst, slice)
	entries(paths, lookup_type, multipath_policy)	**fn_hoho**(src, dst, slice)
		fn_ucmp(src, dst, slice)
Monitoring	**buffer_usage**(ToR)	
	bandwidth_usage(ToR, port)	
	drop_rate(ToR)	

2 USER API

We define API functions, as listed in Table 1, for users to program high-level network protocols as Python scripts without worrying about the low-level system implementation.

Topology APIs. The primitive function for topology configuration is *connect( )*. It adds an optical circuit between two ToRs through their ports in a time slice when the circuit stays still. Using this primitive, we provide built-in functions to generate topology schedules for existing optical DCN architectures, including the *round-robin* schedule enumerating connections across ToR pairs [4, 17], and round-robin variants such as the Opera [16] schedule with a port *offset* in circuit rotation and *h-dimensional* round-robin in EBS [3, 23]. Users can define customized topology schedules in the same way, such as the random connections exemplified in Code 1c. Our framework compiles the user-defined schedule into OCS connections and guides the optical controller to set them up in specified time slices.

ACM ISBN 979-8-4007-0717-9/24/08.
https://doi.org/10.1145/3672202.3673712

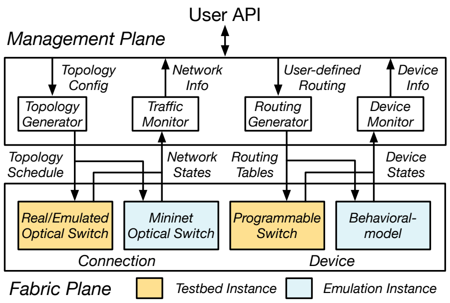

Figure 1: System diagram.

```
net = Network()
net.topology_random(tors=8, ports=2)
paths = net.routing(routing_fn = net.fn_vlb)
entries(paths, lookup_type=SOURCE,
        multipath_policy=RANDOM)
```

(a) Steps to set up a network.

```
def fn_vlb (src, dst, time_slice):
    paths = []; slice1 = time_slice
    for h1 in neighbors(src, slice1):
        (slice2, dst) = earliest_path(src=h1,
        dst=dst, time_slice=slice1, max_hop=1)
        paths.append([(slice1, h1),(slice2, dst)])
    return paths
```

(b) Realizing VLB routing.

```
def topology_random(self, tor_num, port_num):
    slice_num = tor_num * port_num - 1
    for slice_id in range(slice_num):
        ports = list((tor,port) for tor in range(tor_num)
            for port in range(port_num))
        random.shuffle(ports)
        while ports: #ports not empty
            tor1, port1 = ports.pop()
            tor2, port2 = ports.pop()
            self.connect(tor1, port1, tor2, port2, slice_id)
```

(c) Realizing random connections.

Code 1: Code snippets of network applications using the APIs.

Routing APIs. Despite the diversity of routing algorithms for optical DCNs, we define a signature prototype, *routing_fn( )*, that returns a set of paths for a prospective packet from a source ToR to a destination ToR in a particular time slice. The prototype is passed to *routing( )* to generate all paths for all source-destination pairs in all time slices. We materialize the prototype for existing routing algorithms, including *direct-path* [17], *VLB* [4, 17], *EBS* [3, 23], *Opera* [16], *HOHO* [14], and *UCMP* [13], and allow users to implement customized routings following the signature format.

We also provide helper functions: *neighbors( )* that returns connected neighbors for a ToR in a time slice, and *earliest_path( )* that returns the first path between a source ToR and a destination ToR since a given time slice. As maximizing throughput and minimizing latency are common objectives for optical DCNs, these helper functions serve as building blocks for specific routing algorithms, and Code 1b shows how VLB routing is implemented with them.

Finally, we offer *entries( )* to translate paths into lookup table entries and load them onto each ToR. The framework supports source routing and per-hop lookup, with entries compiled accordingly. For multi-path routing, runtime path selection policy is needed. We enable random [4], flow-size-based [13], and queue-occupancy-based [3, 4] selection as required by existing routing algorithms.

Monitoring APIs. Our framework also exposes telemetry APIs for monitoring the network performance, such as the *buffer usage*, *bandwidth usage*, and *packet drop rate* of a ToR, and we will extend them as future requirements arise.

3 FRAMEWORK DESIGN

Fig. 1 illustrates the system diagram for our framework. It consists of the management plane and the fabric plane. The management plane is responsible for network configuration and monitoring, through the user APIs described in §2. The fabric plane mainly comprises the ToR system implemented on programmable switches, along with connectivity management of the OCSes and the integral host system. The host system uses the VMA [2] high-performance user-space library to implement flow pausing [17] and flow aging [13] as required by some routing algorithms.

Our framework can operate in either a testbed environment with actual devices or an emulation environment on Mininet, lowering the hardware barrier for researchers and students. The testbed mode supports various types of OCSes, as well as emulated ones over programmable switches.

As the key component of the framework, the ToR system realizes three major functionalities: (1) time synchronization, (2) routing lookup, and (3) time-scheduled packet forwarding. For (1), we synchronize the ToRs with the optical controller over the varying optical circuits and achieve 28ns sync accuracy. For (2), we abstract a generic routing table format that matches a packet's arrival time slice and destination ToR to determine the departure time slice and egress port for sending the packet. For (3), we leverage the queue pausing feature of Tofino2 switches [12] to buffer packets when the departure time slice is later than the arrival time slice.

4 DEMONSTRATION

We demonstrate our framework on a testbed with 3 Intel Tofino2 switches and 4 servers, each with a ConnectX-6 dual-port NIC. We virtualize 2 physical switches into 4 logical ToRs each, and create 2 virtual hosts on each server by splitting the NIC interfaces into separate namespaces. To support various optical DCN architectures, we emulate OCSes on a Tofino2 switch with tunable time slice durations. Our setup thus contains 8 logical ToRs, each connected to a logical host with a 100Gbps downlink and to the OCSes with 4 10Gbps uplinks to mimic oversubscription in production DCNs.

Architecture implementation with the user APIs. We first demonstrate how our framework simplifies the implementation of optical DCN architectures. As shown in Code 1a, an architecture can be populated with just a few lines of Python code by calling API functions for the topology and routing schemes. We demonstrate the effect of each line of code, by visualizing the generated topology schedule and showing the lookup entries loaded to each ToR.

Code 1b and 1c dive into the implementation of topology and routing functions. In Code 1b, the built-in API *fn_vlb( )* realizes two-hop routing. Hop 1 is via possible ToRs directly connected to the source, obtained using *neighbors( )*. Hop 2 is over the earliest direct circuit from each intermediate ToR to the destination, by calling *earliest_path( )*. Code 1c shows a self-defined topology function to create random connections throughout time slices, by drawing randomly from unused ports and adding circuits through *connect( )*.

Side-by-side comparison running applications. We implement existing topology and routing methods as shown above and perform side-by-side performance comparisons between then with real applications. We generate latency-sensitive traffic with Memcached [1] and generate throughput-intensive traffic with iPerf. The framework enables logging of traffic statistics, such as flow completion times (FCTs). We demonstrate FCT distributions across different routing solutions. We observe that under $50\mu s$ time slices, HOHO exhibits 27% and 88% reduction in 99th percentile FCT for Memcached traffic compared to Opera and VLB, respectively.

REFERENCES

[1] [n. d.]. Memchached. https://memcached.org/. ([n. d.]).

[2] 2024. Mellanox Messaging Accelerator. https://github.com/Mellanox/libvma/blob/master/README. (2024).

[3] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal. 2022. Optimal oblivious reconfigurable networks. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 1339–1352.

[4] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, et al. 2020. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 782–797.

[5] Joshua L Benjamin, Thomas Gerard, Domaniç Lavery, Polina Bayvel, and Georgios Zervas. 2020. PULSE: optical circuit switched data center architecture operating at nanosecond timescales. *Journal of Lightwave Technology* 38, 18 (2020), 4906–4921.

[6] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. 2013. OSA: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking* 22, 2 (2013), 498–511.

[7] Kai Chen, Xitao Wen, Xingyu Ma, Yan Chen, Yong Xia, Chengchen Hu, and Qunfeng Dong. 2015. WaveCube: A scalable, fault-tolerant, high-performance optical data center architecture. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1903–1911.

[8] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. 2017. Enabling Wide-Spread Communications on Optical Fabric with MegaSwitch. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 577–593.

[9] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2010. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 339–350.

[10] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 216–229.

[11] Mehrdad Khani, Manya Ghobadi, Mohammad Alizadeh, Ziyi Zhu, Madeleine Glick, Keren Bergman, Amin Vahdat, Benjamin Klenk, and Eiman Ebrahimi. 2021. SiP-ML: high-bandwidth optical network interconnects for machine learning training. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 657–675.

[12] Jeongkeun Lee. 2020. Advanced congestion & flow control with programmable switches. In *P4 Expert Roundtable Series*. https://bit.ly/3J8x7fw

[13] Jialong Li, Haotian Gong, Federico De Marchi, Aoyu Gong, Yiming Lei, Wei Bai, and Yiting Xia. [n. d.]. Uniform-Cost Multi-Path Routing for Reconfigurable Data Center Networks. In *Proceedings of the ACM SIGCOMM 2024 Conference*.

[14] Jialong Li, Yiming Lei, Federico De Marchi, Raj Joshi, Balakrishnan Chandrasekaran, and Yiting Xia. 2022. Hop-On Hop-Off Routing: A Fast Tour across the Optical Data Center Network for Latency-Sensitive Flows. In *Proceedings of the 6th Asia-Pacific Workshop on Networking*. 63–69.

[15] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. 2014. Quartz: a new design element for low-latency DCNs. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 283–294.

[16] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 1–18.

[17] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. 2017. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 267–280.

[18] Matthew K. Mukerjee, Christopher Canel, Weiyang Wang, Daehyeok Kim, Srinivasan Seshan, and Alex C. Snoeren. 2020. Adapting TCP for Reconfigurable Datacenter Networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 651–666. https://www.usenix.org/conference/nsdi20/presentation/mukerjee

[19] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2013. Integrating microsecond circuit switching into the data center. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 447–458.

[20] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. 2022. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 66–85.

[21] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Eugene Ng, Michael Kozuch, and Michael Ryan. 2010. c-Through: Part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 327–338.

[22] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Zhijao Jia, Dheevatsa Mudigere, Ying Zhang, Anthony Kewitsch, and Manya Ghobadi. 2022. TopoOpt: Optimizing the Network Topology for Distributed DNN Training. *arXiv preprint arXiv:2202.00433* (2022).

[23] Tegan Wilson, Daniel Amir, Vishal Shrivastav, Hakim Weatherspoon, and Robert Kleinberg. 2023. Extending optimal oblivious reconfigurable networks to all n. In *2023 Symposium on Algorithmic Principles of Computer Systems (APOCS)*. SIAM, 1–16.

[24] Dingming Wu, Yiting Xia, Xiaoye Steven Sun, Xin Sunny Huang, Simbarashe Dzinamarira, and TS Eugene Ng. 2018. Masking failures from application performance in data center networks with shareable backup. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 176–190.

[25] Yiting Xia, Mike Schlansker, TS Eugene Ng, and Jean Tourrilhes. 2015. Enabling Topological Flexibility for Data Centers Using OmniSwitch.. In *HotCloud*.

[26] Yiting Xia, Xiaoye Steven Sun, Simbarashe Dzinamarira, Dingming Wu, Xin Sunny Huang, and TS Eugene Ng. 2017. A tale of two topologies: Exploring convertible data center network architectures with flat-tree. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 295–308.