# ExReC: Experimental Framework for Reconfigurable Networks Based on Off-the-Shelf Hardware

Johannes Zerwas
TU Munich, Germany

Chen Avin
Ben-Gurion University of the Negev
Beer-Sheva, Israel

Stefan Schmid
TU Berlin, University of Vienna & Fraunhofer SIT
Germany & Austria

Andreas Blenk
TU Munich & University of Vienna
Germany & Austria

## ABSTRACT

In order to meet the increasingly stringent throughput and latency requirements on datacenter networks, several innovative network architectures based on reconfigurable optical topologies have been proposed. Examples include demand-oblivious reconfigurable topologies such as RotorNet (SIGCOMM 2017), Opera (NSDI 2020), and Sirius (SIGCOMM 2021), as well as demand-aware topologies such as ProjecToR (SIGCOMM 2016). All these architectures feature attractive performance properties using specific prototypes. However, reproducing these experiments is often difficult due to missing hardware and publicly available software. This paper presents a flexible framework for reconfigurable networks based on off-the-shelf hardware, which supports experimentation and reproducibility at a small scale. We describe how our framework, ExReC, can be instantiated with different configurations, allowing us to emulate and study existing architectures. Finally, we demonstrate the application of our approach for different use cases and workloads, including distributed machine learning training.

## CCS CONCEPTS

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

## KEYWORDS

data centers; demand-aware; reconfigurable networks, experiments



*ANCS '21, December 13–16, 2021, Layfette, IN, USA*



ACM ISBN 978-1-4503-9168-9/21/12…$15.00
https://doi.org/10.1145/3493425.3502748

**ACM Reference Format:**
Johannes Zerwas, Chen Avin, Stefan Schmid, and Andreas Blenk. 2021. ExReC: Experimental Framework for Reconfigurable Networks Based on Off-the-Shelf Hardware. In *Symposium on Architectures for Networking and Communications Systems (ANCS '21), December 13–16, 2021, Layfette, IN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3493425.3502748

## 1 INTRODUCTION

The popularity of data-centric applications and cloud-based services in daily life has led to an explosive growth of datacenter traffic. Accordingly, over the last years, significant efforts have been made to improve the throughput of datacenter networks. These efforts also include the physical datacenter topology [11, 24]. A particularly innovative approach revolves around reconfigurable optical topologies [12]: datacenter networks whose physical topology can be changed dynamically over time. In particular, these networks improve throughput by providing "shortcuts": the traffic between, e.g., two racks can be communicated directly, hence avoiding the overheads of multi-hop forwarding (e.g., in terms of bandwidth consumption). These networks are based on emerging optical technologies, such as optical circuit switches or free-space optics. Reconfigurable optical topologies typically rely on reconfigurable switches that provide *dynamic matchings*, e.g., between racks. They can be classified into two types: *demand-oblivious* topologies such as RotorNet [19], Opera [18] and Sirius [2] rely on periodic matchings whereas the dynamic matchings of *demand-aware* topologies such as ProjecToR [10, 16], Eclipse [26], and ReNets [1] are optimized toward the (current) demand.

However, while promising performance results have been demonstrated with various prototypes, the trade-offs between different reconfigurable topologies and their advantages and disadvantages are still not well-understood. What is more, today, it is often challenging to experiment with these technologies, as they are usually based on custom-built prototypes and rely on tailored hardware and software which is not publicly available.

**Table 1: Overview of experimental platforms for reconfigurable networks: name, basic methodology, achieved scale, link rate, if DO is available, if DA is available, if only OTS equipment is used, and if artifacts to reproduce measurements are available.**

| Name | Basis | Scale | Link rate | *Demand-oblivious* | *Demand-aware* | OTS | Source |
|---|---|---|---|---|---|---|---|
| RotorNet [19] | HW | 8 emulated ToRs | 10 Gbps | ✓ Prototype | ✗ | ✗ | n/a |
| Opera [18] | HW | 8 emulated ToRs | 10 Gbps | ✓ emulated (P4) | ✗ | ✓ | n/a |
| Etalon [20] | Emulation | 8 emulated ToRs (128 hosts) | 10 Gbps | ✓ emulated (Click Router) | ✗ | ✓ | ✓ |
| Helios [9] | HW | 24 End-hosts | 10 Gbps | ✗ | ✓ | ✓ | n/a |
| xWeaver[28] | HW | 10 emulated ToRs | 10 Gbps | ✗ | ✓ | ✓ | n/a |
| OSA/WaveCube [4, 5] | HW | 8 emulated ToRs | unclear | ✗ | ✓ | ✓ | n/a |
| Flat-Tree [29] | HW | 24 End-hosts | 10 Gbps | ✗ | ✓ | ✓ | n/a |
| ProjecToR [10] | HW | 3 ToRs | - | ✗ | ✓ Prototype | ✗ | n/a |
| MegaSwitch [6] | HW | 40 End-hosts | 10 Gbps | ✗ | ✓ Prototype | ✗ | n/a |
| Mordia/ReacToR [17, 22] | HW | 23 End-hosts | 10 Gbps | ✗ | ✓ Prototype | ✗ | n/a |
| c-Through [27] | HW | 4 emulated ToRs, 16 hosts | 1 Gbps | ✗ | ✓ emulated (host-based) | ✓ | n/a |
| C-Share [25] | Mininet | 10 leaf switches | 100 Mbps | ✗ | ✓ emulated (OpenVSwitch) | ✓ | n/a |
| This paper | HW | 8 emulated ToRs | 10 Gbps | ✓ emulated (DPDK, VLAN) | ✓ | ✓ | ✓ |

**Our contributions.** This paper presents a flexible framework for building reconfigurable networks that only rely on commercially available off-the-shelf (OTS) hardware. Our framework supports experimentation and reproducibility and can be configured in different ways, allowing us to emulate different existing architectures. In particular, our framework, ExRec, allows emulating hybrid topologies, consisting of both *demand-oblivious* and *demand-aware* components, as they typically appear in the literature [12].

ExRec uses emulation to reduce dependence on hardware that is either expensive or not available. Therefore, it uses an electrical packet switch and a label-based routing approach. Moreover, since today's programmable switches are usually not suited for the packet scheduling logic as required by TDMA-based *demand-oblivious* topologies, the logic is implemented on generic servers. This increases the flexibility for implementing forwarding logic. We demonstrate ExRec in under various workloads, also considering a distributed machine learning (DML) training application.

As a contribution to the research community, as well as to ensure reproducibility, we will make publicly available our entire source code, the exact hardware specifications, and all our experimental artifacts together with this paper.[1]

## 2 BACKGROUND

We consider a reconfigurable topology that embodies a two-layer leaf-spine structure. The spine layer is constituted by $k_{do}$ *demand-oblivious* (DO) and $k_{da}$ *demand-aware* (DA) switches which connect to all leaf nodes. Table 1 summarizes existing experimental evaluations of these two switch types, the scales, and the availability of the components from a hardware and software perspective.

***Demand-oblivious* switches:** Traffic conditions do not drive DO switch configurations. DO switches independently cycle through a pre-determined set of configurations, specifically *matchings*, in a round-robin manner. The matchings are usually chosen so that all top-of-rack (ToR) switches, the leaves, are provided *direct* connectivity to every other ToR within one cycle of all matchings [19]. Each matching is active for time $\delta$, followed by a reconfiguration time of $R_{do}$ to set up the subsequent matching. Thus, a slot lasts for $\delta + R_{do}$. Multiple DO switches can be used in parallel to reduce the length of the cycles [19], hence, to circumvent large periods without connection in the case of many ToRs.

In skewed traffic conditions, DO uses *indirect* forwarding, i.e., packets are forwarded via intermediate ToRs, which has been shown to yield significant efficiency gains in DO topologies [2, 19]. The intermediate ToR, in turn, forwards the packets to the final destination ("2-hop"-forwarding).

DO topologies have been assessed, for instance, in RotorNet [19] and Opera [18]. While RotorNet relies on a specific prototype, Opera is evaluated with a P4 switch-based emulation using only OTS components. Etalon [20] provides an emulation of DO topologies using only OTS components. It emulates multiple ToRs (and racks) per physical server in the testbed and uses time dilation to achieve higher rates. However, it only relies on software-based forwarding solutions and fully virtualizes the topology. The framework is available but only considers a DO topology so far.

***Demand-aware* switches:** DA switches can be configured to create direct connections between ToRs according to the traffic conditions. They have a high degree of freedom and can realize any possible matching. When a circuit is established, it provides constant datarate between ToRs. The reconfiguration time $R_{da}$ is usually larger than for DO switches.

Here, a wider range of experimental evaluations has been conducted. A group of studies relies on commercially available OCSs, e.g., Helios [9], OSA/WaveCube [4, 5], xWeaver [28] or Flat-Tree [29]. Other studies that explore different interconnects, e.g., ProjecToR [10], MegaSwitch [6] or Mordia [22], use prototype implementations that are not available. Finally, DA topologies have also been evaluated using emulations. For instance, c-Through [27] considers

[1] https://github.com/tum-lkn/exrec

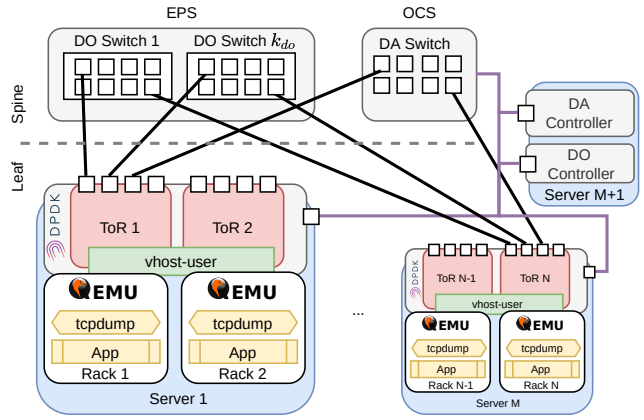



**Figure 1: Overview of ExRec framework.**

host-based scheduling, i.e., the hosts know the active circuits and only send traffic accordingly. C-Share [25] use Mininet [15] and OpenVSwitch for OCS emulation.

All the experimental evaluations of reconfigurable topologies consider either DO or DA switches. In addition, in all but one case, the frameworks are not available.

## 3 THE ExRec FRAMEWORK

ExRec is a flexible framework for building and evaluating different hybrid reconfigurable topologies. It uses OTS equipment for realizing DO and DA switches.

### 3.1 Design Overview

Fig. 1 shows how ExRec emulates a setup with $N$ racks and ToRs (leafs) and $k = k_{do} + k_{da}$ spine switches. This is realized with $M + 1$ physical servers (blue boxes), an electric packet switch (EPS), and an OTS optical circuit switch (OCS) (both grey boxes). $M$ of the servers emulate the ToRs (red boxes), such that each server emulates $x = \frac{N}{M}$ ToRs and racks. The servers have enough physical ports to connect to the EPS that emulates $k_{do}$ DO switches, and to the OCS that realizes the DA topology part with up to $k_{da}$ switches. An additional control server runs two controller processes: one for the DO and one for the DA. The DA process controls the OCS; the DO process sends messages to the ToRs to control the DO links. The control server is connected via a dedicated network.

We use QEMU/KVM [3] to spawn VMs that represent the racks (white boxes). Hosts inside the racks are abstracted: the racks can run either traffic generators like MoonGen [8] and iPerf [14] or applications such as the DML framework Horovod [23] (yellow box "App"). All traffic leaving or entering the VMs is dumped for later analysis. ExRec relies on two design decisions: (1) emulating circuit switching using labels. (2) scheduling traffic directly on the physical servers.

**Label-based routing:** OTS equipment for dynamic topologies is hard to obtain. Either it does not meet requirements for reconfiguration times [30] or it is not available, e.g., in case of *demand-oblivious* Rotor switches [19].

ExRec can emulate a flexible number of DO switches among one or many EPS. The DO component is emulated with a label-based forwarding approach to achieve correct forwarding with low reconfiguration times. The servers add a label that indicates the destination when sending a packet. The EPS is not reconfigured, which reduces complexity.

**Server-based scheduling:** DO topologies rely on buffering and scheduling logic. Implementing such logic is not possible with today's OTS (programmable) switches to the best of our knowledge. Therefore, ExRec moves these tasks to the servers: Each server runs a DPDK application emulating the ToRs. The application fetches traffic from the VMs and schedules it on the links to the DO and DA switches.

### 3.2 Implementation Details

Fig. 2 illustrates the structure of the DPDK application. Each server runs $3 \cdot x + 1$ threads (rounded rectangles) where $x$ is the number of emulated ToRs per server. For every emulated ToR, one TX1, one TX2 and one RX thread are created. One Sync thread is shared by all ToRs on a physical server. We use the DPDK *vhost* library to receive and send packets from and to the VMs.

**Emulating *demand-oblivious* switches:** Transmission via DO switches orients at the design of RotorNet [19], which introduces two sets of destination-based queues per ToR. Such queueing is not available in OTS switches, another reason for emulation. The DPDK application buffers packets leaving the rack in the "local" queues (TX1:2b). For indirect forwarding, packets that are received but are not destined for the ToR (RX:2b), i.e., for which the ToR is an intermediate hop, are put to the "non-local" queues.

The label-based routing uses VLAN tags. When sending, the DPDK application first reads per-queue and per-DO budgets (TX2:1). The budgets indicate per DO link how many packets of each queue can be sent. Then, the thread pulls packets from the corresponding queues (TX2:2b) and adds a VLAN tag that indicates the active matching (TX2:3b). On the EPS, there are pre-installed, static rules that resemble the matchings of the DO switches. The EPS matches the VLAN tag and the incoming port and forwards the packet to the corresponding outgoing port. This source routing-based approach eliminates the need for updating rules on the switch and, thereby, reduces the achievable slot size of the DO emulation. Upon packet reception (RX:1), the DPDK application forwards the packets to the connected racks based on the incoming port of the packet and the active matching (RX:2a). If the packet is not destined to the rack, it is put into the respective non-local queue (RX:2b).

The DO Controller is responsible for cycling through the matchings. It is implemented on top of MoonGen [8];

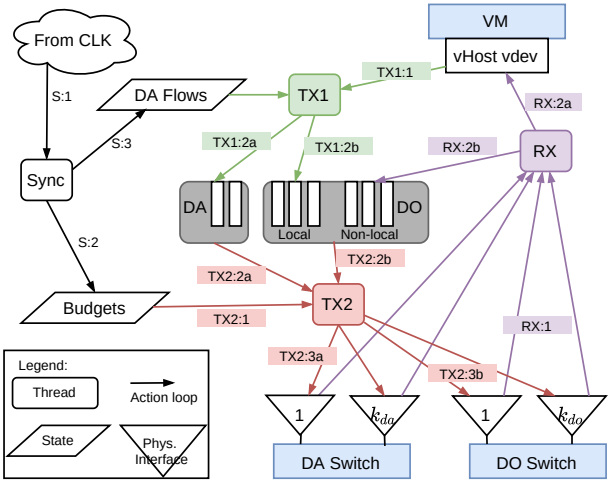



**Figure 2: Overview of the DPDK-based ToR emulation.**

this provides an easy adaptation of slot sizes and emulated reconfiguration times. The DO `Controller` sends the next active VLAN id to all servers. This way, all ToRs use the same VLAN id for the to be sent packets during one slot.

The Sync thread reads VLAN ids from the DO `Controller`'s packets (S:1) and updates the active matchings of the ToRs. This, in turn, triggers a recalculation of the budgets (S:2). Modifying the budget calculation can realize different forwarding policies such as forwarding only directly or valiant load balancing [19]. The current state does not synchronize remaining capacities across the ToRs.

This VLAN-based approach results in negligible reconfiguration time w.r.t. the achieved slot size. Therefore, a dedicated VLAN id indicates reconfiguration of the DO switch. When this VLAN id is received, the application sets all budgets to 0, i.e., stops sending until it receives a VLAN id for a valid matching. Each server runs only one instance of DPDK application which handles the traffic for all emulated ToRs with separate queues and threads. This reduces the jitter of our synchronization method.

**Integrating *demand-aware* switches:** While DO are not yet commercially available, OCSs for realizing DA switches are. Several ports on each NIC can connect via fiber to a commercial, full crossbar OCS. Such an OCS can change the connectivity between the ToRs at run-time: it provides bidirectional links that are reconfigured on-demand in approximately $R_{da} \approx 25$ ms [21]. The DA `Controller` activates the optical links at run-time and then updates the DA flows inside the DPDK application via control plane messages (S:3). DA flows are matched using Layer 3 (source and destination IPs) and Layer 4 (source and destination ports) information. Their packets are put into dedicated queues per DA switch (TX1:2a). Dividing the queues between DA and DO switches avoids head of line blocking in the queues of the DO links.

Once packets are enqueued for the DA switch, even if there would be an upcoming matching on DO switches, they cannot be sent to a DO anymore — using multiple paths is planned for future work. TX2 fetches the packets from the queues and forwards them accordingly (TX2:2a). All DA queues are served in a round-robin fashion, i.e., TX2 fetches a limited number of packets from each queue. Baseline measurements using MoonGen confirm that this mechanism is enough to saturate the DA switch. Moreover, ExRec assures that no other bottlenecks besides the network occur, e.g., by CPU shaping between links in the DPDK application.

Note that ExRec uses an OTS OCS since it is available. Moreover, we think that using available components adds credibility to the measurements. In principle however, the DA switch could also be emulated similarly to the DO switches.

## 4 VALIDATION & EVALUATION

This section first gives details on our testbed and the considered settings. Then, we demonstrate (1) validation of our control mechanism, (2) evaluation of different traffics, and (3) an application example using DML training.

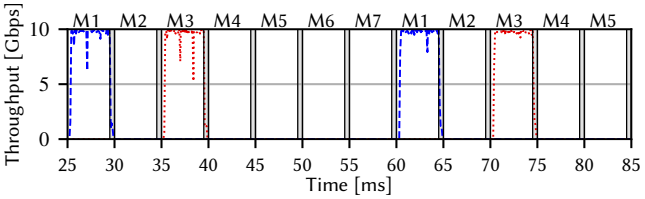
### 4.1 Testbed & Settings

Our testbed consists of four servers running Ubuntu 18.04 (5.15.0-47-generic kernel) with 128 GB of RAM and Intel Xeon Silver 4114 @ 2.2 GHz (20 cores). All servers are equipped with two Intel X710-DA4 4x10G NICs and two 1G Intel onboard-NICs (eight 10G ports and two 1G ports). In principle, this allows exploring configurations from fully DO (four DO switches) to fully DA (four DA switches). For DO emulation, we use a Dell S4048-ON OpenFlow-capable switch [13] and for DA switches a Polatis Series 6000n 32x32 OCS [21]. Based on intensive measurements, the most stable achieved inter-arrival time of control messages is $\approx 0.5$ ms. Hence, to obtain a duty cycle of 90%, all scenarios use a slot of $\delta = 4.5$ ms with an artificial reconfiguration time of $R_r = 0.5$ ms for DO.[2]

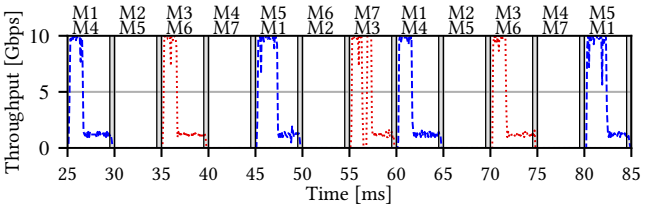
### 4.2 Validation of Topology Components

We verify the behavior of DO and DA switches using four settings: 1 DO switch, 2 DO switches, 1 DO, and 1 DA switch, and 1 DO with indirect forwarding. In particular, for DO switches, we demonstrate how flow rates behave over time when connections between ToRs exist only when matchings are available (cf. §2). For applications that are constantly sending traffic, packets might be buffered if the current matchings do not connect the source and the destination of a flow. We consider setups with 8 ToRs so that there are 7 matchings to be cycled through. Two MoonGen instances generate two packet

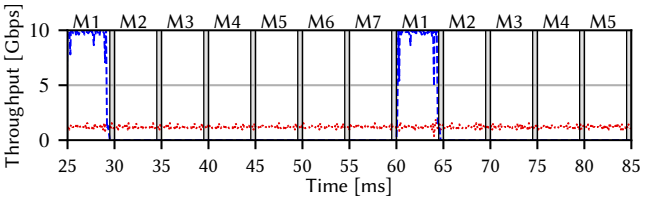
[2]No cross traffic affects our separated management network; however, slight variances in our clock due to hardware or software interference on the controller machine or switch are possible.



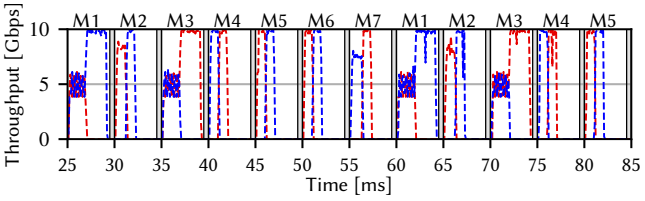

**(a) 1 DO switch.**




**(b) 2 DO switches.**




**(c) 1 DO and 1 DA switches.**




**(d) 1 DO with indirect forwarding.**

**Figure 3: Throughput over time for four configurations.**

streams with constant rate of 1.25 Gbps. They originate at two different ToRs but have the same destination ToR.

Fig. 3 shows the throughput of the flows for a period of 60 ms from the steady-state phase. White areas indicate the slots for DO topology with matchings per DO switch at the top (M1 - M7). Grey shaded areas are periods of reconfiguration.

With one DO (Fig. 3a), the matching M3 serves the red flow, between 35 to 39.5 ms and 70 to 74.5 ms with 10 Gbps. The flow saturates the link for roughly the whole duration of the slot. During the other 6 matchings and the reconfiguration times, packets sent by the source are buffered. The data accumulates to 1.25 Gbps $\cdot$ (6 + 1) $\cdot$ 5 ms = 43.75 MBit. This approximately matches the data that can be sent within one slot (10 Gbps $\cdot$ 4.5 ms = 45 MBit). Thus, when M3 is active, the slot is almost fully utilized at 10 Gbps.

After six slots, the red flow sends again with 10 Gbps for the whole slot duration. This shows (1) that DO emulation allows applications to send within the available slots and (2) that while there is no connection available, traffic is buffered — the desired behavior. The blue flow has the same behavior as the red flow, however, sending when M1 is active.

With the second DO switch (Fig. 3b), the time until a pair of ToRs has a direct connection again is approximately halved. Again, M3 serves the red flow. During this time, all traffic is sent at 10 Gbps until the buffers are empty. Then, the rate drops to 1.25 Gbps, the rate of the traffic source. The red flow is served from 35 to 39.5 ms via DO 1 (upper row), from 55 to 59.5 ms via DO 2 and from 70 to 74.5 ms via DO 1 again. From 39.5 to 55 ms, traffic is buffered during three slots (plus the duration of reconfigurations). Between 59.5 and 70 ms, traffic is buffered only during two slots. These different levels of buffer occupancy reflect in the utilization. The rate of the red flow is longer > 1.25 Gbps during the matching from 55 to 59.5 ms. The blue flow shows a similar, shifted behavior.

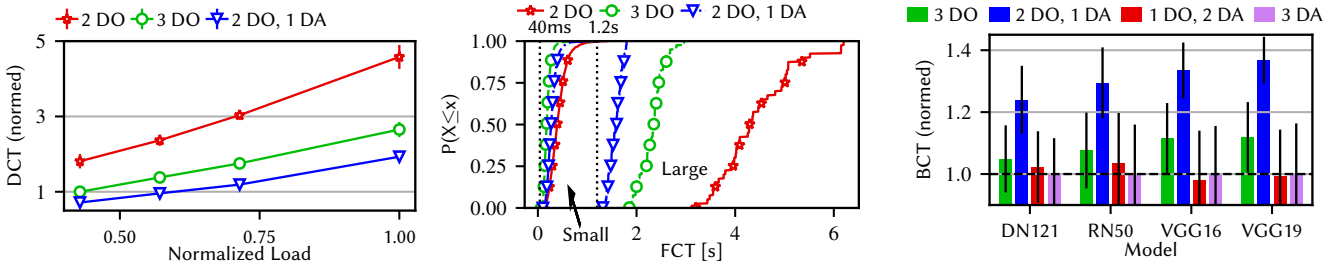
Fig. 3c illustrates the effect of having one DO and one DA switch. The DO switch serves the blue flow as expected: traffic is received only when a matching (M1) is established. In contrast, the DA switch serves the red flow via a direct link between source and destination. The flow constantly sends at the configured rate of the source (1.25 Gbps).

Fig. 3d visualizes 1 DO switch with indirect forwarding. Here, the flows are generated with 1.5 Gbps, and we use a simple policy that dedicates 20% of the remaining volume of a slot to send traffic indirectly. As a result, both flows are served in all slots and use only 20% of the slots when sent indirectly. We observe two behaviors while sending indirect traffic. First, simultaneous sending (M1, M3) and second, sequential sending (M2, M4, M5, M6, M7). The effect comes from how packets are stored on the intermediate ToRs. Overall, this demonstrates that the budget calculation function can realize different forwarding policies. A full re-implementation of RotorLB [19] is out-of-scope.

## 4.3 Measuring & Evaluating Traffic

The next evaluation focuses on more complex traffic scenarios with different traffic intensities and three topology settings (2 DO, 3 DO, and 2 DO & 1 DA) with 8 ToRs. We generate two types of flows: small flows and large flows. The small flows have a size of 6.25$MB$. Both source and destination of flows are sampled uniformly from the 8 ToRs. These flows should be sent via the DO switches since scheduling on a DA switch does not amortize due to its larger reconfiguration time. Besides, each experiment has 4 large flows with a size of at least 250$MB$. With ideal traffic management, these flows should be transmitted via DA switch connections. We vary the arrival rate of all flows and the size of the large ones to increase the load while keeping the ratio between the volume of small and large flows at 2 : 1.

Fig. 4a shows the demand completion time (DCT: finishing all flows) of each setting. The load is normalized by the




(a) DCT.

(b) FCT by flow size (load = 0.43).

**Figure 4: Comparison of DCT (a) and FCT (b) for 3 configurations. Vertical lines indicate ideal transmission times via DO (small flows) and DA switch (large flows).**

**Figure 5: Relative impact of configuration on BCT.**

maximum stable load. DCTs are relative to the result for 3 DO and smallest load. For all settings, the DCT increases with increasing load. The DCT decreases when using more DO switches (3 instead of 2). 2 DO & 1 DA, where the DA switch specifically serves the large flows, further reduces the DCT.

Fig. 4b shows the individual FCTs for a normalized network load of 0.43. The CDFs are separated by flow size, i.e., small flows are the lines left in the figure, large flows on the right. As expected, small flows finish all faster than large flows. Using a DA switch improves the FCT of large flows.

Note the following detail when comparing 3 DO to 2 DO & 1 DA. The topology impacts both flow classes, but the effects are different. For the large flows, 2 DO & 1 DA performs best as it provides a constant rate for them. For the small flows, 3 DO performs best. Here, the waiting time between the slots of a ToR pair is reduced from ≈ 17 ms (2 DO) to ≈ 12 ms, i.e., by one slot. This may be significant for delay-sensitive flows.

## 4.4 Real Application: Distributed ML

ExRec can run real applications, such as the industry-standard DML framework Horovod [23]. Each server has only one Nvidia Tesla T4 GPU [7]; hence, this setup considers only 4 ToRs. We train four models of different sizes: DenseNet121 (DN121), ResNet50 (RN50), VGG16 and VGG19 with different topology configurations for 50 batches (training steps) and report the batch completion time (BCT) in Fig. 5. The values are normalized per trained model to the 3 DA case, which is a full-meshed network, i.e., optimal here.

The observations across the models are consistent. 3 DA obtains the lowest BCT. It is closely followed by 1 DO, 2 DA which uses the DA links to form a ring that matches the mentioned traffic pattern of Horovod. All large flows are forwarded via this ring of DA links and hence, efficiently served. With 2 DO, 1 DA this ring cannot be formed. Here, the BCT is ≈ 22% larger for the smallest model, DenseNet121, compared to the ideal case. Finally, for 3 DO, the BCT is lower again. With 3 DO, the full-meshed is created again in every slot. However, interruptions due to reconfiguration lead to higher batch durations. The average is increased by ≈ 5−10%.

# 5 DISCUSSION & CONCLUSION

This paper presented ExRec, a flexible framework for building reconfigurable networks, which only relies on OTS hardware. We show that ExRec can assess the performance of different topologies combining DO and DA switches. In the following, we discuss the main features and limitations:

**Flexibility.** ExRec can run different combinations of DO and DA topologies going from all DO to all DA with the same testbed setup. Moving the flow scheduling complexity to the physical servers allows the implementation of various forwarding policies for the DO part. By modifying the budget calculation, the forwarding policies can easily be varied. The DA topology can be controlled by an external process, using the API of the OCS and updating forwarding rules in ExRec.

**Scalability.** Due to the available hardware, our evaluation is limited to 8 ToRs. By adding more physical servers, ExRec can also realize larger settings. The size is ultimately limited by the number of VLAN ids, which puts an upper bound on the number of matchings. The size of the spine layer ($k$) depends on the ports per physical server and on the switches. Higher link rates are constrained by the CPU's performance.

**OTS components.** ExRec only uses OTS components, generic servers, OpenFlow switches, and OCSs. A label routing-based emulation replaces DO switches that are not commercially available. In principle, this approach can also be applied to the DA topology. However, using available components as far as possible reduces the assumptions underlying the measurements.

# ACKNOWLEDGEMENTS

This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 864228 - AdjustNet). The work was also funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 438892507. The authors alone are responsible for the content of the paper. We also thank Amaury van Bemten for the fruitful discussions.

## REFERENCES


[1] Chen Avin and Stefan Schmid. 2021. ReNets: Statically-Optimal Demand-Aware Networks. In *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*. 25–39.

[2] 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 *Proc. ACM SIGCOMM*. 782–797.

[3] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *Proc. USENIX ATC*. 1–6.

[4] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. 2014. OSA: An Optical Switching Architecture for Data Center Networks With Unprecedented Flexibility. *IEEE/ACM Transactions on Networking* 22, 2 (April 2014), 498–511.

[5] 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 *Proc. IEEE INFOCOM*. 1903–1911.

[6] 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 *Proc. 14th USENIX NSDI*. 577–593.

[7] Nvidia Corporation. 2019. Nvidia Tesla T4 GPU Datasheet. (2019). Retrieved November 26, 2021 from https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf

[8] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Proc. ACM IMC*. 275–287.

[9] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2011. Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM CCR* 41, 4 (2011), 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 *Proc. ACM SIGCOMM*. 216–229.

[11] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. 2009. BCube: a high performance, server-centric network architecture for modular data centers. *Proc. ACM SIGCOMM*, 63–74.

[12] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. 2021. A Survey of Reconfigurable Optical Networks. *Optical Switching and Networking (OSN), Elsevier* 41 (2021), 100621.

[13] Dell Inc. 2020. Dell S4048-ON Datasheet. (2020). Retrieved November 26, 2021 from https://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell-EMC-Networking-S4048-ON-Spec-Sheet.pdf

[14] iPerf [n. d.]. https://iperf.fr/. ([n. d.]).

[15] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. 2014. Mininet as software defined networking testing platform. In *Proc. International Conference on Communication, Computing & Systems (ICCCS)*. 139–42.

[16] Janardhan Kulkarni, Stefan Schmid, and Pawel Schmidt. 2021. Scheduling Opportunistic Links in Two-Tiered Reconfigurable Datacenters. In *33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 318–327.

[17] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. 2014. Circuit Switching under the Radar with REACToR. In *Proc. 11th USENIX NSDI*. 1–15.

[18] 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 *Proc. 17th USENIX NSDI*. 1–18.

[19] 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 *Proc. ACM SIGCOMM*. 267–280.

[20] Matthew K. Mukerjee, Christopher Canel, Weiyang Wang, Daehyeok Kim, Srinivasan Seshan, and Alex C. Snoeren. 2020. Adapting TCP for Reconfigurable Datacenter Networks. In *Proc. 17th USENIX NSDI (NSDI '20)*. 651–666.

[21] Polatis Series 6000 [n. d.]. https://www.polatis.com/. ([n. d.]).

[22] 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. In *Proc. ACM SIGCOMM*. 447–458.

[23] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. (2018). arXiv:cs.LG/1802.05799

[24] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. 2016. Xpander: Towards optimal-performance datacenters. In *Proc. ACM CoNEXT*. 205–219.

[25] Shay Vargaftik, Cosmin Caba, Liran Schour, and Yaniv Ben-Itzhak. 2020. C-share: Optical circuits sharing for software-defined datacenters. *ACM SIGCOMM CCR* 50, 1 (2020), 2–9.

[26] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. 2018. Costly circuits, submodular schedules and approximate Carathéodory Theorems. *Queueing Systems* 88, 3-4 (2018), 311–347.

[27] 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 *Proc. ACM SIGCOMM*. 327–338.

[28] Mowei Wang, Yong Cui, Shihan Xiao, Xin Wang, Dan Yang, Kai Chen, and Jun Zhu. 2018. Neural network meets DCN: Traffic-driven topology adaptation with deep learning. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 2 (2018), 26.

[29] 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 *Proc. ACM SIGCOMM*. 295–308.

[30] Johannes Zerwas, Wolfgang Kellerer, and Andreas Blenk. 2021. What You Need to Know About Optical Circuit Reconfigurations in Datacenter Networks. In *Proc. 33rd International Teletraffic Congress (ITC 33)*. 1–9.