Duo: A High-Throughput Reconfigurable Datacenter Network Using Local Routing and Control

JOHANNES ZERWAS, TUM School of Computation, Information and Technology, Technical University of Munich, Germany
CSABA GYÖRGYI, University of Vienna and ELTE Eötvös Loránd University, Austria and Hungary
ANDREAS BLENK, Siemens AG, Germany
STEFAN SCHMID, TU Berlin & Fraunhofer SIT, Germany
CHEN AVIN, Ben-Gurion University, Israel

The performance of many cloud-based applications critically depends on the capacity of the underlying datacenter network. A particularly innovative approach to improve the throughput in datacenters is enabled by emerging optical technologies, which allow to dynamically adjust the physical network topology, both in an oblivious or demand-aware manner. However, such topology engineering, i.e., the operation and control of dynamic datacenter networks, is considered complex and currently comes with restrictions and overheads.

We present Duo, a novel demand-aware reconfigurable rack-to-rack datacenter network design realized with a simple and efficient control plane. Duo is based on the well-known de Bruijn topology (implemented using a small number of optical circuit switches) and the key observation that this topology can be enhanced using dynamic ("opportunistic") links between its nodes.

In contrast to previous systems, Duo has several desired features: i) It makes effective use of the network capacity by supporting integrated and multi-hop routing (paths that combine both static and dynamic links). ii) It uses a work-conserving queue scheduling which enables out-of-the-box TCP support. iii) Duo employs greedy routing that is implemented using standard IP longest prefix match with small forwarding tables. And iv) during topological reconfigurations, routing tables require only local updates, making this approach ideal for dynamic networks.

We evaluate Duo in end-to-end packet-level simulations, comparing it to the state-of-the-art static and dynamic networks designs. We show that Duo provides higher throughput, shorter paths, lower flow completion times for high priority flows, and minimal packet reordering, all using existing network and transport layer protocols. We also report on a proof-of-concept implementation of Duo's control and data plane.

1 INTRODUCTION

The performance of many cloud applications, e.g., related to distributed machine learning, batch processing, or streaming, critically depends on the bandwidth capacity of the underlying network topology. High network throughput requirements are also introduced by today's trend of resource disaggregation in datacenters, where fast access to remote resources (e.g., GPUs or memory) is critical for the overall system performance [31, 35]. Accordingly, over the last years, great efforts have been made to improve the throughput of datacenter networks [2, 21, 41, 42].

Emerging optical technologies enable what is now known as *topology engineering*, a particularly innovative approach to improve datacenter performance, by supporting dynamic and real-time reconfigurations of the physical network topology [1, 7, 9, 10, 12, 14, 18, 24, 25, 28, 32–34, 39, 40, 43, 46, 47, 49, 51]. In particular, advanced optical circuit switches enable dynamic physical topologies by providing dynamic input-output ports *matchings* [1, 7, 20, 34]. Reconfigurable datacenter networks (RDCNs) use such switches to establish topological shortcuts (i.e., shorter paths) between racks, hence utilizing available bandwidth capacity more efficiently and improving throughput [20, 33, 34].

Reconfigurable datacenter networks come in two flavors: *oblivious* and *demand-aware* [5, 22]. Oblivious RDCNs such as RotorNet [34], Opera [33], Sirius [7], and Mars [1], rely on quickly and periodically changing interconnects between racks, to emulate a complete graph. Such emulation

was shown to provide high throughput and is particularly well-suited for all-to-all traffic patterns [20]. In contrast, demand-aware RDCNs allow to *optimize* topological shortcuts, that depend on the traffic pattern. Demand-aware networks such as ProjecToR [18], SplayNets [39], Gemini [49], ReNet [6] or Cerberus [20], among others [9, 12, 24, 25, 28, 44, 51], are attractive since datacenter traffic typically features much temporal and spatial structure: traffic is bursty and skewed, and a large fraction of communicated bytes belong to a small number of elephant flows [4, 8, 9, 18, 50, 52]. By adjusting the datacenter topology to support such flows, e.g., by providing direct connectivity between intensively communicating source and destination racks, network throughput can be increased further (even if done infrequently [49]).

However, the operation of reconfigurable datacenter networks comes with overheads and limitations. In general, existing RDCNs typically rely on a hybrid topology which combines static (electrical) and dynamic (optical) parts. While such a combination is powerful [18], current architectures support only fairly restricted routing. First, communication on the (dynamic) optical topology is often limited to one or two hops. This constrains the possible path diversity, and hence capacity, of the optical network [7, 15, 18, 20, 49]. Furthermore, routing is usually *segregated*: flows are either only forwarded along the static or the dynamic network, but not a combination of both [18, 20, 48]. An exception is *cp-Switch* which, however, assumes that capacities of static (electrical) and dynamic (optical) links are different [45]. The restriction to segregated routing also entails overheads as it requires significant buffering while the reconfigurable links are not available. As static links are always available for packet forwarding in hybrid datacenter networks, this segregation entails *non* work-conserving scheduling and a more complex buffer management.

This paper is motivated by the desire to overcome these limitations, and to better exploit the available link resources, by supporting a general *multi-hop* and *integrated* (i.e., non-segregated) routing. Specifically, we envision a datacenter network in which packets can be forwarded using a work-conserving scheduling, along *any* available link, be it static or dynamic, and in which a routing path can combine both link types. Such an integrated routing and work-conserving scheduling also has the potential to avoid long buffering times and hence delays: if a reconfigurable link is currently unavailable, packets can directly be forwarded to the other available (static) links. However, going beyond segregated and 1- or 2-hop routing, requires a novel network control plane: traditional routing protocols based on shortest paths are not designed for highly dynamic topologies and the frequent recomputation of routes can become infeasible [16]. Furthermore, to keep update cost low and provide a high scalability, it is desirable to have small forwarding tables.

To this end, we propose a simpler and more efficient control plane for RDCNs which avoids packets forwarding delays by supporting *local and greedy integrated routing*: the forwarding rules depend on local information only, i.e., the set of direct neighbors as well as information in the packet header (in particular, the destination); they are hence not affected by topological changes in other parts in the network and do not have to be updated under such reconfigurations. This can significantly reduce control plane overheads during topological adjustments, maintaining a simple routing, buffering and control, and is hence well-suited for highly dynamic networks.

In particular, we present Duo, a novel demand-aware reconfigurable datacenter network which leverages such a local control plane using a de Bruijn topology (built from a small number of optical switches), in which static links are enhanced with opportunistic links. Duo uses IP-based logical addressing and operates a receiver-based approach for the efficient detection of elephant flows as well as the local and collision-free scheduling of demand-aware links. Due to its simplicity Duo is well-suited to be realized, e.g., using the Sirius [7] architecture which was originally designed to be demand-oblivious, but can potentially support demand-aware link scheduling.

In summary, we make the following ***contributions***:

		Xpander [29]	Sirius [7] Opera [33]	ProjecToR [18]	Gemini [49]	Cerberus [20]	Duo
Topology Engineering	Demand-aware	No	No	Yes	Yes	Yes	**Yes**
	Update rate	None	Fast	Fast	Slow	Fast	**Fast**
Network Layer	Integrated Multi-hop	Yes	2-hops	1 hop	2-hops	1-hop	**Yes**
	Work conserving	Yes	No	No	Yes	No	**Yes**
	Routing mechanism	IP	NS	IP	IP	NS	**IP**
Transport Layer	Congestion control	TCP	NS	TCP	TCP	NS	**TCP**

Table 1. Recent (R)DCN designs and their properties. (NS is nonstandard).

- We present Duo, a novel, cost effective, and practical reconfigurable datacenter architecture which supports efficient multi-hop, integrated and work-conserving routing, to push the performance limits in datacenter networks.
- The simplicity of Duo relies on the observation that adding shortcuts to a static de Bruijn topology allows to continue supporting greedy local routing, which can be implemented using standard IP and longest prefix matching, with small forwarding tables. This enables a high update rate at low overheads. Duo can be implemented using the Sirius architecture, benefiting from the low cost and power consumption of this architecture, and only requires a small number of matchings. Its control plane can be realized using centralized or distributed algorithms.
- We perform a packet-level evaluation and empirically find that Duo provides a higher throughput compared to the state of the art, static and dynamic, networks. Furthermore, Duo's properties allow us to use TCP out-of-the-box, without the need to develop new transport protocols.
- We further report on a proof-of-concept implementation of the control and data plane of Duo that demonstrates the feasibility of Duo with standard network stacks. The implementation builds around a single P4 switch which we use to emulate a scenario with 16 ToRs.
- As a contribution to the research community, we release our implementation together with this paper.

2 PUTTING DUO INTO PERSPECTIVE

To put Duo into perspective with the most recent datacenter network design proposals, we summarize important properties for such networks in Table 1, and divide them to three categories: topology engineering, network layer, and transport layer. For topology engineering, we indicate if the system uses a demand-aware topology and the supported update rate of the topology (unless it is static). Regarding the network layer, we indicate i) if the routing is fully integrated multi-hop (or only 1 or 2 hops), ii) if the packet scheduling is work-conserving and iii) if the routing can support legacy IP. Finally regarding transport layer, we indicate if the design supports standard, e.g., TCP, or non-standard (NS) congestion control. The table does not compare the *performance* of the different designs, but their main design choices. We empirically evaluate these designs in Section 4.

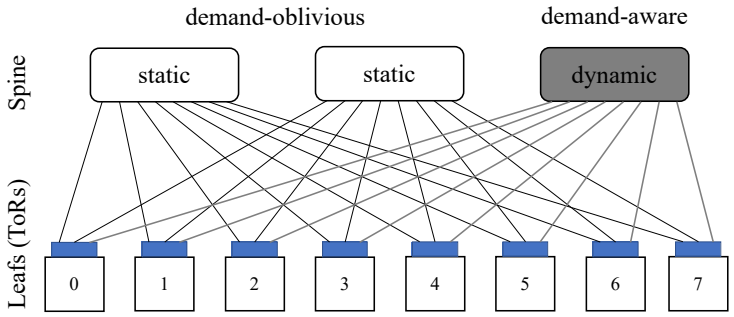


Fig. 1. TMT network example with eight ToR switches and three spine switches from which two are static matchings and one is a dynamic matching. Each spine switch has eight input and output ports.

We first consider Xpander [29], a state-of-the-art *static and demand-oblivious* topology for DCN, which is based on expander graphs. While Xpander has many attractive properties, according to recent results (including the ones in this work), we expect that reconfigurable datacenter networks (i.e., based on dynamic topologies) can provide an improved performance, and in particular, throughput [7, 37].

We next consider Sirius [7] and Opera [33], recent proposals of a *dynamic and demand-oblivious* designs for RDCN. Such topologies have been shown to be very effective as well, but still have several potential deficits. First and foremost, they do not feature demand-aware links: while the role and use of demand-aware topology components in future datacenters is generally still subject to ongoing discussions, empirical studies show that demand-awareness can improve throughput under today's typical skewed workload distributions [20, 49]. Furthermore, current dynamic and demand-oblivious designs are limited to at most 2-hop routing on dynamic links, and are not work-conserving. There also remain open questions regarding the complexity of the control plane, the routing scheme and the transport layer of these systems.

Next we consider systems which are dynamic and demand-aware. Systems like ProjecToR [18] use a combination of demand-aware optical and electric switches, but do not support integrated mutli-hop routing (ProjecToR uses only 1-hop on demand-aware links), are not work-conserving, and the control complexity is not fully determined. Gemini [49] (and more recently [38]), a recent proposal by authors from Google, makes the case for demand-aware links in production level datacenters, but it currently implements only infrequent topology updates (about once a day). Lastly, Cerberus relies on a combination of three topologies: static, dynamic oblivious, and dynamic demand-aware, which together, potentially provide higher throughput. However, it supports only 1-hop routing on the demand-aware links, and its control and routing mechanisms are left abstract and require further investigation.

In contrast to all the above systems, Duo features all the desired properties listed in Table 1. It supports *integrated multi-hop* routing as Xpander, it uses *demand-aware* links as ProjecToR, it is *work-conserving* as Gemini, it enables *fast topology updates* as in Cerberus, and it is based on simple control and IP-based routing. Its properties and especially work-conservation allows us to use standard TCP in Duo, as we will also show empirically.

To the best of our knowledge, we are the first to study and show the benefits of a de Bruijn approach in the context of reconfigurable and demand-aware datacenter networks.

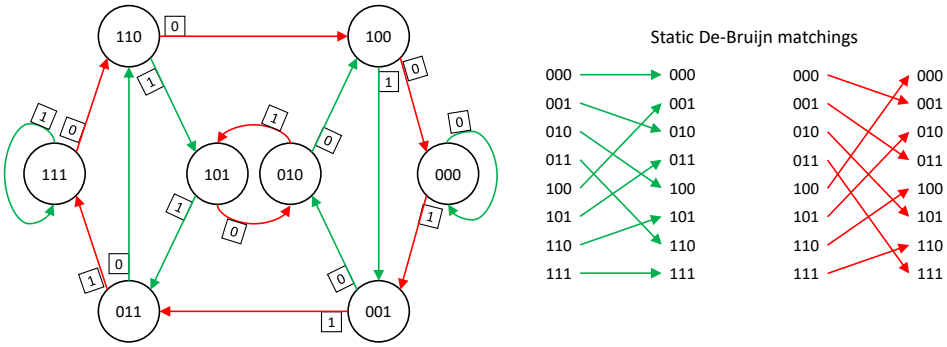


Fig. 2. A $DB(2,3)$ static & directed de Bruijn graph with eight ToRs and its two corresponding matchings (colored in green and red). Each port (edge) is labeled 0 or 1 according to the performed shift operation.

3 THE DUO RDCN DESIGN

The rack-to-rack network provided by Duo is based on the typical ToR-Matching-ToR[1] (TMT) model as it is common in the literature [7, 20, 33, 34]: n top-of-rack (leaf) switches are interconnected by a set of k optical spine switches. Each spine switch provides a $n \times n$ directed matching between its input-output ports. Depending on the switch type, the matching can dynamically change over time. In particular, Duo is hybrid, in the sense that one part of the topology model is static and demand-oblivious, using k_s static spine switches (i.e., static matchings). The second part is dynamic and demand-aware, using k_d reconfigurable spine switches (i.e., dynamic matchings), and $k = k_s + k_d$. Figure 1, presents an example of the TMT model with eight ToR switches and three spine switches, from which two are static and one is dynamic. Each ToR-spine link in the figure represents one directed uplink and one directed downlink. It is important to note that abstractly, we use k spine switches, each implementing an $n \times n$ matching, but each matching can be split across a set of several smaller switches, like in Sirius [7] where spine switches have $\sqrt{n}$ ports[2].

In order to maximize performance, Duo uses dynamic, demand-aware links to provide shorter paths for elephant flows, while other flows are transmitted via the combined (static + dynamic) topology. A key feature of Duo is that it supports *integrated multi-hop* routing across *both* switch types. This is in contrast to previous works that rely on *segregated* and single-hop forwarding for demand-aware links [18, 20]. Moreover routing in Duo is *efficient*, by relying on logical addressing and a local control plane, implementing greedy routing. Thus, links can always be used immediately, with a work-conserving scheduler. To detect elephant flows, Duo leverages a simple sketch, sampling the flow sizes and then adjusting the dynamic links accordingly. In the following, we present the different components of Duo in detail.

3.1 The Hybrid Topology

Duo combines two topologies, a static, demand-oblivious topology (the "*backbone*"), and a dynamic, opportunistic, demand-aware topology. Both topologies are built from matchings in spine switches according to the TMT model, forming a augmented de Bruijn network [11].

- **Static and demand-oblivious de Bruijn topology (backbone):** The static topology of Duo relies on a de Bruijn graph. It is formed by k_s static optical circuit switches or patch panels.

[1] Top of rack switch [2] We note that a smaller number of ports might affect the performance and leave this for future study.

- **Dynamic and demand-aware topology:** The static topology is enhanced by k_d reconfigurable matchings, also implemented with optical circuit switches. The demand-aware (DA) links add *shortcuts* on top of the static de Bruijn topology.

We first discuss and explain the static de Bruijn topology. Our design choice of the de Bruijn topology is based on two main properties: i) It can be built from a small number of matchings (already two matchings suffice in contrast to, e.g., hypercubic topologies that require $\log n$ matchings) and ii) it enables IP-based greedy forwarding also when additional shortcuts are added to the topology, which in turn enables fast and low-overhead topology reconfigurations.

3.1.1 The Static de Bruijn Topology. We start with formally defining the de Bruijn topology [30]. For $i \in \mathbb{N}$, let $[i] = \{0, 1, \ldots, i\}$.

Definition 3.1 (de Bruijn topology). For integers $b, d > 1$, the *b-ary de Bruijn graph of dimension d*, $DB(b, d)$, is a *directed* graph $G = (V, E)$ with $n = |V| = b^d$ nodes and $m = |E| = b^{d+1}$ directed edges. The node set V is defined as $V = \{v \in [b-1]^d\}$, i.e., $v = (v_1, \ldots, v_d), v_i \in [b-1]$, and the directed edge set E is:

$$\{v, w\} \in E \Leftrightarrow w \in \{(v_2, \ldots, v_d, x) : \ x \in [b-1]\} \tag{1}$$

Note that the directed neighbors of node v are determined by a left *shift* operation on the address of v and entering a new symbol $x \in [b-1]$ as the right most (least significant) symbol. It is well known that the de Bruijn topology has the following properties [30]:

(1) Considering self-loops, $DB(b, d)$ is a b-regular directed graph.
(2) $DB(b, d)$ supports greedy routing with paths of length at most d.

The following observation will be relevant for our network design, it is a consequence of Property (1) above and Hall's theorem [23]:

Observation 1. *A $DB(b, d)$ topology can be constructed from the union of b directed perfect matchings*[3].

Figure 2 demonstrates the $DB(2, 3)$ de Bruijn topology with $8 = 2^3$ nodes (ToRs) and two matchings (colored in green and red) that can be combined to create it. Each node in the topology has two outgoing and two incoming directed links (including self loops). The figure shows the *labeled* version of the graph where each edge (or a node outgoing port) is labeled with 0 or 1 according to the shift operation implied by Eq. (1).

It follows from Observation 1 that we can build a $DB(k_s, d)$ topology with k_s static spine switches. From Property 2 it follows that with two static spine switches we can build a topology for n ToRs and with a diameter $\log n$ (when n is a perfect power of two).

3.1.2 Greedy and LPM Routing in de Bruijn Topology. It is well known that the de Bruijn topology supports greedy routing from a source s to a destination t based solely on the address of t. That is, to choose the next-hop toward t each node on the route needs to know the address of t and the address of its neighbors. The next-hop is chosen as the neighbor which minimizes the *de Bruijn distance* to t. The de Bruijn distance between two nodes v, w denoted as $\text{dist}_{\text{DB}}(v, w)$ is the minimum number of *shift* operations needed to transform v's address to w's address. The main observation is that each such shift implies a directed edge and the next-hop in the routing. For example, the de Bruijn distance between node $s = 011$ and $t = 001$ is $\text{dist}_{\text{DB}}(s, t) = 3$ and the route from s to t in $DB(2, 3)$ is $011 \rightarrow 110 \rightarrow 100 \rightarrow 001$ (see also Figure 2). Note that in each hop the distance to t is reducing.

[3] A perfect matching of a graph is a matching in which every vertex is incident to exactly one matching edge.

Algorithm 1: Building the DB Forwarding Table

Function *BuildTable* (at node v)
  for *each neighbor* z_1z_2, z_3 *at port* p **do**
    Add the following entries to the table

Prefix	Port	Path-length
$z_3 * *$	p	3
z_2z_3*	p	2
$z_1z_2z_3$	p	1

  Reduce the forwarding table according to *LPM* rules

Prefix	Port	Path-len
$0**$	0	3
$10*$	0	2
110	0	1

(a) Neighbor 110 on port 0

Prefix	Port	Path-len
$1**$	1	3
$11*$	1	2
111	1	1

(b) Neighbor 111 on port 1

Prefix	Port	Path-len
$0**$	0	3
$10*$	0	2
110	0	1
111	1	1
011	Local	0

(c) Reduced Table for ToR 011

Fig. 3. The results of building the forwarding table (Algorithm 1) of node 011 with neighbors 110 and 111 on the static $DB(2,3)$ de Bruijn graph.

A less-known fact is that routing on the de Bruijn topology can be realized via a simple forwarding table that is based on a *longest prefix match* (LPM) [13]. This results from the fact that the de Bruijn distance from a node $v = (v_1, \ldots, v_d)$ to *all* other nodes can be compactly represented using d entries and LPM. For example, the de Bruijn distance between v and all nodes that their address has a longest-prefix-match to rule $(v_3, v_4, \ldots, v_{d-2}, *, *)$ is two. Namely with two shift operations we can transform v's address to the address of a node that matches the above rule.

To build the forwarding table for a node v it only needs to know the address of each neighbor w and the outgoing port p that connects to it. Algorithm 1 describes the forwarding table building (for simplicity only for the $DB(2,3)$ case) and Figure 3 shows the forwarding table of node 011 and how it is built from its two neighbors 110 and 111. Note that in this example rules $1**$ and $11*$, are removed from the forwarding table of ToR 011 in the *table reduce* process since they will never be used (there will be always a longer match for these rules).

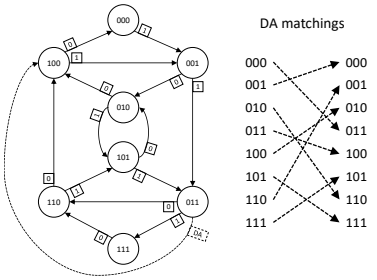
Considering Algorithm 1, we can state the following about the *size* of the forwarding table of each node:

Observation 2. *The longest prefix match forwarding table size of each node in a* $DB(b,d)$ *topology has at most* $bd = O(b \log_b n)$ *entries.*

We can now discuss the DA links and how they are merged into the hybrid topology.

3.1.3 The Dynamic Demand-aware Topology. The simplicity of Duo relies on the observation that adding shortcuts to the static de Bruijn topology is easy and allows to continue supporting greedy and LPM routing. Recall that in our model we have k_d reconfigurable switches which means k_d matchings of size n for DA links. For now consider these $k_d \cdot n$ demand-aware links as arbitrary links. Later we discuss how to choose these links based on the demand.

Let $G = DB(b,d)$ be a de Bruijn topology over the node set V. Let M be a directed matching on $V \times V$. Let $H = G \cup M$ be the union of the directed graphs G and M with the same set of nodes V.



(a) ToR 011 with new DA link to 100.

Prefix	Port	Len
$0**$	DA	3
$00*$	DA	2
100	DA	1

(b) Entries from 100.

Prefix	Port	Len	IP
$0**$	$\{0, DA\}$	3	10.0.0.0/9
$00*$	DA	2	10.0.0.0/10
$10*$	0	2	10.128.0.0/10
100	DA	1	10.128.0.0/11
110	0	1	10.192.0.0/11
111	1	1	10.224.0.0/11
011	Local	0	10.96.0.0/11

(c) Reduced table on ToR 011.

Fig. 4. The new forwarding tables of ToR 011 after the establishment of the DA-link from 011 to 100.

Figure 4 (a) demonstrates this case and shows H, the union of a $DB(2, 3)$ topology with a single demand-aware matching (showing for clarity only one DA link from 011 to 100). Figure 4 (b) and (c) presents the new forwarding table at node 011, which is a result of the added demand-aware link and is constructed using Algorithm 1. If we consider as before the route from $s = 011$ to $t = 001$ it will now be shorter $011 \rightarrow 100 \rightarrow 001$. In fact, all packets that reach node 011 with destination addresses with LPM $00*$ will use the new DA port for forwarding. Note also that routes toward addresses with LPM $0**$, like 010, have now two equal length routes (of length three), for example, $011 \rightarrow 100 \rightarrow 001 \rightarrow 010$ or $011 \rightarrow 110 \rightarrow 101 \rightarrow 010$. More generally, we can claim the following about H.

Claim 1. *Let H be the union of a de Bruijn $DB(b, d)$ topology and a single directed matching over the same set of nodes. If we perform Algorithm 1 on each node in H, then H supports integrated, multi-hop, greedy, LPM routing with forwarding table size of $(b + 1)d$.*

Proof sketch. First we can show that H supports greedy routing, since in each node v and for each destination t the next-hop will be the *neighbor* of v in H with the shortest *de Bruijn distance* to t. While greedy routing on the static topology $DB(b, d)$ reduces the de Bruijn distance in each hop by exactly one, DA links can reduce it by more than one. So the distance is strictly decreasing in each hop until the destination. From the greedy routing it is clear that LPM forwarding will work and that the path is integrated in a multi-hop manner. The forwarding table size is at most $(b + 1)d$ since each node has at most $(b + 1)$ neighbors (b static and one from the DA matching) and for each it needs d entries to represent its de Bruijn distance to destinations. □

Following Claim 1, we can extend the single matching case to more than one matching and support k_d demand-aware matchings. Formally, for integers $k_s, k_d, x \geq 2$ and $n = (k_s)^x$, we denote by $Duo(n, k_s, k_d)$ the Duo topology with $k = k_s + k_d$ spine switches, backbone network $DB(k_s, \log_{k_s} n)$, and k_d demand-aware switches. We can state the following about the hybrid topology of Duo.

Theorem 3.2. *The $Duo(n, k_s, k_d)$ topology with n ToRs and $k = k_s + k_d$ spine switches ($k_d, k_s \geq 2$) supports integrated, multi-hop, greedy, LPM routing with forwarding table size of $O(k \log_{k_s} n)$ and diameter $d \leq \log_{k_s} n$.*

Proof sketch. The proof extends Claim 1 to k_d demand-aware matchings. Since the additional links can only reduce distances the diameter of $Duo(n, k_s, k_d)$ is at most the diameter of $DB(k_s, \log_{k_s} n)$ which is $\log_{k_s} n$. □

Algorithm 2: Centralized (Greedy) DA links setting

```
Function Greedy-DA-links (D - Demand Matrix, k_d - number of DA switches)
    Δ=Largest k_d n demands in D, sorted by volume
    forall (s, t) ∈ Δ from large to small do
        if s, t have available DA ports in switch i then
            Set DA-link (s, t, i)
```

3.2 Scheduling of Demand-Aware Links

Duo relies on a control plane which can use centralized or decentralized scheduling of the DA links. The centralized scheduling benefits from the global view, while the decentralized scheduling supports fast reaction.

We use Sirius' [7] reconfiguration model also for DA links: spine switches use passive gratings while (sending) ToR switches rely on tunable lasers which determine the link to set up in the corresponding switch (matching). This property is useful for the distributed version of the scheduling where the receivers provide permissions to senders to reconfigure links. All algorithms use the command 'Set DA-link (x, y, i)' which means that sender ToR x tunes its laser on (egress) port i to establish a direct link to ToR y via switch i. Recall that each ToR has k egress and k ingress links toward the k spine switches, so we identify egress or ingress port i with switch i.

3.2.1 Centralized scheduling of DA*-links.* The algorithm uses a greedy heuristic to add shortcuts (DA links) to the backbone de Bruijn network. It periodically, every *update period* ρ, determines the new DA-links based on an estimate of the accumulated demand or a measurement of the traffic in the network until the current time. We denote this estimation by a *demand matrix* D. The algorithm sorts the demands in D by decreasing order and for each demand (s, t) in D, it tries to add a DA link to the network. In case the algorithm decides to set a DA link, the link is not available for use during *reconfiguration time* δ, unless it was already setup previously. This results in a duty cycle of $\epsilon = \frac{\rho-\delta}{\rho}$. The algorithm is repeated until no new DA link can been added. Moreover, remaining free, i.e., unmatched, ingress and egress ports are matched. If possible, the matchings are in parallel to links of the static de Bruijn topology, otherwise, a random matching is applied.

Algorithm 2 shows the centralized algorithm, *Greedy*-DA-*links*. The algorithm is a simple version of greedy k-matchings (known also as b-matching for undirected graphs [17]). The algorithm iterates over requests $(s, t) \in D$ that are larger than a threshold v_{th} in decreasing order and only connects a direct link between s and t if they have unmatched ports on the same DA switch i. We note that the the shortcuts added by Greedy-DA-links supports integrated multi-hop, and a similar version of it is easier to implemented in a distributed way, as we explain next.

3.2.2 Distributed scheduling of DA*-links.* Algorithm 3 shows the distributed scheduling algorithm, *Dist*DA. The algorithm combines similar approaches as presented in ProjecToR [18] and Sirius [7]. It implements a distributed, threshold-based greedy k matchings algorithm. A destination-based elephant detection triggers the algorithm. For instance, this can be done in P4 using sketches [36] as we discuss in more details later. If any of the destinations detects a source(-ToR) as elephant it checks if it has available DA-ports. If available, it sends an offer, *PortRequest(ports)*, to the elephant source ToR via the static topology part where *ports* is a list of available ingress ports at the destination. Upon reception, the source/sender checks for an available egress DA-port on its side. If a port is available, it acknowledges the request via a *PortApprove(i)* message and sets the link on port i. If no DA-port is available at the source, the request is declined. The receiver ToR continues to generate

Algorithm 3: Distributed DA-link scheduling

```
1  Function Dist DA() at destination t
2      Upon detection of elephant flow from source s
3      if t has available DA ports then
4          Send PortRequest(ports) to node s
5          if s reply with PortApprove(i) then
6              DA-link (s, t, i) is set (with timeout)
7  Function Dist DA() at source s
8      Upon PortRequest(ports) from destination t
9      if s has available DA ports in ports then
10         Send PortApprove(i) to node t
11         Set DA-link (s, t, i) (with timeout)
12     else
13         Send DeclineRequest
```

PortRequests for other elephants. An agreed DA-link, i.e., the ports at sending and receiving ToR, is reserved for fixed period of time r. Afterwards, the ports can be assigned to new requests and the circuit might be reconfigured. However, the circuit is *not* pro-actively torn down but kept alive until an appropriate request arrives.

We note that while our distributed scheduler is simple, it is effective as we will see next. We leave the study of more sophisticated schedulers (e.g., based on distributed stable matchings [18] or online algorithms [27]) for future work.

3.3 Implementation and Practical Aspects

3.3.1 Implementation and Cost. As mentioned earlier, we envision that Duo could be implemented using the Sirius architecture [7]. Sirius is also captured by the TMT model, but one of its great advantages is that instead of spine switches, Sirius uses a single layer of k gratings. The Arrayed Wavelength Grating Routers (AWGR) are simple and *passive* without moving parts and do not consume power. Still, each grating diffracts ("forwards") incoming light from input to output ports, based on the wavelength, abstractly creating a matching. Reconfiguration is then performed by a physical-layer ToR switch (or directly on servers) equipped with k transceivers containing tunable lasers that can change the wavelength used to carry the data toward the gratings through an optical fiber.

Sirius has been presented as a demand-oblivious architecture which provides fast end-to-end reconfiguration, due to a pre-determined, static schedule that specifies the connectivity at any given fixed-size timeslot. However, the Sirius architecture is in principle also well-suited for demand-aware scheduling, with a slower end-to-end reconfiguration delay.

As Duo differs from Sirius only in the scheduling and routing, the cost and power consumption of Duo will be similar to Sirius. In [7], the authors showed that Sirius' power and cost are about 25% that of an electrically switched Clos network (ESN). That said, unfortunately, a direct comparison of the performance of Duo and Sirius is currently not possible as Sirius' simulation code is not available. Therefore, we concentrate on the comparison to Opera [33] which is similar in the demand-oblivious nature as Sirius. Additionally, we compare to static expander topologies which are also state-of-the-art datacenter networks [29].

3.3.2 IP Addressing and LPM Forwarding. We embed the de Bruijn address into the hosts' IP addresses. Our approach uses IPv4 but can also be implemented using IPv6. Depending on the number of ports per ToR, a single symbol of the de Bruijn address takes one or multiple bits of the IP address: $s = \lceil \log_2(b) \rceil$. Thus, the full de Bruijn address occupies $s \cdot d$ bits of the IP address. In order to use LPM to implement the forwarding, we split the IP address into three parts. The first p bits mark the base network that is assigned to Duo. The following $s' = s \cdot d$ bits identify the ToR by means of the de Bruijn address while the remaining bits identify the host/VM inside the rack, i.e., each ToR is assigned a $/(p + s')$ prefix.

For the example of Figure 4(a), the de Bruijn address can directly be mapped to an IP address/prefix and occupies only 3 address bits. Figure 4(c) shows an exemplary forwarding table for ToR 5 = 011 with 10.0.0.0/8 as a base IP prefix. Following Algorithm 1, each node can build its IP forwarding table locally based on its ToR neighbors' addresses. In particular, when a new DA links is established for a node's port and it knows the ToR address of the new neighbor, the forwarding table can be updated locally (without recomputing shortest paths).

3.3.3 Transport Layer. In general, Duo does not require a customized transport protocol nor complex flow scheduling. It performs well with existing protocols "out of the box", for instance with TCP.

However, we found that handling small and large flows differently, e.g., by using different transport protocols and separating them into different queues, can significantly improve performance further, and reduce flow completion times of the former drastically. Thus, similar to previous approaches such as Opera [33], Duo classifies flows according to a fixed threshold f_{th} into low latency and bulk traffic. In order to improve the performance for low latency flows, NDP [26] was identified as a promising candidate. It also shows good performance in Duo. For large flows ($> f_{th}$), Duo uses TCP for congestion control and re-transmissions.

Each port of a ToR (static and DA) in Duo provides three queues of different, fixed priorities. In order to avoid interference on congestion feedback, TCP acknowledgements and NDP header packets, i.e., acknowledgements or packets that have been stripped, are always forwarded via the high priority queue. NDP data packets use the medium priority queue and TCP data packets use the low priority queue. Packets residing in the queues of a DA-link that is reconfigured are dropped. We acknowledge that this is a radical approach but found that the imposed re-transmissions have only little impact on Duo's performance. Forwarding these packets via the new DA-link resulted in lower throughput.

4 EVALUATION

To evaluate Duo, we use and extend *htsim*. *htsim* is a packet-level simulator which has also been used in previous works, e.g,. [26, 33]. With our simulation, we answer the question of how Duo fares against state-of-the-art topologies such as more common DA topologies, expander-based networks, and Opera [33]. We further explore how Duo behaves under varying traffic conditions and assess the impact of several topology configurations.

4.1 Methodology

4.1.1 Settings. We consider topology configurations with $n = 64, 128$, and 256 racks. Each ToR has 16 ports which are divided into 8 downlink ports for the hosts, and 8 uplink ports. Since Duo implements a directed topology, the uplinks are split into 8 ingress and 8 egress ports. All links have a capacity of 10 Gbps (as in previous work [33]). So the considered topologies have total uplink capacities of 5.12 Tbps, 10.24 Tbps and 20.48 Tbps.

Unless stated differently, we consider reconfiguration delays of $\delta = 100\ \mu$s for DA links. This delays includes both computation and switches setup. The update time is $\rho = 10$ms, i.e., we update the DA links every 10 ms (if needed). Overall, for this settings, the duty cycle is 99%. The reconfiguration delay for demand-oblivious links in Opera is 10 μs and following the default configuration yields a duty cycle of 98% [33]. The packet size is 1500 Bytes. Unless stated differently, each configuration is run for 10 s of simulation time.

4.1.2 Traffic and Flow Size Distributions. The evaluation considers an online traffic scenario in which flows arrive according to a Poisson process. We consider the following four traffic scenarios:
Permutation. The flow sizes are fixed to 500 KB and the local communication pattern follows a rack-to-rack permutation matrix. The permutation matrix was shown to be the worst case traffic for expander and dynamic rotor-based topologies like Opera [20, 37].
Hadoop, Websearch and Datamining. In each case the flow sizes are sampled from the relevant flow size distribution as reported in the literature [3, 19]. The flow size distributions of these cases are similar to a power law distribution, but with different average and maximum flow size. Consequently, most of the flows are small, but the majority of the traffic belongs to large flows [20].

The arrival rate is adapted in order to create load values in the range from 10% to 80% with respect to the total uplink capacity of the ToRs, i.e., 100% means that all hosts are sending at full line rate.

4.1.3 Topologies. Throughout the evaluation, we compare four combinations of topologies and resource management, i.e., DA link configuration and routing algorithms.
Duo. The first competitor uses Duo ($k_s = 2, k_d = 6$) and configures the DA links using the centralized greedy, one-hop approach shown in Algorithm 2. It has knowledge of the remaining volume to be transmitted of all flows that are present in the system when the algorithm is called. Upon establishment of a circuit, it updates the routing information (locally) to incorporate the new DA link. In addition, it ensures that only equal cost (length) paths according to the greedy de Bruijn routing are used.
Segregated. The first baseline (Segr) has the same topology configuration and DA link scheduling algorithm as Duo but can only use *segregated* routing similar to ProjecToR [18]. Flows are routed via a DA link only if it directly connects the source and the destination of the flow. Otherwise, flows are routed on the static topology according to the greedy de Bruijn routing.
Expander. The second baseline (Exp) is a static $k = 8$-regular directed expander topology that is built from k static matchings. For flow allocation, it uses a k-shortest path routing where flows are assigned a fixed path upon their arrival.

All three systems use the priority queueing described in Section 3.3.3. Each port has a low priority queue which can hold 50 data packets and medium and high priority queues of the same size. Flows $\leq f_{th}$ use the NDP [26] implementation and large flows ($> f_{th}$) use the TCP implementation available in *htsim*. If not stated otherwise, the threshold is $f_{th} = 1$ MB.
Opera. The third baseline is Opera. We use the available implementation in *htsim* and the default configuration as provided with the original paper [33]. Flows < 15 MB are sent using NDP [26] via the *Expander* part, whereas large flows use the RotorLB (RLB) transport protocol and rate allocation algorithm. There are two queues per port on the ToRs. The first one can hold 8 data packets and the second queue has the same size for header packets. Note that in addition, Opera makes heavy use of packet queueing on the hosts and its transport protocol is not standard.

4.2 Empirical Results

Our evaluation revolves around the following main *takeaways*:

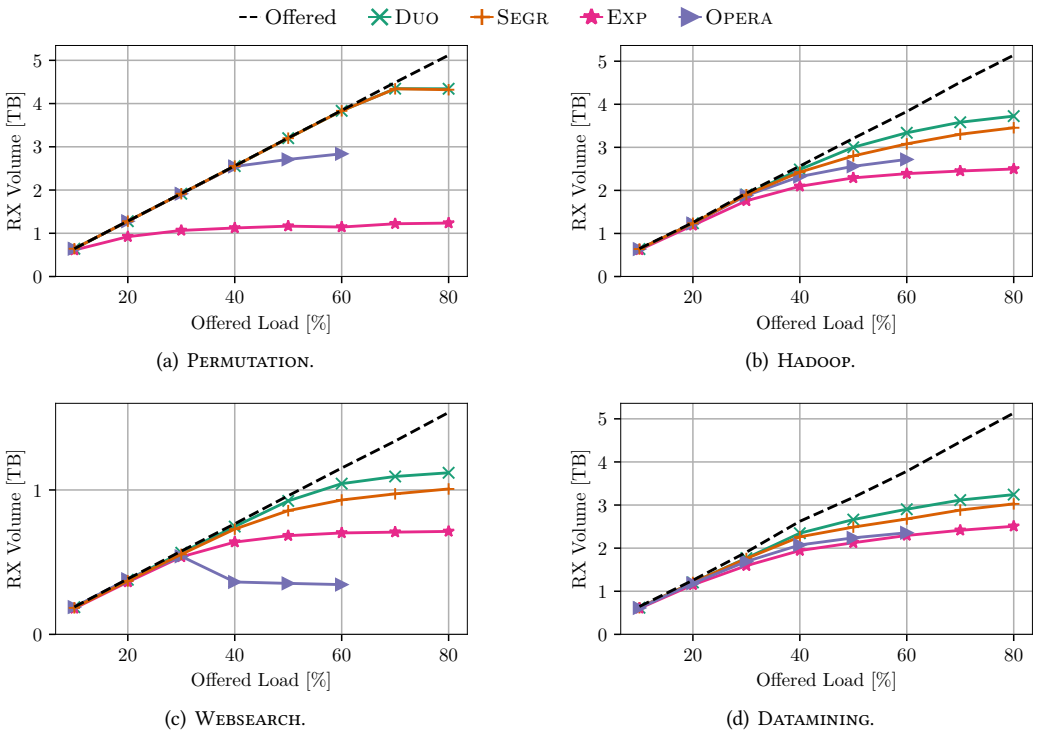


Fig. 5. Total received volume within 10s against total offered load. Duo outperforms the baselines Segr, Exp, and Opera.

4.2.1 **Duo increases system throughput.** Figure 5 presents the total received (RX) volume throughout a simulation run. It compares Duo, Segr, Opera, and Exp for the four traffic patterns. The abscissa shows the offered data relative to the uplink capacity of the ToRs and the simulation duration. The offered data describes the goodput (dashed line), i.e., it does not account for transport layer effects. The RX volume directly correlates with the throughput of the network. Naturally, it increases with the offered data but saturates when the throughput of the network is reached. Recall that Duo, Segr, and Exp use TCP for large flows so the received data is ordered while for Opera the packets mostly arrive out of order due to the oblivious scheduling of the next hop link.

Duo improves over all baselines for three of the traffic patterns. Only for Permutation (Figure 5(a)), it is on par with Segr. Duo and Segr saturate beyond 70% offered load which corresponds to the ratio of DA link capacity to total link capacity (= 6/8 = 0.75). In this case, the DA links are used to exactly match the traffic pattern, i.e., permutation. Thereby, paths of minimal length are provided and the traffic is served efficiently. Opera saturates around 50% which corresponds to the upper bound for the throughput that has previously been derived in theory.[4] Exp starts to saturate already around 20% offered load.

For Hadoop, Websearch[5] and Datamining, the differences between the topologies are smaller. First, we note that Opera and Exp align and start saturating both around 40% for Hadoop and 30% for Websearch and Datamining. For Websearch, the RX volume of Opera drops for loads

[4] Opera is only evaluated up to 60% load due to computation constraints. [5] Websearch is simulated only for 3 s due to the significantly larger number of flows.

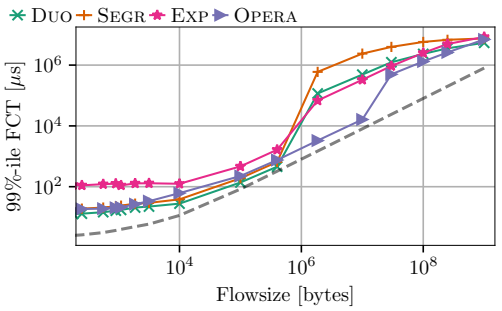


Fig. 6. Comparison of 99%-ile FCT by flow size.

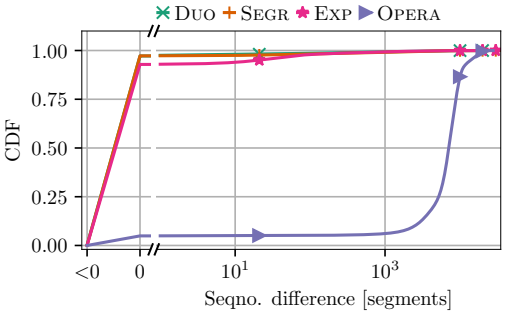


Fig. 7. Difference between received and expected segment sequence number.

> 30%. This is in line with the original paper of Opera which also observed decreasing throughput when the load of Websearch traffic increases [33, Figure 10]. Second, Duo successfully transmits ≈ 15% more data on average across the load values in comparison to Opera and Exp. For instance, Duo successfully transmits 2.75 TB for 50% offered load from Datamining while Exp achieves only 2.27 TB and Opera 2.24 TB. Compared to Segr, Duo transmits 8% more data for high loads on average across the three workloads. This illustrates the benefit of the integrated multi-hop routing provided by Duo.

In the following, the evaluation focuses on the Datamining workload.

4.2.2 **Duo preserves competitive flow completion times.** Besides throughput, the flow completion time (FCT) is an important performance indicator. Figure 6 illustrates the 99%-ile of the FCT per flow size for 40% load. The dashed line indicates the minimum under ideal conditions. For small flows (< 1 MB), Duo slightly outperforms the other topologies. For instance, it achieves FCTs around 13 μs followed by Opera (18 μs) and Segr (20 μs). Exp gives the worst FCTs, approximately one order of magnitude larger compared to Duo.

The advantage of Duo persists until f_{th}, i.e., a flow size of 1 MB is reached. Here, Opera starts to perform better as the curves separate before they converge again for flows > 15 MB. The major reason for these differences is that Duo trades off the lower FCT for small flows against increased values for medium sized flows. The choice of f_{th} to define low latency flows controls this trade off (cf. Section 4.2.5). In addition, it has been shown that *demand-oblivious* topologies with faster configuration speeds, such as Opera, are the best option for medium-sized flows [20]. We note, that Duo consistently performs better than Segr across all flow sizes.

Overall, Duo effectively trades off the higher throughput and slightly better FCTs for small flows against the FCTs of medium flows.

4.2.3 **Duo has a moderate amount of packet reordering.** Figure 7 visualizes the difference of the received and the expected sequence number per packet arrival at the destination. The figure shows the empirical cumulative distribution function. The abscissa is broken into two parts. The two data points in the left part (< 0 and 0) denote the fraction of packets where the difference was negative and equal to 0. The right part visualizes the samples > 0 with a log-scaled abscissa. A negative difference means that a packet is a bad (unnecessary) re-transmission. The values are below 0.5% for all networks. The observations vary for the case ≥ 0. Opera receives only a small fraction of around 5% of the packets in correct order (= 0). For 80% of the received packets, the difference of the sequence numbers lies between 4001 and 20k. While this is a direct consequence of RLB's packet indirection, it still illustrates the complexity that Opera imposes on the receiving

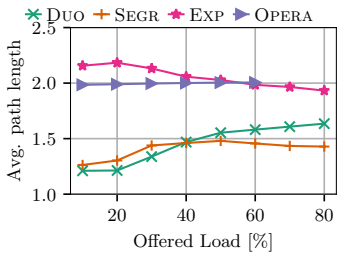


Fig. 8. Comparison of the traffic weighted average path length.

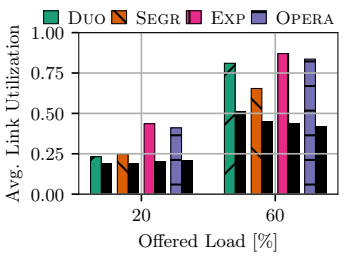


Fig. 9. Comparison of the average link utilization.

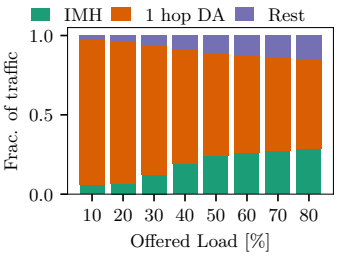


Fig. 10. Comparison of Duo paths' characteristics: IMH vs. 1-hop DA.

hosts. For Exp, ≈ 93% of the packets arrive in order while it is ≈ 97% for Duo and Segr. We conclude that in spite of the DA link and routing reconfigurations, Duo's does not lead to an increased amount of packet re-ordering.

4.2.4 **Duo path characteristics.** To shed light on how Duo's traffic allocation is different from Segr, Exp, and Opera, we consider Figures 8-10. Figure 8 shows the (weighed) average path length in hops calculated based on the traffic allocation. It weighs each path length with the fraction of traffic that arrived at the destination and used a path of this length. Lower values are better and indeed we observe that Duo reduces the path length for all evaluated load values compared to Exp and Opera. However, it only provides shorter paths compared to Segr for loads up to 40% and the advantage to Exp and Opera decreases with higher load, when Duo starts to route flows via integrated multi-hop paths that use both DA and static links. For 20%, Duo has an average path length < 1.25 hops, slightly smaller than Segr (1.3), compared to Opera and Exp which are ≥ 2 hops. For 80% load, Duo uses only 1.63 hops on average, still a 15% improvement to Exp and Opera but 15% larger than Segr (1.42).

However, Segr does not benefit from the shorter average path lengths. To understand this more in detail, Figure 9 shows two assessments of the average link utilization from the steady state of a simulation run. The colored bars are the raw average link utilization. The black bar next to a color bar is the respective *normalized link utilization*. The normalized link utilization $\tilde{u}$ is defined as the ratio between average link utilization u and average path length p, i.e., $\tilde{u} = \frac{u}{p}$. It gives an indication for the throughput. For 20%, u varies for the networks. Duo has the lowest raw link utilization with 0.25 followed by Segr (0.27), Opera (0.41) and Exp (0.45). This is in line with the order for the average path lengths. The normalized link utilization is around 0.2 for all topologies. This means that all topologies can serve the load as also seen in Figure 5(d). In this case, a lower link utilization means more efficient operation and hence, is favorable. The observations change for 60% load. Here, Segr has a lower raw link utilization (0.64) than Duo (0.79). The order for Exp (0.85) and Opera (0.82) remains the same. Also differences in the normalized link utilizations are observed. In line with the observations in Figure 5(d), Duo has the highest value (0.5) followed by Segr (0.44), Exp (0.43), and Opera (0.41). The lower link utilization of Segr is not an indicator for more efficient transmission here. Since Segr is restricted to 1-DA hop paths it is not able to utilize the available capacity as effectively as Duo. Moreover, it routes more flows via the static de Bruijn topology which increases congestion. As a result, Segr provides lower throughput (cf. Figure 5(d)). This confirms that Duo sets DA links that are usable by the greedy routing.

Figure 10 provides more details on the path characteristics. It shows the fraction of traffic that uses a single hop, demand-aware link (1 hop DA) and the fraction of traffic that traverses *integrated multi-hops* (IMH) paths, i.e., paths which use both static and DA links, so their length is at least two

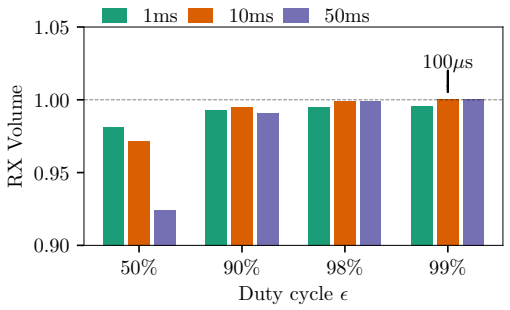


Fig. 11. Impact of period ρ and duty cycle on RX volume. Load is 40%.

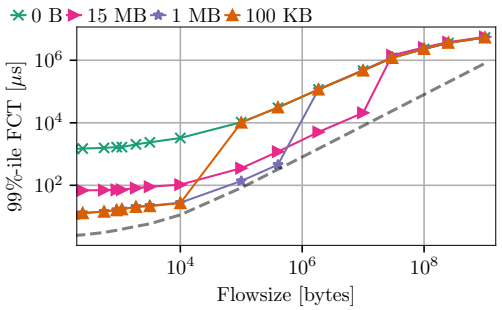


Fig. 12. Impact of small flow threshold on FCTs by flow volume.

hops. With higher load, the share of 1 hop DA traffic decreases, while the fraction of IMH paths increases. This can be explained with the properties of the traffic. For low load, the DA links can support most of the traffic, but as load increases more ToR pairs have demand for shortcuts, and multi-hop paths are needed to enable higher throughput. For these paths, due to its greedy routing, Duo also can use the DA links and the share of IMH traffic increases. Specifically, we observe rather large values for the fraction of 1-DA hop traffic (> 0.9) for loads below 40%. It decreases to ≈ 0.56 with 80% offered load. In contrast, the share of IMH traffic starts < 0.1 and increases towards 0.28 for high load. This demonstrates the importance of the integrated paths property of Duo. Without this property, the throughput would be lower, as the results for Segr show (cf. Figure 5).

4.2.5 **Sensitivity analysis.** Next, we assess how Duo's performance depends on the algorithm configuration. Figure 11 approaches this question on two different dimensions: the optimization period ρ and the duty cycle ϵ. The values are normalized to the result for $\rho = 10$ ms, $\delta = 100\mu$s which gives $\epsilon = 99\%$ and is used in the rest of the analysis.

Overall, the impact of the configuration is small. The values range from 0.92 to 1.0. The received volume increases with the duty cycle since less time is spent during reconfigurations. For instance, $\rho = 1$ms increases from 0.98 at $\epsilon = 50\%$ to 0.99 with $\epsilon \geq 98\%$. The $\rho = 50$ms shows the largest gain due to a higher duty cycle growing from 0.92 to 1.0. Finally, we note that the relation between the three optimization periods is moderated by the duty cycle. While $\rho = 1$ms is dominating for the duty cycle $\epsilon = 50\%$ with $\rho = 50$ms being the worst, the opposite is the case for $\epsilon = 99\%$; this effect can mainly be deduced to the absolute value of the reconfiguration time with low duty cycles.

Another parameter is the flow classification threshold f_{th}. Its impact on the total RX volume is $< 0.1\%$ except for $f_{th} = 15$ MB which has 3% less RX volume. Figure 12 illustrates the impact of f_{th} on the 99%-ile of the FCTs for 40% offered load. We observe the impact of f_{th} in two aspects: 1) using the medium and high priority queues for small flows improves their FCT. For instance, flows of size 180 B have FCTs of ≈ 1.4 ms without priority queuing (0 B). The values are reduced with $f_{th} = 15$ MB to $\approx 67\mu$s and to $\approx 12\,\mu$s with 100 KB and 1 MB. Reducing f_{th}, i.e., considering less flows as small, results in lower FCTs for small flows. 2) the "jump" of the FCT values moves as expected. Flows that are classified as large have around two orders of magnitudes larger FCTs compared to their classification as small flows.

4.2.6 **Duo has low complexity.** Figure 13 illustrates the wall clock time for the simulations against the load. Duo, Segr, and Exp have a slower increase in run-time compared to Opera which frequently runs a costly rate allocation.

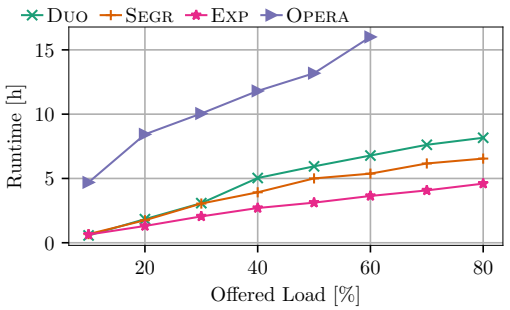


Fig. 13. Comparison of simulation wall clock time.

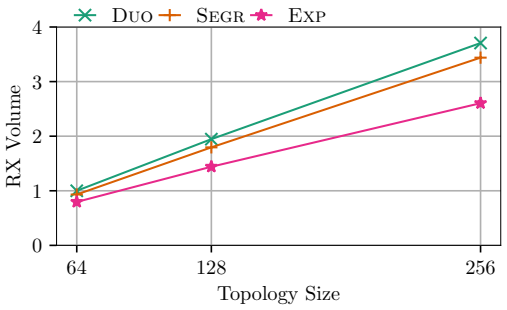


Fig. 14. Impact of topology size on RX volume.

We acknowledge that the simulation run-time does not reflect hardware implementation characteristics but is a first indicator for the computational effort and the complexity of the load balancing protocol in Opera.

4.2.7 **Performance gains persist for larger topologies.** Lastly, we assess Duo on topologies with $n = 128$ and $n = 256$ racks. The traffic is generated based on the Datamining distribution and offers a load of 60% relative to the topology size. Figure 14 illustrates the RX volume after 5 s of traffic and simulation. The results are normalized to Duo with $n = 64$. The simulations runs are shorter than 10s due to scalability. We focus only on Duo, Segr and Exp in this comparison since Opera's simulations do not scale well. All topologies have $k = 8$ uplinks. Comparing the RX volume across the topology sizes, the difference between Duo and Exp increases from 0.21 with 64 racks to 1.1 with 256 racks. For Segr, it increases from 0.07 to 0.27 respectively. This result validates that Duo can provide a higher capacity topology also at larger scales.

5 PROOF-OF-CONCEPT

To demonstrate feasibility, we implemented a proof-of-concept of Duo. In the following, we report on the results of this working prototype.

5.1 Testbed

Components and structure. We implement the data plane logic of Duo using a WEDGE 100BF-32QS Barefoot Tofino 3.2 Tbps switch and the P4 language to emulate 16 ToRs. Figure 15 overviews the testbed (the figure shows a 8 ToR version for readability). The switch (gray box) connects to four servers (blue boxes) 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). The connection is made via a 4x10G breakout cable for each server which connects to the servers' NICs. Each server generates and sends traffic corresponding to four servers in four racks (green boxes) via one of the four links. Separation is realized using *Linux network namespaces*. The Tofino switch emulates the data plane of the ToR switches and the spine layer with its static and dynamic matchings. Eight loopback cables (double headed arrows) represent the links between different switching elements; the QSFP ports are operated in breakout mode (4x10G).

The setup is operated by a single external controller. The controller deploys forwarding rules to separate the racks' traffic, forward it according to the greedy routing and also mimics the behavior of the distributed control plane.

ToR emulation concept. Generally, the emulation considers the two directions of each (10 Gbps) link as separate directed links, hence, ingress and egress directions can be assigned to different

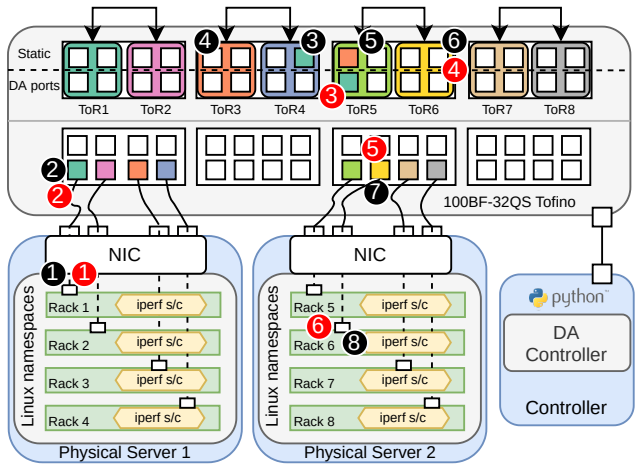


Fig. 15. Illustration of an 8 ToR testbed setup.

ToRs. In Figure 15 for instance, the egress direction of the top left port is used by the green rack (small box); the port next to it is used by the orange rack. The assignment of the ingress direction is visualized by the color of the surrounding box, e.g., for the upper left group of ports, the green ToR.

Static Packet Forwarding vs DA *Forwarding Example.* Rack 1 intends to send a packet to Rack 6; this forwarding goes along a 2-hop path (ToR1 -> ToR3 -> ToR6). First, it sends a packet on the NIC (black circle 1). The Tofino receives the packet (2) and forwards it internally on an outgoing port of ToR1 (3). Please note here how we exploit the differentiation of ingress and egress of the Tofino switch; the outgoing port of ToR1 is not located physically on the same QSFP port where the ingress happens. ToR4 receives the packet via the loopback cable (4) and then sends it out to ToR6 (5). After receiving the packet on ToR6 (6), it is forwarded to Rack6 (7) and finally received (8). Using DA links, ToR1 (red circle 3) can directly transmit the packet to ToR6 (red circle 4). This saves 1 hop. With our flexible P4-based ToR emulation, we can emulate static and DA de Bruijn DC networks.

5.2 Data & Control Plane Implementation

The P4 pipeline consists of two major parts, forwarding and elephant flow detection, which are described in the following.

5.2.1 Forwarding. A crucial aspect of emulating multiple ToRs on a single Tofino is to identify the location of the incoming packets in the emulated network, i.e., at which ToR they arrive. We implement this by a first table lookup which exactly matches the ingress port of the packet and adds the ToR id to a metadata header. The actual de Bruijn forwarding uses a second table that combines the ToR id in the metadata header as an exact key along with an LPM on the packet's destination IP address.

The addressing used in the emulated network follows our approach above (see Section 3.3.2). Since we consider a maximum of 16 ToRs, four bits are needed for the de Bruijn node ids. We use the 12 most significant bits as the base prefix, and the remaining 16 bits for the hosts.

5.2.2 Elephant flow detection. The elephant flow detection algorithm always runs when a packet arrives at its destination ToR. If this packet belongs to an elephant flow, the data plane notifies the control plane using a digest, unless it is listed as an already reported flow.

Per-flow statistics. We define a flow as the aggregate traffic on the ToR-to-ToR level. For a small number of ToRs, maintaining the per flow statistics is feasible. We use an array of low pass filters. They are built-in functionalities (externs) provided by the Tofino Native Architecture and are used to estimate the flow rates. Since we have 16 racks, 256 slots are enough to keep track of the statistics for each ToR. Instead of maintaining 16 different arrays with 16 entries, we use a single $16 \cdot 16$ long array. We get the $4 + 4$ bit long index by concatenating the source and destination ToR ids.

In case, the exact method is infeasible, probabilistic data structures such as sketches can be used.

Elephant flow threshold and detection. We consider only packets larger than 255 bytes to avoid detecting the reverse directions because of the ACKs. We classify a flow as elephant if it sends more packets than a predefined threshold within a predefined time frame. These parameters are configurable and might benefit from fine-tuning for different scenarios. Checking the threshold value needs special attention to avoid complex conditions and using too many Tofino stages[6]. To check one single value, we subtract it from the threshold and check whether the the signing bit becomes 1 indicating an underflow since the threshold was higher than the value.

Preventing control plane flooding. We introduce a simple filtering mechanism to avoid flooding the control plane with digests. After an elephant flow is detected, we set a `reported_elephants` register value from 0 to 4000 for this flow. As long as the threshold is exceeded, each subsequent packet from this decreases this value by 1. The data plane does not report the flow until this value reaches 0 again. Additionally, we have a `known_elephants` table with exact matching on the source and destination addresses. This table is maintained by the control plane and mutes the digests for the inserted source-destination key pairs.

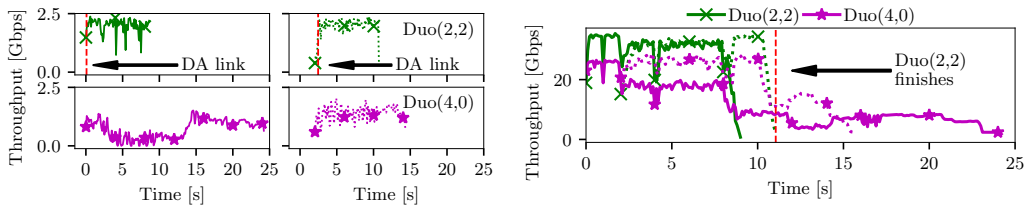
5.2.3 Control plane implementation. The Duo PoC uses a Python-based centralized controller. The controller ignores the available global knowledge and implements the distributed DA link scheduling as presented in Algorithm 3 in a sequential way. That is a single CPU processes the elephant flow notifications from the Tofino. It maintains the currently connected neighbors per-ToR. This in turn is used to derive the forwarding tables with Algorithm 1.

When bootstrapping the network, the controller populates the tables for matching ingress traffic to the correct ToR and installs the forwarding entries following the greedy routing on a static de Bruijn topology. It then starts listening for elephant flow notifications.

Processing notifications. Upon reception of an elephant flow notification, the controller extracts the sending ToR's id t (destination of the elephant flow) and the flow's source ToR id s. In order to disable transmission of further notifications, it adds the flow to the known elephants table. Afterwards, it runs *DistDA* (Algorithm 3) with the state of the destination t to get the port request. The request is passed to *DistDA* along with the state of the source s. If the request is not declined, we obtain the DA link to create and mark the DA ports in s and t as reserved. Finally, the controller checks if any elephant flow has passed its cooldown period and removes it from the `known_elephants` table on the switch and resets the `reported_elephants` register to 1000.

Setting DA links. Our emulation realizes DA links by (re-)assigning egress directions of ports to ToRs and, consequently, adding the needed forwarding rules (cf. Figure 15). Specifically, since the ingress ports of a ToR are fixed, we first infer the egress port that belongs to the loopback cable that connects to the ingress port of the DA link's destination ToR. Then, we update the forwarding table of the DA link's starting ToR following Algorithm 1 and using this egress port (i.e., colors of DA ports in Fig. 15 change over time). Reconfiguration delay is currently not emulated but can be realized by delaying the forwarding table update. In case other DA links must be removed first, the forwarding tables of the involved ToRs are updated accordingly beforehand.

[6] The exact constraints are NDA protected by Intel.



(a) Throughput per flow.

(b) Aggregate throughput per matrix.

Fig. 16. Throughput over time.

5.3 Measurement Results

Settings and traffic input. We consider Duo with 16 ToRs and two configurations: Duo (4,0) and Duo (2,2). The rates on the ToRs' up-links, i.e., the ports that are connected via the loopback cables, is limited to 2.5 Gbps using shapers available on the Tofino switch. The reconfiguration delay is $\delta = 0$ms and DA links are reserved for $r = 200$ms. Moreover, we consider a cooldown time for elephant flows of 200ms during which no notifications are sent. The servers generate traffic according to two permutation matrices which arrive with 2s inter arrival time. Each matrix element corresponds to one TCP flow of 2 Gbyte size. Note that the acknowledgements of the TCP flows can take different paths than the main data. The first matrix is constructed in such a way that all flows must use at least two hops on the static topology. The second matrix is randomly generated.
Analysis. Figure 16 compares the achieved throughput for the two configurations of Duo. Figure 16(a) focuses on the throughput per flow over time. For each configuration, it shows the two flows from both matrices which share the source ToR 2. The flow of the first matrix is shown in the left column, the one of the second matrix in the right column. For Duo (4, 0), the two flows share at least one link, after the second flow has arrived in $t = 2$s. Both do not reach the full link rate and, hence, both have a transmission duration of 24.22s and 14.33s respectively – significantly longer than the theoretic optimum of 6.4s. In particular, the first flow suffers from interference with other flows. In contrast, Duo (2, 2) uses the elephant detection to identify large flows and establishes DA links for each flow (red dashed line). As a result, both flows send at link rate and, hence, finish earlier at 8.6s and 10.73s. This demonstrates that Duo reacts appropriately to elephant flows it detects. Taking a broader perspective, Figure 16(b) illustrates the aggregated throughput per permutation matrix (here a set of 16 flows, dotted and solid lines). Duo (2, 2) achieves a higher throughput per matrix by setting DA links and the overall demand finishes $\approx 54\%$ earlier. For Duo (4, 0), the throughput for the first matrix (purple, solid) decreases when the second matrix arrives. Overall this demonstrates that Duo is realizable.

6 CONCLUSION

To address the limitations and overheads of existing reconfigurable datacenter networks, we proposed Duo, a simple and flexible architecture which supports integrated multi-hop routing and demand-aware links. We understand our work as a next step towards more practical and scalable demand-aware reconfigurable datacenter networks, and believe that it opens several interesting avenues for future research. In particular, in this paper we have primarily focused on the performance of our approach, and, similarly to previous work, less on the resilience. That said, de Bruijn networks are known to not only provide good robustness properties, but also fast recovery, and we leave a detailed empirical study for future research. Furthermore, while we have shown that Duo performs well even when using standard TCP out-of-the-box, it may be interesting to explore more specialized transport protocols to improve throughput further. Finally, we believe that our

approach is applicable to other network topologies that support greedy local routing, beyond de Bruijn topologies.

ACKNOWLEDGMENTS

Research supported by the European Research Council (ERC), consolidator project Self-Adjusting Networks (AdjustNet), grant agreement No. 864228, Horizon 2020, 2020-2025. The work was also funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 438892507.

REFERENCES

[1] Vamsi Addanki, Chen Avin, and Stefan Schmid. 2023. Mars: Near-Optimal Throughput with Shallow Buffers in Reconfigurable Datacenter Networks. In *ACM SIGMETRICS (accepted)*.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. *ACM SIGCOMM Computer Communication Review (CCR)* 38, 4 (Aug. 2008), 63–74.

[3] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proc. ACM SIGCOMM 2010 Conference*. 63–74.

[4] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. 2020. On the Complexity of Traffic Traces and Implications. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 1, Article 20 (may 2020), 29 pages.

[5] Chen Avin and Stefan Schmid. 2019. Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks. *ACM SIGCOMM Computer Communication Review (CCR)* 48, 5, 31–40.

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

[7] 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 2020 Conference*. 782–797.

[8] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proc. ACM SIGCOMM conference on Internet measurement*. ACM, 267–280.

[9] 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.

[10] 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 Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, USA). USA, 577–593.

[11] Nicolaas Govert De Bruijn. 1946. A combinatorial problem. In *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, Vol. 49. 758–764.

[12] Fred Douglis, Seth Robertson, Eric Van den Berg, Josephine Micallef, Marc Pucci, Alex Aiken, Maarten Hattink, Mingoo Seok, and Keren Bergman. 2021. FLEET—Fast Lanes for Expedited Execution at 10 Terabits: Program Overview. *IEEE Internet Computing* 25, 3 (2021), 79–87.

[13] Frank Dürr. 2016. A flat and scalable data center network topology based on De Bruijn graphs. *arXiv preprint arXiv:1610.03245* (2016).

[14] 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 Comput. Commun. Rev. (CCR)* 41, 4 (2011), 339–350.

[15] Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid. 2018. Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures. In *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 89–96.

[16] Pierre Francois, Clarence Filsfils, John Evans, and Olivier Bonaventure. 2005. Achieving sub-second IGP convergence in large IP networks. *ACM SIGCOMM Comput. Commun. Rev. (CCR)* 35, 3 (2005), 35–44.

[17] Harold N Gabow. 2018. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms (TALG)* 14, 3 (2018), 1–80.

[18] 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 2016 Conference*. ACM, 216–229.

[19] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: a scalable and flexible data center network. In *Proc. ACM SIGCOMM 2009 Conference*. 51–62.

[20] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. 2021. Cerberus: The Power of Choices in Datacenter Topology Design (A Throughput Perspective). *Proc. ACM Meas. Anal. Comput. Syst.* 5, 3, Article 38 (dec 2021), 33 pages.

[21] 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. *ACM SIGCOMM Computer Communication Review* 39, 4 (2009), 63–74.

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

[23] P. Hall. 1935. On Representatives of Subsets. *Journal of the London Mathematical Society* s1-10, 1 (1935), 26–30. https://doi.org/10.1112/jlms/s1-10.37.26

[24] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R Das, Jon P Longtin, Himanshu Shah, and Ashish Tanwer. 2014. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, Vol. 44. ACM, 319–330.

[25] Michelle Hampson. 2021. Reconfigurable Optical Networks Will Move SupercomputerData 100X Faster. In *IEEE Spectrum*.

[26] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *Proc. ACM SIGCOMM 2017 Conference*. 29–42.

[27] Bala Kalyanasundaram and Kirk Pruhs. 1993. Online weighted matching. *Journal of Algorithms* 14, 3 (1993), 478–488.

[28] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. 2009. Flyways to de-congest data center networks. In *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*.

[29] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. 2017. Beyond fat-trees without antennae, mirrors, and disco-balls. In *Proc. ACM SICOMM 2017 Conference*. 281–294.

[30] F Thomson Leighton. 2014. *Introduction to parallel algorithms and architectures: Arrays· trees· hypercubes*. Elsevier.

[31] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *Proc. ACM SIGCOMM 2019 conference*. 44–58.

[32] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. 2014. Quartz: A New Design Element for Low-Latency DCNs. *SIGCOMM Comput. Commun. Rev. (CCR)* 44, 4 (Aug. 2014), 283–294. https://doi.org/10.1145/2740070.2626332

[33] 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 Symposium on Networked Systems Design and Implementation (NSDI 20)*. 1–18.

[34] 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 SICOMM 2017 conference*. ACM, 267–280.

[35] Jeffrey C. Mogul and Lucian Popa. 2012. What We Talk About when We Talk About Cloud Network Performance. *ACM SIGCOMM Comput. Commun. Rev. (CCR)* 42, 5 (Sept. 2012), 44–48.

[36] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, Peter Steenkiste, Guyue Liu, Ao Li, Christopher Canel, Adithya Abraham Philip, Ranysha Ware, et al. 2022. Sketchlib: Enabling efficient sketch-based monitoring on programmable switches. In *Proc. 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 743–759.

[37] Pooria Namyar, Sucha Supittayapornpong, Mingyang Zhang, Minlan Yu, and Ramesh Govindan. 2021. A Throughput-Centric View of the Performance of Datacenter Topologies. In *Proc. ACM SIGCOMM Conference* (Virtual Event, USA). 349–369.

[38] 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 *Proc. ACM SIGCOMM 2022 Conference*. 66–85.

[39] Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker. 2016. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Transactions on Networking (ToN)* 24, 3 (2016), 1421–1433.

[40] Roy Schwartz, Mohit Singh, and Sina Yazdanbod. 2019. Online and Offline Greedy Algorithms for Routing with Switching Costs. *arXiv preprint arXiv:1905.02800* (2019).

[41] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM Comput. Commun. Rev. (CCR)* 45, 4 (2015), 183–197.

[42] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. 2012. Jellyfish: Networking data centers randomly. In *Proc. 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 225–238.

[43] Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, and Yueping Zhang. 2010. Proteus: a topology malleable data center network. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 8.

[44] Min Yee Teh, Zhenguo Wu, and Keren Bergman. 2020. Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering. *IEEE/OSA Journal of Optical Communications and Networking* 12, 4 (2020), B44–B54.

[45] Shay Vargaftik, Katherine Barabash, Yaniv Ben-Itzhak, Ofer Biran, Isaac Keslassy, Dean Lorenz, and Ariel Orda. 2016. Composite-Path Switching. In *Proc. 12th International on Conference on Emerging Networking EXperiments and Technologies* (Irvine, California, USA) *(CoNEXT '16)*. 329–343. https://doi.org/10.1145/2999572.2999610

[46] 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.

[47] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Ng, Michael Kozuch, and Michael Ryan. 2011. c-Through: Part-time optics in data centers. *ACM SIGCOMM Comput. Commun. Rev. (CCR)* 41, 4 (2011), 327–338.

[48] Yiting Xia, Xiaoye Steven Sun, Simbarashe Dzinamarira, Dingming Wu, Xin Sunny Huang, and T. S. Eugene Ng. 2017. A Tale of Two Topologies: Exploring Convertible Data Center Network Architectures with Flat-Tree. In *Proc. ACM SICOMM Conference* (Los Angeles, CA, USA). 295–308.

[49] Mingyang Zhang, Jianan Zhang, Rui Wang, Ramesh Govindan, Jeffrey C Mogul, and Amin Vahdat. 2021. Gemini: Practical Reconfigurable Datacenter Networks with Topology and Traffic Engineering. (2021). arXiv:2110.08374

[50] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution Measurement of Data Center Microbursts. In *Proc. 2017 Internet Measurement Conference* (London, United Kingdom) *(IMC '17)*. 78–85. https://doi.org/10.1145/3131365.3131375

[51] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y Zhao, and Haitao Zheng. 2012. Mirror mirror on the ceiling: Flexible wireless links for data centers. *ACM SIGCOMM Comput. Commun. Rev. (CCR)* 42, 4 (2012), 443–454.

[52] Shihong Zou, Xitao Wen, Kai Chen, Shan Huang, Yan Chen, Yongqiang Liu, Yong Xia, and Chengchen Hu. 2014. VirtualKnotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter. *Computer Networks* 67 (2014), 141–153.