Kevin: de Brujin-based topology with demand-aware links and greedy routing

Johannes Zerwas
TU Munich
Germany

Csaba Györgyi
University of Vienna, ELTE Eötvös Loránd University, Austria and Hungary

Andreas Blenk
TU Munich
Germany

Stefan Schmid
TU Berlin, University of Vienna
Germany and Austria

Chen Avin
Ben-Gurion University
Israel

ABSTRACT

We propose Kevin, a novel demand-aware reconfigurable rack-to-rack datacenter network realized with a simple and efficient control plane. In particular, Kevin makes effective use of the network capacity by supporting integrated and multi-hop routing as well as work-conserving scheduling. To this end, Kevin relies on local greedy routing with small forwarding tables which require local updates only during topological reconfigurations, making this approach ideal for dynamic networks. Specifically, Kevin is based on a de Brujin topology (using a small number of optical circuit switches) in which static links are enhanced with opportunistic links.

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. 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 [1, 2]. Accordingly, over the last years, great efforts have been made to improve the throughput of datacenter networks [3–6].

Emerging optical technologies enable a particularly innovative approach to improve datacenter performance, by supporting dynamic reconfigurations of the physical network topology [7–25]. In particular, optical circuit switches allow to provide dynamic connectivity in the form of *matchings* [26, 27]. 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 [10, 11, 26].

Reconfigurable datacenter networks come in two flavors: oblivious and demand-aware [28, 29]. Oblivious RDCNs such as RotorNet [10], Opera [11], and Sirius [30], 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 [26]. In contrast, demand-aware RDCNs allow to *optimize* the topological shortcuts, depending on the traffic pattern. Demand-aware networks such as ProjecToR [17], Gemini [25], or Cerberus [26], among many others [8, 9, 13, 16, 17, 23, 24, 31, 32], 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 [16, 17, 33–36]. 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 [25]).

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 often very powerful [17], current architectures support only fairly restricted routing. First, communication on the (dynamic) optical topology is often limited to a one or two hops, constraining the possible path diversity, and hence capacity, of the optical network [17, 25, 26, 30, 37]. Furthermore, routing is usually *segregated*: flows are either only forwarded along the static or the dynamic network, but not a combination of both [17, 26, 38]. 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 is also not work conserving. Demand-aware RDCNs may introduce additional delays as they require potentially time-consuming optimizations.

This paper is motivated by the desire to overcome these limitations, and to better exploit the available link resources,

	Xpander [40]	Sirius [30]	ProjecToR [17]	Gemini [25]	Cerberus [26]	Kevin
Integrated Multi-hop	Yes	2-hops	1 hop	2-hops	1-hop	**Yes**
Demand-aware	No	No	Yes	Yes	Yes	**Yes**
Work Conserving	Yes	No	No	Yes	No	**Yes**
Topology Update	None	Fast	Fast	Slow	Fast	**Fast**
Routing & Control	Simple	TBD	TBD	Simple	TBD	**Simple**

Table 1: Recent (R)DCN designs and their properties.

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 in a work-conserving manner, along *any* available link, be it static or dynamic, and in which a routing path can combine both link types. Such an integrated and work-conserving routing 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 [39]. 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 flow 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 reconfigurations. This can significantly reduce control plane overheads during topological adjustments, maintaining a simple routing and control, and is hence well-suited for highly dynamic networks. As we will show, the greedy routing approach is also compact and only requires small forwarding tables.

In particular, we present Kevin, a novel demand-aware reconfigurable datacenter network which leverages such a local control plane using a de Brujin topology (built from a small number of optical switches), in which static links are enhanced with opportunistic links. Kevin uses logical addressing, and is based on a receiver-based approach for the efficient detection of elephant flows as well as the local and collision-free scheduling of demand-aware links. The control plane of Kevin can be realized both using centralized or distributed algorithms. In both cases, it reduces buffer requirements and delays, supports very small forwarding tables based on standard longest common prefix matching, and enables fast and local route updates as well as short path lengths. Kevin is well-suited to be realized using the Sirius [30] architecture. In summary, we make the following ***contribution***:

- We present Kevin, a novel and pratical reconfigurable datacenter architecture which supports efficient multi-hop, integrated and work-conserving routing, to push the performance limits in datacenter networks. The simplicity of Kevin relies on the observation that adding shortcuts to a static de Bruijn topology allows to continue supporting greedy local routing.

2 PUTTING KEVIN INTO PERSPECTIVE

To put Kevin into perspective with the most recent proposals for datacenter network designs, we use Table 1. We first consider Xpander [40], a state-of-the-art *static and demand-oblivious* topology, 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.

We next consider Sirius [30], a recent proposal by authors from Microsoft, as an example of a *dynamic and demand-oblivious* topology (similar to [10, 11]). 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 [25, 26]. 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 some open questions regarding the complexity of the control and routing of these systems.

Then there are also systems which are dynamic and demand-aware. Systems like ProjecToR [17] 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 [25], 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

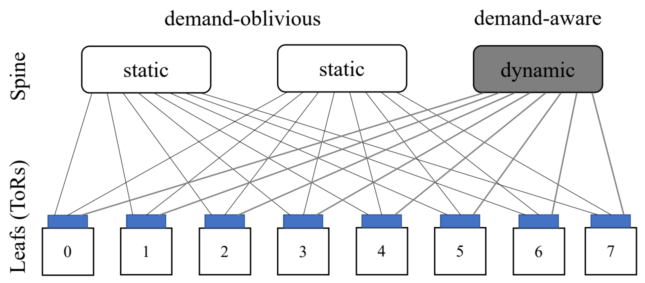


Figure 1: TMT network example with eight ToR switches and three spine switches from which two are static matchings and one is dynamic matchings.

demand-aware links, and its control and routing mechanisms are left abstract and require further investigation.

In contrast to all the above systems, Kevin features all the desired properties listed in Table 1. It supports *integrated multi-hop* routing as in 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 routing.

To put the novelty of our contribution into perspective, we note that the overheads and limitations of shortest path routing has already been studied in several contexts, including dynamic and mobile networks such as ad-hoc networks where greedy approaches such as geo-routing can be an attractive alternative [41]. We are also not the first to observe the benefits of de Bruijn based networks in dynamic settings, and there exist peer-to-peer [42–45] and parallel architectures [46] which rely on de Bruijn graphs. However, to the best of our knowledge, we are the first to study such an approach in the context of reconfigurable and demand-aware datacenter networks.

3 THE KEVIN RDCN

The rack-to-rack network provided by Kevin is based on the ToR-Matching-ToR (TMT) model [10, 26]: 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, Kevin is hybrid, in the sense that one part of the topology model is demand-oblivious and static using k_s spine switches (i.e., static matchings), and the other 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 [30].

In order to maximize performance, Kevin 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 Kevin 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 [17, 26]. Moreover routing in Kevin 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, Kevin leverages a simple sketch, sampling the flow sizes and then adjusting flow paths accordingly. In the following, we present the different components of Kevin in detail.

3.1 The Hybrid Topology

Kevin combines two topologies, a static, demand-oblivious topology (the "*backbone*"), and a dynamic, opportunistic, demand-aware topology into a unified one. Both topologies are built from matching switches according to the TMT model, forming a augmented de Bruijn network [47].

- **Static and demand-oblivious de Bruijn topology (backbone):** The static topology of Kevin relies on a de Bruijn graph. It is formed by k_s static optical circuit switches or patch panels.
- **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 the static de Bruijn topology, and how it can be built from a constant number of matchings (already two matchings suffice).

3.1.1 The Static de Bruijn Topology. We start with formally defining the de Bruijn topology [48]. 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 $n = b^d$ nodes and $m = b^{d+1}$ 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:

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

Note that the directed neighbors of node v are determined by a left *shift* operation on the address of v and entering a

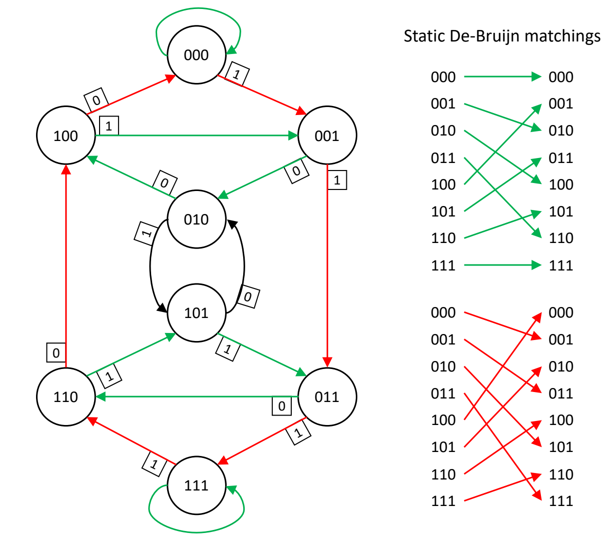


Figure 2: A $DB(2,3)$ static & directed de Bruijn graph with eight ToRs and its two corresponding matchings (colored in green and red). Each port (edge) is labeled 0 or 1 according to the performed shift operation.

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:

(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 [49]:

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

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.

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 $\mathrm{dist}_{\mathrm{DB}}(v,w)$ is the minimum number of *shift* operations needed to transform v address to w 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 $\mathrm{dist}_{\mathrm{DB}}(s,t) = 3$ and route from s to t in $DB(2,3)$ is $011 \to 110 \to 100 \to 001$ (see also Figure 2). Note that in each hop the distance to t is reducing.

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) [50]. 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 Fig. 3 shows the forwarding table of node 011 and how it is built from its neighbors 110 and 111. Note that rule $1**$, is removed from the table in the *reduce* process since it will never be used.

Following 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.*

Algorithm 1: Building the DB Forwarding Table

```
1 Function BuildTable
2     for each neighbor $z_1z_2, z_3$ at port $p$ do
3         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

```
4     Reduce the forwarding table according to LPM
```

Prefix	Port	Path-len
(neighbor 110 on port 0)		
$0**$	0	3
$10*$	0	2
110	0	1
(neighbor 111 on port 1)		
$1**$	1	3
$11*$	1	2
111	1	1

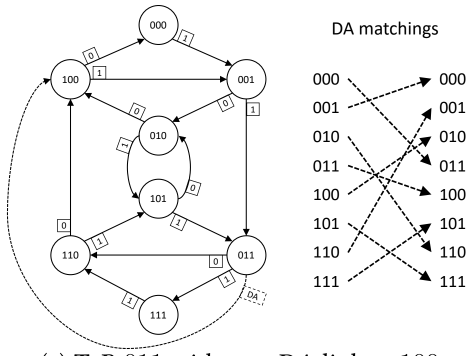
(a) Neighbors' entries

Reduced Table for ToR 011

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

(b) Reduced table

Figure 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) ToR 011 with new DA link to 100.

Prefix	Port	Path-len
$0**$	DA	3
$00*$	DA	2
100	DA	1

(b) Entries from 100.

Prefix	Port	Path-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.

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

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

3.1.3 The Demand-aware Topology. The simplicity of Kevin 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 switches or matchings for DA links. For now consider these $k_d n$ links as arbitrary links. Later we discuss how to choose these links based on flow sizes.

Let $G = DB(b, d)$ be a de Bruijn topology over the set V of nodes. 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. We can claim the following about H.

Claim 1. *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 show that H supports greedy routing, namely the next-hop is the neighbor with the shortest de Bruijn distance to the destination. While greedy routing on the static topology reduces the distance function in each hop by exactly one, DA links can reduce it by more than one hop. From the greedy routing it is clear that LPM forwarding will work and that the path is integrated in a multi-hop manner. □

Figure 4 demonstrates the $DB(2, 3)$ topology with the addition of a single demand-aware matching (showing only one DA link from 011 to 100). The figure also presents the new forwarding table at node 011, which is constructed using Algorithm 1. If we consider as before the route from $s = 011$ to $t = 001$ it will now be shorter $011 \to 100 \to 001$. In fact, all addresses with LPM $00*$ will use the new DA port for forwarding on node 011. Note also that routes toward addresses with LPM $0 * *$, like 010, have now two equal length routes (of length three), for example, $011 \to 100 \to 001 \to 010$ or $011 \to 110 \to 101 \to 010$.

Following Claim 1, we can extend this example 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 $Kevin(n, k_s, k_d)$ the Kevin 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 Kevin.

Theorem 3.2. *The $Kevin(n, k_s, k_d)$ 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 $d = O(\log_{k_s} n)$.*

3.2 Scheduling of Demand-Aware Links

Kevin 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' [30] 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 port i to establish a direct link to ToR y via switch i. Recall that each ToR has k up-link toward the k spine switches so we identify port i with switch i.

3.2.1 Centralized scheduling of DA-links. We consider two centralized algorithms for the scheduling of DA-links. Both algorithms use greedy heuristics to add shortcuts (DA links) to the backbone de Bruijn network. Both algorithms work by periodically (with *period* ρ) determining the new DA-links based on an estimate of the demand or measurement of the traffic in the network. We denote this estimation by a *demand matrix* D. Both algorithms sort the demands in D

Algorithm 2: Centralized (BFS) DA links setting

```
1  Function BFS-DA-links(D - Demand Matrix, $k_d$ - number of DA switches)
2      $\Delta$=Largest $k_d n$ demands in $D$, sorted by volume
3      forall $(s,t) \in \Delta$ from large to small do
4          $(x, switches)$ = FowardBruijn$(s,t)$
5          $(y,i)$ = BackwardBruijnBFS$(s,t,switches)$
6          if $\mathrm{dist}_{\mathrm{DB}}(s,x) + \mathrm{dist}_{\mathrm{DB}}(y,t) + 1 \leq \mathrm{dist}_{Kevin}(s,t)$ then
7              Set DA-link $(x,y,i)$
8  Function FowardBruijn$(s,d)$
9      $v = s$
10     while $v! = t$ do
11         if $v$ has available DA ports then
12             Return $(v, switches)$
13         else
14             $v$ = next-hop node toward $t$
15     Return NULL
16 Function BackwardBruijnBFS$(s,d,switches)$
17     $Q = t, i = 0$
18     while $s \notin Q$ do
19         forall $z \in Q$ do
20             if $z$ has available DA ports in switches then
21                 Return $(z, port)$
22         $i = i + 1$
23         $Q$ = all nodes $x$ with $\mathrm{dist}_{\mathrm{DB}}(x,t) = i$
24     Return NULL
```

Algorithm 3: Centralized (Greedy) DA links setting

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

by decreasing order and for each (s,t) demand in D, they try to add a DA link to the network. In case the algorithm decides to set a DA link, we assume that the *reconfiguration time* is δ and during this time the link is not available for use. Additionally when a DA link is set, it stays connected for a *reservation time* of r before it can be replaced, if needed.

The first algorithm Breadth-First-Search *(BFS)-DA-links*, shown in Algorithm 2, takes a global perspective. For each demand $(s,t) \in D$ in decreasing order, it searches for the *shortest* possible path that could be created between s and t by adding a shortcut to the de Bruijn backbone. From the source s, the search follows the path on the static topology part toward t until a node with at least one available DA-port has been found (line 12), denote it as x. The available DA ports of x are denoted as *switches* (line 4). In turn, a destination-based breadth first search is preformed until a node y with available DA port in *switches* is found (line 5). We denote the available port in y as i. If the new path (with the shortcut) is equal or shorter than the greedy path on the current Kevin topology using static + DA links (line 6), the algorithm creates a shortcut via a DA link between x and y on the i'th spine switch (line 7). Note that initially $x = s$ and $y = t$, but at a later stages of the algorithm it will create integrated multi-hop paths in Kevin.

The second centralized algorithm, *Greedy*-DA-*links*, shown in Algorithm 3, is a simple version of greedy k-matchings (known also as b-matching for undirected graphs [51]). The algorithm iterates over the requests $(s,t) \in D$ in decreasing order and only connects a direct link between s and t if they have available ports on the same DA switch i. The greedy matching is a simplified version of the *BFS-DA-links* algorithm, nevertheless, its shortcuts also support integrated multi-hop as before, 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.*

The distributed scheduling algorithm, *Dist*DA, shown in Algorithm 4, combines similar approaches as presented in ProjecToR [17] and Sirius [30]. It implements a distributed, threshold-based greedy k matchings algorithm. The algorithm is triggered by destination-based elephant detection of flows from a source. For instance, this can be done in P4 using sketches [52] 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 ports. Upon reception, the source/sender checks for an available DA-port on its side. If a port is available, it acknowledges the request via a *PortApprove(i)* message and set the link on port i. If no DA-port is available at the source, the request is declined. The receiver ToR continues to generate *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

Algorithm 4: Distributed DA-link scheduling

```
1  Function DistDA() 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 DistDA() 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
```

sophisticated schedulers (e.g., based on distributed stable matchings [17] or online algorithms [53]) for future work.

3.3 Implementation and Practical Aspects

3.3.1 Implementation and Cost. As mentioned earlier, we envision that Kevin could be implemented using the Sirius architecture [30]. 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, Sirius' architecture is in principle also well-suited for demand-aware scheduling, with a slower end-to-end reconfiguration delay.

As Kevin differs from Sirius only in the scheduling and routing, the cost and power consumption of Kevin will be similar to Sirius. In [30], 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 Kevin and Sirius is currently not possible as Sirius' simulation code is not available, we therefore concentrate on the comparison to static expander topologies which are also state-of-the-art datacenter networks [40].

Recently, Cerberus [26] which can potentially also be built on the Sirius architecture, demonstrated that using 1-hop DA links (and keeping some demand-oblivious dynamic links as in Sirius or RotorNet [10]) can increase the network throughput. We, therefore, believe that besides the conceptual contribution of Kevin, in terms of performance it could enhance any demand-oblivious existing design.

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: 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 Kevin. 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 Fig. 4(a), the de Bruijn address can directly be mapped to an IP address/prefix and occupies only 3 address bits. Using 10.0.0.0/8 as a base IP prefix, an exemplary forwarding table for ToR $5 = 011$ is shown in Figure 4(c). 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).

4 CONCLUSION

To address the limitations and overheads of existing reconfigurable datacenter networks, we proposed Kevin, a simple and flexible architecture which supports integrated multi-hop routing and demand-aware links. Kevin is work conserving and enables fast topology updates and simple control. We argued that a realization of Kevin using a Sirius topology reconfiguration model may be particularly interesting.

We understand our work as a next step towards more practical and scalable demand-aware reconfigurable datacenter networks, and believe that our work opens several interesting avenues for future research. In particular, while we demonstrated the benefits of greedy and local routing on a de Bruijn topology, we believe that our approach is more general and applicable to other network topologies that support greedy local routing.

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] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *SIGCOMM Comput. Commun. Rev. (CCR)*, vol. 42, pp. 44–48, Sept. 2012.

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

[3] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.

[4] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pp. 225–238, 2012.

[5] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *Proc. SIGCOMM Computer Communication Review (CCR)*, vol. 38, pp. 63–74, Aug. 2008.

[7] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen, *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 782–797, 2020.

[8] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng, "Mirror mirror on the ceiling: Flexible wireless links for data centers," *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 42, no. 4, pp. 443–454, 2012.

[9] S. Kandula, J. Padhye, and P. Bahl, "Flyways to de-congest data center networks," in *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2009.

[10] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 267–280, ACM, 2017.

[11] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pp. 1–18, 2020.

[12] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 339–350, 2011.

[13] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, "Firefly: A reconfigurable wireless data center fabric using free-space optics," in *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 319–330, ACM, 2014.

[14] L. Chen, K. Chen, Z. Zhu, M. Yu, G. Porter, C. Qiao, and S. Zhong, "Enabling wide-spread communications on optical fabric with megaswitch," in *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, (USA), pp. 577–593, USENIX Association, 2017.

[15] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav, "Quartz: A new design element for low-latency dcns," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 283–294, Aug. 2014.

[16] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "Osa: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking*, vol. 22, pp. 498–511, April 2014.

[17] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 216–229, ACM, 2016.

[18] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 327–338, 2011.

[19] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker, "Splaynet: Towards locally self-adjusting networks," *IEEE/ACM Transactions on Networking (ToN)*, 2016.

[20] S. B. Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Costly circuits, submodular schedules and approximate carathéodory theorems," *Queueing Systems*, vol. 88, no. 3-4, pp. 311–347, 2018.

[21] R. Schwartz, M. Singh, and S. Yazdanbod, "Online and offline greedy algorithms for routing with switching costs," *arXiv preprint arXiv:1905.02800*, 2019.

[22] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: a topology malleable data center network," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, p. 8, ACM, 2010.

[23] M. Hampson, "Reconfigurable optical networks will move supercomputerdata 100x faster," in *IEEE Spectrum*, 2021.

[24] F. Douglis, S. Robertson, E. Van den Berg, J. Micallef, M. Pucci, A. Aiken, M. Hattink, M. Seok, and K. Bergman, "Fleet—fast lanes for expedited execution at 10 terabits: Program overview," *IEEE Internet Computing*, 2021.

[25] M. Zhang, J. Zhang, R. Wang, R. Govindan, J. C. Mogul, and A. Vahdat, "Gemini: Practical reconfigurable datacenter networks with topology and traffic engineering," *arXiv preprint arXiv:2110.08374*, 2021.

[26] C. Griner, J. Zerwas, A. Blenk, S. Schmid, M. Ghobadi, and C. Avin, "Cerberus: The power of choices in datacenter topology design (a throughput perspective)," in *Proc. ACM SIGMETRICS*, 2022.

[27] E. Feder, I. Rathod, P. Shyamsukha, R. Sama, V. Aksenov, I. Salem, and S. Schmid, "Lazy self-adjusting bounded-degree networks for the matching model," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2022.

[28] C. Avin and S. Schmid, "Toward demand-aware networking: A theory for self-adjusting networks," in *ACM SIGCOMM Computer Communication Review (CCR)*, 2018.

[29] M. N. Hall, K.-T. Foerster, S. Schmid, and R. Durairajan, "A survey of reconfigurable optical networks," in *Optical Switching and Networking (OSN), Elsevier*, 2021.

[30] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen, *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 782–797, 2020.

[31] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network designs of bounded degree," in *Proc. International Symposium on Distributed Computing (DISC)*, 2017.

[32] M. Y. Teh, Z. Wu, and K. Bergman, "Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 12, no. 4, pp. B44–B54, 2020.

[33] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," in *Proc. ACM SIGMETRICS*, 2020.

[34] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 267–280, ACM, 2010.

[35] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, (New York, NY, USA), pp. 78–85, ACM, 2017.

[36] S. Zou, X. Wen, K. Chen, S. Huang, Y. Chen, Y. Liu, Y. Xia, and C. Hu, "Virtualknotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter," *Computer Networks*, vol. 67, pp. 141–153, 2014.

[37] K.-T. Foerster, M. Ghobadi, and S. Schmid, "Characterizing the algorithmic complexity of reconfigurable data center architectures," in *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2018.

[38] Y. Xia, X. S. Sun, S. Dzinamarira, D. Wu, X. S. Huang, and T. S. E. Ng, "A tale of two topologies: Exploring convertible data center network architectures with flat-tree," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), p. 295–308, Association for Computing Machinery, 2017.

[39] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving subsecond igp convergence in large ip networks," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 35–44, 2005.

[40] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla, "Beyond fat-trees without antennae, mirrors, and disco-balls," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 281–294, ACM, 2017.

[41] I. Stojmenovic, "Position-based routing in ad hoc networks," *IEEE communications magazine*, vol. 40, no. 7, pp. 128–134, 2002.

[42] C. Scheideler and S. Schmid, "A distributed and oblivious heap," *Proc. International Conference on Automata, Languages and Programming (ICALP)*, pp. 571–582, 2009.

[43] M. Naor and U. Wieder, "Novel architectures for p2p applications: the continuous-discrete approach," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 3, p. 34, 2007.

[44] P. Fraigniaud and P. Gauron, "D2b: A de bruijn based content-addressable network," *Theoretical Computer Science*, vol. 355, no. 1, pp. 65–79, 2006.

[45] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," in *International Workshop on Peer-to-Peer Systems*, pp. 98–107, Springer, 2003.

[46] A. Louri and H. Sung, "Optical binary de bruijn networks for massively parallel computing: design methodology and feasibility study," *Applied optics*, vol. 34, no. 29, pp. 6714–6722, 1995.

[47] N. G. De Bruijn, "A combinatorial problem," in *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, vol. 49, pp. 758–764, 1946.

[48] F. T. Leighton, *Introduction to parallel algorithms and architectures: Arrays· trees· hypercubes*. Elsevier, 2014.

[49] P. Hall, "On representatives of subsets," *Journal of the London Mathematical Society*, vol. s1-10, no. 1, pp. 26–30, 1935.

[50] F. Dürr, "A flat and scalable data center network topology based on de bruijn graphs," *arXiv preprint arXiv:1610.03245*, 2016.

[51] H. N. Gabow, "Data structures for weighted matching and extensions to b-matching and f-factors," *ACM Transactions on Algorithms (TALG)*, vol. 14, no. 3, pp. 1–80, 2018.

[52] H. Namkung, Z. Liu, D. Kim, V. Sekar, P. Steenkiste, G. Liu, A. Li, C. Canel, A. A. Philip, R. Ware, *et al.*, "Sketchlib: Enabling efficient sketch-based monitoring on programmable switches," NSDI, 2022.

[53] B. Kalyanasundaram and K. Pruhs, "Online weighted matching," *Journal of Algorithms*, vol. 14, no. 3, pp. 478–488, 1993.