D3: An Adaptive Reconfigurable Datacenter Network

Johannes Zerwas
TUM School of Computation, Information and
Technology,
Technical University of Munich
Munich, Germany

Chen Griner
School of Electrical and Computer Engineering,
Ben-Gurion University of the Negev
Beer-Sheva, Israel

Stefan Schmid
TU Berlin & Fraunhofer SIT
Berlin, Germany

Chen Avin
School of Electrical and Computer Engineering,
Ben-Gurion University of the Negev
Beer-Sheva, Israel

Abstract

The explosively growing communication traffic in datacenters imposes increasingly stringent performance requirements on the underlying networks. Over the last years, researchers have developed innovative optical switching technologies that enable reconfigurable datacenter networks (RCDNs) which support very fast topology reconfigurations.

This paper presents D3, a novel and feasible RDCN architecture that improves throughput and flow completion time. D3 quickly and jointly adapts its links and packet scheduling toward the evolving demand, combining both demand-oblivious and demand-aware behaviors when needed. D3 relies on a decentralized network control plane supporting greedy, integrated-multihop, IP-based routing, allowing to react, quickly and locally, to topological changes without overheads. A rack-local synchronization and transport layer further support fast network adjustments. Moreover, we argue that D3 can be implemented using the recently proposed Sirius architecture (SIGCOMM 2020).

We report on an extensive empirical evaluation using packet-level simulations. We find that D3 improves throughput by up to 15% and preserves competitive flow completion times compared to the state of the art. We further provide an analytical explanation of the superiority of D3, introducing an extension of the well-known Birkhoff-von Neumann decomposition, which may be of independent interest.

1 Introduction

Communication traffic in datacenters is growing explosively. This is due to the popularity of data-centric cloud applications such as batch processing and distributed machine learning (ML), and the trend towards resource disaggregation in datacenters [1, 2]. This results in increasingly stringent performance requirements on the underlying datacenter networks (DCNs), which are reaching their capacity limits. Accordingly, over the last few years, researchers have made great efforts to improve the capacity of DCNs [3–9].
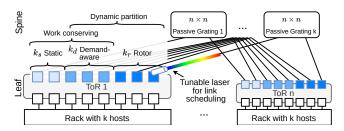

Figure 1: Overview of D3**'s architecture. The solid lines indicate bidirectional links. ToRs connect in a two layer leaf-spine topology to passive gratings. A tuneable laser at the transceivers can adjust the wavelength to select an egress port. There are three links (ports) scheduler classes STATIC, DEMAND-AWARE, and ROTOR.**

A particularly innovative approach is to render DCNs dynamic and *reconfigurable* [10]: emerging optical switching technologies allow to quickly change the network topology. In principle, such reconfigurable datacenter networks (RD-CNs) enable us to better use the network capacity by providing topological shortcuts, hence reducing multi-hop forwarding and saving *bandwidth tax* at the cost of *latency tax* [11]. Indeed, it has been shown that even *demand-oblivious* RDCNs such as RotorNet [12], Opera [13], Sirius [14], and Mars [15], whose topology changes periodically, can significantly improve the throughput in datacenters. *Demand-aware* RDCNs even allow to adapt the topology to account for structure in the workload [16–22], and, e.g., optimize the rack-to-rack interconnect toward elephant flows. Demand-aware networks such as ProjecToR [23], Gemini [24], or Cerberus [11], among others [25–43], successfully exploit the spatial and temporal locality in traffic patterns to improve performance, even if reconfigurations are performed infrequently [24, 43].

Accordingly, the best (dynamic) topology depends on the traffic and applications it needs to serve [11]. For example, shuffling traffic of map-reduce applications, or all-gather in ML, will benefit from demand-oblivious RCDNs, based on optical *rotor* switches [12–14], due to the all-to-all nature of the traffic. At the same time, workloads that are skewed, like reduce-based operations in ML or applications with large flow sizes like in datamining [44] are best served on demand-aware RDCNs, using demand-aware optical switches [24, 28, 30]. Latency-critical mice flows, on the other hand, are best served on static topologies such as fat-trees or Xpander [3–5, 8], which do not have latency tax resulting from reconfiguration delays. Matching traffic patterns to the right topology designs is, hence, critical for performance [11].

While communication patterns in datacenters naturally change over time (e.g., due to increasing loads or evolving applications), it can be difficult in practice to tailor datacenter networks toward their demand. Our proposed system, D3, is an adaptive network design that enables addressing such an evolution in a practical manner, by dynamically splitting its infrastructure between different modes of operations.

Our paper is motivated by recent optical switching technology introduced in Sirius [14] which enables us to perform topology engineering at the Top-of-the-Rack (ToR). Sirius uses nanosecond tunable lasers at the leaves (ToR switches) and passive gratings that route light based on wavelengths at its spine switches. To change the topology, the optical circuits are reconfigured via the lasers. Controlling the topology boils down to what we call *link scheduling*, scheduling the wavelengths that each laser uses at any time. Sirius realizes its RDCN based on a demand-oblivious topology, using demand-oblivious link scheduling. However, as the authors of Sirius already observed, the optical switches can also be used for adaptive link scheduling in a demand-aware manner. In fact, we claim that the switch can even be used in a polymorph manner, to quickly change between demand-oblivious, demand-aware, or static link scheduling. Changing the optical topology, whether demand-oblivious or demand-aware, simply means changing the type of scheduler that controls the tunable lasers. Moreover, different lasers (ports), even at the same ToR, can use different types of link schedulers. This, in principle, enables fully dynamic, *self-adjusting* datacenter networks: networks whose topology can instantaneously be optimized towards the traffic at any time.

Our contribution. This paper explores how to design such flexible self-adjusting datacenter networks. In particular, we propose a novel datacenter architecture, D3, based on the Sirius [14] architecture, but whose topology consists of different sub-topology components (namely a static, a dynamic demand-oblivious, and a dynamic demand-aware

Concept (Approach)	Adopted from	D3 improvement
Demand-aware links (Distributed matching)	ProjecToR [23]	Non-segregated routing
Rotor scheduling (RotorLB)	RotorNet [12]	Local LB
ToR tunable lasers (Passive gratings)	Sirius [14]	Demand-aware ports
Three sub-topologies (Static partition)	Cerberus [11]	Dynamic partition
Greedy routing (de Bruijn topology)	Duo [46]	Rotor ports
Birkhoff–von Neumann (Greedy decomposition)	Eclipse [33]	Mixed decomposition

Table 1: D3 **combines and extends several important concepts from existing systems.**

sub-topology) that can be adjusted quickly toward the evolving traffic. Figure 1 presents D3's architecture, and we explain it in more details in §3. We further study how to support such flexible architectures using efficient and *jointly* optimized link scheduling (to realize the dynamic topology) and packet scheduling (the packet forwarding strategy), as well as network and transport layers. To this end, D3 relies on a decentralized network control plane supporting greedy integrated-multihop, IP-based routing, which allows to quickly and locally react to topological changes without overheads and with minimal packet reordering. D3 employs different transport protocols for the different sub-topologies: latency-sensitive (small) flows are transmitted on the static topology using NDP [45], flows sent via rotors benefit from LocalLB (a local version of RotorLB [12]), and standard TCP is used for flows transmitted over the demand-aware ports.

D3 comes with theoretical underpinnings, and we explain D3's superiority analytically. That is, we prove that a mixture of demand-aware and demand-oblivious scheduling can improve the *demand completion time* of a scheduling that uses only a single type. For our analysis, we contribute a novel extension of the Birkhoff–von Neumann (BvN) matrix decomposition which supports mixed topologies and which may be of independent interest.

We further evaluate D3 using extensive simulations with realistic and synthetic workloads and find that it improves throughput by up to 15% while preserving or even improving flow completion times compared to the state-of-the-art.

We present the main concepts within the paper, and defer some technical details to the Appendix. *The paper raises no ethical concerns.*

Putting things into perspective and related work. To achieve its goals, the design and implementation of D3 stand on the shoulders of giants by combining and extending several important concepts and approaches from existing systems. Table 1 summarizes the main points. Similar to systems such as ProjecToR [23], D3 leverages demand-aware

links that are scheduled in a decentralized manner, using distributed matching algorithms; however, D3 additionally supports non-segregated routing, enabling an improved utilization of the available network resources. Similar to systems such as RotorNet [12], D3 leverages fast rotor scheduling to shuffle some of its traffic in a demand-oblivious manner quickly; however, in contrast to the RotorLB approach in prior work, D3 employs a local approach, which significantly reduces control plane overheads and does not require global synchronization. As mentioned, we use Sirius [14] as the infrastructure for D3, and while the Sirius technology makes D3 feasible, the two systems are conceptually different. First and foremost, Sirius is demand-oblivious as a philosophy that trades throughput with simplicity. To achieve higher throughput (as demonstrated in the evaluation section), D3 requires a more complex, nontrivial control mechanism to enable demand-aware topologies and multi-hop packet forwarding. Like Cerberus [11], D3 is tailored toward the traffic mix it serves, using a network topology that matches the demand; however, while in the Cerberus topology, the partition into static, demand-oblivious, and demand-aware dynamic switches is *fixed*, D3 allows for a dynamic adaption of the topology and its behavior, leveraging a flexible link scheduling (enabled by Sirius technology). Moreover, while Cerberus presented a more conceptual contribution and a flow-based simulation, D3, presents a feasible implementation and a packet-level simulation, as well as transport layer solutions. D3 further builds upon concepts from Duo [46]; in particular, it supports greedy routing by relying on an enhanced de Bruijn topology; however, D3 additionally also supports rotor ports to shuffle traffic quickly. Last but not least, at its core, D3 relies on a greedy Birkhoff-von Neumann (BvN) decomposition algorithm, as also used in Eclipse [33]. For D3, we extend this BvN algorithm and introduce a novel matrix decomposition that supports mixed topology types.
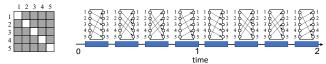
2 Motivation for Links Scheduling

Our work is motivated by the observation that in order to maximize throughput, the datacenter topology must adjust to the specific traffic pattern it serves [11]. We illustrate this with a simple motivating example. Consider an RDCN with five hosts and five racks (single host per rack). In this example, a *single reconfigurable port* in each ToR switch provides a single, ToR-to-ToR, directed matching (the reconfigurable topology) between the racks (and hosts). The link scheduler determines the current matching (topology) to use. The throughput-maximizing topology resp. link scheduling depends on the (5×5) host-to-host demand matrix.

Figure 2 presents three different demand matrices: (a) a sparse demand matrix that forms a permutation, (b) a uniform demand matrix where all host pairs have equal demand,

(a) Permutation matrix and its optimal link scheduling.

(b) Uniform matrix and its optimal link scheduling.

(c) Mixed matrix and its optimal link scheduling.

Figure 2: Motivation for D3: **the optimal dynamic topology resp. link scheduling depends on the demand matrix, which may change over time.**

and (c) a mixed scenario where darker entries denote more demand. Next to each matrix, we show its optimal schedules. The time axis shows two units of time (e.g., 100ms each), and we indicate the time it takes to serve the given demand matrix (the demand-completion time). On the time axis, in blue color, we show the circuit-hold times (i.e., the times that circuits can be used to transmit), and the gaps between them represent the reconfiguration times between two consecutive matchings. Recall that reconfiguration times are typically shorter for demand-oblivious link scheduling (which does not involve any optimization and uses pre-defined matchings) than for demand-aware link scheduling that needs to make decisions, like what links to establish. Figure 2(a): A permutation matrix is best served by a single demand-aware matching that provides direct connectivity between the racks that need to communicate. It requires a single links-change (that takes longer) but then makes no more changes. Figure 2(b): A uniform demand matrix is best served by a fast and oblivious (periodically) changing sequence of matchings (i.e., a rotor switch), always providing one direct connection between each rack pair during a periodic cycle. Figure 2(c): For a mixed demand, a combination of scheduling gives the best results: a demand-aware link scheduling configures a matching for the first unit of time, and then a sequence of fast pre-defined oblivious matchings are used in the second unit of time. We define and show this formally in §5.

We conclude that different types of link schedulers (which have a given but different reconfiguration times) are needed

to maximize the throughput. Moreover, the *type* of link scheduler on a port may change over time. However, realizing such flexible datacenter networks is challenging and requires efficient and practical links and packet schedulers, routing algorithms, and a transport protocol that can support a high degree of dynamics. In the next section, we describe the design of D3 that achieves these goals.

3 The System Design of D3

This section introduces the system design of D3. We first present the general architecture and topology components. Then, we describe in more detail the de Bruijn-based demand-aware sub-topology and the packet scheduler for the demand-oblivious sub-topology. We show how integrated forwarding and dynamic port partitioning can be realized in practice and describe the transport layer of D3 to fully reap D3's throughput benefits. Lastly, we discuss how D3 can be built on top of Sirius and the implications for cost effectiveness.

3.1 Basic Architecture and Link Scheduler

Figure 1 overviews D3's architecture. D3 relies on a two-layer, leaf-spine optical topology that can be described with the ToR-Matching-ToR (TMT) model [11]. The leaf layer consists of n ToR switches T_i ($1 \leq i \leq n$). Each ToR has k up- and k downlink ports of rate r. The latter ones connect to end-hosts so that the network contains $h = n \cdot k$ hosts in total. Since the uplinks are bidirectional, we can further separate them into k unidirectional ingress and k egress ports. The spine layer consists of k passive gratings[1] connected to the ports at the ToRs. Tunable lasers in each port of the ToRs adjust the laser wavelength to select the egress port of the grating, and by that changing a topology link, as was proposed and demonstrated in Sirius [14]. Properly configuring the laser wavelengths at each time slot creates a directed multi-hop topology that is based on a set of k-directed matchings between the ToRs, i.e., the i-th ports in the ToRs are connected to the i-th grating, creating the i-th matching (of size n). This is exactly the structure of the TMT model.

In turn, the k matchings (and the k ports in each ToR) are partitioned into three *link scheduling* classes[2]: STATIC, ROTOR, and DEMAND-AWARE. The sizes of the classes are k_s, k_r, k_d respectively, keeping the constraint $k = k_s + k_r + k_d$. A partition to three classes is implemented by assigning the k ports in each ToR switch to the different link schedulers (STATIC, ROTOR, and DEMAND-AWARE) in a consistent and symmetric way. In all ToRs, the same k_s ports use the STATIC link scheduler, the same k_r ports use the ROTOR link scheduler, and the same k_d ports use the DEMAND-AWARE link

scheduler. Each of the three link scheduling class creates a different sub-topology that are described in the following:

Static sub-topology (STATIC): The k_s static ports do not reconfigure the links over time. The resulting topology can be described as the union of k_s static matchings. The static ports provide basic connectivity between the ToRs and can be used to create k_s-regular graphs, such as expander graphs, providing low latency, multi-hop routing for short flows, and control messages. Specifically, D3 relies on de Bruijn graphs [47] for the STATIC topology, which is described later.

Demand-aware sub-topology (DEMAND-AWARE): The demand-aware topology consists of a collection of k_d DEMAND-AWARE reconfigurable ports per ToR. They can flexibly be reconfigured to *any* possible k_d-regular directed graph between the ToRs and change it over time. For example, these ports can be used to create direct connections between ToR pairs with high communication demand. The pure reconfiguration delay of a link by the lasers has been shown to be in the order of nanoseconds [14]. However, reconfiguring the DEMAND-AWARE ports requires coordination between the ToRs (e.g., for data collection and decision-making). To account for this, we denote by R_d the *reconfiguration delay* of a demand-aware port. The default value we assume is $R_d = 1ms$, but we also evaluate other values in Section 4. The circuit-hold time after each reconfiguration is dynamic (not constant) during the operation of a DEMAND-AWARE port. We require it to be much larger than R_d to achieve a high duty cycle and efficient operation.

Rotor-based sub-topology (ROTOR): The rotor-based topology is formed by the set of k_r ROTOR ports per switch. Each ROTOR port cycles through $n - 1$ predefined matchings, emulating a fully-connected network (i.e., complete graph). Every ROTOR port cycles through the same $n - 1$ matchings but using a different time shift. This synchronization provides an average *cycle time* of $\frac{n-1}{k_r}$ slots to complete a single emulation of a complete graph between all ToRs. The symmetry between the ports' link scheduling facilitates a flexible way to enable or disable a ROTOR link scheduler on a port since the only difference between the ROTOR ports is their time shift. The slot time of the ROTOR link scheduling class is defined by a circuit-hold time, denoted as δ, plus a reconfiguration time denoted as R_r, which includes the physical link reconfiguration (wavelength change) and additional quiet time to empty active links. The *duty cycle* η is the fraction of time traffic can be sent in a slot (i.e., $\eta = \frac{\delta}{\delta + R_r}$). The slot time is tuneable and depends on the reconfiguration time, where a reasonable setup is to achieve $\eta > 90\%$ as in [12–14]. Here, we assume $\eta \approx 98\%$ as in [13].

Since the partition of sub-topologies is determined by the link scheduling classes of the ports, but not by the underlying infrastructure, the assignment of ports to link scheduling

---

[1]Originally, k *active* optical spine switches in the TMT model.
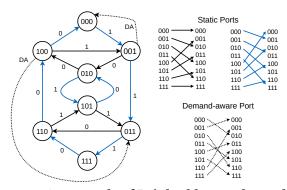[2]Denoted as spine switch types in the TMT model.

Figure 3: An example of D3**'s backbone sub-topology with 8 ToRs: A de Bruijn topology established by two** STATIC **ports and one** DEMAND-AWARE **port. Top right: two matchings for the** STATIC **ports, bottom right: the matching of the** DEMAND-AWARE **ports (for clarity, drawing only two links in the topology).**

classes can change over time and, by that, change the sub-topologies sizes. In contrast to previous proposals [11], this adds a new dimension of demand-awareness to D3 where k_s, k_d, and k_r can dynamically adapt. For instance, a DEMAND-AWARE port can become a ROTOR port if the traffic pattern has changed such that the ROTOR sub-topology component is under-provisioned. While this *dynamic partitioning* of the topology can generally be used for all three classes, D3 limits it to the DEMAND-AWARE and ROTOR ports.

The three sub-topologies have different properties for the packet forwarding behavior: STATIC and DEMAND-AWARE can utilize work-conserving forwarding (in the sense of store and forward), whereas ROTOR requires a *packet scheduling* algorithm to transmit packets successfully. Both parts are introduced in the subsequent sections.

3.2 STATIC and DEMAND-AWARE Topology

To implement the sub-topology with work-conserving forwarding, D3 augments a de Bruijn graph-based static topology built from the STATIC ports with dynamic connections (short-cuts) on the DEMAND-AWARE ports. This concept has recently been shown to be a promising candidate for high-throughput topologies [46]. In particular, and in contrast to previous proposals [11, 23], it supports *integrated multi-hop* routing across links from both the STATIC and DEMAND-AWARE topology parts, i.e., a single packet can traverse both sub-topologies to reach the destination. Moreover, the structural properties of the de Bruijn graph enable (IP-based) *greedy* routing using small forwarding tables [46].

Figure 3 shows an example of such a hybrid de Bruijn topology with eight ToRs using two STATIC ports per ToR, and one DEMAND-AWARE port. Node IDs are in binary representation. The topology is, therefore, a union of three matchings, two STATIC and demand-oblivious, and one dynamic

and DEMAND-AWARE. The matchings are shown in the right half of the figure, and we provide the main details below. Formally, the *static* de Bruijn topology is defined as follows [47]:

Definition 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: $\{v, w\} \in E \Leftrightarrow w \in \{(v_2, \ldots, v_d, x) : x \in [b-1]\}$ where $[i] = \{0, 1, \ldots, i\}$.*

The direct neighbors of a node v are determined by a left *shift* operation on the node's address and appending a new symbol $y \in [b-1]$. For example, in Figure 3 where $b = 2$ is the binary case, shifting 001 and appending 0 to it results in 010, which is one of two the neighbors of node 001 in the static sub-topology. In particular, it is the neighbor via port 0. The second neighbor is 011, via port 1, which indicates appending 1 after the shift operation. A key property of de Bruijn graph-based topologies is that they can be constructed from b directed perfect matchings. Moreover, when using longest prefix matching (LPM), the size of the forwarding table on each node has at most $bd = O(b \log_b n)$ entries [46].

The essence of the recent proposal Duo [46], was to show that augmenting a de Bruijn topology built from k_s STATIC matchings, with a k_d DEMAND-AWARE matchings creates a topology that supports integrated, multi-hop, greedy, work-conserving, LPM-based routing with a forwarding table size of $O((k_s + k_d) \log_{k_s} n)$ and diameter $d \leq \log_{k_s} n$ (Theorem 3.2 within). Moreover, the update cost of a forwarding table upon a DEMAND-AWARE link change is small and can be performed locally by communicating with the new neighbor. Routing on the de Bruijn topology can be implemented using standard IP and packet switching equipment. More details are given in Appendix A and [46]. While Duo has many benefits that we implement in D3, it does not have a ROTOR sub-topology which we discuss next.

3.3 Packet Scheduler for ROTOR Topology

As in previous work [12–14], the highly dynamic ROTOR sub-topology requires a non-work-conserving packet scheduling algorithm to transmit packets successfully. Recall that in this sub-topology, links between ToRs are constantly and systematically changing, emulating a complete graph between the ToRs while being oblivious to the demand. Therefore, packets potentially need to be stored while waiting for their next-hop link to be reconfigured. Like previous work, D3 buffers packets at the end hosts (and not at the ToR switch) and forwards them to a ToR switch based on a ToR-host synchronization protocol and the packet scheduler.

The state-of-the-art approach for this challenging task is to consider both *direct* (single hop) and *indirect* (of at most

	Offloading	Non-local	Local	Indirect (Sync.)
RotorLB (RLB)	Not supported	FS	FS	FS (global)
LocalLB (LLB)	Non-local	FS	FS	Greedy (local)

Table 2: Comparison of RotorLB to D3**'s LocalLB.**

two-hops) routes and use Valiant-based routing [48] (via a random intermediate helper node) to achieve load balancing.[3] This approach was shown to provide high throughput, i.e., the RotorLB (RLB) scheduling proposed with RotorNet and Opera [13, 14]. RLB has for each host a set of *virtual buffers* for each destination, both for *local* traffic that is generated by the host and for *non-local* traffic where the host acts as an intermediate node. Unfortunately, RLB introduces a significant control plane overhead, particularly since intermediate buffers cannot easily handle overflows, and a tight sender-receiver flow control mechanism needs to be implemented. At the beginning of every slot, hosts (in different racks) negotiate the amount of traffic that can be sent indirectly to prevent overflow. In contrast, D3 implements a simpler packet scheduler, *LocalLB* (LLB), that does not require such global synchronization. The decisions of what to send indirectly are made (rack-)locally. In case of an overload of a specific rack/host, i.e., if non-local traffic accumulates, D3 can rely on its efficient backbone topology to do offloading and forward long waiting traffic to its destination. A second difference is the reserved capacity per source. OPERA assumes a uniform distribution of the traffic and, therefore, a priori, applies an equal share of the slot capacity across all (source) hosts in a rack. The initial sending capacity assigned to each host is $\frac{C}{h}$, where C is the slot capacity and h is the number of hosts. D3 does not make such an assumption but allows a more flexible distribution of the resources.

Table 2 summarizes the scheduling process at the beginning of a slot for RLB vs. LLB. First, with LLB each host checks if non-local traffic demand must be offloaded from the ROTOR to the backbone sub-topology. The offloading (Algorithm 2 in appendix) follows a simple, greedy logic. Each host v checks for each destination u (that is not in the same rack) if the local demand exceeds a given threshold d_{off}. If so, all the non-local traffic to u is offloaded to the de Bruijn-based backbone. If the local demand to u is below the threshold, the algorithm compares the total demand from v to u and offloads excess *non-local* demand to the backbone. Note that LLB offloads only non-local demand to the backbone.

Next, the remaining non-local demand is scheduled. The procedure is the same for RLB and LLB using a fair-share (FS) algorithm over source and destination hosts. The same procedure is applied for local direct demand. After this step, RLB creates offers to be exchanged between connected ToRs,
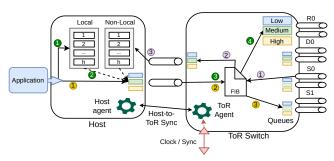
---

[3]For short, time-sensitive packets, Opera [13] uses longer multi-hop paths routing. Still, these packets capture only a small fraction of the total traffic.

Figure 4: Data- and control plane for hosts and ToRs.

thereby synchronizing demand information across the network to perform flow control. In LLB, this step is not necessary, saving complexity. For allocating new indirect traffic, RLB uses again FS to distribute the remaining capacity across the source hosts and, in particular, the destinations per host. In contrast, LLB follows a more greedy share (GS) approach since it can handle buffer overloads (see Algorithm 4 in appendix). Each host selects the destination with the largest demand and then distributes the remaining capacity in the slot to all hosts with remaining local demand again in a greedy way preferring source destination pairs with small leftover demand and capacity. The detailed algorithm is given in the appendix.

3.4 Technical Details

D3's architecture with three sub-topologies comes with several challenges regarding integration and synchronization. In the following, we first cover flow classification, feasible transport protocols, and data and control plane components. Afterward, we describe how packets are forwarded, the synchronization between hosts and ToRs, and sketch the procedure to re-assign ports to a different scheduler.

Flow classification: D3 requires a mechanism to classify flows and to separate traffic onto the three sub-topologies. One option is to estimate the flows' sizes to classify them as done in OPERA [13], CERBERUS [11], and DUO [46]. Also, D3 applies this method to distinguish STATIC and DEMAND-AWARE traffic, for instance, using approaches such as flow aging or based on information from the application. However, the ROTOR sub-topology is particularly suited for uniform traffic patterns, such as flows that belong to the shuffle phase of a map-reduce job. Therefore, in addition, D3 relies on application-level information for traffic classification. That is, either a shim layer between the application and network stack on the host inserts tags that identify ROTOR traffic, or alternatively, transport layer ports are used to identify the applications with uniform communication patterns.

Transport protocols: D3 uses different transport protocols for the three sub-topologies. Latency-sensitive (small) flows

run via the STATIC topology with NDP [45] which has shown good performance for such traffic. Flows that are transmitted via the ROTOR part are sent according to the schedules determined by LocalLB (LLB, Section 3.3). Throughput-sensitive flows use standard TCP for transmission on the STATIC and DEMAND-AWARE ports to efficiently share resources.

Data and Control Plane Components: D3 uses label-based source routing and priority queuing to forward the flows onto the different topology classes and to turn the decisions of LLB into action. Figure 4 visualizes the involved components on the hosts and the ToRs. The example considers a D3 configuration with one ROTOR (R0), one DEMAND-AWARE (D0), and two STATIC ports (S0, S1) on the ToR switch. All three flow classes share the up-link from host to ToR. In order to reduce interference, D3 uses priority queues on both hosts and ToR switches. STATIC flows are given the highest priority, followed by ROTOR and DEMAND-AWARE. Note that queues for STATIC and DEMAND-AWARE traffic are not needed on ROTOR uplink ports, but the queue for ROTOR traffic is needed on both STATIC and DEMAND-AWARE ports to enable the *offload* from ROTOR topology to the de Bruijn backbone. Besides the port queues, there are four more entities involved:

- The *Local* and *Non-local* buffers store packets for transmission via the ROTOR part. Similar to RotorNet and Opera, each of them features a dedicated virtual queue per destination in the network [12, 13].
- `Host agent` runs on every host and sends the individual sizes of the *Local* and *Non-local* buffers to a rack-local coordinator (`ToR agent`). It waits for *pull* messages from the `ToR agent` containing the volume to be sent from the buffer per destination. When sending a packet, the host adds a *slot label* which either indicates the active ROTOR matching (to send via the ROTOR topology) or is 0 if the packet is to be sent via STATIC or DEMAND-AWARE ports. (Every ROTOR port cycles through $n-1$ configurations. The slot label indicates the active configuration for all hosts.)
- `ToR agent` coordinates the ROTOR packet transmission of all hosts in a rack. It can run on the ToR or on one of the hosts. It receives demand information from the `Host agents` and runs LLB. The outcome is sent to the hosts along with the slot label to be used (*pull* messages).
- `FIB` is a single forwarding table on the ToR for all three sub-topologies. Besides the destination IP address, it matches the slot label to obtain the egress port. Thereby, forwarding to ROTOR or DEMAND-AWARE and STATIC can be differentiated. The entries for the ROTOR links can be pre-computed and do not change over time (unless the number of ROTOR ports changes). Forwarding entries for DEMAND-AWARE and STATIC are updated when local links change.

Step-by-Step Forwarding Example: Packets from applications that should be sent via the ROTOR (Green), are put to the corresponding *Local* destination queue ①. If packets are offloaded from ROTOR to the STATIC and DEMAND-AWARE sub-topologies, the `Host agent` takes packets from the queue and directly sends them ②. In this case, the 0 slot label is added to the packet. Otherwise, the packets are sent according to the calculated schedule (cf. Section 3.3) with the slot label as provided by the `ToR agent`. Packets that are sent indirectly via the ROTOR part are encapsulated with the address of the intermediate host. On the ToR, the packet is matched in the FIB ③ and forwarded using the medium priority queue ④.

Packets from applications to be sent via STATIC or DEMAND-AWARE (Non-Rotor), are tagged with 0 as slot label and then sent out to the ToR ①. Here, they are again matched in the FIB ② and forwarded accordingly ③. Traffic received on the ToR's uplinks is forwarded to the destination host ① ②. When a host receives indirect ROTOR traffic, it adds the packets to the corresponding *non-local* queue ③. Packets received on the final destination (host) are forwarded to the application. On the ToR, all packets are matched in the FIB, the slot label is popped, and the packet is sent to the egress port.

Rotor Synchronization: D3 requires two aspects of synchronization for the ROTOR sub-topology. First, ToRs need to synchronize their configuration state (slot) globally across the network. This can be achieved using a global (broadcast) clock signal (red triangle in Figure 4).

Second, D3 needs synchronization between hosts and ToRs to put the decisions of LLB into effect. Therefore, D3 considers a similar approach as RotorNet and Opera. Packets are primarily buffered on the hosts. Demand information is pushed *once* per slot from the `Host agent` to the `ToR agent`, e.g., with RDMA mesages [12]. After having calculated the number of packets to send with LLB, the `ToR agent` pulls traffic from the host, i.e., notifies the `Host agent` about how much to send (e.g., again using RDMA messages). Other approaches might be possible as well. For instance, in its rack-based deployment, Sirius can implement the needed local queues with the buffers on the ToRs. The authors refer to Credit-based flow control mechanisms as available in Infini-Band to avoid buffer explosions on the ToR. However, Sirius negotiates the scheduling almost on a packet-by-packet basis. While this approach reduces the buffer requirements, it comes at the cost of high inter-ToR synchronization effort.

Dynamic Port Partitioning: D3 can dynamically assign ports to different schedulers to adjust the proportions of the sub-topologies. Like a "normal" reconfiguration of a DEMAND-AWARE link, the reconfiguration involves changes in the FIB, which can be computed locally by the `ToR agent` or even prepared offline. In addition, the ROTOR scheduler

has to reload the new schedule for the sending lasers, which can also be pre-calculated offline. The reconfigurations must be coordinated globally, e.g., with a centralized controller. All ToRs must change at the same time and, to minimize the impact on the Rotor packet scheduling, the reconfigurations happen at the end of a Rotor slot. D3 considers two possible types of port-to-scheduler reassignments: (1) Demand-aware to Rotor and (2) Rotor to Demand-aware. In (1), the ToR agent first removes the rules with the respective port from the FIB of the ToR and clears the queues belonging to that port (similar to a Demand-aware link reconfiguration). Then, it removes the port from the Demand-aware scheduler and assigns it to the Rotor one by loading the new Rotor matchings, updating sending laser schedule and the entries in the FIB. Finally, it presents the new matchings to the packet-scheduler (LLB). No updates to the host agent are required since all necessary information for it to operate is included in the pull messages. Changing a port from Rotor to Demand-aware, happens analogously in reversed order.

Practicality and Cost of D3**:** Although D3 comes with several challenges regarding the integration of the three sub-topologies, all the described aspects in the previous sections illustrate the feasibility of D3. Since the Sirius prototype (and simulation code) are not publicly available, we base our observations on a careful study of the paper [14] and personal communication with some of the authors. We claim that the principles of D3 can be implemented and prototyped using the architecture of Sirius. This includes several main components we already discussed: link scheduling (topology reconfiguration) via tunable lasers on both sender and receiver, IP packet forwarding and multi-hop routing, and time synchronization. The control plane can be implemented using *local* rack-based agents and a *global* SDN-based controller that uses the static topology of D3 for control messages. As we mentioned earlier, a control plane for the demand-oblivious links (ports) already exists in Sirius, and a control plane for demand-aware links (ports) was already implemented, e.g., in ProjecToR [23]. Moreover, we claim that since D3 uses the same hardware as Sirius, the cost analyses of Sirius still hold. In [14], it was shown that Sirius is cost-effective compared to a classical electrical-based switched network, i.e., Sirius can reach the same throughput at a lower cost. In turn, this claim follows for D3 as well since, as we show next, D3 improves the throughput of Sirius-like systems, namely demand-oblivious, rotor-based RDCNs.

4 Empirical Evaluation of D3

We evaluate D3 with packet-level simulations using *htsim* [45]. We compare it to two state-of-the-art RDCNs, a Sirius-like, pure demand-oblivious system, Opera [13], and

a demand-aware system, Duo [46], across a range of traffic patterns that build on available empirical flowsize distributions and synthetic patterns. The evaluation focuses on throughput as the main performance metric but also investigates flow completion times and the efficiency of the resource usage.

Settings, Topologies & Traffic: We consider topologies with $n = 64$ ToRs. Each ToR has 16 bi-directional ports, which are equally split into up- and downlinks ($k = 8$). All links have a capacity of 10 Gbps. This results in a total uplink capacity of 5.12 Tbps, which is comparable to [13, 14, 46].

Table 6 (in Appendix) summarizes the used reconfiguration periods. The values are chosen such that the duty cycles in both dynamic topology parts are similar to prior work [13, 46]. Unless stated otherwise, the physical reconfiguration delay is 100 ns for Rotor switches and 1 ms for Demand-aware switches. On the Rotor part, we use a $1.7\mu s$ guard period to empty the fibers before reconfiguration. The evaluation compares the following three systems:

D3: uses the hybrid topology as presented in Section 3. Throughout the evaluation, we vary the proportions of the parts denoted by the tuple (k_s, k_r, k_d). Scheduling of Demand-aware links follows the approach for Duo (see below). We assume that the system has full knowledge of the flows' sizes at any point in time; an assumption also made in prior work [13, 46]. The allocation of flows to the scheduling classes relies on two criteria. First, we assume application-level information to identify traffic for the Rotor part using RLB (see also the traffic description later). Second, a size threshold of 1 MB separates traffic for only the Static part (using NDP) and traffic for the Demand-aware part (using TCP). Lastly, the offloading parameter is $d_{\text{off}} = 1$ packet, which is the volume that can be sent in a single slot and a single path, assuming all-to-all traffic and fair-share.

Duo [46]: has been presented in prior work, but it can be seen as a specific configuration of D3 with $k_r = 0$ (no Rotor part). We consider a configuration with $k_s = 2$ and $k_d = 6$. The Demand-aware links are scheduled greedily based on the remaining demand volume between the ToR pairs according to Algorithm 2 in [46] with a threshold of 10 MB. The algorithm has full knowledge of the flows' sizes upon their arrival. The routing uses the properties of de Bruijn-based greedy routing, i.e., paths combining Static and Demand-aware links are possible. Similar to D3, flows < 1 MB use NDP and larger flows use TCP. Queues in the two systems above can hold 50 data packets of size 1500 B.

Opera/Sirius [13]: is a demand-oblivious, dynamic topology and serves as the second baseline. Since an implementation of Sirius is not available, we use Opera as the closest representative of a Sirius-like system. It periodically cycles through a specifically-generated set of matchings that maintains an
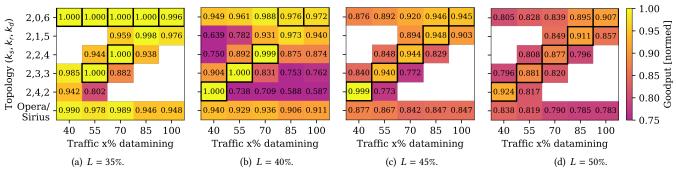
Figure 5: The average goodput over 1s of simulation. The heatmap compares topology configurations (Y-axis) and traffic shares (X-axis) for different loads L. Values are normalized to the offered traffic. The best result is highlighted with a thick border. The white regions are excluded for clarity, they are far from optimal.

expander graph at every time instance. Opera also splits the flows in low-latency and bulk traffic. Low-latency traffic is forwarded via the temporarily static expander part of the topology with NDP, whereas bulk traffic is scheduled and sent with the RotorLB protocol and scheduling [12, 13]. The evaluation uses their default configuration for queue sizes ($8 \cdot 1500$ B) and the threshold for bulk traffic is 15 MB.

Traffic: We consider an online traffic scenario where flows arrive over time according to a Poisson process. To demonstrate that D3 is particularly suited for traffic patterns that mix skewed demands with uniform patterns, we create traces that contain traffic from two distributions and vary the shares of the patterns. We denote as x the *share* of the skewed traffic. For the skewed part, connection pairs are sampled uniformly at random. If not stated otherwise, the flow sizes are sampled from available empirical distributions DATAMINING [44]. For the uniform pattern, we create a demand matrix with one flow of size 112.5 KB for each host pair in different racks. The flows of a matrix arrive over time. The load L is controlled via the arrival rates of the flows. We assume that all three systems can identify traffic belonging to the uniform traffic pattern using application-level information and/or special tags. They are configured in such a way that they forward flows belonging to the uniform traffic via the ROTOR part. We report on 10 runs per setting.

We first present results about throughput, then about individual flow performance and finally about dynamic traffic.

4.1 Throughput Evaluation

We start with evaluating the throughput of D3.

D3 Increases Throughput: Figure 5 visualizes the average goodput of 1 s of simulation for different topology configurations and traffic mixes. The values are normalized to the offered load in the respective run. A value = 1 means that the topology can fully serve the offered traffic. The skewed traffic is sampled from DATAMINING. Comparing the different loads (sub-figures), we note that the normalized values

decrease slightly with increasing load. For $L = 35\%$ (Figure 5(a)), the number of cells = 1 is highest; whereas for $L = 50\%$ (Figure 5(d)), none of the configurations can fully sustain the offered traffic. This is expected as the congestion in the topologies increases.

Looking at the individual heatmaps (e.g., Figure 5(b)), the performance of the topology configurations varies with the traffic share between DATAMINING and uniform. That is, for each share, there is one best configuration. For instance, for $x = 70\%$ (70% DATAMINING and 30% uniform traffic), D3 with $(2, 2, 4)$ achieves the highest goodput, i.e., the normalized values are closest to 1. Moreover, topology configurations with a small number of ROTOR links serve better traffic mixes with higher x. This aligns with the conclusions made in prior work [11]. The share of ROTOR links approximately corresponds to the share of uniform-size flows in the traffic. For instance, the configuration $(2, 2, 4)$ dedicates 75% of the resources to STATIC and DEMAND-AWARE and performs best for $x = 70\%$ in which 70% of the traffic uses these sub-topologies. Here, the normalized goodput is 0.999, compared to 0.892 and 0.875 for $x = 55\%$ and $x = 85\%$ respectively. To conclude, over-provisioning the ROTOR part reduces the goodput more than over-provisioning the DEMAND-AWARE one.

Compared to D3, OPERA performs worse for all considered traffic mixes (and load levels). For instance, for $L = 40\%$, OPERA achieves approximately 6% lower goodput than D3. DUO, which resembles a special case of D3 (with $(2, 0, 6)$), performs the best for the high x. Overall, choosing the correct configuration for D3 is important for maximizing its performance. If not stated otherwise, the following analyses use $L = 40\%$ and $x = 70\%$ sampled from DATAMINING.

LocalLB Throughput & Complexity: In the previous analyses, D3 uses the LocalLB proposed in Section 3.3. This scheduling can run rack-local, which reduces the synchronization overhead compared to running D3 with the RotorLB scheduling [12, Algorithm 1]. Figure 6 shows the goodput and the normalized simulation runtime (wall clock time) of both
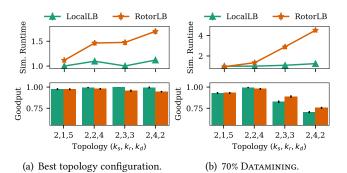
(a) Best topology configuration.   (b) 70% Datamining.

Figure 6: Comparison of D3**'s scheduling (LocalLB) and RotorLB scheduling for load** 40%**.**

scheduling approaches within D3 for $L = 40\%$. We acknowledge that the simulation runtime does not reflect the actual synchronization effort, which also depends on hardware characteristics but can give a first indication. The figure compares two cases: (a) with the topology matching the traffic share, i.e., (2,1,5) is run with $x = 85\%$, (2,2,4) with $x = 70\%$ etc., and (b) with $x = 70\%$. For all configurations, the differences in goodput between D3 and RotorLB are $< 10\%$. For the matching traffic, we observe that LocalLB achieves slightly higher goodput than RotorLB and for the fixed share, the opposite is the case. LocalLB consistently has lower run times than RotorLB. The difference increases with the number of Rotor ports and emphasizes the gain of avoiding inter-ToR synchronization for indirect traffic.

4.2 Sensitivity Analysis

To demonstrate that D3's advantages also persist in other scenarios, we evaluate its sensitivity to system parameters and traffic characteristics. in the following

4.2.1 System parameters.
We start with assessing the impact of reconfiguration times of the Demand-aware topology, higher link capacities, and varying offloading from Rotor to the Demand-aware topology.

Reconfiguration Times: Figure 10 illustrates similar heatmaps as in Figure 5 for $R_d = 10$ ms (Figure 10(a)) and $R_d = 100\mu s$ (Figure 10(b)). The reconfiguration period is adjusted so that the duty cycle remains constant. The load is $L = 40\%$. On a macroscopic level, we observe a similar behavior as before: matching the topology configuration to the traffic mix maximizes the achieved goodput. Comparing the values in detail, no consistent impact of the reduction of R_d (and the implied reduction of the reconfiguration period) is observable. Some combinations have higher, some have lower achieved goodput. Note that the values for Opera are not affected when changing R_d.

Link Capacity & Offloading: To illustrate that D3's benefits also persist for higher link capacities, Figure 7 shows the achieved goodput for scenarios with higher link capacity.

The detailed parameters are summarized in Tables 6 and 6. The load is $L = 40\%$ with the share of Datamining $x = 70\%$.

Overall, we can observe a similar behavior as for 10G. The matching topology achieves the highest throughput. The performance deviations of D3 between 10G and 100G are within the range of single-digit percentages, but there is no consistent pattern. 50G shows slightly different behavior than 10G and 100G for the configurations close to the matching one. Here, the goodput reduces almost symmetrically for too few or too many Rotor links, whereas it is asymmetrically for 10G and 100G. We leave a more detailed investigation for future work. In summary, we conclude that the gains of D3 persist also for other link capacities.
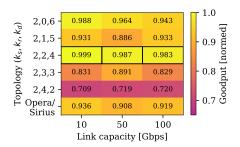
Figure 8 visualizes the impact of the offloading threshold. The link capacity is 10G. The figure compares three values 1, 2, and ∞ and also shows Opera as a reference. The values are normalized to the best configuration with $d_{\text{off}} = 1$ (2,2,4). The achieved goodput varies with the threshold. However, the ranking among the topology configurations persists. The specific impact depends on the topology configuration. For instance for (2,2,4), $d_{\text{off}} = 2$ results in slightly lower goodput than ∞ or 1. In contrast, for (2,1,5), the impact is negligible and for (2,3,3), we observe that the goodput decreases significantly as the threshold is reduced and more traffic is offloaded. Overall, the amount of offloading should be optimized.
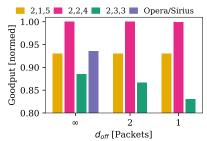
4.2.2 Traffic pattern.
In the following, we assess if D3 outperforms the reference solutions also with other loads, flow size distributions, less bursty traffic and evaluate its robustness to delayed or erroneous flow size information.

Performance for Other Loads & Distributions: Figure 11 summarizes the relative gain in goodput when using D3 compared to the indicated reference configuration. The gain is calculated as $\frac{G_{D3} - G_{ref}}{G_{ref}}$. Note that (2,0,6) (Duo) is also considered as part of D3 here. Figure 11(a) illustrates the gain against the share of skewed traffic for different loads. For $L = 35\%$, the gain varies between $0 - 4\%$ but does not show a clear trend. For $40\% \leq L \leq 50\%$, the behavior is different. For these load values and $40\% \leq x \leq 70\%$, the gain remains almost constant around 10% – neither the load nor the share of skewed traffic shows an impact here.

For a share $\geq 85\%$, the load clearly impacts the gain. It raises from $\approx 10\%$ at $L = 40\%$ to $\approx 16\%$ at $L = 50\%$. In summary, choosing the correct topology configuration becomes more critical with higher traffic skew and load.

Figure 11(b) and 11(c) illustrate the impact of the flow size distribution of the skewed traffic on the gain. In addition to Datamining, they also show the results when using Hadoop [18] and Websearch [49]. The reference configurations are Duo (2,0,6) and Opera, respectively. First, for Duo
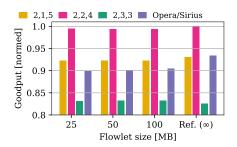
Figure 7: Heatmap of the normalized achieved goodput against the link capacity, and topology configuration.

Figure 8: Bar of the normalized achieved goodput against the LocalLB offloading threshold (d_{off}) and topology configuration.

Figure 9: Barplots of the mean goodput for traffic with large flows split into chunks (flowlets).
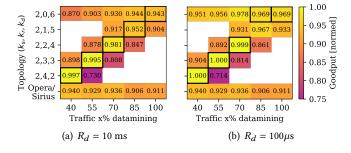

(a) $R_d = 10$ ms

(b) $R_d = 100\mu s$

Figure 10: Goodput averaged over 1s of simulation. Comparison of reconfiguration times of the Demand-aware links with fixed duty cycle. The heatmap compares topology configurations and traffic mixes.

(Figure 11(b)), we observe that the gain diminishes for all distributions when x increases. For higher shares, the optimal configuration has smaller k_r and becomes more similar to (2,0,6). In fact, for $x = 100\%$, (2,0,6) is the best configuration.
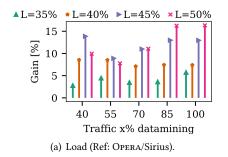
Compared to Opera (Figure 11(c)), D3 shows improvements up to 10% when skewed traffic is sampled from the Datamining or Hadoop distribution. The numbers vary depending on the exact share. For Websearch, the gain strongly grows with x (the upper-end is cut. The max gain is 112% at 100% Websearch). Also, in prior work [13, 46], Opera's throughput was shown to collapse if the amount of traffic routed via the expander part increases, as is the case with the Websearch distribution. As a general trend, we observe that the gains decrease with the average flow size of the distributions (Datamining has the highest, followed by Hadoop and then Websearch). The intuition is that Hadoop and Websearch are per se less skewed than Datamining, which overall benefits Opera. In summary, the gains from D3 persist for other skewed traffic distributions.
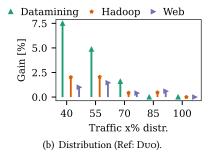
Impact of Reduced Burstiness: In the previous evaluations, the entire volume of the flow is available for transmission upon flow arrival which results in highly bursty traffic when large flows arrive. To assess the performance of D3 in a less bursty scenario, we split large flows into several flowlets (of a given size) arriving sequentially with a certain delay. The flowlet inter arrival time is calculated from the ideal transmission time. The values are normalized to the best solution (2,2,4) in the reference case in which the flowlet size is ∞, i.e., the entire flow arrives at once. Figure 9 varies the size of the flowlets. When matching the topology to the traffic almost no impact of the flowlet size is observervable. Also, the ranking between the topology configurations persists. For the not-matched configurations (2,1,5) and (2,3,3), there are slight variations in the achieved goodput. Also for Opera, the goodput decreases with the flowlet size.

Classification Errors and Delayed Information: D3 builds on the assumption that it has information about the flow size readily available to classify flows. In order to demonstrate its robustness, we relax this assumption along two dimensions: 1) delaying information about the total flow size, 2) introducing errors to the flow classification.

For the first case, Figure 12(a), we delay the flowsize information in the simulation so that the first part of the flow is handled as a small flow. The full information flowsize is then available after the time it would take to transmit the first part over an ideal link. This can also be interpreted as some kind of flow aging. When the full flowsize information is released (emulated as a second flow arrival), the flow can be properly classified and forwarded on the fitting sub-topology. The figure compares delays of 1, 5, and 10MB which correspond to 667, 3334 and 6667 packets respectively.

The figure shows the results without flow classification errors. Note that since the traffic generation for this analysis is not exactly the same as before, attention should be put on the relative differences and a 1:1 comparison of the numeric values to other figures is not meaningful. Comparing the

(a) Load (Ref: OPERA/Sirius).

(b) Distribution (Ref: DUO).
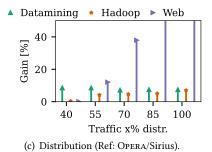
(c) Distribution (Ref: OPERA/Sirius).

Figure 11: Comparison of gain in goodput when using D3 for different flowsize distributions for the skewed traffic (a) and over the offered load (b & c). The reference topology is indicated in the caption.

(a) Fixed error = 0%.
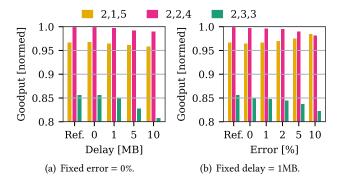
(b) Fixed delay = 1MB.

Figure 12: Barplots of the mean throughput for traffic with relaxed assumption about flowsize information. (a) varies the delay of the information and (b) varies the error.

results to the reference case, the goodput reduces slightly as the flow size information delay increases. This is expected as more traffic is routed via the static topology part. However, the reduction amounts to only $2 - 3\%$ for 10MB delay (which is already larger than the majority of the flows).

The second relaxation (Figure 12(b)), is an additional error in the flow classification (on top of a fixed flowsize information delay of 1MB). Here, $y\%$ of the uniform flows for the Rotor-topology are mis-classified and forwarded via the static topology part. The goodput of the best topology configuration for $x = 70\%$ (2,2,4) continuously reduces as the error increases. The reason is that the actual (allocated) traffic shares changes. The share of traffic allocated to the STATIC and DEMAND-AWARE topology increases. Therefore, we also observe that (2,1,5), which dedicates more resources to those topology parts, shows increasing goodput, up to the point that the ranking changes at 10% error. Lastly, the goodput of (2,3,3) reduces for the very same reason as described above.

4.3 Individual Flows' Performance

This subsection evaluates the impact on the individual flows.

Flow Completion Times (FCT): Figure 13 visualizes the FCTs against the flow sizes for D3 (2,2,4), DUO (2,0,6) and OPERA. The flows are separated into three groups for the sake of readability. The dashed lines indicate the optimal transmission time accounting for queues and propagation time (using the calculations provided by [13]). For small flows ($\leq$ 100 KB, Figure 13(a)), we consider the 99%-ile to understand better how this latency-sensitive traffic is handled. For all sizes, D3 achieves the smallest FCT, followed by OPERA and DUO. Recall that small flows use only the STATIC topology. Compared to D3, DUO's performance suffers from the uniform traffic that cannot be served well over the DEMAND-AWARE links and creates congestion on the STATIC links.

For medium sized flows (99%-ile, Figure 13(b)), D3 consistently performs better than DUO for the same reason as for the small flows. D3 also outperforms OPERA until the point where D3 starts to classify flows for the DEMAND-AWARE part (1 MB). From thereon, flows are using TCP, and there is a strong increase in the FCT, an effect that has already been observed in previous work, e.g., in DUO [46] and similarly also for OPERA [13] (for flows > 15 MB). A special point of interest are flows of size 112.5 KB, which arrive according to uniform matrices. All systems show an increased FCT here. OPERA is the best, followed by D3 (with the optimized configuration) and DUO (2,0,6). This steep increase can be explained by the fact that these flows do not fit in a single RO-TOR slot and need multiple cycles of the ROTOR sub-topology to be transmitted.

The situation changes for the FCTs of large flows ($\geq$ 100 MB, Figure 13(c)). Since for large flows, throughput matters more, we consider here the median value. D3 (2,2,4) performs best followed by DUO (2,0,6) and OPERA. Specifically, the distance between D3 and DUO is smaller compared to OPERA; in fact, the FCTs for DUO and D3 overlap for most flow sizes. This relates to our previous observations regarding the achieved goodput. In summary, integrating ROTOR into a purely demand-aware topology improves the FCT for all flow sizes. Compared to OPERA, D3 trades off benefits for

(a) Small flows.

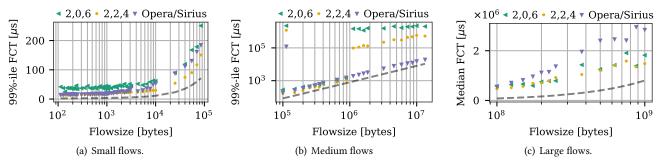(b) Medium flows

(c) Large flows.

Figure 13: Comparison of FCT. Flows separated into three groups. 99%-ile for small flows (a), medium flows (b), and the median for large flows (c). Load is 40%, traffic mix with $x = 70\%$.

the small and large flows against those of medium-sized and the Rotor flows.

Packet Reordering: To take a different perspective on the impact on the individual flows, Figure 14 illustrates the difference between the expected and received sequence number of the packets at the receive. Thereby, it hints at the packet reordering experienced in the different topologies. Again, $L = 40\%$ and 70% Datamining. The abscissae are divided into values < 0 (useless re-transmissions), $= 0$, and more fine-grained for values > 0 (reception of re-ordered packets). Comparing the systems, we observe that the behavior of (2,0,6) and D3 (2,2,4) is similar. For both, $> 80\%$ of the packets arrive in order. This is fundamentally different for Opera for which $< 20\%$ of the packets arrive in order. The shape of the remaining curve has a break around 79 packets which corresponds to the number of packets needed to transmit the flows from the uniform traffic (of size 112.5 kB). We conclude that Opera requires more effort at the receivers to forward packets in order to the applications.

4.4 Dynamic Traffic & Partitioning

As a last aspect, we assess the benefits of dynamic partitioning in D3. Figure 15 shows the goodput over time and the average of a simulation with varying traffic mix. Specifically, x increases from 55% for $< 2s$ to 70% ($2s \leq t < 4s$) to 85% ($4s \leq t < 6s$) and finally reduces to 70% again for $\geq 6s$. For each of the three traffic mixes, the figure shows the goodput of the matching configuration (2,3,3), (2,2,4) and (2,1,5) respectively. For these three static baselines, we see in the period with matching traffic that they achieve the offered load (after some transient phase) but drop in goodput when not matching the traffic. For instance, (2,3,3) significantly drops after 2s. In contrast, for the dynamic partitioning, the achieved goodput aligns with the offered load throughout the whole run. This also becomes evident from the average values. Note that the shown example is hard-coded and only demonstrates the benefits of applying the dynamic partitioning.
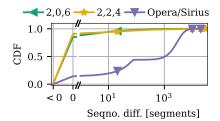
Figure 16 illustrates the rate of retransmissions captured at the senders throughout the simulation run with changing
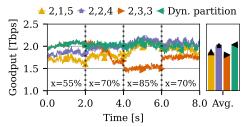
traffic mix. The retransmissions are split by the transport protocol (TCP or NDP) and normalized with the total successfully received volume at the destination per 1ms time window. By design, there are no retransmissions for RLB. The abscissa of the figure is divided to zoom-in to the time ranges close to a change in the traffic pattern, showing ±0.5s around the change. All topology configurations show retransmission rates in a similar range. For NDP, the are no retransmissions. Moreover, in all cases, we observe periodic spikes due to reconfigurations of the Demand-aware links. During these reconfigurations, all packets in the affected queues are dropped, resulting in retransmissions. The reconfiguration of the sub-topologies' sizes (at $t = 2, 4, 6s$) uses the same approach when converting a Demand-aware port to a Rotor one and vice versa (cf. Section 3.4), hence, we observe the effects from Demand-aware reconfiguration also during port conversion. Besides that, no additional impact is observed hinting at the feasibility of port conversions in D3. Note that the reconfigurations are scheduled such that they happen between two slots on the rotor topology; therefore, we also do not expect major impacts there.

5 Theoretical Modeling

This section provides an analytical explanation for the superiority of D3 over Opera/Sirius and Duo in terms of throughput. To this end, we study the *demand completion times* (DCT) of these three systems in a simplified model that captures the essence of the systems. In [11], a direct connection between the DCT, the time to send all the traffic of a demand matrix, and the throughput was proved; basically, it was shown that a shorter DCT implies a higher throughput. This led to a novel method that allows us to account for reconfiguration times when analyzing the throughput of dynamic networks.

We compare the three systems denoted as i) *da-net* where all traffic needs to be sent via the Demand-aware link scheduler (matchings) with a reconfiguration delay of R_d. ii) *rotor-net* where all traffic needs to be sent via the Rotor link scheduler that rotates between $n - 1$ predefined (oblivious) matchings with a reconfiguration delay of R_r. iii) *mix-net*
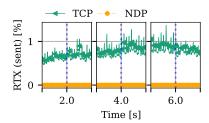
13

Figure 14: Difference in sequence number of received packets. 40% load, 70% Datamining.

Figure 15: Goodput over time and average. Dashed lines indicate changes in the traffic mix.

Figure 16: Retransmission rate per transport protocol at the sender. relative to the received volume.

a simplified D3-based system where traffic can be split (in time) between the Demand-aware link scheduler and the Rotor link scheduler. We assume all systems have a single dynamic port in the ToRs, namely a single matching, but the topology scheduling can change over time. Moreover, all systems send the same fraction of the traffic that belongs to time-sensitive flows toward Static links for multi-hop routing and, therefore, remove it from our analysis.

Following previous work [33, 50, 51], we model the traffic as a *saturated* demand matrix M, i.e., a scaled doubly stochastic matrix where the sum of each row and column is equal to the nodes capacity (depending on the links rates and the time-frame that the matrix captures). In our analysis of the DCT, we assume that delay is only due to transmission and reconfiguration times, and we neglect any delays due to packet loss or congestion. We now consider the DCT for *da-net*, *rotor-net*, and the simplified D3 in turn.

DCT of *da-net* and BvN Decomposition: Similar to previous work [33, 52], we model the DCT of *da-net* using the Birkhoff-von Neumann decomposition [53] and single-hop forwarding. Briefly, the Birkhoff-von Neumann (BvN) theorem guarantees that for any doubly stochastic matrix, M, there exists a matrix decomposition to a set $P(M) = \{P_1, \ldots, P_m\}$ of m different permutation matrices (each row and each column has exactly a single 1 entry and all other entries are 0), each with a corresponding non-negative coefficient $\alpha = \{\alpha_1, \ldots \alpha_m\}$. The sum of the coefficients stratifies the condition $\sum_{i=1}^{m} \alpha_i = 1$ when the matrix is doubly stochastic or $|M|/n$ when the matrix is saturated and $|M|$ it the sum of all entries in the M. The linear combination of these creates the original matrix that is: $\sum_{i=1}^{m} \alpha_i P_i = M$. Given that the Demand-aware link scheduler can emulate any matching, we can individually transmit each matching in P. Transmitting each P_i takes $\frac{\alpha_i}{r}$ plus the reconfiguration time R_d. The DCT for *da-net*, a demand matrix M, an algorithm BvN that computes a decomposition $\mathcal{P}(M)$, and a set of coefficients α, is the sum of transmission and reconfiguration times of each permutation:

$$DCT_{da}(M, \text{BvN}) = \sum_{i=1}^{m} \left(\frac{\alpha_i}{r} + R_d\right) = \frac{|M|}{nr} + mR_d. \quad (1)$$

While it is simple to find a decomposition of an order of $O(n^2)$ terms, the problem of minimizing the number of terms in the decomposition is known to be NP-complete [54]. Note that a minimal number of terms will also minimize the total reconfiguration time and, therefore, the DCT.

The DCT of *rotor-net*: Recall that *rotor-net* uses faster reconfigurations to rotate between $n-1$ predefined matchings, which, when combined in a full cycle, form a complete graph [12]. During each slot, there is a single active matching on which packets are sent (recall that for now, we have a single Rotor link scheduler or port). Each source-destination pair has to wait for all other $n-2$ matchings to rotate until it is available again.

In practice *rotor-net* and similar projects such as Sirius [14] are demand oblivious and use Valiant load balancing [48] routing on *all* packets. In this case, the fraction of direct single-hop traffic will always be 0 and all traffic is sent via two hops. Therefore, to approximate the expected DCT for *rotor-net* in the following sections, we assume that it uses Valiant routing, and we denote it as Val. That is, for a demand matrix M its expected DCT on *rotor-net* is

$$DCT_{rot}(M, \text{Val}) = \frac{2}{\eta r} \frac{|M|}{n}. \quad (2)$$

where $\eta = \frac{\delta}{\delta + R_r}$ is the *duty cycle*. This bound was previously informally [12, 14] and formally [11] claimed.

DCT of *mix-net* and BvN Based Partition: In the above, we provided bounds for two extreme cases: either the whole traffic, M is sent via *da-net* using only Demand-aware link scheduling, or the whole traffic is sent via *rotor-net* using only Rotor link scheduling.

The basic idea of D3 and here *mix-net*, is that a partition of the traffic in M into two parts may lead to better DCT results, namely to split M into one sub-matrix that will be sent using Demand-aware link scheduling and a second sub-matrix that will be sent using Rotor link scheduling.
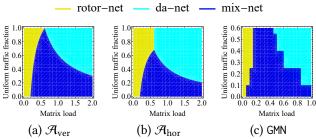
14

(a) $\mathcal{A}_{\text{ver}}$ (b) $\mathcal{A}_{\text{hor}}$ (c) GMN

Figure 17: Comparison of the DCT of *da-net*, *rotor-net*, and *mix-net*. (a) and (b) in the multi-permutation matrix $M(n, m, u, L)$ for $n = 64$ and $m = 20$, (c) on simulation-based traffic. Regions denote the system with the lowest DCT.

We extend the classic BvN problem to the problem of DCT minimization on *mix-net*. As before, let M be a saturated demand matrix. We can use the BvN theorem to decompose M into a set of at most $m \leq n^2$ scaled permutation matrices $\{M_1, M_2 \dots M_m\}$ where $M_i = \alpha_i \cdot P_i$. Since each M_i is a scaled permutation matrix, the sum of any subset of matrices is a scaled double-stochastic matrix. An algorithm $\mathcal{A}$ for *mix-net* needs to perform three operations:

(1) Decompose M into the set P of permutation matrices.
(2) Define the subset of matrices $P^{(da)} \subseteq P$ that will be served using DEMAND-AWARE link scheduling. Let $P^{(rot)} = P \setminus P^{(da)}$ be the set that will be served using RO-TOR link scheduling. We define $M^{(da)} = \sum_{P_i \in P^{(da)}} M_i$ and $M^{(rot)} = \sum_{P_i \in P^{(rot)}} M_i$. Note that $M = M^{(da)} + M^{(da)}$.
(3) Define the packet scheduling for $M^{(da)}$ using the DEMAND-AWARE link scheduling and the packet scheduling for $M^{(rot)}$ using the ROTOR link scheduling.

Given such an algorithm $\mathcal{A}$, the DCT for *mix-net* is

$$DCT_{mix}(M, \mathcal{A}) = DCT_{da}(M^{(da)}) + DCT_{rot}(M^{(rot)}) \quad (3)$$

An optimal algorithm for *mix-net* is an algorithm that minimizes the overall DCT by finding the best decomposition and best partition for $M^{(da)}$ and $M^{(rot)}$. The above definition can be extended to non-saturated matrices (for M, $M^{(da)}$ and $M^{(rot)}$), but for simplicity we keep them saturated for now.

5.1 A Case Study

In this section, we consider a simple case study to evaluate the DCT of the three systems. Let $M(n, m, u, L)$ be an $n \times n$ scaled doubly stochastic demand matrix where the sum of each row and column is equal to L. $M(n, m, u, L)$ is built from a uniform matrix M^u with $|M^u| = uLn$ and a multipermutation matrix M^m with $|M^m| = (1 - u)Ln$. M^m is the union of m scaled permutation matrices, where each row and each column has m entries of $\frac{1-u}{m}L$, and the entries on the diagonal are zero. W.l.o.g. we assume that $0 < u < 1$ and $1 \leq m \leq (n - 2)$. The matrix $M(n, m, u, L)$ is an abstraction

of the simulation traffic where a u fraction of the traffic has uniform flow sizes and a $1 - u$ fraction of the traffic is mostly sparse with large flows, so decomposable to a few matchings.

In Appendix C, we analytically compute the DCT of *da-net*, *rotor-net*, and two versions of *mix-net* ($\mathcal{A}_{\text{ver}}$ and $\mathcal{A}_{\text{hor}}$) for the case study demand matrices. Therefore, for this demand matrix, we can systematically compare when each system has the smallest DCT for $M(n, m, u, L)$. Figure 17 presents such a 2-dimensional DCT map for the reconfiguration times of the simulation setup, with $n = 64$, and $m = 20$ matchings. The Y axis of the map is the fraction of uniform traffic u, and the X axis is the normalized load $\frac{L}{r}$ where r is the link rate from the simulation. Each point on this map represents a different demand matrix $M(n, m, u, L)$. We then divide the map into three regions, and each region shows the range of parameters for which each system *da-net*, *rotor-net*, or *mix-net* has the smallest DCT.

Figure 17 (a) and (b) consider $\mathcal{A}_{\text{ver}}$ and $\mathcal{A}_{\text{hor}}$ for *mix-net*, respectively. There are some interesting observations when comparing the two figures. The blue fraction representing the vertical, $\mathcal{A}_{\text{ver}}$, algorithm is larger in (a), suggesting that, despite that both algorithms send the *same* size m of the permutation set, $\mathcal{A}_{\text{ver}}$ is more efficient. A second observation is to notice the point on the X axis where $DCT_{da} = DCT_{rot}$. Solving for x (by Equations (5) and (6)) where $x = \frac{L}{r}$ we get $x = \frac{\eta(n-1)R_d}{2-\eta}$ which for our parameters is approximately $x \approx 0.6$. Above x at least some matchings will use the DEMAND-AWARE scheduler, and below x at least some matchings will be sent to the ROTOR scheduler. This value for x gives us the bound between the *da-net* and *rotor-net* dominated areas. To generate the three different areas in the figure, we also need to find where *mix-net* is dominant (i.e., its DCT is smaller). Therefore, when, $x \geq 0.6$ (roughly), we need to find where *mix-net* is dominant over *da-net*; that is where the value given by Equation 6 is larger than the DCT value given by Equation 8 for Figure 17 (a) and Equation 7 for Figure 17 (b). When $x < 0.6$ we need to find where *mix-net* is dominant over *rotor-net* that is, where the DCT value given in Equation 5 is larger than the DCT value given by Equation 8 for figure Figure 17 (a) and Equation 7 for Figure 17 (b). Overall, we can see, as expected, that increasing the load will reach a point where *da-net* is the best system (since the longer reconfiguration time is amortized), and decreasing the load will reach a point where *rotor-net* has a smaller DCT. Looking at the Y axis, we find that with a lower u we observe higher success for *mix-net*. Lower u means a higher fraction is represented by the same 20 matchings. This makes the ROTOR scheduler less effective on them and the DEMAND-AWARE scheduler less effective on the uniform traffic, justifying the traffic partition of *mix-net*.

Algorithm 1: The `GreedyMixNet` Algorithm

---

1   Input: $M \in \mathbf{R}^{n \times n}$ scaled double stochastic

2   Initialize output: $P^{(da)} = \emptyset$, $P^{(rot)} = \emptyset$

3   Decompose M to a set permutation matrices $P(M)$ using maximum matchings

4   **forall** $P_i \in P(M)$ **do**

5      $M_i = \alpha_i \cdot P_i$

6   **if** $DCT_{da}(M_i) \leq DCT_{rot}(M_i)$ **then**

7         $P^{(da)} = P^{(da)} \bigcup P_i$

8   **else**

9         $P^{(rot)} = P^{(rot)} \bigcup P_i$

10   $M^{(da)} = \sum_{P_i \in P^{(da)}} M_i$ and $M^{(rot)} = \sum_{P_i \in P^{(rot)}} M_i$

5.2 The Greedy Algorithm

Algorithm 1 presents `GreedyMixNet` (GMN), a natural and simple algorithm to minimize DCT_{mix} for *mix-net*. Given a saturated demand matrix M as input, the algorithm first decomposes M into a set of permutation matrices $P(M)$ and the corresponding scaling factors α. It does so in a standard way, iteratively and greedily, computing a maximum matching in M, setting M_i to be a scaled permutation matrix $\alpha_i P_i$ (e.g., by considering α_i as the minimum value in the matching), and removing M_i from M. This method is guaranteed to decompose M to a set (not necessarily minimal) of scaled permutation matrices, as proven by the BvN theorem. Next, for each M_i, we then add P_i either to $P^{(da)}$ if $DCT_{da}(M_i) \leq DCT_{rot}(M_i)$ and otherwise we add it to $P^{(rot)}$. Since M_i is a scaled permutation matrix, we can easily calculate the *optimal* DCT of each individual matching M_i on both *rotor-net* and *da-net* and determine the partition of matrices to sub-systems. For *da-net* and DCT_{da}, using Equation (1) with $m = 1$ and for *rotor-net* and DCT_{rot} we can use Equation (2). We then define, as described earlier, the traffic that will use the DEMAND-AWARE scheduler as $M^{(da)} = \sum_{P_i \in P^{(da)}} M_i$ and the traffic that will use the ROTOR scheduler as $M^{(rot)} = \sum_{P_i \in P^{(rot)}} M_i$.

The DCT_{mix} of the algorithm is then defined as in Equation (3), and using Equation (1) and (2) as,

$$DCT_{mix}(M, \text{GMN}) = \frac{|M^{(da)}|}{nr} + mR_d + \frac{2}{\eta r} \frac{|M^{(rot)}|}{n}. \quad (4)$$

We can now bound the DCT of *mix-net* with GMN.

Theorem 5.1. *For any saturated demand matrix M,*

$$DCT_{mix}(M, \text{GMN}) \leq \min(DCT_{rot}(M, \text{Val}), DCT_{da}(M, \text{BvN}))$$

We can examine the performance of GMN and *mix-net* on the case study demand from Section 5.1 (i.e., $M(n, m, u, L)$). For this case, we can show that GMN and *mix-net* will match the DCT of the best of the three systems we considered for

$M(n, m, u, L)$. For example, GMN will match the results of each region in Figure 17 (a) (see Appendix C).

5.3 Empirical Evaluation of `GreedyMixNet`

We evaluate GMN on the same traffic as generated for the simulations in Section 4. The results, shown in Figure 17 (c), are similar in spirit to the theoretic results in Figure 17 (a) & (b). This shows how our simplified traffic model is able to capture the essence of more realistic traffic.

To understand this figure recall that `GreedyMixNet` needs to decompose M. We use a simple greedy decomposition algorithm based on BVN in our implementation. Given a demand matrix M we search for the maximal weighted matching and remove it from M. In the general case, we would fully decompose a matrix, however, since each subsequent matching found is getting smaller, we decompose the matrix until there is some fraction of traffic left. While we do not explicitly consider this, we assume small traffic can be sent using the STATIC sub-topology. Let us denote this fraction as L_s. In this section, we consider $L_s = 0.05$, i.e. 5% of the total traffic is assumed to be sent using the STATIC sub-topology. Since the simulation traffic does not necessarily adhere to the saturation requirement, we cannot assume that M is perfectly saturated. Each matching might also not be saturated. We, therefore, use a cutoff value for the matching. As a simple heuristic, we choose the average value over their original matching as the cutoff. Each value in the found matching will be, at most, the average value after the cutoff. In Figure 17 (c), we show results based on five real traffic traces from the simulation as described earlier, in Section 4. The specific traces used are with a load $L = 40\%$ with five different datamining shares (40%, 55%, 70%, 85%, 100%). Furthermore, for the sake of consistency between different loads, we normalize all matrices to the global mean matrix value. The traffic trace itself is a sequence of ToR-level traffic matrices, each representing a total of 0.05 seconds of accumulated remaining traffic from the simulation. That is, after each 0.05 seconds, we gather all pending traffic into an accumulated matrix. Each row represents a different datamining load, and each column represents an accumulated trace. That is, at the cell $(0.3, 0.7)$, we gather the first 14 matrices from the 70% trace. To color the cells we denote cells with less than a fraction of $\frac{1}{12}$ of the total DCT in *rotor-net* pure *da-net* and vise versa. Otherwise, we assume the optimal system is *mix-net*.

In Figure 18, we see the DCT results for four algorithms, `GreedyMixNet`, $\mathcal{A}_{\text{hor}}$, *da-net*, and *rotor-net* on simulated traces; the traffic is the same as in Figure 17 (c) for a load of 40%. Note that we could only implement $\mathcal{A}_{\text{hor}}$ and not $\mathcal{A}_{\text{ver}}$ on these traces. Recall that traffic is generated such that a fraction of the load is from uniform traffic, and a portion is sampled from datamining. To implement $\mathcal{A}_{\text{hor}}$, we sent
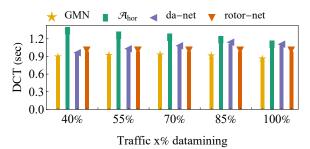
Figure 18: Empirical results on the 40% load traces showing the DCT of `GreedyMixNet`, $\mathcal{A}_{\text{hor}}$, *da-net* and *rotor-net*.

the uniform traffic to *rotor-net* and the rest to *da-net*. To implement $\mathcal{A}_{\text{ver}}$, we would need to know in advance how to decompose the matrix into a set of m 'large' matchings, for *da-net*, and a set of $n - 1 - m$ 'small' matchings, for *rotor-net*, which is not normally possible. This is different than our `GreedyMixNet`, which tries to find an optimal set m matching to send to *da-net*

Regarding the results for *rotor-net*, we see that all the results are the same for *rotor-net*. We can explain this by looking at the DCT of *rotor-net*. In Equation 2, we see that it is a function of the total load and not a function of the structure of the matrix itself. The *total* traffic of the simulated matrices was normalized to the same expected size, that is $t \cdot L \cdot r \cdot n \cdot k$, where t is the time represented by the matrix in seconds, resulting in a similar outcome. For the other algorithms, the BVN decomposition results in a different DCT due to the different sizes of the decomposition.

To conclude, this figure shows that `GreedyMixNet` outperforms all other algorithms, consistently achieving a lower DCT of about 0.88 s. For *da-net*, the DCT increases with DATAMINING share, from about 0.94 s to 1.08 s. The DCT of *rotor-net* is 0.982 s for all DATAMINING shares since all matrices have the same volume.

6   Conclusions

Motivated by the desire to maximize throughput in datacenters whose workloads evolve over time, we presented a self-adjusting datacenter network whose topology can be quickly adapted to dynamically and optimally match the workload requirements. Our architecture, D3, is enabled by Sirius and relies on a novel decentralized control plane with links and packet scheduling.

Our work opens several interesting avenues for future research. In particular, while we have primarily focused on performance, it will be interesting to study how our architecture can be extended to provide resilience under different failure models. It will also be interesting to further study scalability aspects related to the limited or heterogeneous number of ports or laser frequencies.

References

[1] Jeffrey C. Mogul and Lucian Popa. What we talk about when we talk about cloud network performance. *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, 42(5):44–48, September 2012.

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

[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, volume 38, pages 63–74. ACM, 2008.

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

[5] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A fault-tolerant engineered network. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 399–412, 2013.

[6] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[7] Haitao Wu, Guohan Lu, Dan Li, Chuanxiong Guo, and Yongguang Zhang. Mdcube: a high performance network structure for modular data center interconnection. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 25–36. ACM, 2009.

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

[9] Ankit Singla, Chi-Yao Hong, Lucian Popa, and Philip Brighten Godfrey. Jellyfish: Networking data centers, randomly. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, volume 12, pages 17–17, 2012.

[10] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. A survey of reconfigurable optical networks. *Optical Switching and Networking*, 41:100621, 2021.

[11] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. Cerberus: The power of choices in datacenter topology design (a throughput perspective). *Proc. ACM SIGMETRICS*, pages 1–33, 2021.

[12] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proc. ACM SICOMM 2017 conference*, pages 267–280. ACM, 2017.

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

[14] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi,

Benn Thomsen, et al. Sirius: A flat datacenter network with nanosecond optical switching. In *Proc. ACM SIGCOMM 2020 Conference*, pages 782–797, 2020.

[15] Vamsi Addanki, Chen Avin, and Stefan Schmid. Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(1):1–43, mar 2023.

[16] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. *Proc. ACM Meas. Anal. Comput. Syst.*, 4(1):1–29, may 2020.

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

[18] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 123–137. ACM, 2015.

[19] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proc. 9th ACM SIGCOMM conference on Internet measurement*, pages 202–208, 2009.

[20] Jeffrey C Mogul and Lucian Popa. What we talk about when we talk about cloud network performance. *ACM SIGCOMM Computer Communication Review*, 42(5):44–48, 2012.

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

[22] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proc. 2017 Internet Measurement Conference*, IMC '17, pages 78–85, 2017.

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

[24] Mingyang Zhang, Jianan Zhang, Rui Wang, Ramesh Govindan, Jeffrey C Mogul, and Amin Vahdat. Gemini: Practical reconfigurable datacenter networks with topology and traffic engineering. *arXiv preprint arXiv:2110.08374*, 2021.

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

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

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

[28] 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*, 22(2):498–511, April 2014.

[29] Michelle Hampson. Reconfigurable optical networks will move supercomputerdata 100x faster. In *IEEE Spectrum*, 2021.

[30] Fred Douglis, Seth Robertson, Eric Van den Berg, Josephine Micallef, Marc Pucci, Alex Aiken, Maarten Hattink, Mingoo Seok, and Keren Bergman. Fleet—fast lanes for expedited execution at 10 terabits: Program overview. *IEEE Internet Computing*, 25(3):79–87, 2021.

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

[32] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. Enabling wide-spread communications on optical fabric with megaswitch. In *Proc. 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 577–593, USA, 2017.

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

[34] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, 41(4):339–350, 2011.

[35] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating microsecond circuit switching into the data center. In *Proc. ACM SIGCOMM 2013 Conference*, pages 447–458, New York, NY, USA, 2013.

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

[37] Chen Avin and Stefan Schmid. Renets: Statically-optimal demand-aware networks. In *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, 2021.

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

[39] Roy Schwartz, Mohit Singh, and Sina Yazdanbod. Online and offline greedy algorithms for routing with switching costs. *arXiv preprint arXiv:1905.02800*, 2019.

[40] M. Dinitz and B. Moseley. Scheduling for weighted flow and completion times in reconfigurable networks. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1043–1052, 2020.

[41] Janardhan Kulkarni, Stefan Schmid, and Pawel Schmidt. Scheduling opportunistic links in two-tiered reconfigurable datacenters. In *33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2021.

[42] Marcin Bienkowski, David Fuchssteiner, Jan Marcinkowski, and Stefan Schmid. Online dynamic b-matching with applications to reconfigurable datacenter networks. In *Proc. 38th International Symposium on Computer Performance, Modeling, Measurements and Evaluation (PERFORMANCE)*, 2020.

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

[44] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *Proc. ACM SIGCOMM 2009 Conference*, page 51–62, 2009.

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

[46] Johannes Zerwas, Csaba Györgyi, Andreas Blenk, Stefan Schmid, and Chen Avin. Duo: A high-throughput reconfigurable datacenter network using local routing and control. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(1):1–25, 2023.

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

[48] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350–361, 1982.

[49] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference*, pages 63–74, 2010.

[50] Sangeetha Abdu Jyothi, Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. Measuring and understanding throughput of network topologies. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 761–772. IEEE, 2016.

[51] Pooria Namyar, Sucha Supittayapornpong, Mingyang Zhang, Minlan Yu, and Ramesh Govindan. A throughput-centric view of the performance of datacenter topologies. In *Proc. ACM SIGCOMM Conference*, page 349–369, 2021.

[52] Cheng-Shang Chang, Wen-Jyh Chen, and Hsiang-Yi Huang. Birkhoff-von neumann input buffered crossbar switches. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, volume 3, pages 1614–1623. IEEE, 2000.

[53] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucuman, Ser. A*, 5:147–154, 1946.

[54] Fanny Dufossé and Bora Uçar. Notes on birkhoff–von neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 497:108–115, 2016.

A Details on de Bruijn graph-based topology

Prefix	Port	Len	IP
11*	DA	2	10.192.0.0/10
101	$\{1, DA\}$	3	10.160.0.0/11
011	$\{1, DA\}$	3	10.96.0.0/11
010	1	3	10.64.0.0/11
001	1	3	10.32.0.0/11
000	0	3	10.0.0.0/11
100	Local	0	10.128.0.0/11

Table 3: Forwarding table at node 100 **in Figure 3.**

Table 3 shows the forwarding table of node 010 in Figure 3 (including the DEMAND-AWARE port). The table includes the forwarding port IDs, 0, 1 or DA, the path length to destinations using this rule, and the IP representation. Besides the logical addressing in the de Bruijn graph, it also illustrates how these addresses can be embedded into IP(v4) addresses and, thereby, how the de Bruijn graph-based topology component can be realized with today's packet switching equipment. Note also that multiple, equal cost paths are supported.[4]

B Additional Material for the Design of D3

Variable	Description
n	Total number of ToRs.
k	Total number of spines
k_r	Total number of rotor spines
T_i	ToR i, $1 \le i \le n$
R_i	Rotor i, $1 \le i \le k_r$
$h = n \cdot k$	Total number of hosts
$C = r \cdot \delta$	Slot active capacity (volume to send)
r	Link rate
$\bar{C} = \frac{C \cdot k_r}{k}$	host ToR rotor capacity
$h_{i,j}$	Host i on ToR j (can be translated to an integer for indexing)
$u_{i,j}$	Uplink i on ToR j, ($1 \le i \le k_r$)
$T_{u_{i,j}}$	ToR connected on uplink i on ToR j
Input	
$D^l_{i,j}$	current local demand from host i to host j
$D^n_{i,j}$	current non-local demand from host i to host j
Output	
$S^{ld}_{i,j}$	direct local volume to send from host i to j
$S^n_{i,j}$	non-local volume to send from host i to j
$S^{li}_{i,j,k}$	indirect local volume to send from host i to j with final destination k

Table 4: Notation.

---

[4]More details on embedding the de Bruijn address into the IP addresses, ports IDs, and the table update procedure are provided in [46].

B.1 D3 **Packet Scheduling (LocalLB)**

This section provides the algorithms underlying D3's packet scheduling. First, Table 4 summarizes the used notation. In particular, it defines the input (sizes of the local and non-local buffers) and output data structures (allocated volumes from local and non-local to the destination and from local to an intermediate host). Algorithm 3 is a sub-routine used by LocalLB. The routine is summarized in pseudo-code from prior work (the code base) from OPERA [13]. Given a matrix of demand to be sent, it applies a 2-dimensional fair share over the sending and receiving capacities (given as vectors). Therefore, it repeats sweeping the rows (l. 5f) and then the columns (l. 7f) until the output values converged.

Algorithm 4 is the full packet scheduling algorithm with the major steps as described in Section 3.3. It takes a per ToR perspective (the algorithm runs on ToR t); all data structures are rack-local but the algorithm listing partially uses global indices to simplify notation.

In lines 2-6, the algorithm allocates the second hop non-local traffic and local traffic that can be sent directly to the destination. For each buffer type, it iterates over the ROTOR ports and per port collects the demand information (local or non-local respectively) of all source and destination hosts that are connected by the port. Then, it allocates the demand using the 2-dimensional fairshare (FS2D, l. 5). The second part (lines 7- 21) allocates traffic to be sent via an intermediate host. Again, the algorithm iterates over the ROTOR ports. Per port, it iterates over the hosts in the destination rack (with the order changing in a round-robin fashion between calls of the packet scheduling). For each such candidate intermediate host, it searches for the destination with the highest demand per source host (lines 9-13). Then, it iterates over these demands in non-decreasing order. If a demand z_i exceeds the remaining capacity in the slot to the intermediate host, the equal-share of the remaining capacity is given to all remaining demands > 0 (lines 15-18). If the demand does not exceed the remaining capacity to the intermediate host, it is greedily allocated (l. 20f). Note that the algorithm does not evaluate the capacity of the slot from the intermediate node to the actual destination but relies on D3's offloading (Algorithm 2) to compensate for overloads.

B.2 D3 **FIB**

To illustrate the content of ToRs' FIBs, we consider a network with eight ToRs. We consider a point in time at which STATIC and DEMAND-AWARE ports are connected according to the scenario depicted in Figure 3. In addition to the ports shown there, each ToR has one ROTOR port. The IP addressing scheme follows the description in [46]. We use bits 22-24 to encode the de Bruijn node addresses. The lower 21 bits can be used to address hosts inside the rack. The rules match on a

#	Slot Label	Dst. Addr.	Port
1	0	10.192.0.0/10	DA
2	0	10.160.0.0/11	$\{1, DA\}$
3	0	10.96.0.0/11	$\{1, DA\}$
4	0	10.64.0.0/11	1
5	0	10.32.0.0/11	1
6	0	10.0.0.0/11	0
7	0	10.128.0.0/11	Local
8	1	10.96.0.0/19	{ROTOR }
9	2	10.128.0.0/19	{ROTOR }
10	3	10.160.0.0/19	{ROTOR }
11	4	10.192.0.0/19	{ROTOR }
12	5	10.224.0.0/19	{ROTOR }
13	6	10.0.0.0/19	{ROTOR }
14	7	10.32.0.0/19	{ROTOR }
15	*	*	drop

Table 5: Examplary forwarding table for node 100 following the STATIC and DEMAND-AWARE topologies in Figure 3 with one additional ROTOR port.

combination of exact match (for the slot label) and LPM (for the destination address). To give an example, Table 5 shows the FIB for node 100. The upper part (note that the order was chosen arbitrarily and does not necessarily reflect the order in memory/the order of the lookup) shows the forwarding rules for the STATIC and DEMAND-AWARE part and essentially matches Table 3. The only addition is an exact match on the slot label to be 0. The lower part of the FIB shows the rules for the ROTOR forwarding. Here, the matching value of the slot label depends on the active slot. The match also considers the destination address to perform forwarding if multiple ROTOR ports are present. Packets that do not match any rule are dropped.

Algorithm 2: D3 Offloading (from Rotor to de Bruijn) at host v

---

1 **for** *dest $u = 1 \ldots h$, (v and u not in same rack)* **do**
2   **if** $D_{v,u}^l > d_{\text{off}}$ **then**
3   Offload $D_{v,u}^n$ to backbone topology
4   **else**
5   **if** $D_{v,u}^l + D_{v,u}^n > d_{\text{off}}$ **then**
6   Offload $D_{v,u}^l + D_{v,u}^n - \frac{C}{k}$ from $D_{v,u}^n$ to
         backbone

Algorithm 3: 2-dimensional fair share (from OPERA's repository [13]).

---

1 $I \in \mathbf{R}^{N \times M}, c0 \in \mathbf{R}^N, c1 \in \mathbf{R}^M$

2 Initialize output: $O \in \mathbf{R}^{N \times M}$

3 Count elements > 0 in I as n^+

4 while $n^+ > 0$ *and max iterations not reached* **do**

5   **for** $i = 1 \ldots N$ **do**

6 $t[i]$ = 1-dimensional fair share on $I[i]$ with capacity $c0[i] - \sum_{j=1 \ldots M} O[i, j]$

7   **for** $j = 1 \ldots M$ **do**

8 $t2$ = 1-dimensional fair share on j-th column of t with capacity $c1[j] - \sum_{i=1 \ldots N} O[i, j]$

9   Update t with $t2$

10   Update I, O with t

11   Zero out rows $I[i]$ where $c0[i] = 0$

12   Zero out columns $I[j]$ where $c1[j] = 0$

13   Count elements > 0 in I as n^+

C Analysis of Case Study

We first consider the DCTs of *rotor-net* and *da-net*. For *rotor-net*, we can use Equation (2), so

$$DCT_{rot}(M(n, m, u, L)) = 2\frac{|M(n, m, u, L)|}{\eta n r} = \frac{2L}{\eta r}. \quad (5)$$

For *da-net*, we can decompose the matrix into the m original permutation matrices and additionally a set of $(n-1-m)$ permutation matrices for the rest of the uniform matrix. In total, we have $n - 1$ matchings. Using Equation (1), we get the following result,

$$DCT_{da}(M(n, m, u, L)) = \frac{L}{r} + (n - 1)R_d. \quad (6)$$

In both cases, we note that the DCT is a function of the the sum of a row L.

Next, we study *mix-net*. For this, we need first to find a partition of $M(n, m, u, L)$ into $M^{(da)}$ and $M^{(rot)}$. We consider two potential algorithms: $\mathcal{A}_{hor}$ and $\mathcal{A}_{ver}$.

The $\mathcal{A}_{hor}$ algorithm (horizontal) is motivated by the simulation setup and sets $M^{(da)} = M^m$ and $M^{(rot)} = M^u$ where all uniform traffic is sent with the ROTOR link scheduler, and all the permutation matrices are sent with the DEMAND-AWARE scheduler. The DCT_{mix} in this case is

$$DCT_{mix}(M(n, m, u, L), \mathcal{A}_{hor}) = DCT_{da}(M^m) + DCT_{rot}(M^u)$$
$$= u\frac{L}{r} + mR_d + (1 - u)\frac{2L}{\eta r}. \quad (7)$$

Algorithm 4: D3 Packet-Scheduling (LocalLB) for ToR t

---

1 $C_i^{TX} \leftarrow \bar{C}, C_{i,b}^{RX} \leftarrow \frac{C}{k}, \forall i = 1 \ldots k, \forall b = 1 \ldots k_r$

   // Allocate Non-local (n) on the 2nd hop and local (l)

2 for $x \in \{n, l\}$ **do**

3   **for** $b = 1 \ldots k_r$ **do**

4 $W_{i,j} \leftarrow D_{h_{i,t}, h_{j,T_{b,t}}}^x \quad \forall i = 1 \ldots k, j = 1 \ldots k$

5 $S^x \leftarrow FS2D(W, C^{TX}, C^{RX})$

6   Update C^{RX}, C^{TX}, D^x

   // Allocate new indirect traffic on the remaining capacity

7 for $b = 1 \ldots k_r$ **do**

     // Iterate over all hosts in ToR connected via b

8   **for** $\forall a \in \{h_{j,T_{b,t}}, j = 1 \ldots k\}$, *order RR over slots* **do**

      // Find the largest demand per source host capped by remaining sending capacity

9 $V_{i,m} \leftarrow D_{h_{i,t}, h_m}^l \quad \forall i = 1 \ldots k, m = 1 \ldots nk$

10 $v_{i,m} = v_{i,m} - \frac{C}{k}$ and zero all < 0

11 $y_i \leftarrow \arg\max_m v_{i,m}, \forall i = 1 \ldots k$

12 $z_i \leftarrow \min\{v_{i,y_i}, C_i^{TX}\}$

13 $n_z \leftarrow$ number of $z_i > 0$

14   **for** $\forall i$ *where* $z_i > 0$ *in non-decreasing order of* z_i **do**

15   **if** $z_i > C_{a,b}^{RX}$ **then**

16   Allocate $\frac{C_{a,b}^{RX}}{n_z}$ for all i with $z_i > 0$

17   Update C^{TX}, C^{RX}, D^l

18   Break

19   **else**

20   Allocate z_i for i and update C^{TX}, C^{RX}, D^l

21   Decrement n_z

For the $\mathcal{A}_{ver}$ algorithm (vertical), we again decompose $M(n, m, u, L)$ into a set of m matchings sent with the DEMAND-AWARE scheduler, but here the total traffic in the m matchings also includes traffic from the uniform matrix; the remaining matrix is sent with the ROTOR scheduler. The DCT is determined by the average row size of $M^{(da)}$ and $M^{(rot)}$. In $M^{(da)}$, a cell form one of the m matchings, has $L\frac{1-u}{m}$ bits from M^m and $L\frac{u}{n-1}$ bits from M^u. For the remaining entries, each of the $(n - 1 - m)$ non-zero entries has $L\frac{u}{n-1}$ bits. The DCT_{mix}

Type	Parameter	10 G			50 G			100 G		
		Opera	D3	Duo	Opera	D3	Duo	Opera	D3	Duo
Rotor	R_r	$10\mu s + 1.7\mu s$	$100ns + 1.7\mu s$	NA	$10\mu s + 0.74\mu s$	$100ns + 0.74\mu s$	NA	$10\mu s + 0.62\mu s$	$100ns + 0.62\mu s$	NA
	δ	$33.46 \cdot R_r$	$54.56 \cdot R_r$	NA	$13.65 \cdot R_r$	$14.65 \cdot R_r$	NA	$10.93 \cdot R_r$	$12.88 \cdot R_r$	NA
	Duty cycle	97%	98%	NA	93%	93%	NA	92%	92%	NA
Demand-aware	R_d	NA	$1ms = 10k \cdot R_r$	$1ms$	NA	$100\mu s = 2k \cdot R_r$	$100\mu s$	NA	$100\mu s = 1k \cdot R_r$	$100\mu s$
	δ_d	NA	$49R_d$	$49R_d$	NA	$49R_d$	$49R_d$	NA	$49R_d$	$49R_d$
	Duty cycle	NA	98%	98%	NA	98%	98%	NA	98%	98%

Table 6: Reconfiguration parameters.

is then

$$DCT_{mix}(M(n, m, u, L), \mathcal{A}_{ver}) =$$
$$\frac{Lm}{r}\left(\frac{1-u}{m} + \frac{u}{n-1}\right) + mR_d + \frac{2Lu}{\eta r}\left(\frac{n-1-m}{n-1}\right). \quad (8)$$

C.1 Formal results

Formally for $M = M(n, m, u, L)$, we can show the following:

Claim 1. *Let M, a demand matrix be $M = M(n, m, u, L)$, than the following holds for the algorithm* GMN.

$$DCT_{mix}(M, \text{GMN}) =$$
$$= \min\{DCT_{mix}(M, \mathcal{A}_{ver}), DCT_{da}(M, \text{BvN}), DCT_{rot}(M, \text{Val})\} \quad (9)$$

PROOF. After decomposing M using BvN, we first observe that there are only two sizes of permutations matrices in the decomposition. The first, has a size of $\frac{uL}{n-1}$ per row. These permutations were made directly from the matrix M^u. And an other set of permutations made from the overlap between M^u and M^m, these will have a row size of $\frac{(1-u)L}{m(n-1)} + \frac{uL}{n-1}$. Since GMN tests each permutation on an individual basis, it has exactly three options. Send all permutations to *da-net*, send all permutations to *rotor-net*, or send the smaller of the two types to *rotor-net* and the larger to *da-net*. Since the second set with the size of $\frac{(1-u)L}{m(n-1)} + \frac{uL}{n-1}$ is always larger GMN will be emulating $\mathcal{A}_{ver}$. Lastly, note that GMN will never send the smaller permutations to *da-net* and the larger to *rotor-net*. This is true for any matrix, as it's a results of the DCT of both networks being linear. □

The following is a simple observation of Claim 1.

OBSERVATION 1. *Let $P \in P_\pi$ be a permutation matrix.* mix-net *has an equal DCT to either* da-net *or* rotor-net *for P,*

$$DCT_{mix}(P, \text{GMN}) = \min(DCT_{rot}(P, \text{Val}), DCT_{da}(P, \text{BvN})) \quad (10)$$

Note that P is a special case in $M(n, m, u, L)$. The observation stems from the fact that GMN only schedules the entire matrix P on either *mix-net* or *rotor-net*.

Next, we prove Theorem 5.1 which holds for any saturated demand matrix.

Theorem C.1. *For any saturated demand matrix M,*

$$DCT_{mix}(M, \text{GMN}) \leq \min(DCT_{rot}(M, \text{Val}), DCT_{da}(M, \text{BvN})) \quad (11)$$

That is, the DCT of mix-net *will be equal or less than the DCT of either* rotor-net *or* da-net.

PROOF. To see why this is the case, let us denote the set of matchings sent on *da-net* as $M^{(da)} = \{M_i : DCT_{da}(M_i) \leq DCT_{rot}(M_i)\}$. And let $M^{(rot)} = \{M_i : DCT_{rot}(M_i) \leq DCT_{da}(M_i)\}$ be the set of matchings sent on *rotor-net*. Since for an individual permutation matrix *mix-net* has lower or equal DCT to either *da-net* or *rotor-net* as from Observation 1, and since both functions describing the DCT either system (in Equation (1) and Equation (2)) are linear in the total load of either $M^{(da)}$ or $M^{(rot)}$, the results follows. □