Distributed Self-Adjusting Tree Networks

Bruna Peres, *UFMG, Brazil*
Otavio A. de O. Souza, *UFMG, Brazil*
Olga Goussevskaia, *UFMG, Brazil*
Chen Avin, *BGU, Israel*
Stefan Schmid, *University of Vienna, Austria*

***Abstract*—We consider the problem of designing *dynamic* network topologies that *self-adjust* to the (possibly changing) traffic pattern they serve. Such demand-aware networks currently receive much attention, especially in the context of datacenters, due to emerging technologies supporting the fast reconfiguration of the physical topology. We present the first *fully distributed*, provably efficient self-adjusting network. Our network called *DiSplayNet* relies on algorithms that perform *decentralized* and *concurrent* topological adjustments to account for changes in the demand. We present a rigorous formal analysis of the correctness and performance of *DiSplayNet*, which can be seen as an interesting generalization of analyses known from sequential self-adjusting datastructures. We also report on results from extensive trace-driven simulations.**

I. Introduction

Traditionally, the topology of a communication network is *fixed* and *oblivious* to the traffic pattern it serves. For example, today's datacenter topologies, ranging from fat-trees over hypercubes to expander graphs [1], [2], are optimized toward static and demand-oblivious properties such as the degree, the diameter, or the mincut. But even the logical topology of peer-to-peer networks (the *overlays*) is usually optimized toward static objectives (usually degree and diameter), and dynamic changes are limited to handle joins and leaves, e.g., in order to reestablish the same static properties mentioned above.

This paper is motivated by the more orthodox and less explored question of how to design network topologies which dynamically *self-adjust* toward the demand. The question is timely: emerging technologies based on optical circuit switches, 60 GHz wireless, and free-space optics, allow to *reconfigure* the (physical) topology of communication networks at runtime [3], [4], [5], [6], [7]. For example, such reconfigurable networks allow to establish direct links between two frequently communicating pairs of racks in a datacenter, e.g., using digital micromirror devices.

Dynamically reconfigurable networks can be attractive: Some empirical studies show that for certain traffic patterns, a traffic-aware topology can achieve a performance similar to a demand-oblivious full-bisection bandwidth network at 25-40% lower cost [3], [6]. In general, the higher the given (spatial and temporal) locality of the communication pattern, the higher the possible gains of self-adjusting networks.

However, while the technologies enabling more flexible networked systems are maturing, today, we do not have a good understanding of how to actually *exploit* these flexibilities.

Putting Things Into Perspective. The vision of self-adjusting networks is similar in spirit to the vision of self-adjusting datastructures introduced by Sleator and Tarjan: In their seminal work [8], Sleator and Tarjan proposed *splay trees*, a new kind of Binary Search Tree (BST) which self-adjusts to its usage pattern, moving more frequently accessed elements closer to the root: this moving cost is likely to be compensated in the future due to reduced lookup times. In particular, Sleator and Tarjan proved upper bounds on the amortized cost of splay trees.

The main difference between datastructures and communication networks is that in the former, requests always originate from the *root* (i.e., the pointer to the BST root), whereas in the latter, requests occur between *node pairs* (e.g., top-of-rack switches in datacenters, or peers). A first proposal to generalize splay trees to networks, short *SplayNet*, has been presented in [9]. In *SplayNet*, communication happens between arbitrary node pairs in the network and nodes communicating more frequently perform local transformations and become topologically closer to each other over time. In particular, node pairs located in different subtrees move toward their least common ancestor: there is no need to move all the way to the network root in this case.

While *SplayNets* have been proven to be optimal for some specific traffic patterns and have some interesting additional features such as support for local routing, they are operated centrally and are inherently sequential.

To the best of our knowledge, the fundamental question of how to design *distributed*, i.e., decentralized and concurrent, dynamically self-adjusting network topologies, has not been explored in the literature so far. Surprisingly, we are also not aware of any *distributed* analysis of the performance of classic self-adjusting splay trees under concurrent reconfigurations (existing performance analyses of concurrent datastructures such as *CBTrees* [10] are sequential).

Our Contributions. This paper presents *DiSplayNet*, the first fully distributed (*decentralized* and *concurrent*) self-adjusting (splay tree) network which comes with formal performance guarantees. *DiSplayNet*s are of constant degree and rely on distributed algorithms which adapt the topology to the workload *automatically* and in an *online* manner (i.e., without knowledge of future demand).

This paper proposes two natural metrics to evaluate the performance of any distributed self-adjusting network: (1) The amortized work, which is similar to the performance measures used in the context of self-adjusting data structures. It measures the cost of routing on and adjusting the network. (2) The makespan, which measures the time it takes to serve a set of communication requests. Our main technical contri-

bution is a rigorous amortized analysis of DiSplayNet. We show the proposed algorithm is deadlock- and starvation-free and derive formal worst-case guarantees on both amortized work and makespan. To the extent of our knowledge, this is the first upper bound for the work needed to fulfill an arbitrary sequence of requests using self-adjusting networks in a concurrent setting, as existing upper bounds only apply to sequential/centralized settings [10], [9], [8].

We also report on simulation results (based on real datacenter workloads from the ProjecToR [3] project and from Facebook [11]) which complement our formal analysis. In particular, our results indicate that decentralization does not come at a price of additional reconfiguration work, and can significantly reduce the makespan and increase the communication throughput of a network. By comparing our results to a static optimal network, we also shed light on when *DiSplayNet* is able to leverage temporal locality. We find that even if the demand does not feature any temporal locality (but requests are chosen *i.i.d.*) our approach does not perform much worse than an optimal static network which has complete knowledge of the demand ahead of time; and when the demand features some temporal structure, *DiSplayNet* soon outperforms statically optimal networks.

Paper Organization. Section II presents the model. In Section III, we describe our algorithm and analyze it subsequently in Section IV. We report on our simulations in Section V. After reviewing related work (Section VI), we conclude in Section VII.

II. Model

Our objective is to design distributed algorithms for self-adjusting networks which come with provable performance guarantees. The network should connect a set $V = \{v_1, \ldots, v_n\}$ of n nodes (e.g., top-of-rack switches or peers). The input to the network design problem is a traffic demand, given as a sequence $\sigma = (\sigma_1, \sigma_2 \ldots \sigma_m)$ of m communication requests $\sigma_i = (s_i, d_i)$ occurring over time, with source s_i and destination d_i; m can be infinite. We use b_i to denote the time when a request $\sigma_i = (s_i, d_i) \in V \times V$ is generated, and e_i to denote the time in which it is completed. The times between successive arrivals between requests is assumed to be at least one. The sequence σ is revealed over time, in an online manner: the algorithm does not have any information about the future requests σ_j at time $t < b_j$. Moreover, the sequence σ can be arbitrary: in particular, in our formal analysis we consider a worst-case scenario, where σ is chosen *adversarially*, in order to maximize the cost of a given distributed algorithm. When serving these communication requests, the network can adjust, and we denote the sequence of network topologies over time by $G_1, G_2, \ldots$. However, we require that each G_i belongs to some "desirable" graph family $\mathcal{G}$. In particular, for scalability reasons, the networks should be of constant degree.

In this paper we focus on constant-degree tree networks only. Our motivation is that trees are a most basic graph family and we envision that the self-adjusting links constitute only a subset of the topology, a usual assumption in such networks [3]. More specifically, we are interested in tree networks that are *locally routable*, i.e., dynamic topological changes do not require the global recomputation of routes. As networks based on Binary Search Trees (BST) provide these properties [9], we will focus on them in the following, and denote the family of BST networks by $\mathcal{T}$.

In order to minimize reconfiguration costs and adjust the topology smoothly over time, the tree is reconfigured locally through local *rotations* that preserve the BST properties: in the spirit of the usual pointer-machine models [12], nodes change a constant number of links to their neighborhoods (at constant cost). Accordingly, we will denote the tree at time t computed by a given distributed algorithm (possibly accounting for the communication requests σ_t with $t' < t$) by $T_t \in \mathcal{T}$.

Cost model: We will refer to local reconfigurations as *steps* (a set of rotations). In particular, we will assume that each step, which involves a constant number of link changes, has a cost of $O(1)$ (more details will follow). Similarly, we assume that communication costs one unit per link.

As we will see, the algorithm presented in this paper is aggressive in how it moves communicating nodes together: the communication cost of our algorithm is always in the order of the reconfiguration cost. Hence, for our asymptotic analysis, it will be sufficient to consider reconfiguration costs only.

Time model: In order to study concurrency, we divide the execution time in *rounds*: in a round, multiple (independent) nodes can make local reconfigurations (steps) concurrently.

Our objective is to **minimize** the cost both in terms of *work* (number of reconfiguration steps and routing cost) and the cost in terms of *time* (time to process a given set of requests).

Definition 1. ***Work cost:*** *Consider any initial tree T_0 and a sequence of communication requests $\sigma = (\sigma_1, \ldots, \sigma_m)$. We define the total cost as the number of steps (local rotations) to fulfill all requests.*

In terms of time, we aim to minimize the makespan:

Definition 2. ***Time cost:*** *Consider any initial binary tree T_0 and a sequence of communication requests $\sigma = (\sigma_1, \ldots, \sigma_m)$.*
Makespan: $T(T_0, \sigma) = \max_{1 \le i \le m} e_i - \min_{1 \le i \le m} b_i$.

We are interested in the worst-case performance over arbitrary sequences of operations (rather than individual operations), and hence, conduct an *amortized analysis* [13]. In our model, the amortized cost can be described as the average cost per request for a given sequence σ of communication requests.

Definition 3. ***Amortized cost:*** *For a sequence of communication requests $\sigma = (\sigma_1, \ldots, \sigma_m)$, if $c(\sigma_i)$ is the (time or work) cost of the communication request $\sigma_i \in \sigma$, the amortized cost is defined with respect to the worst sequence σ and initial tree T_0,*
Amortized cost: $\mathcal{C}_A = \max_{\sigma, T_0} \frac{1}{m} \sum_{\sigma_i \in \sigma} c(\sigma_i)$.

III. Distributed SplayNets

The distributed algorithm to implement *DiSplayNet* presented in this section relies on the following key concepts:

1) *Local reconfigurations:* In order to adjust the network topology locally without violating local routing properties, we leverage the **zig**, **zig-zig**, and **zig-zag** operations known from splay trees (see Definition 4).
2) *Independent clustering:* In order to facilitate concurrent adjustments while avoiding deadlocks, we compute (in a distributed manner) local *clusters*: clusters are coordinated by a node requesting steps (i.e., a cluster *master*), and can be updated in parallel, without interference.
3) *Prioritization:* In order to avoid starvation, we prioritize requests according to their timestamp (b_i).

In the following, we will elaborate on each of these components in more detail.

A. Order Preserving Transformations

To perform local routing and order preserving local reconfigurations, our algorithm requires that each node u stores the identifiers of its direct neighbors in the BST tree, i.e., its parent ($u.p$), its left child ($u.l$), its right child ($u.r$), as well as the smallest ($u.smallest$) and the largest ($u.largest$) identifiers currently present in the sub-tree rooted at u.

Upon a request $\sigma_i = (u, v)$, the nodes u and v start moving towards each other, by performing local reconfigurations that preserve the search-tree property. A *DiSplayNet* implements such topological updates using the **zig**, **zig-zig**, and **zig-zag** operations, known from *splay trees* [8]. We refer to such a local reconfiguration as a *step*:

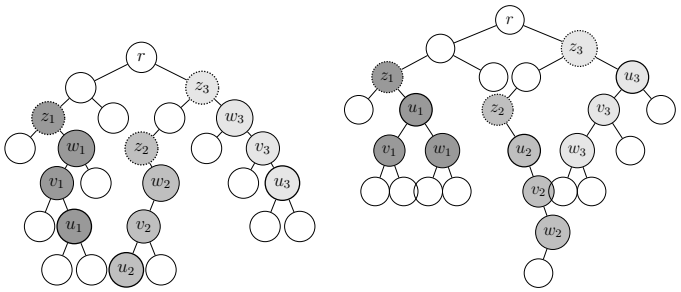
Definition 4. ***Step*** $step_t(u)$***:*** *Steps in* DiSplayNet *are performed through rotations that preserve the BST properties. The link updates in the network because of a step performed by a node u in round t depend on the relative positions of u, its parent v and its grandparent w. Note that a zig comprises a single rotation, while a zig-zig and zig-zag are composed of double rotations.*

Unlike in splay trees, in *DiSplayNet*, nodes are not splayed to the root. Rather, upon a request $\sigma_i = (u, v)$, nodes u and v are rotated only toward their lowest common ancestor:

Definition 5. ***Lowest Common Ancestor (LCA):*** *The lowest common ancestor of two nodes $(u, v) \in V$ at time t, is the closest node to u and v that has both u and v as descendants. A node can be the lowest common ancestor of itself and another node.*

Consider two nodes u and v, such that in round t, $u.p = v$. If in round $t + 1$, $u.p \neq v$, we say that v changed the link to u. When performing a step, we consider that the highest node sharing a link is responsible for that link, i.e., the parent node is in charge of a link to a child. Therefore, if a link from a node v to node u with $u.p = v$ must be updated because of a $step_t(x)$, v is responsible for informing u about the link change.

In order to deal with concurrency, i.e., facilitate (and maximize) simultaneous transformations while maintaining a consistent BST and avoiding deadlocks and starvation, we need nodes to come in consensus on which step to participate in. Towards this end, our distributed algorithm for *DiSplayNet* computes an independent set of *clusters*



(a) Round t: "Before" (b) Round $t + 1$: "After"

Fig. 1. Example of 3 concurrent steps (clusters $\mathcal{C}_t(u_1) \dots \mathcal{C}_t(u_3)$: u_i: *requesters*, z_i: *masters*)

Definition 6. ***Cluster*** $\mathcal{C}_t(u)$***:*** *Consider a step $step_t(u)$ performed by some node $u \in T_t$ in round t. Nodes that have a link to a child, which changes a a result of $step_t(u)$ form a cluster $\mathcal{C}_t(u)$. In each cluster, only a* single *node can perform one step: this ensures consistency of the local reconfigurations; the other nodes in the cluster are then* locked *i.e.,* paused *for this reconfiguration. Each cluster contains exactly one* **requester** *and exactly one* **master***, which together coordinate the step: For a given $step_t(u)$, u is the* requester *in $\mathcal{C}_t(u)$. The* master node *is the highest node in the tree participating in $step_t(u)$.*

Figure 1 presents an example of three concurrent clusters $\mathcal{C}_t(u_1)$ through $\mathcal{C}_t(u_3)$, where u_i is the requester and z_i is the master of each cluster, in consecutive rounds t and $t + 1$.

B. Reasoning About Progress

Before we proceed, we introduce a useful concept to describe and reason about (sequential and concurrent) tree adjustment algorithms and their executions: the *progress matrix* $\mathcal{M}$. The progress matrix $\mathcal{M}$ is a function of σ, an algorithm $\mathcal{A}$ and T_0, and it is fully determined by the choice of these three parameters. Each row in $\mathcal{M}$ represents a request $\sigma_i \in \sigma$, and each column represents a round t. Each element $M_{\sigma_i,t}$ in the matrix indicates if at round t the request σ_i makes progress (✓) or is paused (✗). In addition, before being generated or after being fulfilled, the request's status in the matrix is represented by the inactive sign (-). We say that a request is *active* from the moment it enters the system and until it was served, after which it becomes inactive. We consider that a request $\sigma_i(s_i, d_i)$ makes progress at time t if one step ($step_t(s_i)$ or $step_t(d_i)$) is performed in t. Otherwise, if σ_i is active and does not make progress at time t, we say that σ_i is *paused*. A request σ_i is prevented from making progress when another node in its neighborhood (or cluster) is making progress (as described in Sections III-A and III-C).

The progress matrix can also be used to represent executions of sequential algorithms, such as *SplayNet* [9]. To simplify the understanding, we start with this case accordingly. In a

nutshell, the (sequential) algorithm *SplayNet* splays s_i and d_i upwards upon request $\sigma_i = (s_i, d_i)$. First the source s_i is splayed until it becomes an ancestor of the destination, after which d_i is splayed until it becomes a child of s_i. Only after this has been achieved, the next request $\sigma_{i+1} = (s_{i+1}, d_{i+1})$ is processed. Table I presents an example of the progress matrix for such an algorithm. Once a request σ_i enters the network, it makes progress until it is fulfilled. By the sequential nature of the algorithm, nothing can cause a request that is making progress to pause. When it is completed, and only then, the next request is allowed to progress.

We can also see the work cost in the progress matrix $\mathcal{M}$: the check marks (✓) represent progress, and their total number corresponds to the total work. To measure the time cost per request, we can sum the number of columns in which the inactive sign (-) does not appear. The makespan (see Definition 2) is represented by the number of columns in the progress matrix, i.e., the total number of rounds for all the nodes to complete the requests. In prior work it has been shown that the amortized cost in terms of work for a retrieval tree is $O(\log n)$ per operation, in sequential [8] and distributed [14] scenarios.

However, the decentralized algorithm we present in the following allows for *concurrent steps*. That is, multiple communication pairs are active simultaneously and are performing steps in parallel.

C. Distributed Reconfiguration Algorithm

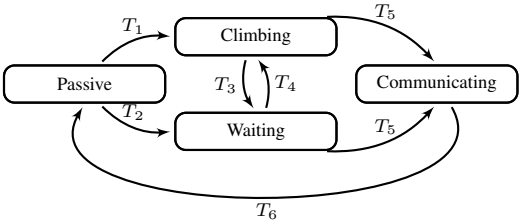
With these concepts in mind, we can now present our algorithms in detail. Essentially, each node in *DiSplayNet* executes:

while(*true*) { **execute** *clusterStep()* }

DiSplayNet can be best described in terms of a state machine, executed by each node in parallel. Each node can be in one of four states:

1) *Passive:* A node is in passive state at time t if it is not the source or destination of any request in $\sigma_i \in \sigma, b_i \leq t$
2) *Climbing:* A node s_i (or d_i) is climbing at time t if it has an *active request*: $\exists \sigma_i(s_i, d_i) \in \sigma, b_i \leq t$ and the distance $d_t(s_i, d_i) > 1$, and additionally s_i (or d_i) $\neq LCA_t(s_i, d_i)$;
3) *Waiting:* A node s_i (or d_i) is waiting at time t if it has an active request and $s_i = LCA_t(s_i, d_i)$.
4) *Communicating:* A node s_i or d_i is communicating at time t if $\exists \sigma_i(s_i, d_i) \in \sigma, b_i \leq t$ and $d_t(s_i, d_i) = 1$.

Figure 2 shows the possible state transitions. In order to ensure deadlock and starvation freedom, concurrent splaying steps are chosen according to a *priority* in *DiSplayNet*. Given two requests σ_i and σ_j, we say that σ_i has a higher priority than σ_j if $i < j$. An older request in the network has a higher priority than a more recent request . Note that, a node s in the *waiting* state might be removed from the LCA position by a splaying step with higher priority. If that happens, s returns to the *climbing* state and resumes requesting splaying steps. Finally, when s and d meet, they communicate.



T_1: Started a request and is not LCA (makes request *active*)
T_2: Started a request as LCA (makes request *active*)
T_3: Reached LCA
T_4: Pushed down from LCA position by a higher priority node
T_5: Source and destination meet
T_6: Fulfill the request (makes request *inactive*)

Fig. 2. State Transition Diagram for a node in *DiSplayNet*

To synchronize the process between the nodes, the distributed algorithm proceeds in *rounds*. Each round is composed of five phases, summarized in Algorithm 1: (1) Cluster Requests; (2) Top-down Acks; (3) Bottom-up Acks; (4) Link Updates; (5) State Updates.

Each node u maintains a local buffer, containing a queue of cluster requests, generated by itself, its right or its left child, one of its four grandchildren or one of its eight great-grandchildren. In each round, each request $\mathcal{C}_u$ is sent upwards until reaching its *master* (2 hops ancestor in case of a zig, and 3 hops ancestor in case of a zig-zig or zig-zag operation). Once all requests have been received, the highest priority request is acknowledged top-down, from master to requester. If its request is the highest priority request it has received in phase 1, upon receiving a top-down acknowledgment, the requester sends an acknowledgment upwards to the master. We say that neighboring nodes form a cluster $\mathcal{C}_u$ if all of them received one top-down and one bottom-up acknowledgment for request($\mathcal{C}_u$).

D. Concurrent Progress Matrix

To illustrate how progress is made in a concurrent scenario, let us look at the progress matrix of *DiSplayNet*, illustrated in Table II. In the concurrent setting (unlike in a sequential model), instead of a row representing a request σ_i, each row represents an individual source or destination node, since both nodes s_i and d_i work simultaneously. Moreover, since a node $v \in V$ can participate in several requests, e.g., $\sigma_i(v, d_i)$ and $\sigma_j(s_j, v)$, it might be assigned several rows, e.g. s_i and d_j in the progress matrix.

IV. Analysis

In this section we formally analyze the correctness and performance of *DiSplayNet*. Firstly we prove that the decentralized reconfiguration underlying *DiSplayNet* is deadlock-free. Subsequently, we present an amortized analysis of the work (reconfiguration cost) of *DiSplayNet* under worst-case request sequences. Finally, we derive an upper bound on the makespan, i.e., the time it takes to serve a batch of communication requests.

TABLE I

PROGRESS MATRIX $\mathcal{M}(\sigma, \text{SPLAYNET}, T_0)$. EACH COLUMN REPRESENTS ONE ROUND. A ROW REPRESENTS THE EXECUTION TIME-LINE OF A REQUEST $\sigma_i \in \sigma$. IN EACH ROUND, A REQUEST CAN MAKE PROGRESS (✓), BE PAUSED (✗) OR BE INACTIVE (-).

	1	2	3	4	5	6	7	8	...	$i-6$	$i-5$	$i-4$	$i-3$	$i-2$	$i-1$	i
σ_1	✓	✓	✓	✓	-	-	-	-	...	-	-	-	-	-	-	-
σ_2	-	✗	✗	✗	✓	✓	✓	-	...	-	-	-	-	-	-	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
σ_{m-1}	-	-	-	-	-	-	-	-	...	✓	✓	-	-	-	-	-
σ_m	-	-	-	-	-	-	-	-	...	✗	✗	✓	✓	✓	✓	-

TABLE II

CONCURRENT PROGRESS MATRIX $\mathcal{M}(\sigma, \textit{DiSplayNet}, T_0), \sigma = (\sigma_1(s_1, d_1), \sigma_2(s_2, d_2), \sigma_3(s_3, d_3))$.

	1	2	3	...	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$	$t+6$	...	t'	...	t''
s_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
d_1	✓	✓	✓	...	✓	✓	✓	✓	✓	✓	...	-	✓	...	-
s_2	-	✓	✓	...	✓	✓	✓	✓	-	-	...	-	-	...	-
d_2	-	✓	✓	...	✓	✓	✗	✓	-	-	...	-	-	...	-
s_3	-	-	✓	...	✗	✗	✗	✗	✓	✗	✗	...	✓	...	-
d_3	-	-	✓	...	✗	✗	✗	✗	✗	✗	✗	...	✓	...	-

In the following helper lemma, we argue that different clusters do not interfere and form "independent sets", which is useful to prove deadlock freedom and required for the amortized performance analysis.

Lemma 1. *Link updates are consistent and clusters disjoint.*

Proof. The proof follows from the fact that each node acknowledges at most one cluster $\mathcal{C}_t(u)$ request in round t, originated by the highest priority node u in its neighborhood (Algorithm 1, phases 2,3). Therefore, no node can belong to more than one cluster simultaneously. Moreover, all links in the network are updated simultaneously in each round, which maintains consistency (Algorithm 1, phase 4). □

In Theorem 1 we show that *DiSplayNets* is deadlock- and starvation-free.

Theorem 1. DiSplayNets *are deadlock- and starvation-free.*

Proof. Proof by induction. **Base Case:** Consider the first request $\sigma_1(s_1, d_1) \in \sigma$. Since σ_1 has the highest priority in σ and, by Lemma 1, clusters are disjoint and link updates are consistent, no other $\sigma_i(s_i, d_i) \in \sigma$ can prevent nodes s_1 or d_1 from making progress in every round, until $d(s_1, d_1) = 1$. Therefore, $\sigma_1(s_1, d_1)$ has no obstructions and will complete without any pauses. **Hypothesis:** All requests $\sigma_j \in \sigma \mid 1 \leq j \leq i-1$ have completed in round t. **Step:** Consider the request $\sigma_i(s_i, d_i)$. By the induction hypothesis, all requests with higher priority have completed in round t, so σ_i is the request with highest priority in the network. Thus, it has no more obstructions and will complete without any pauses. □

In order to compute the worst case cost over arbitrary sequences, we conduct an amortized analysis of the performance of *DiSplayNet*. We introduce a potential function to amortize actual costs. Consider a *DiSplayNet* instance T_t in round t. Let size $s_t(u), u \in T_t$ denote the number of nodes in the subtree of node u, including u. We define the **rank** of node u as $r_t(u) = \log_2(s_t(u))$ and the total rank $r(T_t)$ as the sum of the ranks of all nodes in T_t. Note that the maximum size and rank of a node is n and $\log_2 n$, respectively. The potential of a given *DiSplayNet* instance T_t in round t is then the sum of the ranks of all nodes in the tree: $\phi(T_t) = \sum_{i=1}^{n} r_t(i)$. In the potential method, the amortized cost $\hat{c}_t(u)$ of an operation $step_t(u)$ is the actual cost $c_t(u)$, plus the increase in potential $\delta_t(u)$ due to the operation $step_t(u)$, where $\delta_t(u) = \phi(T_t) - \phi(T_{t-1})$. This gives us: $\hat{c}_t(u) = c_t(u) + \phi(T_t) - \phi(T_{t-1})$.

To understand the amortized analysis, it is useful to revisit the sequential Progress Matrix (see Table I). From the sequential splay tree and *SplayNet* analysis it follows that fulfilling a request $\sigma_i \in \sigma$ consists of p_i steps, represented by a sequence of p_i of consecutive checks in a row in M_{σ_i}. Each check mark represents a node performing a step of cost $O(1)$. Thus, the actual cost to fulfill σ_i is $\sum_{t=1}^{p_i} O(1)$. To calculate the amortized cost of σ_i, we must calculate the total potential change (Δ), summing the individual changes per step (δ), which gives us $\delta_t(\sigma_i) = \sum_{t=1}^{p_i} \phi(T_t) - \phi(T_{t-1})$. This summation results in a telescoping series in which all terms cancel except the first and the last. Thus, the amortized cost of request σ_i can be represented by: $p_i + \phi(T_{e_i}) - \phi(T_{b_i})$. From this, we later derive a total amortized cost of $O(\log n)$ per request.

In the concurrent scenario the analysis is more challenging. We can only guarantee that the source and destination nodes from the highest priority request ($\sigma_1(s_1, d_1)$) have consecutive ✓ in the progress matrix. For all the other nodes s_i and d_i, the consecutive progress can be interrupted, resulting in several consecutive progress sequences. An interruption can cause the potential to change drastically, i.e., for each consecutive progress sequence we can have, in the worst case, a change in potential of $\log n$. The following lemma allows us to compute potentials based on columns.

Lemma 2. *Given a* DiSplayNet $\mathcal{T}$ *and the resp. progress matrix* $\mathcal{M}$*, in any column of* $\mathcal{M}$*, corresponding to a round* t*, all nodes making progress at* t *belong to separate clusters.*

Proof. Each cluster is a set of nodes participating in a single step (Lemma 1). For each node u making progress in

Algorithm 1 *ClusterStep()* (one round)

1: Cluster Requests (3 time-slots)

if Climbing for some $\sigma_i(s,d)$ **then**
send $request(\mathcal{C}_u)$ upward;
insert $request(\mathcal{C}_u)$ into buffer;

upon receiving $request(\mathcal{C}_w)$**:**
insert $request(\mathcal{C}_w)$ into buffer;
forward $request(\mathcal{C}_w)$ upward;

2: Top-down Acks (3 time-slots)

get highest priority $request(\mathcal{C}_x)$ in buffer;
if Master($request(\mathcal{C}_x)$) **then**
send Ack($request(\mathcal{C}_x)$) downward;

upon receiving top-down ack $request(\mathcal{C}_w)$**:**
if $w = x$ **then**
forward Ack($request(\mathcal{C}_w)$) toward requester;

3: Bottom-up Acks (3 time-slots)

upon receiving top-down ack($request(\mathcal{C}_u)$**):**
if Requester($request(\mathcal{C}_u)$) and $u = x$ **then**
send Ack($request(\mathcal{C}_u)$) up toward master;
create $\mathcal{C}_u$;
join $\mathcal{C}_u$;
else
forward Ack($request(\mathcal{C}_u)$) toward master;
join $\mathcal{C}_u$;

4: Link Updates (3 time-slot)

if in($\mathcal{C}_u$) **then**
update links according to $\mathcal{C}_u$;

5: State Updates (1 time-slot)

update state; ▷ Figure 2
clear buffer;
leave cluster;

round t, u is either climbing or waiting. Thus, for each node u making progress at t (or column t in $\mathcal{M}$), either there is a cluster $\mathcal{C}_t(u)$ of nodes participating in $step_t(u)$, in which u is the requester; or there is a cluster of size 1 ($\mathcal{C}_t(u) = \{u\}$) of a node that is waiting. No node in $\mathcal{C}_t(u)$ but u can make progress, since each cluster has only one requester, and only the requester node makes progress. □

At the heart of our amortized analysis lies the following observation: the total potential change in one round which consists of multiple steps, is simply the sum of the potential changes of the individual clusters.

Lemma 3. *Consider a* DiSplayNet *instance T_t and let $\mathcal{C}_t$ be the set of clusters in round t. The total potential change in round t is $\delta_t = \sum_{\forall \mathcal{C}_t(j) \in \mathcal{C}_t} \delta(\mathcal{C}_t(j))$.*

Proof. The potential of T_t is the sum of the ranks of all nodes in $u \in T_t$. A $step_t(u)$ can only change the rank of nodes in cluster $\mathcal{C}_t(u)$. By Lemma 1, clusters are disjoint, i.e., a node cannot be in more than one cluster at a time. Thus, only one cluster can change the rank of a node per round. Therefore, $\phi(T_{t+1}) - \phi(T_t) = \delta_t = \sum_{\forall \mathcal{C}_t(j) \in \mathcal{C}_t} \delta_t(\mathcal{C}_t(j))$. □

Lemma 4. *[8] Consider a* DiSplayNet *instance T_t and let δ_t be the total potential change in round t, caused by a single $step_t(u)$. We have that:*

- $\delta_t(u) \leq 3(r_t(u) - r_{t-1}(u)) - 2$*, if the step is a zig-zig or zig-zag;*
- $\delta_t(u) \leq 3(r_t(u) - r_{t-1}(u))$*, if the step is a zig.*

Thereby, since we can represent the potential change for each step in terms of the rank change of the requester node, and combining Lemmas 2 and 3, we obtain the amortized cost to perform all steps in round t:

$$\hat{c}(\mathcal{C}_t) = \sum_{\forall \mathcal{C}_t(j) \in \mathcal{C}_t} c(step_t(j)) + \sum_{\forall \mathcal{C}_t(j) \in \mathcal{C}_t} \delta(\mathcal{C}_t(j))$$

where $c(step_t(j))$ is the actual cost to perform $step_t(j)$.

Definition 7. ***Bypass****: Consider a sequence of communication requests $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ and a pair of active requests $\sigma_i = \{s_i, d_i\} \in \sigma$ and $\sigma_j = \{s_j, d_j\} \in \sigma$, such that σ_i has higher priority, i.e., $i < j$. We say that a node $n_i \in \sigma_i$ bypasses a node $n_j \in \sigma_j$ if, in some round t, n_i is a descendant of n_j and, in round $t+1$, n_i becomes an ancestor of n_j.*

A bypass can only happen if distance $d_t(n_i, n_j) \leq 2$ and n_i performs a $step_t(n_i)$ and n_j participates in cluster $\mathcal{C}_t(n_i)$, of which n_i is the *requester* (Definition 6). Note that, when a node is bypassed, its subtree can decrease in size. Since the potential of a subtree is a function of its size, and the only operation that can decrease the size of the subtree of a node with an active request is a bypass, we have the following observation: *a node can only lose potential due to a concurrent higher-priority request as a result of a bypass.*

Lemma 5. *Given a sequence of communication requests $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, a source or destination node of a request $\sigma_i \in \sigma$ with priority i can be bypassed by at most $2(i-1) = O(m)$ concurrent requests.*

Proof. Consider one pair of nodes $n_i \in \sigma_i = (s_i, d_i)$ and $n_j \in \sigma_j = (s_j, d_j)$, where $i < j$. The first observation is that n_i can bypass n_j at most once. Consider, by contradiction, that n_i bypasses n_j for the second time in some round t. We know that n_i has previously bypassed n_j in some round $t' < t$. By Definition 7, in round t' node n_i was a descendant of n_j and in round $t'+1$ it became its ancestor. Therefore, in order to bypass n_j for the second time in round t, node n_i must have been bypassed by node n_j in the time interval $[t'+1, t-1]$, However, this is not possible by Algorithm 1, since the priority of n_j is lower than that of n_i. (Note that node n_j can only become an ancestor of n_i as a result of a *step(n_j)*. In case n_j is carried upwards by some other node n_h, as a result of a *step(n_h)*, n_i would not be part of the subtree of n_j as a result.)

Since a node of priority can only by bypassed by the source or destination nodes of requests with higher priority, a node that belongs to the lowest-priority request σ_m can suffer the most bypasses in σ, which is at most $2(m-1)$. □

Lemma 6. *Consider a* DiSplayNet T_0 *on* n *nodes and a sequence of communication requests* $\sigma = (\sigma_1, \ldots, \sigma_m)$*. The amortized work cost of any* $\sigma_i \in \sigma$ *is* $\mathcal{C}_A = O(m \log n)$.

Proof. Consider the (concurrent) progress matrix $\mathcal{M}$ for a given *DiSplayNet* $\mathcal{T}$. For each row in $\mathcal{M}$, there are sequences of consecutive rounds in which some node makes progress. By Lemma 5, a node can be bypassed at most $2(m-1)$ times, i.e., there can be at most $2(m-1)$ pauses in each row of $\mathcal{M}$ that causes the node to drop potential. Thus, for each source or destination node, there are at most $2m$ sequences of rounds in which it makes progress and rises potential. Consider that, for each row u of $\mathcal{M}$, each progress interval i starts in round i_s, ends in round i_f and has length p_i (rounds). Then, the total potential change to perform all steps requested by node u is upper bounded by:

$$\begin{aligned}
\Delta(u) &\leq \sum_{i=1}^{2m} \sum_{t=i_s}^{i_f} \delta_t(u) \qquad (1)\\
&\leq \sum_{i=1}^{2m} \left(\sum_{t=i_s}^{i_f} (3(r_t(u) - r_{t-1}(u)) - 2) + 1 \right)\\
&\leq \sum_{i=1}^{2m} ((3(r_{i_f}(u) - r_{i_s}(u)) - 2p_i) + 1)\\
&\leq 6m(\log n) + 2m - \sum_{i=1}^{2m} 2p_i
\end{aligned}$$

Observe that each zig-zig and zig-zag has a cost 2 and a zig has a cost 1. Splaying a node u consists of at most $2m$ sequences of p_i zig-zig or zig-zag steps, plus one zig at the end of each interval. So the actual cost is upper bounded by $\sum_{i=1}^{2m} 2p_i + 2m$, and the amortized cost to complete any request $\sigma_i \in \sigma$ is $O(m \log n)$. □

Theorem 2. *Consider a* DiSplayNet T_0 *on* n *nodes and a sequence of communication requests* $\sigma = (\sigma_1, \ldots, \sigma_m)$*. The total work cost to fulfill* σ *is* $O(m(m+n)\log n)$.

Proof. By Lemma 6, the amortized cost to fulfill each request $\sigma_i \in \sigma$ is $O(m \log n)$. Since the net potential drop over σ is at most $nm \log n$, the result follows. □

Theorem 3. *Consider a* DiSplayNet T_0 *on* n *nodes and a sequence of communication requests* $\sigma = (\sigma_1, \ldots, \sigma_m)$*. The makespan of* σ *is* $O(m(m+n)\log n)$.

Proof. The time cost of a request $\sigma_i \in \sigma$ is equal to the number of rounds in which it performs steps, or makes progress, plus the number of rounds in which it is paused. As illustrated in the progress matrix $\mathcal{M}$ (Table II), each paused round of a request σ_i's must overlap in time with a step (work) performed by a higher-priority request in σ. Therefore, the makespan is upper bounded by the total number of non-paused rounds in $\mathcal{M}$, i.e., the maximum total number of steps (work) of all m requests, given in Theorem 2. □

V. Simulations

To complement our formal worst-case analysis and to shed light on the performance of *DiSplayNet* under more realistic workloads, both in terms of work cost and time (makespan and throughput) we conducted simulations. In this section, we report on our main insights.

A. Setup and Baselines

To generate request workloads, we leverage two datasets, collected and published by the ProjecToR project [15] and Facebook [11], henceforth denoted by DS1 resp. DS2:

Dataset DS1 (*i.i.d.* over ProjecToR): This dataset describes a probability density function over $8,367$ communication pairs in a network consisting of $n = 128$ nodes (top of racks), randomly selected from 2 production clusters, running a mix of workloads, including MapReduce-type jobs, index builders, and database and storage systems. We sampled $m = 10,000$ requests independent and identically distributed (*i.i.d.*) in time from the provided traffic matrix and repeated each experiment 100 times. The original IDs of the nodes were randomized before each simulation.

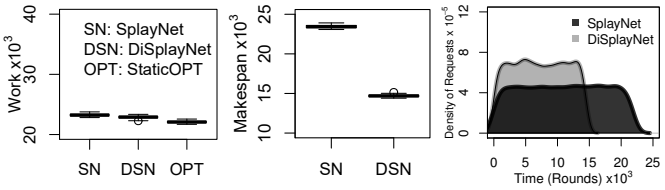
Dataset DS2 (Facebook): This dataset consists of *Fbflow*[1] raw samples from three production clusters at Facebook. The per-packet sampling is uniformly distributed with rate 1:30k; flow samples are aggregated every minute; and node IPs are anonymized. We focused on cluster A only and processed the data as follows. Firstly, we removed all inter-cluster or intra-rack requests, keeping only inter-rack requests within the same cluster. Then, we globally sorted the requests by timestamp. Finally, we mapped the anonymized IPs to a consecutive value range starting at 0. This resulted in a sequence of $m = 48,485,220$ requests, originated in a 24-hour time window, in a network comprised of $n = 159$ nodes.

Our simulations are event-driven and based on the Sinalgo [16] network simulator. In order to generate a sequence over time, we assumed a Poisson distribution for the request arrival, with $\lambda = 0.05$.

Locality of reference (DS1 x DS2): DS1 presents significantly higher spatial locality than DS2, which is possibly due to the limited sampling rate of *Fbflow*. In DS1, some source-destination pairs are responsible for 20% of all communication, whereas in DS2, no node pairs account for more than 0.15% of overall traffic. Even though high spatial locality is present in DS1, there is no temporal locality, given that the requests are *i.i.d.* over time. In DS2, on the other hand, the temporal locality is higher, since the request sequence was generated according to the provided timestamps.

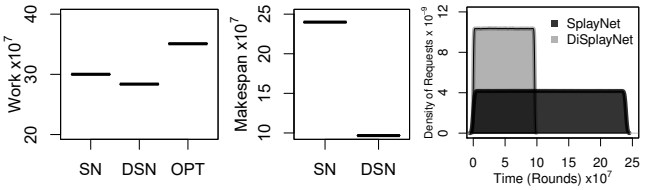
Baselines: To better understand and compare the simulation results, we implemented two baseline algorithms. Specifically,

[1]Fbflow is a network monitoring system that samples packet headers from Facebook's machine fleet.



(a) Work (b) Makespan (c) Throughput PDF

Fig. 3. DS1 (*i.i.d.* over ProjecToR): high spatial, low temporal locality



(a) Work (b) Makespan (c) Throughput PDF

Fig. 4. DS2 (Facebook): low spatial, moderate temporal locality

to study the benefit of dynamic reconfiguration, we implemented a "statically optimum" algorithm, using the dynamic program from [9]: a static binary search tree which is demand-aware and optimized towards the request frequency distribution of a given communication sequence. This baseline has the advantage that it knows the distribution *ahead of time* and does not incur any reconfiguration costs, but only communication costs (one unit cost per link). To investigate the benefit and limitation of concurrency, we implemented a *sequential* baseline, based on the algorithm from [9], henceforth referred referred to as *SplayNet*.

B. Work: A Price of Decentralization?

***DiSplayNet* x *SplayNet*:** The decentralized nature of *DiSplayNet* is likely to introduce an overhead compared to a central and sequential, and hence optimized, approach to reconfigure networks. This is also suggested by our formal worst-case bounds. To verify whether our formal bounds are too pessimistic and to measure the work overhead empirically, we ran several experiments using different request workloads. Interestingly, our simulation results suggest that the overhead in terms of work is negligible compared to a centralized algorithm. Figures 3a and 4a plot the total work, measured in number of steps performed by *DiSplayNet* and the two baselines, for datasets DS1 and DS2, respectively. The results show that there is indeed little difference in the work between our concurrent scheme and the sequential one. *DiSplayNet* performs close to *SplayNet* in practice, suggesting that our worst-case upper bound may be improved.

Dynamic reconfiguration x static optimum: Analyzing the total work performed by an optimum static network (which knows σ a priori), we can see that, for dataset DS1, it is slightly lower than that performed by *SplayNet* and *DiSplayNet*. Since DS1 has high spatial but no temporal locality, the static optimum computes the best topology for the given request frequency distribution and, since requests are distributed i.i.d. in time, dynamic reconfiguration cannot improve on that. For dataset DS2, however, we can see that static optimal performs more work than *SplayNet* and *DiSplayNet*, which shows that dynamic network reconfiguration is able to optimize the network topology dynamically over time, exploiting the temporal locality in the request sequence. Finally, note that the amortized work of sequential trees is asymptotically optimal and cannot be improved, i.e., in the order of the static optimum [8].

C. Makespan and Throughput

Concurrent adjustments turn out to greatly improve the amount of communication requests which can be handled by the network. In Figures 3b and 4b, we compare the makespan of *SplayNet* and *DiSplayNet*, for datasets DS1 and DS2, respectively, i.e., the time it takes to serve a batch of communication requests. In Figures 3c and 4c, we compare the throughput of the two schemes, measured as the number of completed requests per round during the entire simulation, as a PDF (Probability Distribution Function). Note that there is no static optimum baseline in these plots, since the measures of makespan or throughput do not apply to a static network topology without a specification of a communication model.

It can be seen that, compared to the sequential execution of *SplayNet*, *DiSplayNet* significantly improves both the makespan and the throughput, for both datasets. In the sequential execution, the makespan is roughly the same as the total work cost. In the distributed setting, on the other hand, the makespan is approximately a factor of 1.5 shorter for DS1 and a factor of 3.0 for DS2. The gain in time cost is greater in DS2 than in DS1, which can be explained by the low spatial locality of DS2. Since requests are spatially more uniformly spread in DS2, possibly due to the limited sampling rate of *Fbflow*, there arise more opportunities for concurrency in the network. In DS1, on the other hand, where some source-destination pairs concentrate as much as 20% of all data traffic, the local queues at some nodes can get long, making request completion sequential. Potentially, if we could increase the sampling granularity of DS2, increasing its spatial locality, the gains of dynamic and distributed network reconfiguration could be even higher than those seen in our experiments.

VI. Related Work

Reconfigurable networks have been explored both in the context of datacenters, e.g., [3], [4], [5], [6], in wide-area networks [17], [18], [19], and, more traditionally, in the context of overlays [20], [21]. See [22] for a recent algorithmic taxonomy of the field. Many existing network design algorithms rely on estimates or snapshots of the traffic demands, from which an optimized network topology is (re)computed periodically [23], [24], [25], [26], [27]. However, they do not account for the actual reconfiguration costs. In contrast, we in this paper present a more refined model, accounting also for the reconfiguration costs, and allowing us to study (within our model) the tradeoff between the benefits and costs

of reconfigurations. Other interesting solutions are *dynamic skip graphs* [28] which minimize the average routing costs between arbitrary communication pairs by performing topological adaptation to the communication pattern, and Flattening [14] which optimizes the communication cost of point-to-point requests over a k-ary tree, by performing local tree transformations according to the request pattern. However, these solutions do not come with any concurrency support or analysis.The paper closest to ours is *SplayNet* [9]. However, *SplayNet* is based on centralized algorithms (e.g., rely on a global controller or scheduler), and is purely sequential. In contrast, we in this paper present the first distributed, i.e., decentralized and concurrent implementation of *SplayNet*. This is a non-trivial extension, both in terms of the result and the required techniques (e.g., ensuring liveness is straightforward in a centralized architecture). The distributed setting fundamentally changes basic notions such as the working set (in a distributed setting, keeping working set nodes close to the root is insufficient) and makes it impossible to amortize costs by employing the usual telescopic sum approach [8], [9]. Finally, we would like to point out that an early version of this work appeared as a brief announcement at DISC 2017 [29].

VII. Conclusion

We understand our work as a first step, and believe that it opens interesting directions for future research. In particular, on the theory side, it will be interesting to study lower bounds for our algorithm and the problem in general, and investigate the optimality of the performance bounds derived in this paper. On the more applied side, it will be interesting to study the integration and use of self-adjusting links of the topology with links that are not self-adjusting: for example, in the context of datacenters, self-adjusting networks are usually used *in addition to* fixed infrastructures [3].

Acknowledgments. Research supported in part by CAPES, CNPq and FAPEMIG, as well as by the German-Israeli Foundation for Scientific Research and Development (G.I.F. No I-1245-407.6/2014).

References

[1] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and trade-offs," *IEEE Communications Surveys & Tutorials*, 2017.

[2] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla, "Beyond fat-trees without antennae, mirrors, and disco-balls," in *Proc. ACM SIGCOMM*, 2017, pp. 281–294.

[3] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect," in *Proceedings of the 2016 ACM SIGCOMM*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 216–229.

[4] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter, "Circuit switching under the radar with reactor," in *NSDI*, vol. 14, 2014, pp. 1–15.

[5] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 339–350, 2010.

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

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

[8] D. Sleator and R. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.

[9] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker, "Splaynet: Towards locally self-adjusting networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1421–1433, Jun. 2016.

[10] Y. Afek, H. Kaplan, B. Korenfeld, A. Morrison, and R. E. Tarjan, "Cb-tree: A practical concurrent self-adjusting search tree," in *Proceedings of the 26th International Conference on Distributed Computing*, ser. DISC'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 1–15.

[11] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM SIGCOMM*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 123–137.

[12] P. Bose, K. Douïeb, and S. Langerman, "Dynamic optimality for skip lists and b-trees," in *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008, pp. 1106–1114.

[13] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[14] M. K. Reiter, A. Samar, and C. Wang, "Self-optimizing distributed trees," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008, pp. 1–12.

[15] "Projector dataset," www.microsoft.com/en-us/research/project/projector-agile-reconfigurable-data-center-interconnect, 2016.

[16] D. C. Group, "Sinalgo - simulator for network algorithms," http://disco.ethz.ch/projects/sinalgo/index.html, 2007, accessed 10-May-2017.

[17] S. Jia, X. Jin, G. Ghasemiesfeh, J. Ding, and J. Gao, "Competitive analysis for online scheduling in software-defined optical wan," in *Proc. IEEE INFOCOM*, 2017.

[18] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing bulk transfers with software-defined optical wan," in *Proc. ACM SIGCOMM*, 2016, pp. 87–100.

[19] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill, "Radwan: rate adaptive wide area network," in *Proc. ACM SIGCOMM*. ACM, 2018, pp. 547–560.

[20] C. Scheideler and S. Schmid, *Automata, Languages and Programming: 36th Internatilonal Collogquium, ICALP 2009, Rhodes, greece, July 5-12, 2009, Proceedings, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ch. A Distributed and Oblivious Heap, pp. 571–582.

[21] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. IEEE INFOCOM*, vol. 3, 2002, pp. 1190–1199.

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

[23] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: a topology malleable data center network," in *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2010.

[24] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network design with minimal congestion and route lengths," in *Proc. IEE INFOCOM*, 2019.

[25] ——, "Demand-aware network designs of bounded degree," in *Proc. International Symposium on Distributed Computing (DISC)*, 2017.

[26] C. Avin, A. Hercules, A. Loukas, and S. Schmid, "rdan: Toward robust demand-aware network designs," in *Information Processing Letters (IPL)*, 2018.

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

[28] S. Huq and S. Ghosh, "Locally self-adjusting skip graphs," arXiv:1704.00830, 2017.

[29] B. Peres, O. Goussevskaia, S. Schmid, and C. Avin, "Brief announcement: Concurrent self-adjusting distributed tree networks," in *Proc. International Symposium on Distributed Computing (DISC)*, 2017.