Distributed Self-Adjusting Tree Networks

Bruna S. Peres, Otávio A. O. Souza, Olga Goussevskaia, Chen Avin, and Stefan Schmid

Abstract—The performance of many data-centric cloud applications critically depends on the performance of the underlying datacenter network. Reconfigurable optical technologies have recently introduced a novel opportunity to improve datacenter network performance, by allowing to dynamically adjust the network topology according to the demand. However, the vision of self-adjusting networks raises the fundamental question how such networks can be efficiently operated in a scalable and distributed manner.

This paper presents $DiSplayNet$, the first fully distributed self-adjusting network. $DiSplayNet$ relies on algorithms that perform decentralized and concurrent topological adjustments to account for changes in the demand. We propose two natural metrics to evaluate the performance of distributed self-adjusting networks, the *amortized work* (the cost of routing on and adjusting the network) and the *makespan* (the time it takes to serve a set of communication requests). We present a rigorous formal analysis of the work and makespan of $DiSplayNet$, which can be seen as an interesting generalization of analyses known from sequential self-adjusting datastructures. We complement our theoretical contribution with an extensive trace-driven simulation study, shedding light on the opportunities and limitations of leveraging spatial and temporal locality and concurrency in self-adjusting networks.

Index Terms—Self-adjusting networks, decentralization, concurrency, datacenters, amortized analysis, trace-driven simulations

1 Introduction

The performance of many cloud-based applications critically depends on the underlying network, requiring datacenter networks to provide low latency and high bandwidth. For instance, in distributed machine learning applications that periodically require large data transfers, the network is increasingly becoming a bottleneck [1]. Similarly, stringent performance requirements are introduced by today's trend of resource disaggregation in datacenters where fast access to remote resources (e.g., GPUs or memory) is pivotal for the overall system performance [1]. Building systems with strict performance requirements is especially challenging under bursty traffic patterns as they are commonly observed in datacenter networks [2], [3], [4], [5], [6].

- *B.S. Peres was and O.A.O.Souza and O. Goussevskaia are with the Computer Science Department, Universidade Federal de Minas Gerais, 31270-901 Belo Horizonte, Brazil. E-mails: bperes@dcc.ufmg.br, oaugusto@dcc.ufmg.br, olga@dcc.ufmg.br*
- *Chen Avin is with the School of Electrical and Computer Engineering, Ben Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel. E-mail: avin@bgu.ac.il*
- *Stefan Schmid is with the Faculty of Computer Science, University of Vienna, Währinger Straße 29, 1090 Vienna, Austria. E-mail: stefan_schmid@univie.ac.at*

In order to deal with the explosively growing traffic in datacenters, researchers have developed several innovative datacenter topologies over the last years. While traditionally datacenter networks rely on (multi-rooted) fat-tree topologies [7], [8], [9], more recent proposals revolve around hypercubic topologies [10], [11] or topologies based on expanders [12], [13], among others. All these networks have in common that their topology is *fixed* and *oblivious* to the traffic pattern it serves. In particular, these topologies are optimized toward static and demand-oblivious properties such as the degree, the diameter, or the mincut.

This paper is motivated by an intriguing alternative vision of how datacenter networks should be operated: namely in a demand-aware manner, dynamically adapting the topology towards the workload it serves. Such self-adjusting networks are empirically motivated: many measurement studies show that datacenter traffic features much spatial and temporal structure [2], [14], [15], which can be exploited. For instance, it could be attractive to connect two frequently communicating racks *directly*, rather than routing their traffic in a multi-hop fashion [16]. Self-adjusting networks have received significant attention over the last years and are enabled by emerging reconfigurable optical technologies [17], e.g., based on optical circuit switches, 60 GHz wireless, and free-space optics [18], [19], [20], [21], [22]. All these technologies allow to reconfigure the physical communication topology at runtime. It has been shown that dynamically reconfigurable networks can be attractive and, e.g., achieve a performance similar to a demand-oblivious full-bisection bandwidth network at $25-40\%$ lower cost [18], [21]. 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 datacenter networks are maturing, today, we still do not have a good understanding of how to actually *exploit* these flexibilities. Indeed, self-adjusting networks still lack theoretical foundations, from performance metrics to optimization. In particular, the notion of self-adjusting networks raises the basic question how such networks can be operated in a scalable and hence distributed manner. This paper aims to make a first step to address these questions, by considering a most basic network topology, a self-adjusting tree.

Putting Things Into Perspective. The approach proposed in this paper is motivated by the observation that 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 [23], 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 [24]. 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. In order for such sequences of local topology changes to be executed in real-time using any of the emerging reconfiguration hardware, such as optical switches [17], each network node should be able to decide and physically change its communication links in a distributed and concurrent manner.

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* [25] 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 relies on distributed algorithms which adapt the topology to the workload automatically, 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 contribution is an amortized analysis of DiSplayNet. We show that the proposed algorithm is deadlock- and starvation-free, and we derive formal worst-case guarantees on both amortized work and makespan. To the best of our knowledge, this is the first upper bound on the work needed to serve an arbitrary sequence of requests using self-adjusting networks in a concurrent setting, as existing bounds apply to sequential/centralized settings [23], [24], [25].

We also report on simulation results (based on synthetic traces and real datacenter workloads, including from Microsoft [18] and Facebook [26]) which complement our formal analysis. Our results indicate that decentralization does not come at a price of additional reconfiguration work, and can significantly increase the throughput of a network. By comparing our results to an optimal static 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. The remainder of this paper is organized as follows. Section 2 presents the model. In Section 3, we describe our algorithm and analyze it subsequently in Section 4. We report on our simulations in Section 5. After reviewing related work (Section 6), we conclude in Section 7.

2 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 requests arrivals are 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 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 *tree* networks only. Our motivation is that trees are the most basic graph family and we envision that the self-adjusting links constitute only a subset of the topology, the usual assumption in such networks [18]. 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 [24], we will focus on them 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 [27], nodes update a constant number of links to their neighborhoods

(at constant cost). We denote the tree at time t computed by a given distributed algorithm (possibly accounting for the communication requests $\sigma_{t'}$ 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: $T(T_0, \sigma) = \max\limits_{1 \le i \le m} e_i - \min\limits_{1 \le i \le m} b_i$.

We are interested in the worst-case performance over arbitrary sequences of operations (rather than individual operations), and hence, conduct an *amortized analysis* [28]. 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 a request $\sigma_i \in \sigma$, the amortized cost is defined with respect to the worst sequence σ and initial tree T_0,*
Amortized cost: $\mathcal{C}_A = \max\limits_{\sigma, T_0} \frac{1}{m} \sum\limits_{\sigma_i \in \sigma} c(\sigma_i)$.

Note that we do not explicitly define the routing or the control communication costs because, in our model, data is sent only when the distance to the destination is one ($d_t(s, d) = 1$), so the routing and control costs are dominated by the work and time costs of the meeting requests.

3 Distributed Reconfiguration

At the heart of DiSplayNet lies a distributed topology reconfiguration algorithm. It is based on the following 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).

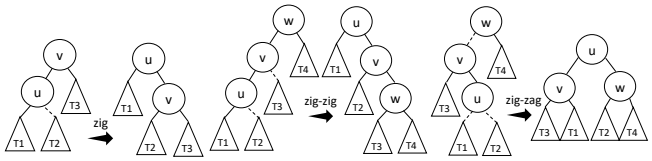
In the following, we elaborate on each of these components in more detail.

3.1 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. DiSplayNet implements such topological updates using the **zig**, **zig-zig**, and **zig-zag** operations, known from *splay trees* [23]. 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. The three types of steps: (1) **zig**; (2) **zig-zig**; and (3) **zig-zag** are shown in Figure 1. Note that a zig consists of a single rotation, while a zig-zig and zig-zag consist of double rotations.*



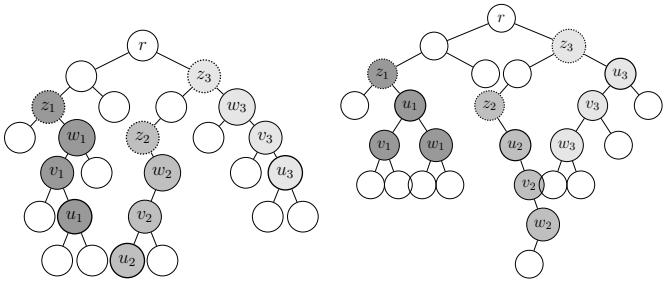
(a) zig (b) zig-zig (c) zig-zag

Fig. 1. Splaying step types (dashed lines represent the updated links).

As pointed out in [23], a step not only moves a node upwards in the tree, preserving BST properties, but also roughly halves the depth of every node along the access path. This halving effect makes splaying efficient in an amortized sense and is a property not shared by other, simpler rotation heuristics, such as single rotations [29] or move-to-root [30]. 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



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

Fig. 2. Ex: 3 concurrent steps (clusters $\mathcal{C}_t(u_1)\ldots\mathcal{C}_t(u_3)$: u_i: *requesters*, z_i: *masters*)

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*.

Definition 6. ***Cluster*** $\mathcal{C}_t(u)$***:*** *Consider the set of links that are modified as a result of a* $step_t(u)$*, performed by some node* $u \in T_t$ *in round* t*. The parent node of each such link belongs to a set of nodes that is referred to as the cluster* $\mathcal{C}_t(u)$*.*[1] *In each cluster* $\mathcal{C}_t(u)$*,* u *is the only node that can perform a step: this ensures consistency of the local reconfigurations; the other nodes in the cluster are then* locked *(*paused*) in round* t*. Each cluster contains exactly one* **requester** *and exactly one* **master***, which together coordinate the step. For a given* $\mathcal{C}_t(u)$*,* u *is the* requester*. The* master node $m \in \mathcal{C}_t(u)$ *is the highest node in the tree* $m \in T_t$ *participating in* $step_t(u)$*.*

Figure 2 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 $t+1$.

3.2 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, $\mathcal{A}$ and T_0, i.e., it is fully determined by 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 3.1 and 3.3).

The progress matrix can also be used to represent executions of sequential algorithms, such as $SplayNet$ [24]. 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 1 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 [23] and distributed [31] 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.

3.3 Algorithm

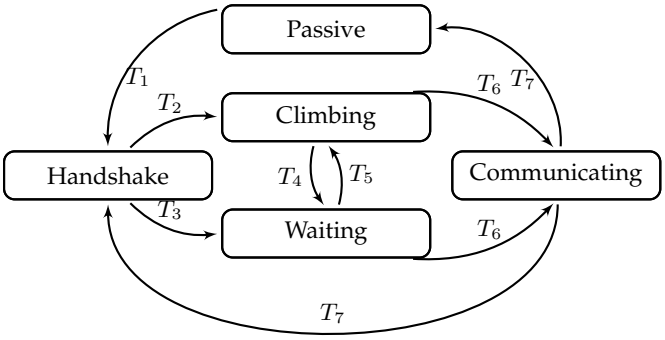
With these concepts in mind, we can now present our algorithms in detail. DiSplayNet can be best described in terms of a state machine, executed by each node in parallel. Figure 3 shows the possible state transitions. Each node can be in one of five states:

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) *Handshake:* A node s_i (or d_i) is in handshake state at time t if it has an *active request*, i.e., $\exists\sigma_i(s_i, d_i) \in \sigma, b_i \leq t$, but an agreement between s_i and d_i on starting working on σ_i has not been reached;
3) *Climbing:* A node s_i (or d_i) is climbing at time t if there is an active request $\sigma_i(s_i, d_i) \in \sigma, b_i \leq t$ for which a handshake has been completed; additionally s_i (or d_i) $\neq LCA_t(s_i, d_i)$;
4) *Waiting:* A node s_i (or d_i) is waiting at time t if there is an active request $\sigma_i(s_i, d_i) \in \sigma, b_i \leq t$ for which a handshake has been completed and either s_i or d_i is an ancestor of the other, i.e., is the $LCA_t(s_i, d_i)$.
5) *Communicating:* A node s_i or d_i is communicating at time t if there is an active request $\sigma_i(s_i, d_i) \in$

1. $|\mathcal{C}_t(u)| = 4$ if $step_t(u)$ is a zig-zig or zig-zag, $|\mathcal{C}_t(u)| = 3$ if it is a zig.

TABLE 1
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	...	$t-6$	$t-5$	$t-4$	$t-3$	$t-2$	$t-1$	t
σ_1	✓	✓	✓	-	-	-	-	...	-	-	-	-	-	-	-
σ_2	-	✗	✗	✓	✓	✓	-	...	-	-	-	-	-	-	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
σ_{m-1}	-	-	-	-	-	-	-	...	✓	✓	-	-	-	-	-
σ_m	-	-	-	-	-	-	-	...	✗	✗	✓	✓	✓	✓	-



T_1: Upon new request start handshake
T_2: Started a request and is not LCA
T_3: Started a request as LCA
T_4: Reached LCA
T_5: Left LCA due to higher-priority node
T_6: Source and destination nodes meet
T_7: Fulfill the communication request

Fig. 3. DiSplayNet state transition diagram (updateState()).

Algorithm 1 DiSplayNet preStep()

1: Start New Handshake (1 time-slot)
if *Handshake* for some $\sigma_i(s,d)$ **then**
send $sync(\sigma_i)$ to d;
insert $sync(\sigma_i)$ into sync buffer;

2: Send Ack to Source (1 time-slot)
upon receiving each $sync(\sigma_j)$**:**
insert $sync(\sigma_j)$ into sync buffer;
get highest priority $sync(\sigma_k)$ in sync buffer;
if $k \neq i$ **then**
send $Ack(sync(\sigma_k))$ to source node of σ_k;

3: Send Ack to Destination (1 time-slot)
upon receiving $Ack(sync(\sigma_i(s,d)))$**:**
send $Ack(Ack(sync(\sigma_i)))$ to d;
updateState(); //goto clusterStep(); (Fig. 3, Alg. 2)
clear sync buffer;

4: End Handshake (1 time-slot)
upon receiving $Ack(Ack(sync(\sigma_k(s,d))))$**:**
updateState(); //goto clusterStep();
clear sync buffer;

$\sigma, b_i \leq t$ for which a handshake has been completed and $d_t(s_i, d_i) = 1$.

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 source and destination meet, they communicate and all the data is sent.

To synchronize the process between the nodes, the distributed algorithm proceeds in *rounds*. Each round is comprised of a sequence of three routines. Essentially, each node in DiSplayNet executes:

while (*true*) **do** {*preStep()*; *clusterStep()*; *sendData()*;}

In the beginning of each round, four time-slots are reserved for a three-way *handshake* procedure between the source-destination node pairs in the *Handshake* state. This routine is described in Algorithm 1 ($preStep()$) and is necessary to check if the destination node of a request is available to start the communication. The source node begins by sending a handshake request to the destination node of that message. The destination node maintains a priority queue (sync buffer) with possibly other such requests and commits to the one with the highest priority. We assume that the small control messages exchanged during the handshake phase are sent at no additional cost over a one-hop communication backbone.[2]

Next each node executes the *cluster step* routine, comprised of five (synchronized) phases, summarized in Algorithm 2 ($clusterStep()$): (1) Cluster Requests; (2) Top-down Acks; (3) Bottom-up Acks; (4) Link Updates; (5) State Updates. Each node $u \in V$ maintains a local (cluster) buffer, containing a priority queue of cluster requests, generated by itself, its right or 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 to its *master* (a 2-hop ancestor in the case of a zig, or a 3-hop ancestor in the case of a zig-zig or zig-zag). 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. A set of nodes form a cluster $\mathcal{C}_u$ if they have received a top-down and a bottom-up acknowledgment for request($\mathcal{C}_u$).

2. This assumption is justified by the fact that hybrid architectures, where self-adjusting links are integrated into a static network, are likely to prevail [20].

Algorithm 2 DiSplayNet clusterStep()

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

2: Top-down Acks (3 time-slots)
  get highest priority $request(\mathcal{C}_x)$ in cluster 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($request(\mathcal{C}_w)$) to 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 to master;
      create $\mathcal{C}_u$;
      join $\mathcal{C}_u$;
    else
      forward Ack($request(\mathcal{C}_u)$) to 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)
  updateState(); (Fig. 3)
  clear cluster buffer;
  leave cluster;

Finally, if the distance between the source and the destination of an active message is one, the two enter the *Communicating* state, the data is transferred to the destination, and both nodes either return to the *Passive* or *Handshake* states (Fig. 3).

3.4 Concurrent Progress Matrix

In a concurrent setting, instead of each row in the progress matrix representing a request σ_i, each row represents an individual source or destination node, since both nodes s_i and d_i work in parallel. 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.

To illustrate how progress is made in a concurrent scenario, let us look at an example request sequence $\sigma = (\sigma_1(s_1, d_1), \sigma_2(s_2, d_2), \sigma_3(s_3, d_3))$ in a DiSplayNet, illustrated in Figure 4 (the destination node d_1 (in T_1 or T_2) is not depicted in the picture, and we assume that it performs steps towards its source node s_1 during the depicted five rounds). The corresponding progress matrix is depicted in Table 2. In rounds t and $t+1$ (Figure 4(a)-(b)), node s_3 is paused by the source s_1 node of request σ_1, while nodes s_2 and d_2 make progress; in round $t+2$ (Figure 4(c)), node s_3 is bypassed by s_2; in round $t+3$ (Figure 4(d)), s_3 is bypassed by d_2, and nodes s_2 and d_2 meet and exchange data; in round $t+4$ (Figure 4(e)), request σ_2 is no longer active, so s_3 makes progress again; because σ_3 has lower priority than σ_1, it will remain locked (paused or bypassed) by either s_1 or d_1, until σ_1 completes and is no longer active.

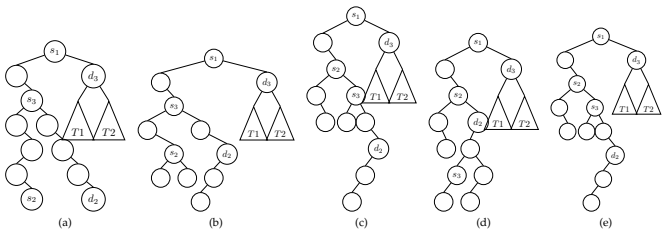


Fig. 4. Example of concurrent execution of a DiSplayNet, progress matrix in Table 2.

4 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.

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. *All link updates are consistent and clusters are 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 2, phases 2,3). Therefore, no node can belong to more than one cluster. Moreover, all links in the network are updated simultaneously in each round, which maintains consistency (Algorithm 2, phase 4).

More specifically, Algorithm 2 ensures that only the nodes that change a link to a child will belong to a cluster $\mathcal{C}_t(u)$. For the sake of contradiction, assume there is a node $p \in \mathcal{C}_t(u)$ that is the parent of node $c \notin \mathcal{C}_t(u)$, and that during $step_t(u)$, the link between p and c is modified. Let us consider another concurrent step $step_t(x)$ in round t, such that $c \in \mathcal{C}_t(x)$. $step_t(x)$ cannot update the link between p and c, because $p \notin \mathcal{C}_t(x)$. □

An essential property of DiSplayNets is that they never enter deadlock.

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

Proof. Proof by induction on the priority of σ_i, the lowest-priority request in σ that has not been completed before round $t > 0$. **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 $d(s_1, d_1) = 1$.

TABLE 2
Concurrent progress matrix: $\mathcal{M}(\sigma, DiSplayNet, T_0), \sigma = (\sigma_1(s_1, d_1), \sigma_2(s_2, d_2), \sigma_3(s_3, d_3))$

	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	-	-	✓	...	✗	✗	✗	✗	✗	✗	✗	...	✓	...	-

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 before round t. Note that the induction hypothesis does not exclude the possibility that $\exists \sigma_k, k > i$, completed before 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. Thus, it has no more obstructions and will complete without interruptions. □

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 1). 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 in round 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 round t, u is either climbing or waiting. Thus, for each node u making progress in round 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. Every node in $\mathcal{C}_t(u)$ but u cannot 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. *[23] 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 2, 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 can only be 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:

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

Splaying a node u from its current location up to the LCA of itself and its destination, possibly facing $2(m-1)$ bypasses, consists of at most $2m$ sequences of p_i $steps_t(u)$, each comprised by a double rotation (zig-zig or zig-zag), plus possibly one single rotation (zig) at the end of each interval. The last single rotation is needed in case in some time slot t, the distance between node u and the LCA becomes equal to one.

Observe that each double rotation has cost 2 and a single rotation has cost 1. So the *actual cost* is upper bounded by $\sum_{i=1}^{2m} 2p_i + 2m$. By replacing this value in Eq. 1, we conclude that 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 2), 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. □

5 SIMULATIONS

To complement our formal worst-case analysis and to shed light on the performance of DiSplayNets 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.

TABLE 3
Datasets: n nodes, m requests, T(σ) temporal, NT(σ) non-temporal (spatial) locality

Dataset	n	m	T(σ)	NT(σ)
ProjecToR [18]	128	10,000	very low	very high
PFabric [33]	144	10,000	very high	very low
Multigrid [34]	1,024	1,000,000	high	high
Bursty [2]	1,024	10,000	very high	very low
Facebook [35]	159	1,000,000	very low	very low
Splay Tree	128	10,000	very low	low

5.1 Setup and baselines

Our simulations are event-driven and based on the Sinalgo [32] network simulator. In order to generate a sequence over time, we assumed a Poisson distribution for the request arrival, with $\lambda = 0.05$.

To better understand and compare the simulation results, we implemented three baseline algorithms. To study the benefit of dynamic reconfiguration performed by DiSplayNets (**DSN**), we implemented two static networks:

- **BT:** a balanced static BST network. This baseline represents the optimum topology when all pairs of nodes are equally likely to communicate.
- **OPT:** a statically optimum BST network, implemented using the dynamic program from [24], which is demand-aware and optimized towards the request frequency distribution of a given communication sequence. This baseline has the advantage of knowing the distribution *ahead of time*.

Neither of the two static baselines incur any reconfiguration costs, but only communication costs (one unit cost per link). Furthermore, we implemented a sequential self-adjusting network, to investigate the benefit and limitation of concurrency:

- **SplayNet (SN):** a sequential self-adjusting network, based on the algorithm from [24].

5.2 Datasets

To generate request workloads, we leverage two datasets collected from three real systems (ProjecToR [36], Multigrid [34], and Facebook [35]) and three artificially generated datasets (PFabric [33], Bursty [2] and Splay Tree), as shown in Table 3. We divided the input workloads into test groups according to their type of locality, as follows:

High non-temporal and low temporal locality (ProjecToR): The ProjecToR dataset [36] describes the distribution of communication probability between $8,367$ pairs of nodes in a network of $n = 128$ nodes (top of racks), randomly selected from 2 production groups, running between Map Reduce operations, index builders, database and systems storage. We sampled a sequence of $m = 10,000$ independent and identically distributed requests ($i.i.d.$) in time by the communication matrix provided and repeated each experiment 30 times. A single source-destination pair is responsible for 28% of all communication, and only a few pairs are responsible for as much as 80% of all communication. This dataset has no temporal and high non-temporal locality.

High temporal and low non-temporal locality (PFabric and Bursty): The PFabric (0.8 trace) dataset [33] consists of a synthetic trace generated by executing simulations scripts in NS2, with a Poisson arrival process, reproducing a web search (high) workload scenario. When a flow arrives, the source and destination nodes are chosen uniformly at random. We sampled a sequence of $m = 10,000$ communication requests from a network of $n = 144$ nodes. This dataset presents high temporal and non-temporal locality.

The Bursty dataset was generated artificially to have very high temporal and very low non-temporal locality, following a methodology for synthetic workload generation, presented in [2]. For each request ($m = 10,000$), a pair of source-destination nodes (s, d) was chosen uniformly at random from a network of $n = 1024$ nodes. With high probability ($p > 0.9$), the request from s to d was repeated, until a different pair was chosen (with probability $1 - p$).

High temporal and non-temporal locality (Multigrid): The Multigrid dataset [34] consists of high-performance computing applications, such as solutions of Poisson's equations, hyperbolic components of the Navier-Stokes equation, and solution for elliptical linear system models. We collected a sample of $m = 1,000,000$ requests for a network of $n = 1024$ nodes. This dataset presents high levels of locality, both temporal and non-temporal.

Low locality (Facebook and Splay Tree): The Fbflow trace consists of real-system Fbflow[3] samples collected from three production clusters on Facebook. The per-packet sampling is uniformly distributed with rate 1:30000, 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 $m = 1,000,000$ requests, originated in a 24-hour time window, in a network comprised of $n = 159$ nodes. In this dataset no source-destination pair accounts for more than 0.15% of overall traffic, so the spatial locality is low. Moreover, due to the way the samples were collected, this dataset has low temporal locality.

Since DiSplayNets are an extension of self-adjusting data structures, it is interesting to analyze their performance on request sequences encountered in self-adjusting data structures. The Splay Tree dataset consists of artificial communication sequences ($m = 10,000, n = 1024$), where each message's destination is the root node of the network and the source is chosen randomly, following a normal distribution ($std = 0.8$) over the remaining $n - 1$ nodes.

5.3 Results

In order to evaluate the performance of DiSplayNet relative to the baselines, we made empirical measurements of the total (reconfiguration) work, total routing cost of the static baselines, throughput and makespan of the dynamic self-adjusting networks, as well as the number of concurrent clusters and number of pauses and bypasses of DiSplayNets, for different communication patterns.

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

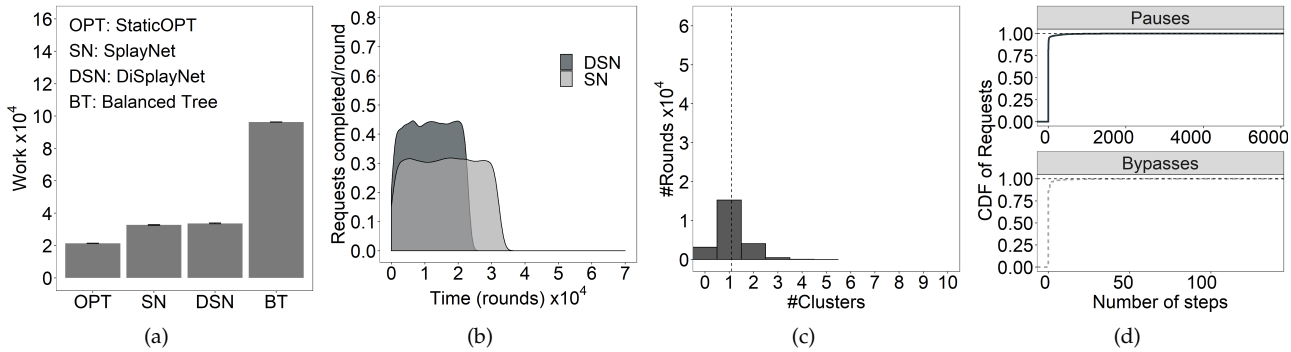


Fig. 5. ProjecToR ($n = 128, m = 10,000$): High non-temporal, low temporal locality

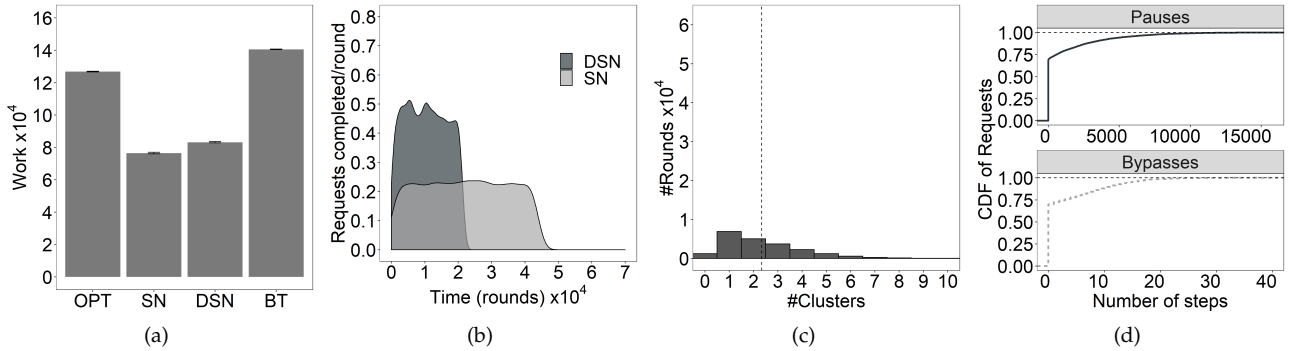


Fig. 6. Bursty ($n = 1024, m = 10,000$): high temporal, low non-temporal locality

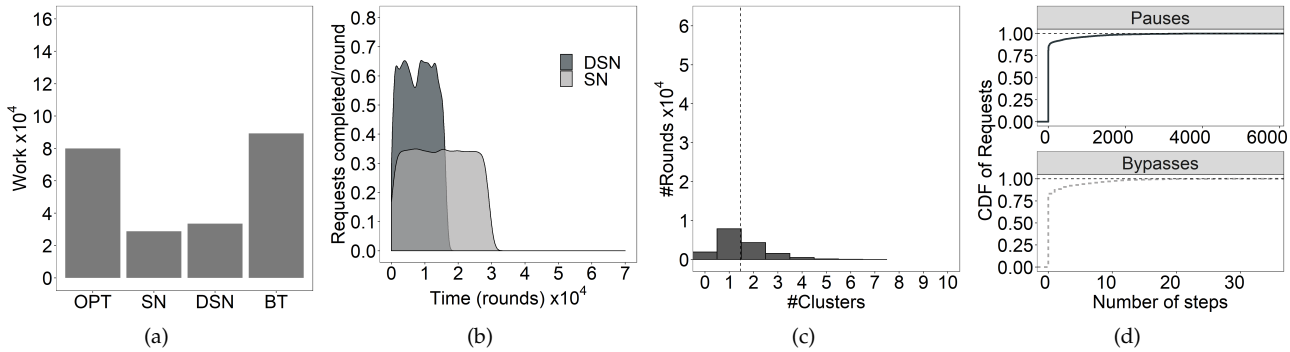


Fig. 7. PFabric trace 0.8 ($n = 144, m = 10,000$): high temporal, low non-temporal locality

5.3.1 Work: a price of decentralization?

$DiSplayNet \times SplayNet$: 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 5a, 6a, 7a, 8a, 9a, and 10a show the total work, measured in number of steps performed by DiSplayNet and the baselines, for all datasets. The results show that there is indeed little difference between the concurrent algorithm and the sequential one. In the facebook dataset DiSplayNet actually performed less work than SplayNet, as shown in Figure 10a. This shows that the analytical upper bound on the number of bypasses (conflicts between parallel splaying operations), derived in Lemma 5, is quite pessimistic. A bypass, therefore, represents a rare event in practice, which allows for efficient parallel work execution by DiSplayNet, as we discuss in the next section.

Dynamic reconfiguration $\times$ static baselines: For the datasets with low temporal locality (ProjecToR 5a, Splay Tree 9a and Facebook 10a) the total work performed by an optimum static network (recall that OPT knows σ a priori) is slightly lower than that by $SplayNet$ and $DiSplayNet$. This is expected, given that OPT, by definition, is the optimum topology for the given request frequency distribution. Since requests are distributed i.i.d. in time, dynamic reconfig-

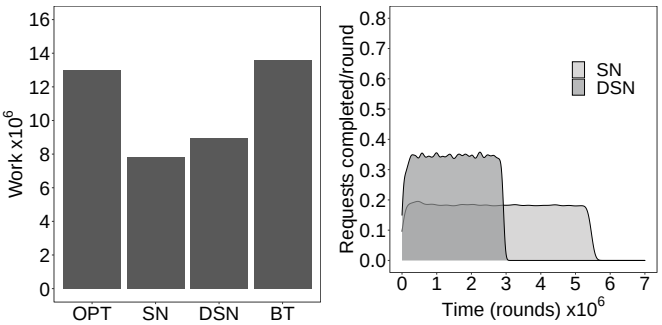


Fig. 8. Multigrid ($n = 1,024, m = 1,000,000$): high temporal, high non-temporal locality

uration cannot improve on that. For datasets with high temporal locality (Bursty 6a, PFabric 7a and Multigrid 8a), on the other hand, we can see that OPT 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. The static balanced BST (BT) presents work cost close to OPT in all datasets, except in ProjecToR 5a, where the high spatial locality leverage unbalanced tree topologies.

5.3.2 Throughput

Concurrent adjustments turn out to greatly improve the amount of communication requests which can be handled by the network. In Figures 5b, 6b, 7b, 8b, 9b, and 10b, we compare the throughput of dynamic self-adjusting networks, measured as the number of completed requests per round during the entire simulation, for all datasets. Note that there are no static baselines 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.

$DiSplayNet \times SplayNet$: It can be seen that, compared to the sequential execution of SplayNet, DiSplayNet significantly improves the throughput, for all datasets, except Splay Tree, where the throughput plots are exactly the same (Figure 9b). In the latter, since all requests are addressed to the root, the execution of DiSplayNets becomes sequential. The gain in time cost of DiSplayNets is especially pronounced in workloads with either low spatial (Bursty 6b and Facebook 10b) or high temporal (Bursty 6b, PFabric 7b and Multigrid 8b) locality. In the ProjecToR trace (5b), on the other hand, where temporal locality is low and spatial locality is high, the difference in throughput is less pronounced. Because some source-destination pairs generate as much as 80% of all data traffic, causing the local queues at these nodes to get long, the request completion is forced to become sequential, even in the concurrent implementation.

5.3.3 DiSplayNet: zooming into concurrent execution

In this section we take a closer look into the concurrent execution of DiSplayNet. In particular, we analyze the number of active clusters per simulation round (Figures 5c, 6c, 7c, 9c, and 10c) and the Cumulative Distribution Function (CDF) of the number of pauses and bypasses per communication request (Figures 5d, 6d, 7d, 9d, and 10d). Recall that a pause can increase the time cost but not the work cost of a request, whereas a bypass may increase both.

Concurrent clusters: Analyzing the distribution of the number of active clusters per round, we observe that there were 0 clusters in a few rounds. This means that no new requests were generated in those rounds and all the previous requests had already completed, i.e., the network was idle. Furthermore, in some rounds, there was only 1 active cluster, i.e., the execution was sequential. This occurred more frequently in the ProjecToR (5c) and Splay Tree (9c) datasets. In the Bursty (6c), PFabric (7c) and Facebook (10c)) datasets, we observe greater concurrency, as the number of concurrent active clusters per round reached values as high as 10.

Pauses and bypasses: Analyzing the CDFs of the number of bypasses performed in each communication request, we observe that a bypass occurred less than 30 times in more than 99% of requests, in all datasets. The number of pauses per request was considerably higher, reaching values as high as $6,000$ in the workloads with $m = 10,000$ requests and $50,000$ in the Facebook dataset, with $m = 1,000,000$ requests. Nevertheless, by analyzing the CDFs, we can see that such long waiting time is a relatively rare event, remaining below 500, $2,000$ and $5,000$ in 99% of requests in the real-world workloads ProjecToR (5d), Facebook (10d), and PFabric (7d), respectively. In the Splay Tree workload (9d) the number of pauses and bypasses was close to 0, given its sequential execution. In the Bursty workload (6d), the number of pauses was slightly higher than in the real-world workloads, due to its artificially high temporal locality. There is a relationship between higher concurrency and higher number of pauses, given that a higher number of active clusters per round (as in the Bursty dataset 6c) results in more conflicts among concurrent requests.

5.3.4 Final Remarks

It is worth noting that, in a real reconfigurable hardware scenario, the cost of a network reconfiguration will be significantly higher than that of forwarding a packet through one link. In this work, in order to make a platform-independent evaluation, in all experiments, we assumed that the static baselines incur only communication costs, one unit per link, and the dynamic approaches incur a cost of one unit for each single rotation, and a cost of two units for each double rotation. The actual gains in performance, however, will greatly depend on the specifics of each network reconfiguration technology, as well as on the pace of technological development and innovation in this sector.

6 Related Work

Reconfigurable networks have been explored both in the context of datacenters, e.g., [18], [19], [20], [21], in wide-area networks [37], [38], [39], and, more traditionally, in the context of overlays [40], [41]. See [42] 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 [43], [44], [45], [46], [47], [48], [49]. However,

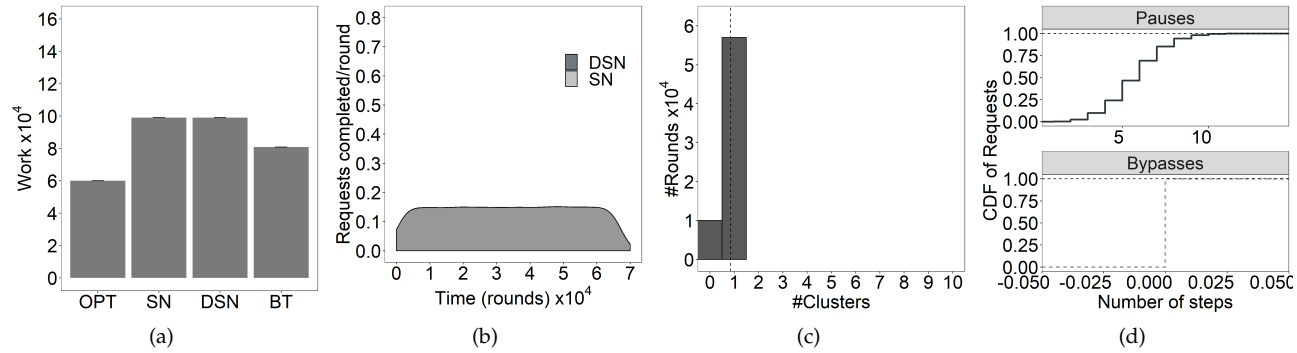


Fig. 9. Splay Tree ($n = 1024, m = 10,000$): low spatial and temporal locality

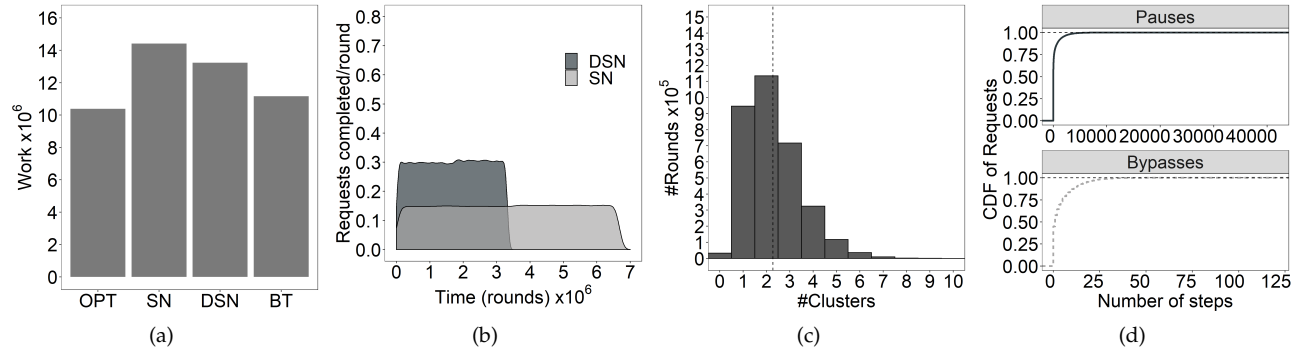


Fig. 10. Facebook ($n = 159, m = 1,000,000$): low spatial and temporal locality

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* [50] which minimize the average routing costs between arbitrary communication pairs by performing topological adaptation to the communication pattern, and Flattening [31] 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 work closest to ours is $SplayNet$ [24]. 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 [23], [24].

7 Conclusion

Motivated by emerging reconfigurable datacenter networks whose topology can be adapted toward the traffic in a demand-aware manner, we developed and analyzed $DiSplayNet$, the first self-adjusting network which is fully distributed. $DiSplayNet$ comes with analytical performance guarantees and also performs well in our simulations under different real-world and synthetic workloads.

Our work opens several interesting avenues for future research. On the theory front, 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 front, it will be interesting to study the integration and use of self-adjusting links of the topology with links that are not self-adjusting: hybrid architectures are likely to prevail also in the future.

Acknowledgments

The authors would like to thank Capes, Fapemig and CNPq. Research supported by the European Research Council (ERC), grant agreement 864228 (AdjustNet), 2020-2025

References

[1] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "HPCC: High Precision Congestion Control," in *Proc. of the ACM Special Interest Group on Data Communication*, 2019, pp. 44–58.

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

[3] J. Woodruff, A. W. Moore, and N. Zilberman, "Measuring burstiness in data center applications," in *Proc. 2019 Workshop on Buffer Sizing*, 2019.

[4] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proc. of the 2017 Internet Measurement Conference*, 2017, pp. 78–85.

[5] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proc. of the 1st ACM workshop on Research on enterprise networking*, 2009, pp. 73–82.

[6] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of tcp throughput collapse in cluster-based storage systems." in *FAST*, vol. 8, 2008, pp. 1–14.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[8] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *USENIX NSDI*, 2013, pp. 399–412.

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

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

[11] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: a high performance network structure for modular data center interconnection," in *Proc. ACM Int. Conference on Emerging Networking Experiments and Technologies (CONEXT)*, 2009, pp. 25–36.

[12] 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.

[13] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers, randomly," in *USENIX NSDI*, vol. 12, 2012, pp. 17–17.

[14] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. 9th ACM Internet Measurement Conference (IMC)*, 2009, pp. 202–208.

[15] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.

[16] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proc. ACM SIGCOMM*, 2017, pp. 267–280.

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

[18] 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 *SIGCOMM Conference*. ACM, 2016, pp. 216–229.

[19] 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 *Proc. of the 11th USENIX Conference on Networked Systems Design and Implementation*. USA: USENIX Association, 2014, pp. 1–15.

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

[21] 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.

[22] 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.

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

[24] 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.

[25] Y. Afek, H. Kaplan, B. Korenfeld, A. Morrison, and R. E. Tarjan, "The CB Tree: A Practical Concurrent Self-Adjusting Search Tree," *Distrib. Comput.*, vol. 27, no. 6, p. 393–417, Dec. 2014.

[26] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.

[27] 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.

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

[29] J. R. Bitner, "Heuristics that dynamically organize data structures," *SIAM J. Comput.*, vol. 8, no. 1, pp. 82–110, 1979.

[30] B. Allen and I. Munro, "Self-organizing binary search trees," *J. ACM*, vol. 25, no. 4, pp. 526–535, Oct. 1978.

[31] M. K. Reiter, A. Samar, and C. Wang, "Self-optimizing distributed trees," in *IEEE IPDPS*, 2008, pp. 1–12.

[32] D. C. Group, "Sinalgo - simulator for network algorithms," http://disco.ethz.ch/projects/sinalgo/index.html, 2007, accessed 10-January-2020.

[33] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "PFabric: Minimal near-Optimal Datacenter Transport," in *ACM SIGCOMM Conference*, 2013, p. 435–446.

[34] U. DOE, "Characterization of the DOE mini-apps." https://portal.nersc.gov/project/CAL/overview.htm, (accessed August 2021).

[35] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Conference*, 2015, pp. 123–137.

[36] "ProjecToR dataset," www.microsoft.com/en-us/research/project/projector-agile-reconfigurable-data-center-interconnect, (accessed August 2021).

[37] 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.

[38] 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.

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

[40] C. Scheideler and S. Schmid, "A distributed and oblivious heap," in *ICALP*, ser. Lecture Notes in Computer Science (LNCS), vol. 5556. Springer, 2009, pp. 571–582.

[41] 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.

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

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

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

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

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

[47] 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.

[48] K.-T. Foerster, M. Pacut, and S. Schmid, "On the complexity of non-segregated routing in reconfigurable data center architectures," in *ACM SIGCOMM Computer Communication Review*, 2019.

[49] T. Fenz, K.-T. Foerster, S. Schmid, and A. Villedieu, "Efficient non-segregated routing for reconfigurable demand-aware networks," in *Proc. IFIP Networking*, 2019.

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

Bruna Soares Peres Bruna Peres received her PhD degree in Computer Science in 2019 from Universidade Federal de Minas Gerais (UFMG). She is currently working as a software engineer at Google Brazil. Her research interests include algorithm design and analysis for computer networks.

Otávio Augusto de Oliveira Souza Otávio A. de O. Souza received the M.Sc degree in Computer Science in 2020 from Universidade Federal de Minas Gerais (UFMG), Brazil, where he is currently a PhD candidate in Computer Science. His research interest is in computer networking.

Olga Goussevskaia Olga Goussevskaia is an associate professor of computer science at Universidade Federal de Minas Gerais (UFMG), Brazil. She received her PhD in computer science in 2009 from ETH Zurich, Switzerland. Her main research interests include modeling, algorithm design and analysis in communication networks, with emphasis on distributed systems, wireless networks and complex systems.

Chen Avin Chen Avin is an associate professor in the School of Electrical and Computer Engineering at the Ben Gurion University of the Negev, Israel which he joined in October 2006. He received the B.Sc. degree in communication systems engineering from Ben Gurion University, Israel, in 2000, and his M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles (UCLA), CA, USA, in 2003 and 2006, respectively. His current research interests are: data-driven graphs and networks algorithms, modeling and analysis with emphasis on demand-aware networks, distributed systems, social networks and randomized algorithms for networking.

Stefan Schmid Stefan Schmid is professor for computer science at the University of Vienna, Austria. He received his MSc (2004) and PhD (2008) from ETH Zurich, Switzerland. Subsequently, Stefan Schmid worked as postdoc at TU Munich and the University of Paderborn (2009), in Germany. From 2009 to 2015, he was a senior research scientist at T-Labs in Berlin, Germany, and from 2015 to 2018, an Associate Professor at Aalborg University, Denmark. His research interests revolve around the fundamental algorithmic problems of networked and distributed systems.