Lazy Self-Adjusting Bounded-Degree Networks for the Matching Model

Evgeniy Feder*, Ichha Rathod†, Punit Shyamsukha† Robert Sama‡ Vitaly Aksenov*, Iosif Salem‡, Stefan Schmid§

*Computer Technologies, ITMO University, Russia

†Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

‡Faculty of Computer Science, University of Vienna, Austria

§TU Berlin, Germany & Faculty of Computer Science, University of Vienna, Austria

***Abstract*—Self-adjusting networks (SANs) utilize novel optical switching technologies to support dynamic physical network topology reconfiguration. SANs rely on online algorithms to exploit this topological flexibility to reduce the cost of serving network traffic, leveraging locality in the demand. While prior work has shown the potential of SANs, the theoretical guarantees rely on a simplified cost model in which traversing and adjusting a single link has uniform cost.**

We initiate the study of online algorithms for SANs in a more realistic cost model, the Matching Model (MM), in which the network topology is given by the union of a constant number of bipartite matchings (realized by optical switches), and in which changing an entire matching incurs a fixed cost α. The cost of routing is given by the number of hops packets need to traverse.

Our main result is a lazy topology adjustment method for designing efficient online SAN algorithms in the MM. We design and analyze online SAN algorithms for line, tree, and bounded degree networks in the MM, with cost $\mathcal{O}(\sqrt{\alpha})$ times the cost of reference algorithms in the uniform cost model. We report on empirical results considering publicly available datacenter network traces, that verify the theoretical bounds.

***Index Terms*—self-adjusting networks, matching model, online algorithms**

I. Introduction

Data center network traffic is currently growing explosively, roughly doubling each year while the capacity of packet switches doubles only roughly every two years [1], [2]. To tackle this mismatch, the networking community is currently making major efforts to innovate data center networks and their traffic handling, among other approaches.

Optical switches (but also other networking hardware [3]) allow *reconfiguring* the physical network topology, thus giving a wide range of flexibility in network topology design. Harnessing this flexibility to design more efficient (data center) networks is a research direction that has attracted recent attention from both network theoreticians and practitioners. The goal is to design *self-adjusting networks (SANs)*, i.e., networks that automatically adapt their topology depending on changes in the demand, to reduce routing costs. Such adaptations bear potential, as the traffic demand patterns in data center networks are skewed [4].

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 864228 "Self-Adjusting Networks (AdjustNet)". Vitaly Aksenov is supported by JetBrains Research.

In this work, we study SANs from an algorithmic point of view. SAN algorithms dictate how the network topology should change when there are shifts in the traffic demand, and especially, in the set of large "elephant flows" [3], [4]. In particular, in this paper we consider a model where the network needs to serve routing requests which arrive over time, in an online manner. Existing SAN algorithms are based on a uniform cost model where both traversing and changing a link have unit cost [5], [6]. This is a useful basic model that enabled the first algorithmic results. In practice, however, switching hardware usually allows to reconfigure the topology on a per-matching granularity, and changing a matching in a demand-aware manner is more costly than traversing a link (e.g., in terms of time) [3], [7].

The Matching Model (MM) proposed in [7] addresses this discrepancy, by assuming that traversing a single link has unit cost and changing the whole topology G to a new one G' comes at a fixed cost. Any topology can be defined as a union of matchings over the set of nodes and the MM assumes that rearranging the edges (links) of a single matching comes at a fixed cost (e.g., time), say α. Thus the total cost for adjusting the whole topology to a new one is the product of α and the number of matchings needed to construct the topology. In this paper we focus on scalable topologies where the maximum degree Δ is a constant and thus the topology reconfiguration cost in the MM is $\mathcal{O}(\alpha)$, as the number of matchings needed is constant as well. This model better fits systems and hardware properties and early work has shown its relevance [1], [8], [9]. However, so far, we lack algorithmic and analytical techniques for this model.

This paper presents a first analysis of the Matching Model and describes efficient online algorithms for this model. Concretely, we study online algorithms that start from an initial topology and upon each request, they first serve the request and then decide whether to adjust the topology, where adjustment would incur a fixed cost $\mathcal{O}(\alpha)$. As the sequence of routing requests is unknown a priori, we compare our SAN algorithms with the performance of state-of-the-art algorithms in the uniform cost model (lower bounds are known only in some cases [5] and relate to the entropy of the request sequence).

Note that adjusting the topology upon every request in the MM is inefficient for $\alpha \in \omega(1)$, as this could incur a

multiplicative factor of α in the cost of serving sequences of routing requests, even for trivial cases. For example, consider a line network and any sequence for which the source is always the same node and the destinations are arbitrary: $((s, d_1), (s, d_2), \ldots)$. In the uniform cost model, fixing the source s at one edge of the line network and, upon every request (s, d_i), moving the destination d_i such that it becomes adjacent to s (move to front) is asymptotically optimal [10]. But the same approach in the MM is α times more costly.

In this paper, we present our results for the MM in three steps; we start with line topologies, then we move to tree topologies, and we finally reach our main goal, which is bounded-degree networks. Our main contribution is a method for designing efficient online SAN algorithms in the MM, when compared to reference SANs in the uniform cost model. We cache a constant amount of topology adjustments and then *lazily* apply them by switching to a topology that is a result of all cached adjustments when it is most beneficial to pay the cost α of topology reconfiguration.

Our method of *lazy* topology reconfiguration transforms a self-adjusting algorithm from the uniform-cost model to one in the MM. We show that in the three bounded-degree topology families we studied, the SANs in the MM cost $\mathcal{O}(\sqrt{\alpha})$ times the algorithm cost in the uniform cost model, which is a clear improvement from the naive α factor that we mentioned earlier. We start by demonstrating our method in line topologies, where we show that it achieves static optimality in the MM. As an intermediate step, we present LAZYSPLAYNET, an online SAN in the MM, with cost $\mathcal{O}(\sqrt{\alpha})$ times the cost of SPLAYNET [5], a state-of-the-art SAN for tree networks in the uniform cost model. Our main algorithmic result is LAZYRENET, an online SAN in the MM, with cost $\mathcal{O}(\sqrt{\alpha})$ times the cost of RENET, a statically-optimal self-adjusting bounded-degree network in the uniform cost model. As RENET is a hybrid network, combining a static and a dynamic topology, our paper contributes to understanding both pure and hybrid networks in the MM.

We complement our theoretical analysis by testing our algorithms over publicly available data sets of data center network traces (e.g. from Facebook's data centers). Our findings confirm our theoretical results, in a number of datasets with varying skewness in the demand. We also explore connections of our results to self-adjusting data structures where requests originate from a single node, and operations describe search rather than routing tasks. In particular, we prove an $\mathcal{O}(1)$ competitive ratio for search in a line topology.

Paper organization. In Section II, we present related work. In Section III, we introduce basic definitions such as Self-Adjusting Network (SAN) algorithms and static optimality, and then we define two models, the Standard Model (SM) and the Matching Model (MM). In sections IV and V, we present our algorithms and analyses for search and routing sequences in the MM, for line and binary tree networks, respectively. In Section VI, we present and analyze LAZYRENET. In Section VII, we present experimental analysis of our algorithms and their evaluation, before concluding the paper.

II. RELATED WORK

An overview of the challenges and subdomains of self-adjusting networks was presented in [11], while the first thorough study that appeared was about SPLAYNET [5]. Demand-aware networks of bounded degree were first studied by Avin et al. [12]. Given the communication pattern and a bound on the node degree they design a static network topology that minimizes the expected path length. Below, we refer to two common optimality conditions: static and dynamic optimality. An online self-adjusting algorithm is statically-optimal if it performs as well as a static topology with perfect knowledge of the request sequence. An self-adjusting algorithm is dynamically-optimal if it is constant-competitive. Moreover, the MM was introduced in [7]. In the following, we present related work for the uniform-cost.

The majority of related work for bounded-degree SANs appears mostly in connection to the uniform-cost model. As advocated in [5], [11], many useful tools from self-adjusting data structures can be used in the design of self-adjusting networks. In the context of our paper, we mention related work for bounded-degree networks, and specifically for line and tree-based topologies as well as for more general bounded-degree SANs.

Designing SANs for line topologies has been proven a hard task as shown in [13]. In the static case it reduces to the Minimum Linear Arrangement problem, which is NP-hard [14]. In the online case, [13] showed a $\Omega(\log n)$ lower bound on the competitive ratio. Meanwhile, there is a rich literature on search requests on a line topology, representing an (unsearchable) linked list data structure. In a uniform cost model the move-to-front algorithm of Sleator and Tarjan is 2-competitive [10], whereas Albers and Janke gave a recent overview of existing constant-competitive deterministic and randomized online algorithms [15]. In short, the move-to-front rule, moves the accessed element to the front of the line (list). A notable case is the paid-exchange model (P^d), where each swap of two adjacent items incurs a cost d. Albers et al. presented a deterministic 4.56-competitive algorithm in [16].

Tree-based SANs were introduced in SplayNet [5], a networks-equivalent idea of the statically-optimal splay trees [17]. Splay-trees include three tree rotations (zig, zig-zig, zig-zag, cf. Figure 2) that adjust a tree upon an access request. [5] also introduces a set of lower bounds for specific cases, but general lower bounds in the uniform-cost model are yet to be studied. To our knowledge, [18] is the only distributed bounded-degree SAN in the literature ([19] also present a distributed version of their SAN, but the maximum degree is not bounded and in expectation is $\Theta(\log n)$). Avin et al. [20] present dynamically-optimal single-source tree networks, in a restricted model, in which the topology is always a tree which can be adjusted only by swapping neighboring nodes. For such networks, they prove a working-set lower bound and present deterministic and randomized algorithms with complexity that matches the lower bound.

Avin and Schmid presented RENETs in [6], which is a

statically optimal bounded-degree SAN in the uniform cost model, under a sparsity restriction on the communication sequences. Their topology is a union of *ego-views*, i.e., low-degree stars or splay trees rooted at each node and including all recently communicated nodes. All nodes are connected via a static control network, which includes the network coordinator and exists for relaying control messages (e.g. adding a new route or deleting all edges when the network grows to a limit). Thus, a RENET is a hybrid network combining static and dynamic links, in contrast to the SANs we mentioned in this section. We elaborate on RENETs in Section VI.

III. DEFINITIONS AND MODELS

We define notions regarding the topologies, models, and optimality conditions, to be used in our algorithms and analyses.

Basic topologies. We model a network as a set of n computational nodes with connections among them, forming an undirected graph $G = (V, E)$, where $n = |V|$, G belongs to $\mathcal{G}$ and $\mathcal{G}$ is a family of undirected graphs which models the allowed network topologies. Throughout the paper we will use the terms network and graph interchangeably. *Connected topology* is a family of graphs, in which there exists a route between all pairs of nodes. *Line topology* is a connected topology, where two nodes (*head* and *tail*) have degree one, and all other nodes have degree two. *Tree topology* is a connected topology, that does not contain any cycles. That is, each node but one (*root*) has a *parent*. If each node has no more than two children the topology is named as *Binary tree topology*. We present the RENET topology in Section VI.

Self-Adjusting Networks (SANs). We define SAN algorithms and some related notions. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m) = ((u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m))$, where $u_i, v_i \in V$, be a sequence of *routing requests* to forward a packet from node u_i to v_i. If we assume that our topology has a distinguished node S, e.g., head for Lists and root for Trees, then instead of routing requests we perform *search requests* from node S when $u_i = S$ for all i and the notation of these requests will be $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, where $\sigma_i \in V$ (single-source networks).

The main goal of our work is to manipulate a dynamic graph, so as to optimize the total cost for processing all requests. We can solve this task in two manners: 1) statically, i.e., the graph does not change during the execution, or 2) dynamically, i.e., the graph can change during the processing of requests.

Definition 1. *A* static optimization task *is a task in which given all requests in advance we have to build a network with topology $G_{static} \in \mathcal{G}$ that does not change during or in between the requests. Such a graph G_{static} needs to optimize the total cost function* $\mathrm{sumCost}(static, G_{static}, \sigma) = \sum_{i=1}^{m} l_i$*, where l_i is the path length in edges to process request σ_i.*

Definition 2. *In a* dynamic optimization task*, we assume that we can change the network after each request. We are provided with an arbitrary initial network (before the first request arrives), which we denote by $G_0 \in \mathcal{G}$. Our task is to build an algorithm $\mathcal{A}$ that adjusts the network at any time instant G_i, $i = 0, 1, \ldots, m$, and minimizes the total cost, which is calculated as* $\mathrm{sumCost}(\mathcal{A}, G_0, \sigma) = \sum_{i=1}^{m} (routingCost(G_{i-1}, \sigma_i) + adjustmentCost(G_{i-1}, G_i))$*, where $routingCost(G_{i-1}, \sigma_i)$ is the path length in edges of G_{i-1} to process request σ_i and $cost(G_{i-1}, G_i)$ is the adjustment cost to reconfigure the network from step $i-1$, $G_{i-1} \in \mathcal{G}$, to step i, $G_i \in \mathcal{G}$.*

An algorithm that performs requests and adjusts the network at each step from $G_{i-1} \in \mathcal{G}$ to $G_i \in \mathcal{G}$ is called *Self-Adjusting Network algorithm*. SAN algorithms aim to exploit temporal locality by changing the topology to reduce the distance between frequently communicating nodes, while balancing the reconfiguration costs. In this work we assume that all changes to the network topology are controlled by a coordinator C. The coordinator is connected to all nodes in V and controls a structure of the network, as it is the case for the controller node(s) in an SDN control plane. We neglect the communication cost with the coordinator.

Models. We introduce the Standard Model (uniform-cost) and the newly proposed Matching Model [7]. Let algorithm $\mathcal{A}$ be a SAN algorithm. We define each model by specifying the cost of each operation.

Definition 3 (Standard Model (SM))**.** *In the SM the cost of traversing or adjusting a single edge (link) is equal to* 1*. Thus, $routingCost(G_{i-1}, \sigma_i)$ is the length of the route in G_{i-1} and $adjustmentCost(G_{i-1}, G_i)$ is the number of edges that change between G_{i-1} and G_i (single edge addition or deletion costs 1).*

In this work, we focus on the recently introduced Matching Model (MM) [7]. The motivation for this model comes from the physical properties of a next-generation dynamic networks based on optical switches [3]. Each physical node has a constant number Δ of physical sockets — a sender and a receiver. A socket represents a possible connection with another node.

Theorem 1 (Vizing's theorem [21])**.** Edge coloring of the graph *is a color function $C : E \to \{1, \ldots, \Delta\}$, where incident edges have different color values. Every undirected graph with bounded degree Δ can be provided with edge coloring using at most $\Delta + 1$ different colours.*

Let Δ be the maximum degree of any graph from $\mathcal{G}$. Thus, by Vizing's theorem, at each step i the edges of graph G_i can be edge colored using at most $\Delta + 1$ colors. Thus, our graph G_i can be represented by $\Delta + 1$ matchings.

In the MM, the coordinator C can perform the following command: ask all nodes simultaneously to change the edges of the matching with the chosen color at *fixed* cost α. Since G_i has constant number of matchings (more precisely, $\Delta + 1$) by Vizing theorem, the network can be changed from G_i to any $G_{i+1} \in \mathcal{G}$ by paying the cost $\alpha \cdot (\Delta + 1) = \mathcal{O}(\alpha)$ (Δ is constant for bounded-degree networks, which are the focus of this work).

Definition 4 (Matching Model (MM))**.** *In the MM, the routing cost is defined as in the SM and the adjustment cost per request, $adjustmentCost(G_{i-1}, G_i)$, is $c_{\sigma_i} \cdot \alpha$, where c_{σ_i} is the number of matchings that the SAN algorithm changed between G_{i-1} and G_i on i-th request.*

We remark that for bounded degree networks c_{σ_i} is a constant. Moreover, the adjustment cost of $\mathcal{O}(\alpha)$ counts both the computation of the new topology G_i by the coordinator C and the physical reconfiguration cost. Although [7] presents a more elaborate model, Definition 4 abstracts further details that are not pertinent to this paper.

Optimality of SAN algorithms. Two desirable optimality properties of online SAN algorithms are static and dynamic optimality [11]. Let $\mathrm{sumCost}(\text{static}, G, \sigma)$ be the cost of the algorithm that computes a fixed network topology that minimizes the cost of serving a given sequence of communication requests, when no adjustments are allowed (Definition 1). A SAN algorithm $\mathcal{A}$ is called *statically optimal* if for every sequence of requests σ and for every starting configuration G_0, $\mathrm{sumCost}(\mathcal{A}, G_0, \sigma) = \mathcal{O}(\mathrm{sumCost}(\text{static}, G_{static}, \sigma))$, where G_{static} is the offline (optimal) static topology. Similarly, a SAN algorithm $\mathcal{A}$ is called *dynamically optimal* if for every sequence of requests σ and for every starting configuration G_0, $\mathrm{sumCost}(\mathcal{A}, G_0, \sigma) = \mathcal{O}(\mathrm{sumCost}(OPT, G_0, \sigma))$, where OPT is optimal online algorithm with perfect knowledge over σ. We relax the notion of static optimality by calling a SAN algorithm *c-statically optimal* if for each sequence of requests σ and for each starting configuration G_0, $\mathrm{sumCost}(\mathcal{A}, G_0, \sigma) \leq c \cdot \mathrm{sumCost}(\textit{static}, G_{static}, \sigma)$, where *static* is the algorithm that computes an optimal fixed topology that minimizes the cost of serving σ.

IV. Lazy Line Networks

We first expose our *lazy* topology adjustment method in line network topologies. We start with single-source communication sequences (search requests). In the Standard Model (SM) we are provided with a dynamically optimal Move-To-Front (MTF) algorithm [10]. We note that in the Matching Model (MM) the *"move-to-front"* operation costs α. Thus, we amortize this cost increase by not adjusting the network at each search request, but when a threshold of routing cost has been reached. The following straightforward optimization of the MTF algorithm for the MM gives an improved theoretical bound:

- maintain a counter for each node, being zero at initialization,
- on each request for a node, we increase the node's counter by one,
- If the counter becomes α, we perform a move-to-front operation on this node (thus the network adjustment cost will be amortized over α operations).

We refer to this algorithm as "Lazy Move-To-Front". It is not surprising that "Lazy MTF" is statically optimal in the MM: "Lazy MTF" is exactly the deterministic version of the randomized COUNTER algorithm in P^d from [22, Section 3.3] which is shown to be constant competitive (hence also statically optimal). Here we present a simple proof of this optimality condition under an additional assumption: the number of performed requests should be quite large.

To prove our claim we use the following lemmas from the original paper on Move-To-Front [10]. Let $\phi_T(x)$ be the MTF potential function of element x at request $T \in \{1, \ldots, m\}$, which (as in [10]) is the number of inversions with element x in the MTF list with respect to the optimal list (OPT). An inversion is a pair (x, y) of items such that x occurs before y in the MTF list and it occurs after y in list of OPT. Also $cost(x)$ is the distance of x from the list front.

Lemma 1 ([10])**.** *Let $p_T(x)$ be the position of element $\sigma_T = x$ in the MTF list at time T and j be the position of element σ_T in the OPT list. Then, $j \geq p_T(x) - \phi_T(x)$.*

Lemma 2 ([10])**.** $\mathrm{cost}(\sigma_T) + \Delta\phi_T(x) = 2(p_T(x) - \phi_T(x)) - 1$

We re-prove these lemmas in the proof of the following theorem and in the context of the MM.

Theorem 2. *The "Lazy Move-To-Front" algorithm is statically optimal in the Matching Model if $|\sigma| \geq \alpha \cdot \frac{n(n+1)}{2}$.*

Proof. We prove this theorem using a standard amortization argument. For that we introduce a potential function for the Lazy MTF after T requests: $\Phi_T = \sum \alpha \cdot \phi_T(x)$, where $\phi_T(x)$ is the MTF potential function of element x, which is the number of inversions with element x in MTF list with respect to the statically optimal list (OPT). For simplicity we assume without loss of generality that Move-To-Front and OPT start with the same list, so, the initial potential is zero. (Note, that if this potential is not zero we can amortize its value across all requests, since their number exceeds the maximal potential.)

Our potential function changes only on the α-th request per element. So, we split the requests to a fixed element x into blocks of length α. Now, we want to compare the total cost of requests in each block in OPT and Lazy MTF list.

i) The cost of α operations on x in the list of OPT is straightforward. It is equal to $\alpha \cdot j$, where j is the position of x in OPT list.

ii) Consider α requests to the element x. Let $p_{T_i}(x)$ be the position of x in Lazy MTF after the T_i-th request where i is the index of the request to x and T_i is the index of the request in σ. Thus, the total cost of the first $\alpha - 1$ requests that access x is equal to $\sum_{i=1}^{\alpha-1} p_{T_i}(x)$.

Let us perform the α-th access to x. We briefly revise the proof of Lemma 2 and use the same amortization idea here.

The change in the MTF potential due to moving x to the front is equal to the sum of two contributions (you can see these contributions in Figure 1): 1) x no longer contributes anything to the potential, as it no longer has any elements in front of it, so, we have a decrease of the potential by $\phi_{T_\alpha}(x)$; 2) all $p_{T_\alpha}(x) - 1$ elements that were in front of x now have an additional element in front of them, x itself, and thus their potential might be increased by one. The total number of these

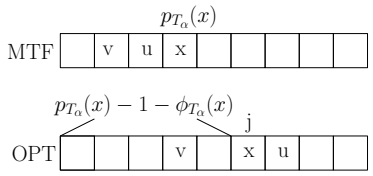

Fig. 1: Illustration for Lemmas 1 and 2. u increases $\phi_{T_\alpha}(x)$ by 1 and v does not affect $\phi_{T_\alpha}(x)$ (the number of nodes like v is at most $(p_{T_\alpha}(x) - 1 - \phi_{T_\alpha}(x))$). After we move to front $\phi_{T_\alpha+1}(x) = 0$, x starts increasing $\phi_{T_\alpha+1}(v)$ and does not affect node u.

elements is $p_{T_\alpha}(x) - 1 - \phi_{T_\alpha}(x)$, because all nodes, which did not affect on $\phi_{T_\alpha}(x)$ before, are affected now. So, a total change of MTF potential of x is at most $-\phi_{T_\alpha}(x) + (p_{T_\alpha}(x) - 1 - \phi_{T_\alpha}(x)) = p_{T_\alpha}(x) - 1 - 2 \cdot \phi_{T_\alpha}(x)$. Moreover we have to pay $(\Delta + 1) \cdot \alpha$ for the network adjustment where $\Delta = 2$.

By that, we measure how the potential changes after the move-to-front adjustment. The amortized cost of the α-th operation is equal to: $p_{T_\alpha}(x) + \Delta\Phi + 3\alpha = p_{T_\alpha}(x) + \alpha \cdot (p_{T_\alpha}(x) - 1 - 2 \cdot \phi_{T_\alpha}(x)) + 3\alpha$.

Then the total cost of all α operations in a block equals:

$$\begin{aligned}
&\sum_{i=0}^{\alpha-1} p_{T_i}(x) + p_{T_\alpha}(x) + \Delta\Phi + 3 \cdot \alpha = \\
&= p_{T_0}(x) + p_{T_1}(x) + ... + p_{T_\alpha}(x) - \\
&- \alpha \cdot \phi_{T_\alpha}(x) + \alpha \cdot (p_{T_\alpha}(x) - 1 - \phi_{T_\alpha}(x)) + 3 \cdot \alpha = \\
&= (p_{T_0}(x) - \phi_{T_\alpha}(x)) + (p_{T_1}(x) - \phi_{T_\alpha}(x)) + ... + \\
&+ (p_{T_\alpha}(x) - \phi_{T_\alpha}(x)) + \alpha \cdot (p_{T_\alpha}(x) - \phi_{T_\alpha}(x) - 1) + 3 \cdot \alpha
\end{aligned} \tag{1}$$

At first, we know that $p_{T_0}(x) \leq p_{T_1}(x) \leq ... \leq p_{T_\alpha}(x)$ since the node can only be moved further in the list due to other move-to-front nodes. Secondly, from Figure 1, we can see that $(p_{T_\alpha}(x) - 1 - \phi_{T_\alpha}(x)) \leq j - 1$, so $(p_{T_\alpha}(x) - \phi_{T_\alpha}(x)) \leq j$. Recall that j is the position of x in OPT. By these two statements we can conclude that $(p_{T_i}(x) - \phi_{T_\alpha}(x)) \leq (p_{T_\alpha}(x) - \phi_{T_\alpha}(x)) \leq j$. By making a substitution into equation (1) we get

$$(1) \leq j + j + ... + j + \alpha \cdot (j-1) + 3 \cdot \alpha \leq \alpha \cdot (2 \cdot j + 2) < 4 \cdot \alpha \cdot j.$$

So, the total cost is strictly less than four times the cost of OPT list, which concludes (ii).

In the statement of the theorem we have a restriction on the number of requests σ. If $|\sigma| \geq \alpha \cdot \frac{n(n+1)}{2}$ we do not have to think about requests in the "tail" of σ (i.e., the requests which belong to a non-finished block, for which a MTF operation does not occur by the end of the sequence). It is due to the fact that $\mathrm{sumCost}(\text{Lazy MTF}, G_0, \sigma) = \omega(\alpha n^2)$ and all requests from the tail, which are not going to be amortized by move-to-front, can be amortized over all $|\sigma|$ requests: at each node we can set counter to $\alpha - 1$ and the potential is $\mathcal{O}(\alpha n^2)$. □

V. Lazy Tree Networks

We now turn to apply our lazy topology reconfiguration method in tree networks. We first expose the design principles of our method (Section V-A) and then apply it to splay trees [17] and SplayNet [5]. In Sections V-B and V-C we present and analyze LazySplayTree and LazySplayNet, respectively. We show that our lazy algorithms in the Matching Model (MM) incur a cost of $\mathcal{O}(\min(\sqrt{\alpha}, \log n))$ times the cost of the original algorithms in the Standard Model (SM), which we show to be tight.

A. Lazy topology reconfiguration

Consider a self-adjusting algorithm ALG over a graph (which can be a search data structure or a network topology) in the SM, which we want to adapt in the MM. We will denote the adapted version of ALG in MM by $LazyALG$. If we simply run ALG in the MM ($LazyALG = ALG$), then we get that $cost_{MM}(LazyALG, G_0, \sigma) = \alpha \cdot cost_{SM}(ALG, G_0, \sigma)$, where G_0 is the initial graph, σ is a sequence of (search or routing) requests, and $cost_X(A, G_0, \sigma)$ is the cost of algorithm A in model $X \in \{SM, MM\}$ with initial topology G_0 and sequence σ. To improve the factor of α, we simply perform adjustments less often, by introducing our lazy topology reconfiguration method.

We design $LazyALG$, given ALG, as follows. Let us divide the list of requests σ into *epochs*. During one epoch the graph maintained by $LazyALG$ remains unmodified and the graph maintained by *ALG* adjusts exactly as in the SM. An epoch continues until the total cost of operations in $LazyALG$ exceeds α. After that $LazyALG$ synchronizes (copies) its graph with the graph maintained by ALG, resets the epoch cost counter to zero, and moves to a new epoch.

In SANs, $LazyALG$ adjusts the physical network topology, while ALG is a local computation running at the network coordinator, emulating the network. In our context, we are interested in the cost of routing and network reconfiguration, thus local computations as the ones done by the coordinator running *ALG* are ignored in the cost calculation.

We aim to calculate the ratio $\frac{\mathrm{sumCost}_{MM}(LazyALG, G_0, \sigma)}{\mathrm{sumCost}_{MM}(\textit{static}, G_{static}, \sigma)}$, where *static* is the statically optimal algorithm, i.e. the algorithm that has the perfect knowledge of σ, but can only compute a static graph and perform no adjustments (the cost notation does not require G_0 in this case). This ratio measures how close $LazyALG$ is to static optimality; in case the ratio is a constant $LazyALG$ is statically optimal. Let us multiply and divide this cost ratio by $\mathrm{sumCost}_{SM}(ALG, G_0, \sigma)$. By that we obtain: $\frac{\mathrm{sumCost}_{MM}(LazyALG, G_0, \sigma)}{\mathrm{sumCost}_{SM}(ALG, G_0, \sigma)} \cdot \frac{\mathrm{sumCost}_{SM}(ALG, G_0, \sigma)}{\mathrm{sumCost}_{MM}(\textit{static}, G_{static}, \sigma)}$

We know that $\frac{\mathrm{sumCost}_{SM}(ALG, G_0, \sigma)}{\mathrm{sumCost}_{SM}(\textit{static}, G_{static}, \sigma)}$ is equal to some value c_{ALG}, if ALG is statically optimal in the SM, and also that $\mathrm{sumCost}_{MM}(\textit{static}, \sigma) = \mathrm{sumCost}_{SM}(\textit{static}, G_{static}, \sigma)$, since *static* outputs a fixed graph. Thus, the cost ratio equals $\frac{\mathrm{sumCost}_{MM}(LazyALG, G_0, \sigma)}{\mathrm{sumCost}_{SM}(ALG, G_0, \sigma)} \cdot c_{ALG}$. Recall that $c_{SplayTree} = \mathcal{O}(1)$ as splay trees are statically optimal, but we are not aware of c_{SplayNet}.

Let us split now the numerator and the denominator of the ratio (without c_{ALG}) into epochs. Let i be the index of an epoch and m be the number of epochs. Suppose that

G_i is the graph right after the i-th epoch and $\sigma^{(i)}$ be the requests performed during i-th epoch. By using the inequality $\frac{a_1+a_2+\ldots+a_m}{b_1+b_2+\ldots+b_m} \leq \frac{c\cdot b_1+c\cdot b_2+\ldots+c\cdot b_m}{b_1+b_2+\ldots+b_m} = c$, where $c = \max\limits_{i=1\ldots m} \frac{a_i}{b_i}$, we get that:

$$\frac{\text{sumCost}_{MM}(LazyALG, G_0, \sigma)}{\text{sumCost}_{SM}(ALG, G_0, \sigma)} = \frac{\sum\limits_{i=1}^{m} \text{sumCost}_{MM}(LazyALG, G_{i-1}, \sigma^{(i)})}{\sum\limits_{i=1}^{m} \text{sumCost}_{SM}(ALG, G_{i-1}, \sigma^{(i)})} \leq \max_{i=1\ldots m} \frac{\text{sumCost}_{MM}(LazyALG, G_{i-1}, \sigma^{(i)})}{\text{sumCost}_{SM}(ALG, G_{i-1}, \sigma^{(i)})}$$

Thus, we focus on finding a lower bound for $\text{sumCost}_{SM}(ALG,\ G_{i-1}, \sigma^{(i)})$ and an upper bound for $\text{sumCost}_{MM}(LazyALG, G_{i-1},\ \sigma^{(i)})$ for each epoch. In the following, we consider and bound only the ratios of the epochs, not the whole execution.

B. Search requests

We start with LAZYSPLAYTREE, which is the outcome of applying our lazy topology reconfiguration method to the splay tree algorithm. LAZYSPLAYTREE achieves a $\mathcal{O}(\min(\sqrt{\alpha}, \log n))$-ratio with respect to splay tree in the SM. If α is regarded as a constant, then LAZYSPLAYTREE is statically optimal in the MM.

Lemma 3. *The* LAZYSPLAYTREE *algorithm is a $\mathcal{O}(\sqrt{\alpha})$-statically optimal algorithm.*

Proof. At first, we introduce all the necessary notions. Let T_{ST} and T_{LST} be the trees maintained by the splay tree (ST) and LAZYSPLAYTREE (LST) algorithms, respectively. Let us divide the requests into epochs $\sigma^{(i)}$. Let $\sigma_j^{(i)}$ be the j-th request in the i-th epoch. Let s_{ij}^{ST} and s_{ij}^{LST} be the cost of the corresponding request $\sigma_j^{(i)}$ in T_{ST} and T_{LST} respectively.

Now, we give an intuition (and overview) on how the proof works. We consider two cases. Suppose that each request in T_{LST} takes less than $\sqrt{\alpha}$, thus, we did at least $\sqrt{\alpha}$ requests in total during the epoch. This means that the total cost in ST is at least $\sqrt{\alpha}$ per epoch while the total cost of the operations in the epoch in LST $\approx \alpha$, thus the ratio is approximately $\sqrt{\alpha}$. Now, in the second case at least one operation in LST takes more than $\sqrt{\alpha}$. In ST we also visited the corresponding node by visiting more than $\sqrt{\alpha}$ nodes on the path. This means that, in total, the operations of the epoch in ST spends more than $\sqrt{\alpha}$. But since the total cost of the operations in the epoch in LST is approximately α, the ratio is approximately $\sqrt{\alpha}$. Below we prove the theorem more formally.

By the definition of the epochs, we have for any epoch i. it holds that $\sum_{j=1}^{|\sigma^{(i)}|-1} s_{ij}^{ST} < \alpha$ and $\sum_{j=1}^{|\sigma^{(i)}|} s_{ij}^{ST} \geq \alpha$. At the end of the epoch, i.e., after the $|s_i|$-th request, by our lazy topology reconfiguration method we synchronize the data structures and make T_{LST} to be exactly as T_{ST} structure. Without loss of generality we consider an epoch i. We have two cases.

Case 1. If $s_{ij}^{LST} \leq \sqrt{\alpha}$ for all j then during the epoch we perform more than $\sqrt{\alpha}$ operations: each operation costs less than $\sqrt{\alpha}$ and we end the epoch when the total cost exceeds α. Then $\text{sumCost}_{MM}(LST, G_{i-1}, \sigma^{(i)}) \leq \alpha + \sqrt{\alpha} + 4 \cdot \alpha$ because the cost of the last operation does not exceed $\sqrt{\alpha}$ and we perform an adjustment on $\Delta + 1 = 4$ matchings, and $\text{sumCost}_{SM}(ST, G_{i-1}, \sigma^{(i)}) \geq \sqrt{\alpha}$ since we do more than $\sqrt{\alpha}$ operations of cost at least one. So, our complexity of static optimality cannot exceed $\frac{\alpha+\sqrt{\alpha}+4\cdot\alpha}{\sqrt{\alpha}} = \mathcal{O}(\sqrt{\alpha})$.

Case 2. If there exists j for which $s_{ij}^{LST} \geq \sqrt{\alpha}$ (if there are several of them we take the one with the maximal cost) we should consider two cases: $s_{ij}^{ST} \geq \sqrt{\alpha}$ and $s_{ij}^{ST} < \sqrt{\alpha}$. The second case is possible when before the request j in our epoch we have performed splay operations, which decreased the depth of requested node v. This is the only explanation why the change of the depth happens since at the beginning of the epoch the trees ST and LST were the same.

To begin with, we want to prove that $\sum_{k=1}^{j} s_{ik}^{ST} \geq s_{ij}^{LST}$, i.e., the total cost of the requests on the splay tree from the start of the epoch is greater than one operation in LAZYSPLAYTREE.

Let us name the rotation operations during the splay: zig, zig-zig and zig-zag, as splay-steps. At first, we need to understand why and how the depth of our node can change during the epoch. Consider Figure 2. We can see that one splay-step can change the depth of all nodes in the tree.

Look at all splay-step operations that affected the depth of node v. Let this set be SS, Δd be the difference between the depth of node v in T_{ST} and T_{LST}, Δd_s and c_s be the differences in depth before and after splay operation s and the total cost of finding the route and the adjustment in the current lazy topology reconfiguration method s, i.e., the node's depth.

$\Delta d = \sum_{s\in SS} \Delta d_s \underset{\text{by Figure 2}}{\leq} \sum_{s\in SS} c_s \leq \sum_{k=0}^{j-1} s_{ik}^{ST}$

$d_v^{LST} = \Delta d + d_v^{ST} \leq \sum_{k=1}^{j-1} s_{ik}^{ST} + d_v^{ST} = \sum_{k=1}^{j} s_{ik}^{ST}$

It means that $\sum\limits_{k=1}^{j} s_{ij}^{ST} \geq d_v^{LST} = s_{ij}^{LST} \geq \sqrt{\alpha}$. Let us substitute this into static optimal complexity estimation during the epoch: $\frac{\text{sumCost}_{MM}(LST, G_{i-1}, \sigma^{(i)})}{\text{sumCost}_{SM}(ST, G_{i-1}, \sigma^{(i)})} \leq \frac{\alpha + s_{ij}^{LST} + 4\cdot\alpha}{\sum\limits_{k=0}^{j} s_{ik}^{ST}} = \frac{5\cdot\alpha}{\sum\limits_{k=0}^{j} s_{ik}^{ST}} + \frac{s_{ij}^{LST}}{\sum\limits_{k=0}^{j} s_{ik}^{ST}} \leq 5 \cdot \sqrt{\alpha} + \mathcal{O}(1) = \mathcal{O}(\sqrt{\alpha})$ □

In Lemma 3 we proved that LAZYSPLAYTREE is $\mathcal{O}(\sqrt{\alpha})$-statically optimal. But what should we do, if α is too big (for example, $\Omega(n)$)? We modify our algorithm for $\alpha > \log^2 n$: we use a static balanced tree of height $\log n$ instead of LAZYSPLAYTREE. Then we show that our analysis is tight.

Theorem 3. LAZYSPLAYTREE *is a $\mathcal{O}(\min(\sqrt{\alpha}, \log n))$-statically optimal algorithm in the MM.*

Proof. In Lemma 3 we proved that LAZYSPLAYTREE is $\mathcal{O}(\sqrt{a})$-statically optimal (Lemma 3). And if $\alpha > \log^2 n$ we use the static balanced tree algorithm which gives us $\mathcal{O}(\log n)$-static optimality. □

Theorem 4. *The complexity bound of* LAZYSPLAYTREE *is tight, i.e. can achieve at most $\mathcal{O}(\min(\sqrt{\alpha}, \log n))$-static optimality.*

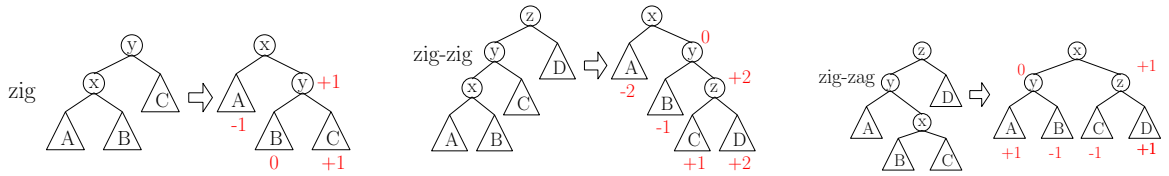

Fig. 2: Change of depth in zig, zig-zig and zig-zag cases. In the zig case cost for adjustment and routing cost are 3 and 1 respectively, the maximum change of depth is 1. In the zig-zig case these costs are 6, 2 and 2. In zig-zag - 4, 2 and 2.

Proof. Since, we provided an algorithm with $\mathcal{O}(\min(\sqrt{\alpha}, \log n))$ static optimality ratio, we want to prove that it is in fact the best ratio for our lazy topology reconfiguration method. For that we need to find a pattern of requests, that will always achieve $\mathcal{O}(\sqrt{\alpha})$ ratio.

The worst case for LAZYSPLAYTREE is when we access one element for the whole epoch. Let us consider the number of operations in tree T_{ST}. Let d be the depth of the accessed node. In LAZYSPLAYTREE we will perform $\lceil \frac{\alpha}{d} \rceil$ requests. In a splay tree the first request costs d, while the next requests cost one since we splayed the node to the root. So, the cost is equal to $C(d) = d + \lceil \frac{\alpha - d}{d} \rceil$.

Note, that simply $C(d) \geq d + \frac{\alpha}{d} - 1$ which by AM-GM (arithmetic mean-geometric mean) inequality exceeds $2 \cdot \sqrt{\alpha} - 1$. The last inequality is satisfied with $d = \sqrt{\alpha}$.

So, consider the following requests. We access a node at depth $\sqrt{\alpha}$. This gives us, that $\text{sumCost}_{SM}(ST, G_{i-1}, \sigma^{(i)}) = C(\sqrt{\alpha}) = \mathcal{O}(\sqrt{\alpha})$ and $\text{sumCost}_{MM}(LST, G_{i-1}, \sigma^{(i)}) \leq 5 \cdot \alpha + \sqrt{\alpha}$. It means, that in the worst case we achieve $\mathcal{O}(\sqrt{\alpha})$-static optimality.

But the tree is not obliged to have a node at depth $\sqrt{\alpha}$. For example, when $a = n^4$ the height of the tree obviously does not exceed $n \leq \sqrt{\alpha} = n^2$. In this case, we are trying to get another bound that depends only on n.

We repeat our pattern to request one node the whole epoch. For that we take a node at level $\log n$. Thus, $\frac{\alpha}{C(d)} \geq \frac{\alpha}{\log n + \frac{\alpha}{\log n}}$ which in turn does not exceed $\frac{\log n}{2}$ since $\sqrt{\alpha} \geq \log n$.

It means that the static optimality ratio lies between $\frac{\log n}{2} < c < \sqrt{\alpha}$. So, the lower bound for the c-static optimality of LAZYSPLAYTREE is $\mathcal{O}(\min(\sqrt{\alpha}, \log n))$. □

Remark 1. *Theorem 3 can be applied to any self-adjusting tree. The proof is split into two parts. In the first part if all the depth of the requested nodes are less than $\sqrt{\alpha}$ we have nothing to change. However, in the second case we have to prove the fact that if we access a node of the tree that is on depth D at the start of the epoch, the original tree (not the lazy one) has to make at least D operations during the epoch before the access. However, it is obviously true, since our structure is a tree and we have to "visit" all the nodes that are above the target node.*

C. Routing requests

In the previous subsection, we applied our lazy topology reconfiguration method to the splay tree algorithm. Here, we show how to extend it to SplayNet algorithm and obtain a SAN for the MM. The key difference in our analysis is that we consider the distance of the route between two nodes, instead of the depth from the root, as we did in the previous section.

Lemma 4. *For any starting tree G_0 and any list of requests σ,* $\text{sumCost}(\text{LAZYSPLAYNET}, G_0, \sigma) = O(\sqrt{\alpha} \cdot \text{sumCost}(\textit{SplayNet}, G_0, \sigma))$.

Proof. The proof is similar to the proof of Lemma 3 with one change: the cost of the request becomes the length of the path between two nodes.

We briefly overview the proof. We recommend to become familiar with the proof of Lemma 3 since it follows exactly the same steps. We consider two cases:

a) Case 1: All requests in LAZYSPLAYNET cost not more than $\sqrt{\alpha}$. It means that we will do more than $\sqrt{\alpha}$ requests in SplayNet. So, our complexity of optimality cannot be more than $\frac{\alpha + \sqrt{\alpha} + 4 \cdot \alpha}{\sqrt{\alpha}} = \mathcal{O}(\sqrt{\alpha})$.

b) Case 2: We get a request that costs more than $\sqrt{\alpha}$ in LAZYSPLAYNET and, again, we want to prove that the sum of costs of the requests in SplayNet exceeds $\sqrt{\alpha}$. At first, we say that in one lazy topology reconfiguration method the maximum difference of lengths between nodes in a tree cannot not be more than the cost to perform the lazy topology reconfiguration method. However, in each lazy topology reconfiguration method we change no more than three links.

Then, let us sum all the costs of operations that affect the distance between the requested nodes. We get $\sum_{k=1}^{j} s_{ik}^{SN} \geq l_{st}^{LSN} = s_{ij}^{LST} \geq \sqrt{\alpha}$ and we use this property next: $\frac{\text{sumCost}_{MM}(LSN, G_{i-1}, \sigma^{(i)})}{\text{sumCost}_{SM}(SN, G_{i-1}, \sigma^{(i)})} \leq \frac{\alpha + s_{ij}^{LSN} + 4 \cdot \alpha}{\sum_{k=0}^{j} s_{ik}^{SN}} = \frac{5 \cdot \alpha}{\sum_{k=0}^{j} s_{ik}^{SN}} + \frac{s_{ij}^{LSN}}{\sum_{k=0}^{j} s_{ik}^{SN}} \leq 5 \cdot \sqrt{\alpha} + \mathcal{O}(1) = \mathcal{O}(\sqrt{\alpha})$. □

In Lemma V-C we proved that LAZYSPLAYNET is $\mathcal{O}(\sqrt{\alpha})$-statically optimal. But what should we do, if our α is too large (for example, $\Theta(n^4)$)? We modify our algorithm when $\alpha > \log^2 n$: we use the static balanced tree of height $\log n$ instead of Self-Adjusting algorithm.

Theorem 5. *For any starting tree G_0 and any list of requests σ,* $\text{sumCost}(\text{LAZYSPLAYNET}, G_0, \sigma) = \mathcal{O}(\min(\sqrt{\alpha}, \log n) \cdot \text{sumCost}(\textit{SplayNet}, G_0, \sigma))$

Algorithm 1: LAZYRENET

Input: A set of nodes & σ, a sequence of m comm. requests
Output: $G^T_{\text{LAZYRENET}}$, physical topologies for $T = 1, \ldots, m$ and *totalCost* the cost of serving σ
Macros:
G^T_{RENET}: RENET topology at time T (cached)
$G^T_{\text{LAZYRENET}}$: LAZYRENET topology at time T (physical)

$G^0_{\text{LAZYRENET}} \leftarrow G^0_{\text{RENET}}$; /* initial topology */
epochCost $\leftarrow 0$;
totalCost $\leftarrow 0$;
for *time* $T = 1, 2, \ldots, m$ **do**
  serve request $\sigma_T = (s_T, d_T)$ in $G^{T-1}_{\text{LAZYRENET}}$;
  $G^T_{\text{RENET}} \leftarrow \text{RENET}(G^{T-1}_{\text{RENET}}, \sigma_T)$; /* new cached */
  totalCost $\leftarrow$ *totalCost* $+ routingCost(G^{T-1}_{\text{LAZYRENET}}, \sigma_T)$;
  epochCost $\leftarrow$ *epochCost* $+ routingCost(G^{T-1}_{\text{LAZYRENET}}, \sigma_T)$;
  if *epochCost* $\geq \alpha$ **then**
    /* sync physical to cached */
    $G^T_{\text{LAZYRENET}} \leftarrow G^T_{\text{RENET}}$;
    epochCost $\leftarrow 0$;
    totalCost $\leftarrow$ *totalCost* $+ \alpha$;
  else
    $G^T_{\text{LAZYRENET}} \leftarrow G^{T-1}_{\text{LAZYRENET}}$; /* not adjust */

Proof. In Lemma 3 we proved that LAZYSPLAYNET is $\mathcal{O}(\sqrt{a})$-statically optimal (Lemma 3). And if $\alpha > \log^2 n$ static balanced tree algorithm is $\mathcal{O}(\log n)$-statically optimal. □

To prove the lower bound for the lazy algorithm we do exactly as the proof of Theorem 4: for each epoch, we ask for a pair with distance $\sqrt{\alpha}$ before the epoch and then repeat this request $\sqrt{\alpha} - 1$ times. Thus, we obtain the following.

Theorem 6. LAZYSPLAYNET *cannot achieve better complexity than* $\mathcal{O}(\min(\sqrt{\alpha}, \log n))$*-static optimality.*

VI. BOUNDED DEGREE NETWORKS IN MM

We now turn to study LAZYRENET in the Matching Model (MM), which is the product of applying lazy topology adjustment to RENET [6]. As in Section V, we show that the LAZYRENET complexity is asymptotically bounded by $\sqrt{\alpha}$ times the complexity of RENET.

a) RENETs recap: A RENET [6] is designed as a union of *ego* (i.e. individual) views of each node. The ego view of a node is either a star centered at a node and connected to recently communicated nodes (if they are less than the degree bound Δ) or a splay tree including these nodes, otherwise. We give an overview of RENETs below.

A RENET is a SAN with node degree bounded by Δ that we can define as $G_t = (V, E_{coord} \cup E_t)$. The subgraph (V, E_{coord}) is used for contacting the network coordinator C and is static throughout the algorithm's execution (we assume it has diameter c). The subgraph (V, E_t) is the dynamic part of the network and is subject to change at any time t.

Initially, E_0 is empty. Upon a request $\sigma_T = (s_T, d_T)$, $T \in \{1, \ldots, m\}$, if a route does not exist, s_T asks C to add a route. If both s_T and d_T are *small* nodes, i.e. if they have less than Δ edges, then C adds a direct edge between s_T and d_T. A node u becomes *large* when its degree becomes equal to Δ. In that instant, the coordinator deletes all direct links of u, creates a splay-tree (ego-tree) including all of u's former neighbors, and connects u to the splay-tree root. Communication from u to any node v in the ego-tree of u is done by following the route dictated by binary search and it is followed by splaying v to the root of the ego-tree. If a small node v is a part of an ego-tree, e.g. of u, when it becomes large, we pick a small *hepler* node, add it in both ego-trees of u and v and use it as a relay in when u and v communicate. If E_T becomes full (e.g. when there are no small nodes to pick or when $|E_T|$ reaches a threshold), the coordinator deletes all nodes in E_T (reset).

b) Complexity proof: If α, the reconfiguration cost in MM, is greater than $\log^2 n$, then for every epoch T we use the static arbitrary binary search tree, G_T, which gives us $\mathcal{O}(\log n)$ complexity of static-optimality. Thus, for the rest of the section we consider α to be less than $\log^2 n$. We summarize LAZYRENET in Algorithm 1. Equivalently to LAZYSPLAYNET, we maintain RENET logically in the coordinator and LAZYRENET as the physical network. The proof of Theorem 7 bases on the static nature of RENET between resets and reduces to similar arguments to the proof of Theorem 3, as splay trees are building blocks of RENET.

Theorem 7. *For every initial graph* G_0 *and list of communication requests* σ*,* $\text{sumCost}(\text{LAZYRENET}, G_0, \sigma) = \mathcal{O}(\sqrt{\alpha} \cdot \text{sumCost}(\text{RENET}, G_0, \sigma))$.

VII. EXPERIMENTAL EVALUATION

In this section, we show how different values of α affect performance, i.e., the total number of steps, of our algorithms in order to serve communication requests. We compare our algorithms to the ones in the standard model (SM): LAZYSPLAYNET to SPLAYNET, and LAZYRENET to RENET. Our empirical results confirm our theoretical results, but also show that in practice, the performance can often be better.

Setup and data: The code for the algorithms was written in Java and Python and we have provided public access in the following repositories: RENET [23], LAZYRENET [24], SPLAYNET and LAZYSPLAYNET [25].

For our experiments, we took requests sampled uniformly at random from four datasets in [4]: Facebook datacenter workload [26], a high performance computing (HPC) workload [27], a workload on ProjectToR [3], and a synthetic pFabric (pFab) [28] workload. For each scenario with a fixed number of k nodes, we processed each raw sample of requests by (i) selecting the first k nodes that appear in the raw sample, (ii) deleting all requests that contain the remaining nodes, and (iii) keeping as many requests as the target size of the request sequence (10^6 in Section VII-A and 10^4 in Section VII-B).

A. LAZYSPLAYNET *network*

We run LAZYSPLAYNET and SPLAYNET on four datasets. We restrict all datasets to 10^6 requests on: Facebook with 10^4 nodes, HPC with 500 nodes, ProjectToR with 121 nodes, and pFab with 144 nodes (the latter three datasets are small).

Figure 3 shows for each workload how α affects the ratio of the cost of LAZYSPLAYNET and the cost of SPLAYNET

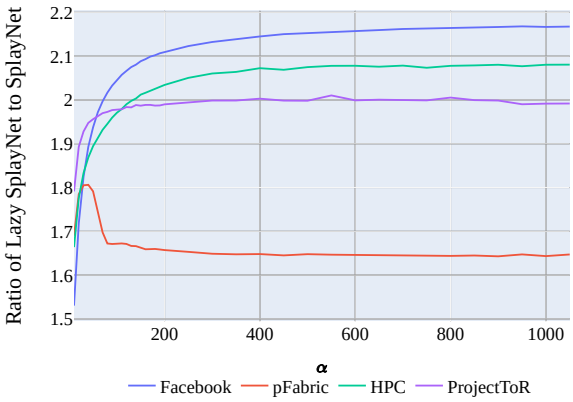

Fig. 3: Total cost of LAZYSPLAYNET in comparison to standard SPLAYNET in SM.

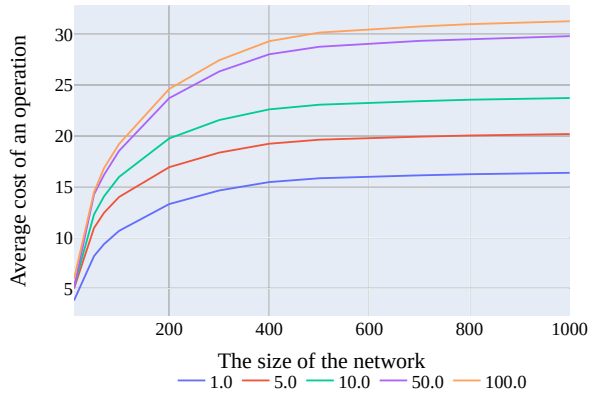

Fig. 4: Total cost of operations per different sizes of the network and α on Facebook dataset.

in the SM. For all workloads except for pFab, the plot seems to reflect the square root of α which was predicted by our Theorem 2. The results for pFab are a little bit different. This happens due to the fact that the requests fit very well on the tree and, thus, almost all requests cost $\mathcal{O}(1)$. Therefore, there are $\mathcal{O}(\alpha)$ requests between the synchronization and this synchronization can be amortized over the previous requests.

In Figure 4 we show how the average cost of an operation of LAZYSPLAYNET depends on the size of the network on the Facebook dataset, for different values of α. We chose this real-world dataset, because it is large and representative of datacenter traffic. As expected the plot looks similar to a logarithmic function.

B. LAZYRENET *network*

We run LAZYRENET and RENET on four datasets. Due to the high computational complexity of the RENET algorithm we were only able to run it on smaller datasets than in the previous subsection. We restrict them to 10^4 requests on: Facebook with 100 nodes, HPC with 20 nodes, ProjectToR with 88 nodes, and pFab with 96 nodes (node quantities were reduced due to the longer running time of RENET).

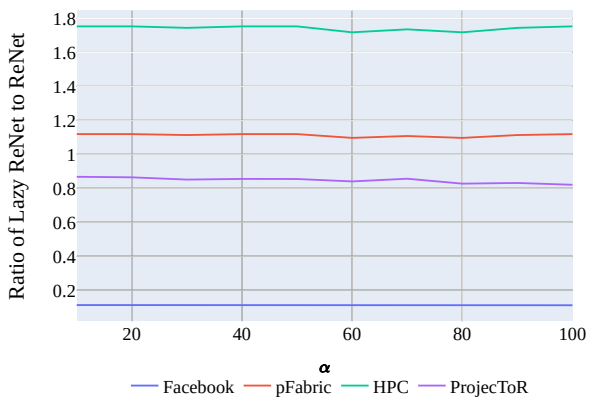

Fig. 5: Total cost of LAZYRENET in comparison to standard RENET in SM.

Figure 5 shows for each workload how α affects the ratio of the cost of LAZYRENET and the cost of RENET in SM. By our Theorem 7, we know that the ratio of LAZYRENET and RENET does not exceed $\mathcal{O}(\sqrt{\alpha})$. However, in practice, as can be seen on the plot, we get almost constant ratio which differs from SPLAYNET. This difference can be explained by the significantly increased flexibility that RENETs provide in contrast to the tree topology of SPLAYNET, which allows the network to better fit to the demand. The difference in the performance of each dataset is partly explained by the different degree of locality in these datasets [4, Fig. 2]. Specifically, the Facebook dataset has the highest temporal locality and the best performance. As part of future work, we aim to further investigate this behaviour in a larger scale.

VIII. CONCLUSION AND FUTURE WORK

We initiated the study of online algorithms for reconfigurable networks in the more realistic matching model. In particular, we presented a lazy topology reconfiguration method for designing self-adjusting networks in the Matching Model (MM), using reference algorithms in the Standard (uniform-cost) Model (SM). We showed the we can use a lazy version of the move-to-front rule in line topologies to achieve static optimality. We further presented LAZYSPLAYNET and LAZYRENET, which are obtained by applying our lazy topology reconfiguration method to SPLAYNET and RENET. We showed that the cost of LAZYSPLAYNET and LAZYRENET in MM is asymptotically bounded by $\mathcal{O}(\min(\sqrt{\alpha}, \log n))$ times the cost of SPLAYNET and RENET in the SM. Our theoretical results are accompanied by experiments on real-world data from datacenter network traces, confirming that the theoretical bounds are respected, while LAZYRENET performed better in practice. We believe that our work opens new opportunities to further improve the $\sqrt{\alpha}$ factor and design future self-adjusting networks in the more realistic Matching Model. The authors have provided public access to their code and data (see Section VII).

REFERENCES

[1] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*. ACM, 2017, pp. 267–280. [Online]. Available: https://doi.org/10.1145/3098822.3098838

[2] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, H. Liu, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising: a decade of clos topologies and centralized control in google's datacenter network," *Commun. ACM*, vol. 59, no. 9, pp. 88–97, 2016. [Online]. Available: https://doi.org/10.1145/2975159

[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 Conference*, 2016, pp. 216–229.

[4] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 1, pp. 1–29, 2020.

[5] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker, "Splaynet: Towards locally self-adjusting networks," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1421–1433, 2015.

[6] C. Avin and S. Schmid, "Renets: Statically-optimal demand-aware networks," in *2nd Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Virtual Conference, January 13, 2021*, M. Schapira, Ed. SIAM, 2021, pp. 25–39. [Online]. Available: https://doi.org/10.1137/1.9781611976489.3

[7] C. Avin, C. Griner, I. Salem, and S. Schmid, "An online matching model for self-adjusting tor-to-tor networks," *CoRR*, vol. abs/2006.11148, 2020. [Online]. Available: https://arxiv.org/abs/2006.11148

[8] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, R. Bhagwan and G. Porter, Eds. USENIX Association, 2020, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/mellette

[9] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen, and H. Williams, "Sirius: A flat datacenter network with nanosecond optical switching," in *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*, H. Schulzrinne and V. Misra, Eds. ACM, 2020, pp. 782–797. [Online]. Available: https://doi.org/10.1145/3387514.3406221

[10] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.

[11] C. Avin and S. Schmid, "Toward demand-aware networking: A theory for self-adjusting networks," *ACM SIGCOMM Computer Communication Review*, vol. 48, no. 5, pp. 31–40, 2019.

[12] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network designs of bounded degree," *Distributed Computing*, pp. 1–15, 2019.

[13] C. Avin, I. van Duijn, and S. Schmid, "Self-adjusting linear networks," in *Stabilization, Safety, and Security of Distributed Systems - 21st International Symposium, SSS 2019, Pisa, Italy, October 22-25, 2019, Proceedings*, ser. Lecture Notes in Computer Science, M. Ghaffari, M. Nesterenko, S. Tixeuil, S. Tucci, and Y. Yamauchi, Eds., vol. 11914. Springer, 2019, pp. 368–382. [Online]. Available: https://doi.org/10.1007/978-3-030-34992-9_29

[14] U. Feige and J. R. Lee, "An improved approximation ratio for the minimum linear arrangement problem," *Information Processing Letters*, vol. 101, no. 1, pp. 26–29, 2007.

[15] S. Albers and M. Janke, "New bounds for randomized list update in the paid exchange model," in *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[16] S. Albers and J. Westbrook, "Self-organizing data structures," in *Online algorithms*. Springer, 1998, pp. 13–51.

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

[18] B. Peres, A. d. O. Otavio, O. Goussevskaia, C. Avin, and S. Schmid, "Distributed self-adjusting tree networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 145–153.

[19] C. Avin, I. Salem, and S. Schmid, "Working set theorems for routing in self-adjusting skip list networks," in *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 2020, pp. 2175–2184. [Online]. Available: https://doi.org/10.1109/INFOCOM41043.2020.9155495

[20] C. Avin, K. Mondal, and S. Schmid, "Dynamically optimal self-adjusting single-source tree networks," in *Latin American Symposium on Theoretical Informatics*. Springer, 2020, pp. 143–154.

[21] V. G. Vizing, "On an estimate of the chromatic class of a p-graph," *Discret Analiz*, vol. 3, pp. 25–30, 1964.

[22] N. Reingold, J. Westbrook, and D. D. Sleator, "Randomized competitive algorithms for the list update problem," *Algorithmica*, vol. 11, no. 1, pp. 15–32, 1994.

[23] https://github.com/punit-iitd/ReNets/tree/master/ReNet_new/src.

[24] https://gitlab.com/robertsLab/lazyrenets.

[25] https://github.com/evgeniyfeder/diploma-benchs.

[26] 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 Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[27] U. DOE, "Characterization of the doe mini-apps," https://portal.nersc.gov/project/CAL/doe-miniapps.htm, 2016.

[28] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.