Locally Self-Adjusting Tree Networks

Chen Avin[1], Bernhard Haeupler[2], Zvi Lotker[1], Christian Scheideler[3], Stefan Schmid[4]
[1] Ben Gurion University, Israel; {avin,zvilo}@cse.bgu.ac.il
[2] Massachusetts Institute of Technology (MIT), USA; hauepler@mit.edu
[3] University of Paderborn, Germany; scheideler@upd.de
[4] TU Berlin & Telekom Innovation Laboratories, Germany; stefan@net.t-labs.tu-berlin.de

***Abstract*—This paper initiates the study of self-adjusting networks (or distributed data structures) whose topologies dynamically adapt to a communication pattern σ. We present a fully decentralized self-adjusting solution called *SplayNet*. A *SplayNet* is a distributed generalization of the classic splay tree concept. It ensures short paths (which can be found using local-greedy routing) between communication partners while minimizing topological rearrangements. We derive an upper bound for the amortized communication cost of a *SplayNet* based on empirical entropies of σ, and show that *SplayNets* have several interesting convergence properties. For instance, *SplayNets* features a provable online optimality under special requests scenarios. We also investigate the optimal static network and prove different lower bounds for the average communication cost based on graph cuts and on the empirical entropy of the communication pattern σ. From these lower bounds it follows, e.g., that *SplayNets* are optimal in scenarios where the requests follow a product distribution as well. Finally, this paper shows that in contrast to the Minimum Linear Arrangement problem which is generally NP-hard, the optimal static tree network can be computed in polynomial time for any guest graph, despite the exponentially large graph family. We complement our formal analysis with a small simulation study on a Facebook graph.**

I. Introduction

In the 80s, Sleator and Tarjan [25] introduced an appealing new paradigm to design efficient data structures: Rather than optimizing traditional metrics such as the search tree depth in the *worst-case*, the authors proposed to make data structures *self-adjusting* and considered the *amortized cost* as the performance metric—the "average cost" per operation for a given sequence s of lookups. The authors described *splay trees*, self-adjusting binary search trees where frequently accessed elements are moved closer to the root, improving the average access times *weighted by the elements' popularity*. The popularity distribution need not be known in advance and may even change over time.

Inspired by these ideas, this paper initiates the study of a *distributed generalization* of splay trees towards networks. We consider a distributed data structure, e.g., a structured peer-to-peer (p2p) system or Distributed Hash Table (DHT), where nodes (the "peers") communicating more frequently should become topologically closer to each other (i.e., the routing distance is reduced). This contrasts with most of today's structured peer-to-peer overlays whose topology is often optimized in terms of static global properties only, such as the node degree or the longest shortest routing path.

This paper focuses on a most fundamental network, a distributed binary search tree (BST) network. Such networks are a natural first extension of classic data structures. Moreover, they facilitate simple and local routing. Generally, tree structures constitute a fundamental building block of more complex networks such as skip graphs [3], and many network protocols also rely on spanning trees or cycle-free backbones. Saying this, we are not offering binary search trees as a "new" network topology, but see our work as a first step towards a better understanding of the design and limitations of self-adjusting networks whose structure needs to be adapted dynamically over time, and must be maintained in a distributed manner.

Contribution and Organization. We describe a fully distributed self-adjusting tree network called SplayNet (Section IV-B). Unlike in classical splay tree where requests (i.e., lookups) always originate from the same root, in *SplayNets* communication happens between arbitrary node pairs in the network. Our goal is to minimize the average routing distance between nodes in the network, and hence frequent communication partners are moved closer together. This may not only reduce communication delays, but also the amount of bandwidth or energy used in the system. In addition to the self-adjusting features, we still want to ensure desirable static properties (e.g., degree or routability).

We derive an upper bound for the amortized performance of *SplayNets* (Section IV-C) in scenarios where the communication patterns can change arbitrarily over time, and we show different locality and convergence properties of *SplayNets* (Section IV-D). Concretely, we prove that the average communication cost is bounded by the sum of the empirical entropies of sources and destinations in the request sequence. Moreover, we prove

that if the communication pattern σ describes a *multicast tree* or a *special matching*, then *SplayNets* are online optimal.

We also shed light on the static (offline) version of our problem, which can be regarded as a variant of the classic *Minimum Linear Arrangement (MLA)* problem [8]. In the static case the sequence of requests is known beforehand, but a single, immutable best tree to serve the requests has to be chosen for the entire sequence. We derive several lower bounds for the static problem which are based on the empirical entropies of the communication pattern σ (Sections IV-C and V). In particular, we can show that if the request pattern follows a product distribution, *SplayNets* have the static optimality property (i.e., the amortized cost of the *SplayNet* is of the same order as of the optimal static tree). We will also see that unlike the MLA problem which is generally NP-hard, the optimal static tree network can be computed in polynomial time *for any guest graph* (Section IV-A). We are not aware of any other arrangement problem for an exponential family of graphs for which a polynomial-time solution exists.

Our theoretical study is complemented with a brief discussion of simulation results on the Facebook online social network (Section VI).

II. Related Work

Self-adjusting networks have many applications, ranging from self-optimizing peer-to-peer topologies over green computing (e.g., due to reduced energy consumption) [9] to adaptive virtual machine migrations [2], [23], microprocessor memory architectures [16], and grids [4]. Other self-adjusting routing scheme were considered, e.g., in scale-free networks to overcome congestion [26].

Peer-to-peer networks are particularly interesting dynamic systems as they are very transient and members continuously join and leave. In this sense, a peer-to-peer system can never be fully "repaired" but must always be fully functional. Today, peer-to-peer networking is a relatively mature field of research, and there are many solutions to maintain desirable network properties under both randomized [22] as well as worst case [14] membership changes, and some peer-to-peer networks are even self-stabilizing [12] in the sense that they quickly converge to a desirable topology (e.g., a hypercube) from an arbitrary connected structure. However, none of these systems are self-adjusting to the demand.

Our work builds upon classic literature on self-adapting *data structures*, in particular upon the seminal work of Sleator and Tarjan on splay trees [25]: Splay trees are optimized binary search trees which move more popular items closer to the root in order to reduce the average access time. Splay trees and its variants (e.g., *Tango trees* [5] or *multi-splay trees* [28]) have been studied intensively for many years (e.g. [1], [25]), and the famous *dynamic optimality conjecture* continues to puzzle researchers: The conjecture claims that splay trees perform as well as any other binary search tree algorithm. [5], [28]

In contrast to the classical splay tree data structures, our paper studies a *distributed variant* where "lookups" or requests cannot only originate from a single root, but where communication happens between all pairs of nodes in the network. Hence, we are in the realm of *distributed data structures* or *networking*.

An intensively studied field related to the static variant of our problem is the *Minimum Linear Arrangement* (MLA) problem [6], originally introduced by Harper [8] to design error-correcting codes with minimal average absolute errors on certain classes of graphs and later used in many other domains such as for modeling of some nervous activity in the cortex [18] or job scheduling [19]. From our perspective, MLA can be seen as an early form of a "demand-optimized" embedding on the *line* (rather than the tree as in our case): Given a set of communication pairs, the goal is to flexibly arrange the nodes on the line network such that the average communication distance is minimized. While there exist many interesting algorithms for this problem already, e.g., with sublogarithmic approximation ratios [7] or polynomial-time executions for special guest graphs [6], no non-trivial results are known about distributed and local solutions.

More generally, we currently witness a renaissance of embedding problems in the context of *network virtualization* [21]. Also, many variants of tree embeddings arise in the context of *phylogenetic trees* ("trees of life") [20].

We are only aware of one paper on self-optimizing overlay networks: Leitao et al. [15] study an overlay supporting gossip or epidemics on a dynamic topology. In contrast to our work, their focus is on unstructured networks (e.g., lookup or routing is not supported), and there is no formal performance guarantee.

III. Model and Preliminaries

Given an arbitrary and unknown pattern of communication (or routing) requests σ between a set of nodes $V = \{1, \ldots, n\}$, this paper attends to the general problem of finding good *communication networks* G out of a family of *allowed networks* $\mathcal{G}$. Each topology $G \in \mathcal{G}$ is a graph $G = (V, E)$, and we define a set of *local transformations* on graphs in $\mathcal{G}$ to transform one member $G' \in \mathcal{G}$ to another member $G'' \in \mathcal{G}$. We seek to adapt our topologies smoothly over time, i.e., a changing

communication pattern leads to "local" changes of the communication graph over time.

This paper focuses on the special case where $\mathcal{G}$ is the set of *binary search trees*, henceforth simply called *tree networks*. Besides their simplicity, such networks are attractive for their low node degree and the possibility to route locally: given a destination identifier (or address), each node can decide locally whether to forward the packet to its left child, its right child, or its parent (see Appendix A). The local transformations of tree networks are called *rotations*. Rotations are the minimal and local transformations that preserve the binary search property: informally, a rotation in a sorted binary search tree changes the local order of three connected nodes, while keeping subtrees intact. Note that it is possible to transform any binary search tree into any other binary search tree by a sequence of local transformations (e.g., by induction over the subtree roots).

We consider a simplified synchronous model where first a communication request arrives, then local network transformations can be performed, the request is satisfied (i.e., the traffic routed), and finally further network transformations are performed. Let $\sigma = (\sigma_0, \sigma_1 \dots \sigma_{m-1})$ be a sequence of m *communication requests* where $\sigma_t = (u, v) \in V \times V$ denotes that a packet needs to be sent from a *source* u to a *destination* v. The source and destination at time t are denoted by $\mathrm{src}(\sigma_t)$ and $\mathrm{dst}(\sigma_t)$, respectively.

Sometimes, for ease of presentation, we will regard the *communication requests* σ as inducing a *request graph* (or equivalently: a *request matrix*) $R(\sigma) = (V(\sigma), E(\sigma))$ over the vertices V; the edges $E(\sigma)$ of $R(\sigma)$ are annotated with frequency information. (When clear from the context, we will often omit σ in $R(\sigma)$, $V(\sigma)$, $E(\sigma)$, and simply write R, V, E.)

Concretely, the node set V of R is given by the set of nodes participating in σ, i.e., $V = \{v : \exists t, v \in \sigma_t\}$, and the set of directed edges E is given by $E = \{\sigma_t : t \in 0, \dots m-1\}$. The weight $w(e)$ of each directed edge $e = (u, v) \in E$ is the frequency $f(u, v)$ of the request from u to v in σ. In the following, we will sometimes simply write $w(u, v)$ to denote the weight $w(e)$ of edge e. For example, in some scenarios the communication pattern between the nodes V may also form a tree (e.g., a multicast tree), or a complete graph, or a set of disconnected components (e.g., describing a clustered communication pattern).

Let $\mathcal{A}$ be an algorithm that given the request σ_t and the graph $G_t \in \mathcal{G}$ at time t, transforms the current graph (via local transformations) to $G_{t+1} \in \mathcal{G}$ at time $t+1$. We will use the notation $\mathcal{A} = \perp$ to refer to a static (i.e., non-adjusting) "algorithm" which does not change the communication network over time.

The cost of the network transformations at time t are denoted by $\rho(\mathcal{A}, G_t, \sigma_t)$ and capture the number of rotations performed to change G_t to G_{t+1}; when $\mathcal{A}$ is clear from the context, we will simply write ρ_t. We denote with $\mathrm{d}_G(\cdot)$ the distance function between nodes in G, i.e., for two nodes $v, u \in V$ we define $\mathrm{d}_G(u, v)$ to be the number of edges of a *shortest* path between u and v in G, and we assume messages are routed along the shortest paths. For a given sequence of communication requests, the cost for an algorithm is given by the number of transformations and the distance of the communication requests plus one (i.e., also a request (u, u) comes at a minimal cost of one unit).

More formally, we will make use of the following definitions.

Definition 1 (***Average and Amortized Cost***)**.** *For an algorithm $\mathcal{A}$ and given an initial network G_0 with node distance function $\mathrm{d}(\cdot)$ and a sequence $\sigma = (\sigma_0, \sigma_1 \dots \sigma_{m-1})$ of communication requests over time, we define the* (average) cost *of $\mathcal{A}$ as:*

$$\mathrm{Cost}(\mathcal{A}, G_0, \sigma) = \frac{1}{m} \sum_{t=0}^{m-1} (\mathrm{d}_{G_t}(\sigma_t) + 1 + \rho_t) \qquad (1)$$

The amortized cost *of $\mathcal{A}$ is defined as the worst possible cost of $\mathcal{A}$, i.e.,* $\max_{G_0, \sigma} \mathrm{Cost}(\mathcal{A}, G_0, \sigma)$.

Like for classic splay trees [25], our yardstick to evaluate the obtained costs of a self-adjusting algorithm is the cost to serve the same requests σ on an optimal static tree network.

Definition 2 (***Optimal Static Cost***)**.** *The optimal static cost for a given communication sequence σ is defined as the cost* $\mathrm{Cost}(\perp, G^*, \sigma) = \frac{1}{m} \sum_{t=0}^{m-1} (\mathrm{d}_{G^*}(\sigma_t) + 1)$ *where $\perp$ denotes a static algorithm that does not change the topology, and $G^* \in \mathcal{G}$ is the graph in the allowed graph family $\mathcal{G}$ that minimizes the cost with respect to σ.*

The *entropy* of the communication pattern σ turns out to be a useful parameter to evaluate the performance of self-adjusting *SplayNets*.

A. Entropy and Empirical Entropy

For a discrete random variable X with possible values $\{x_1, \dots, x_n\}$, the entropy $H(X)$ of X is defined as $\sum_{i=1}^{n} p(x_i) \log_2 \frac{1}{p(x_i)}$ where $p(x_i)$ is the probability that X takes the value x_i. Note that, $0 \cdot \log_2 \frac{1}{0}$ is considered as 0. For a joint distribution over X, Y, the *joint entropy* is defined as $H(X, Y) = \sum_{i,j} p(x_i, y_j) \log_2 \frac{1}{p(x_i, y_j)}$. Also

recall the definition of the *conditional entropy* $H(X|Y)$: $H(X|Y) = \sum_{j=1}^{n} p(y_j) H(X|Y = y_j)$.

Since the sequence of communications σ is revealed over time and may not be chosen from a fixed probability distribution, we are often interested in the *empirical entropy* of σ, i.e., the entropy implied by the communication *frequencies*. Let $\hat{X}(\sigma) = \{f(x_1), \ldots, f(x_n)\}$ be the empirical entropy measure of the frequency distribution of the *communication sources* (origins) occurring in the communication sequence σ, i.e., $f(x_i)$ is the frequency with which a node x_i appears as a source in the sequence, i.e., $f(x_i) = (\#x_i \text{ is a source in } \sigma)/m$. The empirical entropy $H(\hat{X})$ is then defined as $\sum_{i=1}^{n} f(x_i) \log_2 \frac{1}{f(x_i)}$. Similarly, we define the empirical entropy of the *communication destinations* $H(\hat{Y})$ and analogously, the empirical conditional entropies $H(\hat{X}|\hat{Y})$ and $H(\hat{Y}|\hat{X})$.

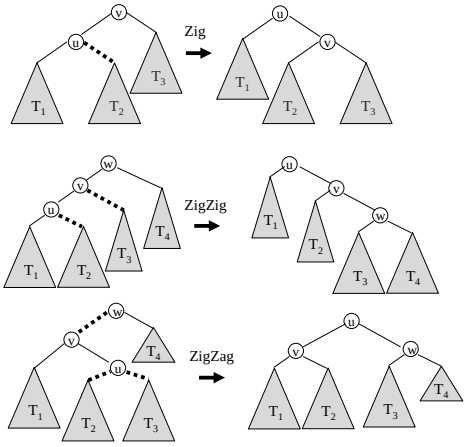

Fig. 1: Basic rotations of splay trees. The dashed bold lines indicate adjacency relationships which are not maintained during the operation.

B. Splay Trees

Our work can be regarded as a distributed generalization of splay trees, binary search trees whose topology adapts to the lookup sequence. Indeed, assuming that all requests originate from the same node, the *SplayNet* problem becomes equivalent to the classic splay tree problem. In the following, we hence briefly review the concept of splay trees.

For a node set V with unique identifiers (IDs) or values, we consider the family $\mathcal{B}$ of the set of all binary search trees over the IDs of V. Let $s = (v_0, v_1, \ldots, v_{m-1})$, $v_i \in V$, be a sequence of lookup requests. In the classic *offline* problem (i.e., for algorithm $\mathcal{A} = \perp$), the goal is to find the best search tree $T^* \in \mathcal{B}$ that minimizes the cost *Cost*$(\perp, T^*, s)$.

We will make use of the following two well-known properties of optimal binary search trees.

Theorem 1 ([13])**.** *An optimal binary search tree T^* that serves s with minimum cost can be found via dynamic programming.*

Note however that the computation of T^* is more complicated than simply using greedy Huffman coding [11] on the frequency distribution of the items in s.

Theorem 2 ([17])**.** *Given s, for any (optimal) binary search tree T:*

$$\text{Cost}(\perp, T, s) \geq \frac{1}{\log 3} H(\hat{Y}) \tag{2}$$

where $\hat{Y}(s)$ is the empirical measure of the frequency distribution of s and $H(\hat{Y})$ is its empirical entropy.

We can use some tools developed for dynamic binary search trees also in our generalized setting. In particular, as we will see, our *SplayNet* algorithm applies splay tree operations in the smallest subtrees connecting two communication partners. The basic operation to adjust a splay tree is called *splaying* (see Algorithm 1) which consists of the classic `Zig`, `ZigZig`, and `ZigZag` rotations. In a nutshell, the main idea of the online splay tree algorithm introduced by Sleator and Tarjan [25] is to rotate (using the uniquely defined sequence of `Zig`, `ZigZig`, and `ZigZag` operations) the currently accessed element directly to the root of the tree. Interestingly, this rather aggressive scheme to promote elements already after a single access, results in a good performance.

For our analysis of the distributed setting, we can also adapt the *Access Lemma* [25] by Sleator and Tarjan. It is reviewed in the following. In [25], in order to compute the amortized time to splay a tree, each node v in the tree is assigned an arbitrary weight $w(v)$. Then the *size* of a node v, $s(v)$ is defined as the sum of the individual weights of all the nodes in the subtree rooted at v. The so-called rank $r(v)$ of a node v is the logarithm of its size, i.e., $r(v) = \log(s(v))$. Sleator and Tarjan showed the following:

Lemma 1 (Access Lemma [25])**.** *The amortized time to splay a tree with root r at a node u is at most $3 \cdot (\mathrm{r}(r) - \mathrm{r}(u)) + 1 = O(\log(\mathrm{s}(r)/\mathrm{s}(u)))$.*

Following this lemma, Sleator and Tarjan were able to show that splay trees are optimal with respect to static binary search trees:

Algorithm 1 Splay Tree Algorithm ST

1: (* upon lookup (u) *)
2: **splay** u to root of T

Theorem 3 (Static Optimality Theorem [25] - rephrased)**.** *Let ST denote the ST algorithm. Let s be a sequence of lookup requests where each item is requested at least once, then for any initial tree T* $\text{Cost}(ST, T, s) = O(H(\hat{Y}))$ *where $H(\hat{Y})$ is the empirical entropy of s.*

IV. Splay Networks

This section presents and analyzes our distributed algorithm *Double Splay* that adapts *SplayNets* according to the demand. Moreover, we will briefly sketch an algorithm to compute optimal static tree networks.

A. Optimal Static Tree Network

Before we introduce our online algorithm, it is interesting to discuss the design of optimal static networks, i.e., networks which do not adapt to the demand dynamically but are optimized for a given static request sequence. The static problem can be seen as a variant of the Minimum Linear Arrangement (MLA) problem. MLA asks for an arrangement of nodes on the line such that the expected path length (i.e., the weighted average of the hop lengths of the communication paths between nodes) is minimized. Note that from this perspective, we can see the task of designing an optimal static network as a "Minimum Search Tree Arrangement" (MSTA) problem. It turns out that unlike the MLA which is NP-hard, the optimal static tree network can be computed in polynomial time. We are not aware of any other graph family of exponential size for which a polynomial time minimal arrangement exists for arbitrary guest graphs.

The main insight needed to solve the MSTA problem is that the problem for the entire tree can be decomposed into optimal subproblems for smaller trees, and that the demand to a given node in a subtree can be *decoupled* from nodes outside a given subtree: the precise topological structure of the nodes outside a subtree does not matter. A detailed description of our dynamic programming algorithm and the optimality proof can be found in the appendix (Appendix B).

Theorem 4. *The optimal static tree network for a sequence σ can be computed in time $O(n^3)$, where n is the number of nodes.*

B. The Double Splay Algorithm DS

Let us now turn to the more interesting self-adjusting networks. The main idea of our *double splay* algorithm DS is to perform splay tree operations in *subtrees* covering the different communication partners. Concretely, consider a communication request (u, v) from node u to node v, and let $\alpha_T(u, v)$ denote the lowest common ancestor of u and v in the current network T. For an arbitrary node x, let $T(x)$ be the subtree rooted at x.

The formal algorithm listing of DS is shown in Algorithm 2: When a request (u, v) occurs, DS first simply splays u to the lowest common ancestor $\alpha_T(u, v)$ of u and v, using the classic splay operations Zig, ZigZig, ZigZag from [25] (see Figure 1). We assume that the **splay** function returns the tree resulting from these operations. Subsequently, the idea is to splay the destination node v to the child of the lowest common ancestor $\alpha_{T'}(u, v)$ of u and v in the resulting tree T'. Observe that this common ancestor is u itself ($u = \alpha_{T'}(u, v)$), i.e., we define the double splay algorithm DS to splay v such that it becomes a child of u. (Note that the child is uniquely defined: if $u > v$, v will be the left, if $u < v$, the right child of u.)

Algorithm 2 Double Splay Algorithm DS

1: (* upon request (u, v) in T *)
2: $w := \alpha_T(u, v)$
3: $T' :=$ **splay** u to root of $T(w)$
4: **splay** v to the child of $T'(u)$

C. Basic Upper and Lower Bounds

The amortized communication cost of DS can be upper bounded by the entropy of the sources and destinations of the requests.

Theorem 5. *Let σ be an arbitrary sequence of communication requests, then for any initial tree T_0,*

$$\text{Cost}(DS, T_0, \sigma) = O(H(\hat{X}) + H(\hat{Y}))$$

where $H(\hat{X})$ and $H(\hat{Y})$ are the empirical entropies of the sources and the destinations in σ, respectively.

Proof: The claim is a consequence of Lemma 1. For any node v let $s(v)$ denote the total number of times v appears as a source in σ, and let $d(v)$ denote the total number of times v appears as a destination. We assign each node $v \in V$ two weights $s(v)/m$ and $d(v)/m$ and analyze the two basic operations of DS separately: first splaying the source to the common ancestor and second splaying the destination to the new common ancestor. The cost $Cost(\text{DS}, T, \sigma)$ can be computed as

$$\frac{1}{m}\sum_{i=1}^{m}\left(\textbf{splay } \text{src}(\sigma_t) \text{ to } w_t + \textbf{splay } \text{dst}(\sigma_t) \text{ to } w'_t\right)$$

where w_t is the lowest common ancestor of $\mathrm{src}(\sigma_t)$ and $\mathrm{dst}(\sigma_t)$ at time t and w'_t is the child of w_t after the first splay operation. Similarly to the proof of Lemma 1, we define the *size* of a subtree T as the sum of the weights of all nodes in T. Since the size of a source $v \in V$ is at least $s(u)/m$ and the size of any node is at most 1 (analogously for the case of destinations: just replace $s(u)$ by $d(v)$), by Lemma 1 we can compute $\mathit{Cost}(\mathtt{DS}, T, \sigma)$ as:

$$\begin{aligned}
\mathit{Cost}(\mathtt{DS}, T, \sigma) &\leq \frac{1}{m}\sum_{i=1}^{m}\big(3\cdot(r(\mathrm{src}(\sigma_t)) - r(w_t)) \\
&\quad + \ 3\cdot(r(\mathrm{dst}(\sigma_t)) - r(w'_t)) + 2\big) \\
&\in O(\frac{1}{m}(2m + \sum_{i=1}^{n} s(i)\log(\frac{m}{s(i)}) \\
&\quad + \sum_{j=1}^{n} d(j)\log(\frac{m}{d(j)}) \\
&\in O(H(\hat{X}) + H(\hat{Y}))
\end{aligned}$$

■

Next we derive a simple lower bound for the expected path length in a distributed tree network. It is related to the *conditional empirical entropy* of the request sequence.

Theorem 6. *Given a request sequence σ, for any optimal (binary search) tree network T:*

$$\mathrm{Cost}(\perp, T, \sigma) \in \Omega(H(\hat{Y}|\hat{X}) + H(\hat{X}|\hat{Y})) \qquad (3)$$

Proof: For any node $x \in V$, let $\hat{Y}_x$ denote the frequency distribution of the destinations *given that* the source is x. Consider an optimal tree T with root x. Following Property 2, the average path length of requests is $\Omega(H(\hat{Y}_x))$. Considering the optimal tree for each source, we have a cost of at least

$$\sum_{i=1}^{n} f(x_i) H(\hat{Y}_{x_i}) \in \Omega(H(\hat{Y}|\hat{X}))$$

A similar argument holds for the destinations: since for each destination y the cost for its requests in an optimal tree where y is the root is at least $H(\hat{X}_y)$, where $\hat{X}_y$ denotes the frequency distribution of the sources given that the destination is y. ■

Theorems 5 and 6 are relatively general and there remains a gap between upper and lower bound. It is hence interesting to study some concrete examples in more detail.

First, we observe that DS achieves an optimal amortized cost for all request patterns following a *product distribution*: the probability $p(u, v)$ of a communication request (u, v) can be described by the product of the activity levels $p(u)$ and $p(v)$ of the nodes, i.e., $p(u, v) = p(u) \cdot p(v)$. Hence, from the independence it follows that the entropy of the communication sources given the destinations equals the entropy of the communication sources only (i.e., $H(\hat{X}|\hat{Y}) = H(\hat{X})$), and vice versa for the destinations ($H(\hat{Y}|\hat{X}) = H(\hat{Y})$).

Corollary 1. *DS is asymptotically optimal if σ follows a product distribution.*

However, there are also simple examples where the gaps remain open. Figure 2 gives three exemplary request patterns $R(\sigma)$ which help us to better understand the gap between our upper and lower bound. All three request graphs in Figure 2 are bounded degree graphs, i.e., $H(\hat{Y}|\hat{X}$ and $H(\hat{X}|\hat{Y})$ are upper bounded by a constant while $H(\hat{Y})$ and $H(\hat{X})$ can be as high as $\log n$. In Scenarios (a) and (b) the lower bound is actually zero since given the source there is no ambiguity about the destination and vice versa. In Section V we will derive an improved lower bound on the cost of an optimal static network. On that occasion, we will revisit our examples here and show that some gaps can be closed.

D. Convergence for Special Request Patterns

In this section we consider special request patterns that highlight some of DS's locality and convergence properties. In particular, we will discuss scenarios where DS achieves an optimal performance. For this section we consider communication requests of infinite length where every request in σ is repeated infinitely many times.

Locality. We first study a scenario which shows the locality properties of DS. In this scenario, requests are clustered, and so is the resulting tree of DS.

Definition 3 (Cluster Scenario)**.** *In a* cluster scenario *the communication pattern σ partitions the nodes in k* contiguous and disjoint *intervals $I_1 \dot\cup I_2 \dot\cup I_3 \dot\cup \ldots \dot\cup I_k$ where nodes within an interval I_j have consecutive numbers and where communication only happens between node pairs in the interval. In particular, a request (u, v) implies that u and v belong to the same interval: $(u, v) \in \sigma \rightarrow \exists j : \quad u, v \in I_j$.*

Theorem 7. *In a cluster scenario σ, DS features the following two properties:*

1) *DS will eventually construct a* SplayNet *in which for any communication pair $(u, v) \in I_j$, for any $j \in \{1, \ldots, k\}$, u and v are connected by a local path which only includes nodes from I_j.*
2) *Once this local routing property is established, it will never be violated again.*

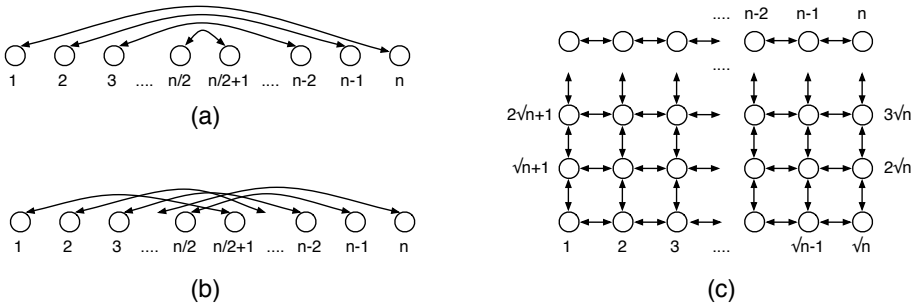

Fig. 2: Three exemplary requests graphs: (a) a laminated set, (b) a random matching, and (c) a 2-dimensional grid

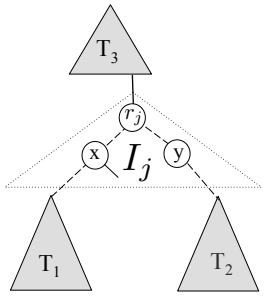

Fig. 3: Illustration for proof of Claim 1 and Theorem 7.

Proof: For any BST T and an interval I_j let $T(I_j)$ denote the smallest size sub-tree of T such that all the nodes of I_j are in $T(I_j)$ and r_j be the root of $T(I_j)$. Let $\mathrm{out}(\mathrm{T}(\mathrm{I_j}))$ denote the number of requests $(u, v) \in \sigma$ s.t. either u or v are in $T(I_j)$, but *not both*.

For a cluster scenario σ and a BST T consider a potential function $\phi = \sum_{j=1}^{k} \mathrm{out}(\mathrm{T}(\mathrm{I_j}))$. We will prove the theorem by showing first that when $\phi = 0$ for any $(u, v) \in I_j$, $j \in \{1, \ldots, k\}$, u and v are connected by a local path which only includes nodes from I_j and ϕ remain zero after any request. Next will show that when $\phi > 0$, DS cannot increase ϕ, and there is a request in σ that will reduce it.

We start with the following claim.

Claim 1. *Any two nodes $x < y$ in I_j, $j \in \{1, \ldots, k\}$ in a tree $T(I_j)$ with $\mathrm{out}(\mathrm{T}(\mathrm{I_j})) = 0$ are connected via a path that contains only nodes from I_j. This property is invariant for all future requests in σ.*

Proof: Consider the generic situation in Figure 3. For all nodes not in I_j there are 3 possible locations: in (possible empty) sub-trees T_1, T_2, or T_3. Clearly, T_1 contains only IDs smaller than I_j and must be attached to the smallest ID in I_j, T_2 contains only IDs larger than I_j and must be attached to the largest ID in I_j. T_3 can contain either smaller or larger IDs depending on whether r_j is a left or a right son of its parent. If $\mathrm{out}(\mathrm{T}(\mathrm{I_j})) = 0$ there are no requests between T_1 or T_2 to T_3. Therefore requests within T_1, T_2 and T_3 remain within the corresponding subtrees, and so $\mathrm{out}(\mathrm{T}(\mathrm{I_j}))$ remains 0 after each request. Moreover for any $x, y \in I_j$, the path contains only nodes from I_j and any request within I_j splays only IDs within I_j, hence the claim holds. ■

Now observe that if $\phi = 0$, every $\mathrm{out}(\mathrm{T}(\mathrm{I_j})) = 0$, and so ϕ will remain zero also in the future; for any $(u, v) \in I_j$, their path only includes nodes from I_j.

Next we claim that no request can increase ϕ, and that if $\phi > 0$, there is a request in σ that will decrease the potential function. Consider $T(I_j)$ and any request (u, v). If $(u, v) \in I_j$ then $\mathrm{out}(\mathrm{T}(\mathrm{I_j}))$ does not change. If $u, v \in T_i$ for $i = 1, 2$ or 3, then again $\mathrm{out}(\mathrm{T}(\mathrm{I_j}))$ does not change. If $u \in T_1$ or $u \in T_2$ and $v \in T_3$, then the lowest common ancestor of u, v is in T_3 and after DS of u and v, $\mathrm{out}(\mathrm{T}(\mathrm{I_j}))$ will decrease by one. Therefore ϕ can only decrease. Now if $\phi > 0$, there is some j for which $\mathrm{out}(\mathrm{T}(\mathrm{I_j})) > 0$. Take the request (u, v) that is a witness; after this request $\mathrm{out}(\mathrm{T}(\mathrm{I_j}))$ will decrease by one, so ϕ will decrease. So overall given a cluster σ, ϕ will decrease to zero which proves the theorem. ■

Optimal Scenarios. DS sometimes achieves an optimal performance by converging to the optimal static network implied by $R(\sigma)$. We have already encountered an example where *SplayNets* are asymptotically optimal, namely if σ describes a product distribution (cf Corollary 1). In the following, we will examine optimal scenarios in more detail.

A first class of optimal scenarios are *laminated* communication requests, cf Figure 2 a).

Definition 4 (Laminated Scenario)**.** *In a* laminated scenario *it holds for the communication pattern σ that for any two pairs (u_1, v_1) and (u_2, v_2) in σ, $\min\{u_1, v_1\} <$*

$\min\{u_2, v_2\} < \max\{u_2, v_2\} < \max\{u_1, v_1\}$.

We have the following result.

Theorem 8. *In communication scenarios with laminated communication patterns σ, DS will eventually converge to (and stay at) a tree where any communication pair $\{u, v\}$ is adjacent.*

Proof: We can assign a *level* to any laminated pair $p_1 = (u_1, v_1)$ depending on the number of pairs $p_2 = (u_2, v_2)$ nested in p_1, i.e., for which $\min\{u_1, v_1\} < \min\{u_2, v_2\} < \max\{u_2, v_2\} < \max\{u_1, v_1\}$. Let us refer to the level of the outermost pair as level 0. Our proof works by induction over these nestings, from lower to higher levels. Concretely, we first show that after a pair $p_1 = (u_1, v_1)$ of level 0 communicated, it will stay adjacent forever. To see this, consider any other pair $p_2 = (u_2, v_2)$ inside the interval of p_1. By the definition of DS, after a request (u_1, v_1), v_1 is the right (resp. left) child of u_1 if $u_1 < v_1$ (resp. $u_1 > v_1$). We will show that this implies that the nodes u_2, v_2 must be both in the same subtree, and can hence not change the adjacency relationship of u_1 and v_1 anymore. The claim follows by case distinction: (1) If $u_1 < v_1$, the left subtree of v_1 is the only subtree which can contain nodes w with $u_1 < w < v_1$; however, these are exactly the nodes fulfilling the lamination property $\min\{u_1, v_1\} < w < \max\{u_1, v_1\}$. If $u_1 > v_1$, the right subtree of v_1 is the only subtree which can contain nodes w with $u_1 < w < v_1$ and hence $\min\{u_1, v_1\} < w < \max\{u_1, v_1\}$. With the link of level i being stable, we can recursively prove the stability of a link of level $i+1$, as again, the influence of any corresponding laminated pair is restricted to the corresponding subtree. ∎

Combining the last two theorems of clustered and laminated patterns it is possible to prove optimality in a non-crossing matching scenario. For a request (u, v) in σ, let $I_{(u,v)}$ denote the interval $[\min(u, v), \max(u, v)]$.

Definition 5 (Non-Crossing Matching Scenario)**.** *In a* non-crossing matching scenario*, it holds for the communication pattern σ that for any two pairs (u_1, v_1) and (u_2, v_2) in σ, either:*

1) $I_{(u_1,v_1)} \subsetneq I_{(u_2,v_2)}$ or $I_{(u_2,v_2)} \subsetneq I_{(u_1,v_1)}$, or
2) $I_{(u_1,v_1)} \cap I_{(u_2,v_2)} = \emptyset$.

It follows from the definition that the request graph $R(\sigma)$ must describe a matching, and for each request (u, v) there are no other requests that enter or leave (i.e., *cross*) the interval $I_{(u,v)}$.

If $R(\sigma)$ describes a non-crossing matching scenario, DS will converge to an optimal solution.

Theorem 9. *In a non-crossing matching scenario σ, DS will eventually converge to (and stay at) a tree where any communication pair $\{u, v\} \in \sigma$ is adjacent.*

The proof follows from the observation that a non-crossing matching scenario is a cluster scenario (which will converge by Theorem 7) and within each cluster it is a laminated scenario (which will converge by Theorem 8).

To give one more example where DS is optimal, we consider the multicast tree scenario.

Definition 6 (Multicast Tree Scenario)**.** *In a* multicast tree scenario*, it holds for the communication pattern σ that the request graph $R(\sigma)$ forms a rooted and sorted binary tree.*

Theorem 10. *If the communication pairs in σ form a multicast tree, DS will eventually converge to the optimal static solution, i.e., to $R(\sigma)$.*

Proof: Let us label the links of $H = R(\sigma)$ by the node they are pointing to. Thus, the node identifiers in H imply an *order* on the links. The proof is by induction over the order of these links. Concretely, we will consider the following kind of *maximal link sets* $S = \{(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)\}$. A link set S is called a maximal link set iff the following holds: if u_s is the smallest source node in $\{u_1, \ldots, u_k\}$ (i.e., $u_s = \arg\min\{u_1, \ldots, u_k\}$), then S includes all links (u_i, v_i) of order $v_i \geq u_s$. We will partition the nodes into such link sets and prove inductively over link sets with decreasing order, that once such a maximal link set S is stable, future communication requests cannot change S anymore. Moreover, we will prove termination, i.e., there are always requests that lead to an increase of the maximal link set.

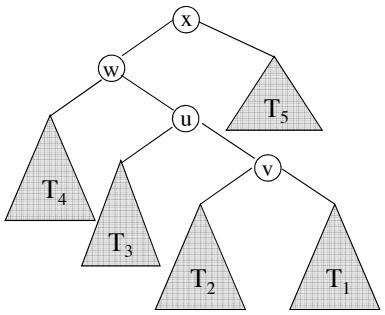

Fig. 4: Illustration for proof of Theorem 10.

Induction hypothesis: Let us consider the highest order link $e = (u, v)$ first, i.e., v is the overall maximal node in H. We claim that a maximal link set

$S = \{(u,v),(u_1,v_1),\ldots,(u_k,v_k)\}$ will emerge which includes all links (u_i,v_i) with nodes $u_i \geq u$. To see this, we first note that once the link (u,v) is requested, it will never change again in the future. We have the following general situation in the current graph (Figure 4): v has a right subtree T_1 and a left subtree T_2, the parent u on the left having itself a left subtree T_3; u has a left parent w with a left subtree T_4 and a right parent x with right subtree T_5. Clearly, all nodes $u_i > u$ must be located in the subtree T_2, as subtrees T_1 and T_5 must be empty as v is the overall largest node in the network, and subtrees T_3 and T_4 can only contain smaller nodes than u. The link (u,v) will remain stable after it has been used for the first time, as (u,v) is both a link in H as well as in the current graph, and as two edges in a binary search tree cannot intersect in the identifier space (cf Figure 1). In order to show that also all other links in S stabilize, we can focus on subtree T_2 and observe that the first time v calls the root node r_2 of subtree T_2, the link (u,r_2) cannot change anymore either, as naturally, all requests inside T_2 fall either in the left or the right subtree of r_2. By a second, simple induction, it follows that the entire subtree T_2 will converge from the roots of the subtrees, which proves the induction hypothesis that such a stable set S will emerge.

Induction step: By the induction hypothesis, we can assume that there exists a stable set $S = \{(u_1,v_1),\ldots,(u_k,v_k)\}$ consisting of all links with incident nodes larger than the minimal source node u_s in the set S. For the induction step, we need to show that a larger maximal link set S' will emerge from S. Consider a maximal link set $S' = \{(u'_1,v'_1),\ldots,(u'_k,v'_k)\} \cup S$ (a laminated partition), assume that v is the largest node in $S' \setminus S$, consider the link (u,v), and let S' such that u is the smallest source in S'. The situation for link (u,v) can again be described by Figure 4. Compared to the induction hypothesis, the situation is slightly more complex now. We again consider the different subtrees in turn. Clearly, all nodes in T_1 and T_5 are larger than v and hence the corresponding links inside T_1 and T_5 as well as the links with endpoints there must be stable by the induction hypothesis. The subtrees T_2 and T_3 cannot communicate due to the link (u,v) which is part of the communication tree by definition. Thus, we can again prove by induction that the subtree T_2 will converge from the root, and the claim follows. ∎

V. Lower Bounds

This section studies lower bounds for serving requests on an optimal static tree in more detail. We first present a lower bound based on interval cuts, and then derive a lower bound based on edge expansion.

A. Interval Cuts Bound

A good lower bound can be obtained by using an intriguing connection to *request graph cuts*. It is intuitively clear that if the request graph $R(\sigma)$ exhibits large cuts, it can be more difficult to find a tree network that accommodates the requests well. But one has to be careful when defining the problematic cuts, as even graphs with many large cuts can sometimes be embedded optimally.

We start with a definition of an *interval cut*.

Definition 7 (Interval Cut)**.** *Arrange the nodes V on a one dimensional line in an ascending, sorted order, and let $I_j^\ell \subseteq V$ denote the set of nodes corresponding to the subinterval of length ℓ covering the nodes of order $j, j+1, ..., j+\ell-1$. Let $\bar{I}_j^\ell$ denote the remaining nodes, i.e., $\bar{I}_j^\ell = V \setminus I_j^\ell$. For a weighted directed graph $G(V,E)$, the* interval cut*,* $\text{Cut}_{in}(I_j^\ell, G)$ *is the set of edges in G pointing to nodes in I_j^ℓ, and the interval cut* $\text{Cut}_{out}(I_j^\ell, G)$ *is the set of edges in G originating at nodes in I_j^ℓ and pointing outwards. Formally*

$$\text{Cut}_{in}(I_j^\ell, G) = \Big\{(u,v) : v \in I_j^\ell, u \in \bar{I}_j^\ell, (u,v) \in G\Big\}$$
$$\text{Cut}_{out}(I_j^\ell, G) = \Big\{(u,v) : u \in I_j^\ell, v \in \bar{I}_j^\ell, (u,v) \in G\Big\}$$

The weight of a cut $\tilde{c}$ is defined as the sum of the weights of its edges, $w(\tilde{c}) = \sum_{(u,v)\in\tilde{c}} w(u,v)$.

We consider the *request graph* $R(\sigma)$ where a directed edge represent a source-destination pair in σ and edge weights represent the frequency of the communication requests. An interval cut $\tilde{c}$ in $R(\sigma)$ is a set of source-destination pairs, so for an interval cut $\tilde{c}$ the conditional frequency distribution of the sources is $\hat{X}_{\tilde{c}}$ and the one of the destinations is $\hat{Y}_{\tilde{c}}$.

Given an interval cut $\tilde{c}$, we will denote the weighted empirical entropy of $\tilde{c}$ as:

$$\tilde{H}(\tilde{c}) = w(\tilde{c})\big(H(\hat{X}_{\tilde{c}}) + H(\hat{Y}_{\tilde{c}})\big)$$

We can lower bound the average communication cost of an (optimal) binary search tree T as follows.

Theorem 11. *Given a request sequence σ, for any (optimal) binary search tree T:*

$$\text{Cost}(\perp, T, \sigma) \in \Omega\left(\max_i \min_{j,\ell} \tilde{H}(\text{Cut}_{in}(I_j^\ell, R(\sigma)))\right) \quad (4)$$

$$\text{Cost}(\perp, T, \sigma) \in \Omega\left(\max_i \min_{j,\ell} \tilde{H}(\text{Cut}_{out}(I_j^\ell, R(\sigma)))\right) \quad (5)$$

where $i \in [0, \log n - 1]$, $j \in [1,n]$ and $\ell \in [\frac{n}{2^{i+1}}, \frac{n}{2^i}]$ and $[a,b]$ is the set of integers between a and b (including the boundary nodes).

Proof: We will only prove Eq. (5), as Eq. (4) can be proved in an analogous way. Let $\tilde{c}^*$, be a cut that maximizes Eq. (5) and let i^* be the corresponding i. Now consider T. Let v be a node s.t. $\frac{n}{2^{i^*}} > |T(v)| \geq \frac{n}{2^{i^*+1}}$ (such a subtree must exist in any T due to Claim 3). Let u be the parent of v. Let $\ell^* = |T(v)|$ and let us choose j^* such that the set of nodes of $T(v)$ is $I_{j^*}^{\ell^*}$. Such a j must exist due to Claim 2. The result now follows from Property 2, since all the requests from $I_{j^*}^{\ell^*}$ to $I_{j^*}^{\ell^*}$ must cross edge (u, v) and entail a cost of at least $\Omega(\tilde{H}(\tilde{c}^*))$. To make the last statement more clear, consider an optimal binary tree (with root v) that needs to serve lookups from a frequency distribution $\hat{X}_{\tilde{c}^*}$. The cost of the lookups will be at least $\Omega(H(\hat{X}_{\tilde{c}^*}))$. The same holds for the destinations $\hat{Y}_{\tilde{c}^*}$. ∎

B. Edge Expansion Bound

Another lower bound can also be obtained by using concepts related to graph expansion, and in particular the *conductance* [24] and the *edge expansion* [10] of graphs. We need the following definitions: Let $G(V, E)$ be a directed weighted graph. We assume the edge weights are normalized, i.e., the sum of all edge weights is one: $\sum_{(u,v)\in E} w(u, v) = 1$.

Definition 8 (Conductance Entropy)**.** *The* cut $E(S, \bar{S})$ *is the set of outgoing edges from* S*:* $E(S, \bar{S}) = \{(u, v) : u \in S, v \in \bar{S}, (u, v) \in E\}$*. The* weight of a cut $E(S, \bar{S})$*,* $W(S)$ *is the sum of the weights of the edges in the cut.*

$$W(S) = \sum_{(u,v)\in E(S,\bar{S})} w(u, v)$$

A distribution of the sources $\mathrm{src}(S)$ *of a cut* $E(S, \bar{S})$ *is defined for the set of nodes in* S *that are also in* $E(S, \bar{S})$ *as follows: the probability (weight) of each* $u \in S$ *and* $E(S, \bar{S})$ *is defined as*

$$w_S(u) = \sum_{\substack{(u,v)\in E(S,\bar{S}) \\ u\in S}} w(u, v)/W(S).$$

Similarly the distribution $\mathrm{dst}(S)$ *is defined over the destinations in* $E(S, \bar{S})$*,* $\mathrm{src}(S)$ *such that the probability of* v *being a destination in* $\bar{S}$ *is:*

$$w_{\bar{S}}(v) = \sum_{\substack{(u,v)\in E(S,\bar{S}) \\ v\in \bar{S}}} w(u, v)/W(S).$$

The entropy of a cut $(S, \bar{S})$ *is defined as:*

$$\varphi_H(S) = W(S)\left(H(\mathrm{src}(S)) + H(\mathrm{dst}(S))\right) \quad (6)$$

The conductance entropy *of a graph is defined as:*

$$\phi_H(G) = \min_{S\subseteq V} \varphi(S) \quad (7)$$

We can now claim the following:

Theorem 12. *Given a request sequence* σ*, for any (optimal) binary search tree* T*:*

$$\mathrm{Cost}(\perp, T, \sigma) \in \Omega(\phi_H(R(\sigma))) \quad (8)$$

The proof is similar to the arguments of the proof of Theorem 11. Note that $\phi_H(G)$ can be at most $O(\log n)$ since $W(s)$ is at most 1 and the entropy is at most $\log n$.

The *edge expansion* [10] of a graph G is defined as:

$$h(G) = \min_{0<|S|\leq \frac{n}{2}} \frac{E(S, \bar{S})}{|S|} \quad (9)$$

For the special case where the request graph $R(\sigma)$ is a constant degree d-regular graph with uniform weights and an edge expansion α, we can claim the following.

Theorem 13. *Given a request sequence* σ *s.t.* $R(\sigma)$ *is a* d*-regular graph with uniform weights and edge expansion* α *then* $\mathrm{Cost}(\perp, T, \sigma) \in \Omega(\log(\alpha n))$.

The claim follows since if we take S to be $\Omega(n)$ the entropy of the cut must be $\Omega(\log(\alpha n)$. We now revisit the examples of Figure 2 and elaborate on their bounds using the above theorems.

C. Examples

Consider a request graph that forms a 2-dimensional grid as in Figure 2 c). For this scenario, Theorem 13 gives a tight lower bound of $\Omega(\log n)$, since the edge expansion of the 2-dimensional grid is of order $1/\sqrt{n}$. For the random matching case of Figure 2 b), Theorems 12 and 13 only give a constant lower bound since the expansion of a matching is zero (the graph is not even connected). However, with Theorem 11 that only considers *interval cuts* (and not all cuts as in edge expansion), we can get a tight lower bound of $\Omega(\log n)$ for these cases.

VI. Simulations

In order to complement our theoretical insights, we conducted simulations on a connected subset of the *Facebook online social network* (obtained from [27]) with roughly 63K nodes and 800K edges. The user identifiers (IDs) are chosen according to a breadth-first search from the largest degree node. We consider two simplistic communication patterns σ: (1) In the first scenario (`RW`), a rumor spreads randomly from one user to the next along friendship links, i.e., communication occurs along a random walk of the friendship graph. We use a parameter p to tune the locality of the random walk: with probability p, a given communication request (u, v) is repeated, and with probability $(1 - p)$ a new

request (v, w) is generated, where w is a neighbor of v chosen uniformly at random. (2) In the second scenario (MATCH), we compute a random maximal matching on the Facebook graph and cycle through all matched edges which represent the (sequential) communication requests. Figure 5 shows the results when applying DS

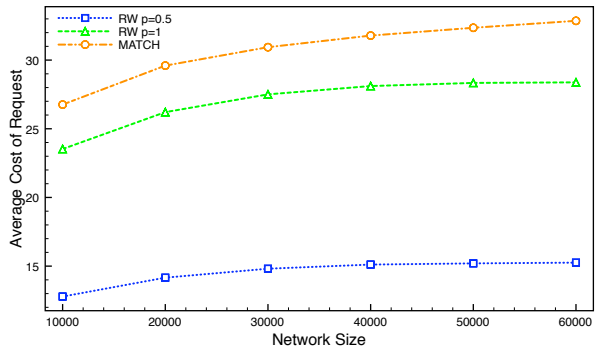

Fig. 5: Average cost of *SplayNet* requests under different scenarios and as a function of the network size.

to subgraphs of the Facebook graph of increasing sizes. We ran our experiments sufficiently long to study the steady state performance (e.g., the random walk is much longer than the network size). The variance of our results is hence very low. We observe that in the random walk scenario, the average request route length is significantly shorter than in the matching scenario, even for $p = 1$. This demonstrates that DS performs better on localized requests. Naturally, the request locality for lower p can be exploited to reduce the amortized cost. In the matching however, the random nature of the matchings cause a high cost as implied by Theorem 11.

VII. Conclusion

We regard our work as a first step towards the design of novel distributed data structures and networks which adapt dynamically to the demand. The main simplification used in this paper regards the restriction to the tree topology, and the generalization to more complex and redundant networks is an open question. Moreover, similarly to [25], we have focused on the amortized costs of *SplayNets*, and an interesting direction for future research regards the study of the competitive ratio achieved by our algorithm under arbitrary communication patterns.

References

[1] B. Allen and I. Munro. Self-organizing binary search trees. *J. ACM*, 25:526–535, 1978.

[2] D. Arora, M. Bienkowski, A. Feldmann, G. Schaffrath, and S. Schmid. Online strategies for intra and inter provider service migration in virtual networks. In *Proc. IPTComm*, 2011.

[3] J. Aspnes and G. Shah. Skip graphs. *ACM Transactions on Algorithms (TALG)*, 3(4):37–es, 2007.

[4] D. Batista, N. da Fonseca, F. Granelli, and D. Kliazovich. Self-adjusting grid networks. In *Proc. IEEE International Conference on Communications (ICC)*, pages 344–349, 2007.

[5] E. D. Demaine, D. Harmon, J. Iacono, and M. Patrascu. Dynamic optimality - almost. In *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 484–490, 2004.

[6] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

[7] U. Feige and J. Lee. An improved approximation ratio for the minimum linear arrangement problem. *Information Processing Letters*, 101(1):26–29, 2007.

[8] L. H. Harper. Optimal assignment of numbers to vertices. *J. SIAM*, (12):131–135, 1964.

[9] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *Proc. 7th USENIX Conference on Networked Systems Design and Implementation*, 2010.

[10] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–562, 2006.

[11] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[12] R. Jacob, A. Richa, C. Scheideler, S. Schmid, and H. Täubig. A polylogarithmic time algorithm for distributed self-stabilizing skip graphs. In *Proc. 28th ACM Symposium on Principles of Distributed Computing (PODC)*, 2009.

[13] D. Knuth. Optimum binary search trees. *Acta informatica*, 1(1):14–25, 1971.

[14] F. Kuhn, S. Schmid, and R. Wattenhofer. Towards worst-case churn resistant peer-to-peer systems. *Distributed Computing Journal (DC)*, 22(4):249–267, 2010.

[15] J. Leitao, J. Marques, J. Pereira, and L. Rodrigues. X-bot: A protocol for resilient optimization of unstructured overlay networks. *IEEE Transactions on Parallel and Distributed Systems*, 99, 2012.

[16] M. Lis, K. Shim, M. Cho, C. Fletcher, M. Kinsy, I. Lebedev, O. Khan, and S. Devadas. Brief announcement: distributed shared memory based on computation migration. In *Proc. ACM SPAA*, pages 253–256. ACM, 2011.

[17] K. Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, 1975.

[18] G. Mitchison and R. Durbin. Optimal numberings of an n n array. *SIAM J. Algebraic Discrete Methods*, 7(4):571–582, 1986.

[19] R. Ravi, A. Agrawal, and P. N. Klein. Ordering problems approximated: Single-processor scheduling and interval graph completion. In *Proc. ICALP*, 1991.

[20] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. 4:406–425, 1987.

[21] G. Schaffrath, S. Schmid, and A. Feldmann. Optimizing long-lived cloudnets with migrations. In *Proc. IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2012.

[22] C. Scheideler and S. Schmid. A distributed and oblivious heap. In *Proc. 36th ICALP*, 2009.

[23] Y. Shang, D. Li, and M. Xu. Energy-aware routing in data center network. In *Proc. ACM SIGCOMM Workshop on Green Networking*, pages 1–8, New York, NY, USA, 2010. ACM.

[24] A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

[25] D. Sleator and R. Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.

[26] M. Tang, Z. Liu, X. Liang, and P. M. Hui. Self-adjusting routing schemes for time-varying traffic in scale-free networks. *Phys. Rev. E*, 80(2):026114, Aug 2009.
[27] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proc. 2nd ACM Workshop on Online Social Networks (WOSN)*, pages 37–42, 2009.
[28] C. C. Wang, J. Derryberry, and D. D. Sleator. O(log log n)-competitive dynamic binary search trees. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 374–383, 2006.

APPENDIX

A. Properties of Binary Search Trees

This section presents some basic properties of binary search trees which are frequently exploited in our proofs. We consider a binary search tree T of size n and nodes IDs $[1, 2, \ldots, n]$. The following fact is an obvious consequence of the binary search structure.

Fact 1. *For two nodes $x < y$ in a binary search tree T, let w be the lowest common ancestor of x and y. It holds that $x \leq w \leq y$.*

The next two claims are needed for our lower bounds.

Claim 2. *Let T be any binary search tree. For any node $v \in T$ the sub-tree $T(v)$ contains the IDs of a* contiguous *interval, i.e., there exist j and ℓ, s.t. I_j^ℓ equals the set of nodes of $T(v)$.*

This can be shown easily by contradiction. Also the following claim is simple:

Claim 3. *Let T be any binary tree of size $|T| = n$. Then for every $i = 0, 1, \ldots, \lfloor \log n \rfloor - 1$ there exists a node v s.t. $\frac{n}{2^i} > |T(v)| \geq \frac{n}{2^{i+1}}$.*

Proof: Let x_0 be the root of T. Define x_{j+1} as the root of the largest subtree of $T(x_j)$ (break ties arbitrarily) if there exist any. Clearly for each $x_j \in T, j > 0$ we have $|T(x_j)| < |T(x_{j-1})|$. Now for each i in the range above let $v_i = x_j$ for the minimal j s.t. $|T(x_j)| < \frac{n}{2^i}$. Since $|T(x_{j-1})|$ is at least $\frac{n}{2^i}$, it holds that $|T(x_j)| \geq \frac{n}{2^{i+1}}$. ■

Sorted binary tree *networks* are attractive for their low degree and the support for a simple and local routing.

Claim 4. *Sorted binary tree networks facilitate local routing.*

Proof: Basically, a local routing can be achieved by exploiting Claim 2 as follows. Let us regard each node u in the tree network T as the root of a (possibly empty) subtree $T(u)$. Then, a node u simply needs to store the smallest identifier u' and the largest identifier u'' currently present in $T(u)$. This information can easily be maintained, even under the topological transformations performed by our algorithms. When u receives a packet for destination address v, it will forward it as follows: (1) if $u = v$, the packet reached its destination; (2) if $u' \leq v \leq u$, the packet is forwarded to the left child and similarly, if $u \leq v \leq u''$, it is forwarded to the right child; (3) otherwise, the packet is forwarded to u's parent. ■

B. Optimal Static Network

The optimal static tree T which minimizes the sum of the weighted node distances: $\min \sum_{(u,v)\in R(\sigma)} d_T(u, v)$. can be computed by dynamic programming. Let V denote the set of ordered nodes and let R denote the request matrix (i.e., the frequency of a given ordered communication pair). We will index subproblems by intervals I on V, and will refer to all nodes outside I by $\hat{I} = V \setminus I$. For each node v in an interval I, we can compute the aggregate demand towards v (v's weight) from nodes outside I: $W_I(v) := \sum_{u\in\hat{I}} w(u, v)+w(v, u)$ Let W_I denote the corresponding vector consisting of all nodes in the interval I

The algorithm is based on the observation that the demand from outside the considered interval can be "decoupled" with the aggregate weight. The cost of a given tree T_I on I can be computes as follows: $\mathit{Cost}(T_I, W_I) = [\sum_{u,v\in I}(d(u, v)+1)\cdot w(u, v)] + D_I \cdot W_I$ where D_I in the scalar product $D_I \cdot W_I$ is the vector denoting the distance of the nodes in I from the root of T_I, i.e., the vector of the *depths* of the nodes.

Dynamic programming is then based on merging optimal sub-intervals. In order to compute the optimal tree T_I^* for an interval I partitioned into subintervals I' and I'', we exploit the computed optimal substructures for the sub-intervals and choose the best overall root x. For the induction hypothesis, a single node tree has cost zero. The total tree cost can be expressed as follows (the additive 1 has to be added in the end): $\mathit{Cost}(T_I^*, W_I) = \min_{x\in I} \mathit{Cost}(T_{I'}^*, W_{I'} + \mathit{Cost}(T_{I''}^*, W_{I''}) + \sum_{v\in I'} W_{I'}(v)+\sum_{v\in I''} W_{I''}(v)$ Note that when choosing a new root x, all nodes except x are pushed one level down in the tree.

Our algorithm terminates with an overall solution when I represents the entire node set. Since there are $O(n^2)$ intervals I and since merging two trees requires testing $O(n)$ different root candidates, the runtime is at most cubic in the number of nodes. (The W_I values can be computed within the same asymptotic order.)