SplayNet: Towards Locally Self-Adjusting Networks

Stefan Schmid*, Chen Avin*, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, Zvi Lotker

***Abstract*—This paper initiates the study of locally self-adjusting networks: networks whose topology adapts dynamically and in a decentralized manner, to the communication pattern σ. Our vision can be seen as a distributed generalization of the self-adjusting datastructures introduced by Sleator and Tarjan [22]: In contrast to their splay trees which dynamically optimize the lookup costs from a *single node* (namely the tree root), we seek to minimize the routing cost between arbitrary *communication pairs* in the network.**

As a first step, we study distributed binary search trees (BSTs), which are attractive for their support of greedy routing. We introduce a simple model which captures the fundamental tradeoff between the benefits and costs of self-adjusting networks. We present the *SplayNet* algorithm and formally analyze its performance, and prove its optimality in specific case studies. We also introduce lower bound techniques based on interval cuts and edge expansion, to study the limitations of any demand-optimized network. Finally, we extend our study to multi-tree networks, and highlight an intriguing difference between classic and distributed splay trees.

I. Introduction

In the 1980s, Sleator and Tarjan [22] proposed an appealing new paradigm to design efficient Binary Search Tree (BST) datastructures: rather than optimizing traditional metrics such as the search tree depth in the *worst-case*, their *splay* datastructure *self-adjusts* to its usage pattern, moving more frequently accessed elements closer to the root. A natural performance metric to evaluate a self-adjusting system is the *amortized cost*: the "average cost" for a worst case sequence of operations (of a certain class).

Since this seminal work, self-adjusting datastructures have been studied intensively, and various more efficient self-adjusting datastructures such as *Tango* BSTs [7] or *multi-splay trees* [23] have been proposed. In particular, the famous *Dynamic Optimality* conjecture [7] continues to puzzle researchers: the conjecture claims that splay trees perform as well as any other binary search tree algorithm up to a constant factor.

In contrast to these flexible classic datastructures, today's distributed datastructures and networks are still optimized toward static metrics, such as the diameter or the length of the longest route: the self-adjusting paradigm has not spilled over to distributed networks yet.

We, in this paper, initiate the study of a distributed generalization of self-optimizing datastructures. This is a non-trivial generalization of the classic splay tree concept: While in classic BSTs, a *lookup request* always originates from the same node, the tree root, distributed datastructures and networks such as skip graphs [2], [13] have to support *routing requests* between arbitrary pairs (or *peers*) of communicating nodes; in other words, both the source as well as the destination of the requests become variable. Figure 1 illustrates the difference between classic and distributed binary search trees.

In this paper, we ask: Can we reap similar benefits from self-adjusting *entire networks*, by adaptively reducing the distance between frequently communicating nodes?

As a first step, we explore fully decentralized and self-adjusting Binary Search Tree networks: in these networks, nodes are arranged in a binary tree which respects node identifiers. A BST topology is attractive as it supports greedy routing: a node can decide locally to which port to forward a request given its destination address.

A. Our Contributions

This paper makes the following contributions.

1) We initiate the study of self-adjusting distributed datastructures and introduce a formal model accordingly. Our model is simple but captures the fundamental tradeoff between the benefits of self-adjustments (namely shorter routing paths) and their costs (namely reconfigurations).
2) We present a self-adjusting distributed BST called *SplayNet*. *SplayNet* is a natural generalization of the classic splay tree algorithm which "splays" communication partners to their common ancestor. *SplayNet* is fully decentralized in the sense that all topological adjustments as well as routing are local.
3) We formally analyze the performance of *SplayNet* (in terms of amortized costs). In particular, we show that the overall cost is upper bounded by the *empirical entropies* of the sources and destinations in the communication pattern; a simple lower bound follows from *conditional empirical entropies*. We also prove the optimality of our approach in specific case studies, e.g., when the communication pattern follows a product distribution. Finally, we also present a dynamic programming algorithm to optimally solve the offline problem variant in polynomial time.
4) We introduce novel lower bound techniques to study the limitations of self-adjusting networks. These techniques are based on *interval cuts* and *edge expansion*, and may

S. Schmid is with the TU Berlin & Telekom Innovation Laboratories, Germany, e-mail: stefan@net.t-labs.tu-berlin.de

C. Avin and Z. Lotker are with the Department of Communication Systems Engineering, Ben-Gurion University of the Negev, e-mail: {avin,zvilo}@bgu.ac.il.

C. Scheideler is with the University of Paderborn, Germany, e-mail: scheideler@upb.de

M. Borokhovich is with the University of Texas at Austin, USA, e-mail: michaelbor@gmail.com

B. Haeupler is with the School of Computer Science, Carnegie Mellon University, e-mail: haeupler@cs.cmu.edu.

Conference versions of this work were published in [3], [5].

Research supported by the German Israeli G.I.F. Research Grant I-1245-407.6/2014.

* The first two authors contributed equally to this work.

be of independent interest and find applications beyond the setting studied in this paper.

5) Finally, we initiate the discussion of more complex self-adjusting networks, namely topologies consisting of multiple trees. We make the interesting observation that in contrast to classic datastructures where the self-adjustment benefits of multiple trees is limited, in a distributed setting, a single additional BST can sometimes reduce the amortized cost dramatically.

In summary, our work shows that while some algorithmic concepts of traditional splay trees can be generalized to networks, the distributed setting requires new analytical tools. Moreover, our results highlight that self-adjustment benefits can indeed be reaped also in the context of networks; for multi-tree networks, these benefits can even be significantly higher than in classic datastructures.

In general, we regard our study as a first step, and believe that our model and results open a rich area for future research.

B. Paper Organization

The upcoming Section II introduces our formal model and provides the reader with the necessary background. Section III describes an offline algorithm to compute optimal static distributed BSTs and presents the *SplayNet* approach. Section IV derives entropy-based upper and lower bounds on the performance of *SplayNets*, and Section V studies the locality and convergence properties of the *SplayNet* algorithm in specific scenarios. Section VI then derives improved lower bounds which allow us to show the optimality of *SplayNets* in additional scenarios. In Section VII we initiate the discussion of datastructures based on multiple BSTs. After reviewing related work in Section VIII, we conclude our contribution in Section IX. In the Appendix, some additional technical details are provided.

II. Model and Background

We consider a set of n nodes (or *peers*) $V = \{1, \ldots, n\}$ interacting according to a certain *communication pattern*. The pattern is modeled by $\sigma = (\sigma_0, \sigma_1 \ldots \sigma_{m-1})$: a sequence of m *communication requests* where $\sigma_t = (u, v) \in V \times V$, with source u to destination v, henceforth sometimes denoted by $\mathrm{src}(\sigma_t)$ and $\mathrm{dst}(\sigma_t)$, respectively.

Our goal is to find a *communication network* G which connects the nodes V according to the communication pattern: additionally, G must be chosen from a certain family of *desired topologies* $\mathcal{G}$, for example, the set of tree topologies (the focus of this paper), expander graphs, or low-diameter networks, etc. Each topology $G \in \mathcal{G}$ is a graph $G = (V, E)$. We distinguish two problem variants: (1) A *static variant* where G can be optimized towards the communication pattern σ in the sense that it can exploit, e.g., long-term characteristics of σ, however, G is fixed and cannot change over time. (2) A *self-adjusting variant* where G can be adapted over time.

Generally, it is desirable that networks are adjusted smoothly, and we are interested in *local transformations*: changing communication pattern leads to "local" adjustments of the communication graph over time.

As mentioned above, this paper focuses on a setting where $\mathcal{G}$ represents the set of *binary search trees* (BSTs), henceforth sometimes simply called *tree networks*. Besides their simplicity, BSTs 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 for details.

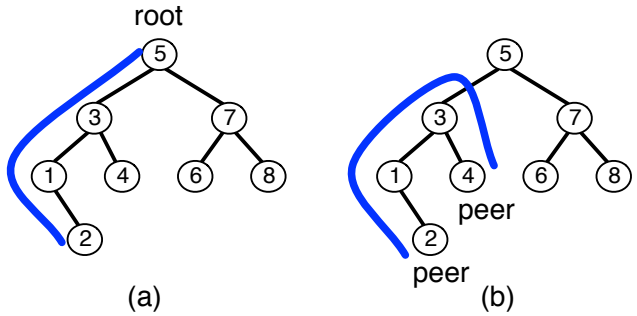


Fig. 1: (a) Classic BST vs (b) distributed BST: Classic splay tree datastructures optimize the distance of the elements from the root (the lookup cost), while in a distributed datastructure, communication occurs between arbitrary nodes (the *peers*). Also note that in a BST network, requests also travel upwards in the tree; nevertheless, as we will see, routing decisions are completely local.

The local transformations of tree networks are called *rotations*. Informally, a rotation in a sorted binary search tree changes up to three adjacency relationships, 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).

For our formal analysis, we consider a simplified synchronous model where first a communication request arrives, then local network transformations can be performed, and finally, the request is satisfied (i.e., the traffic routed). In this paper, we are often interested in a setting where the requests σ are drawn at random from a fixed but unknown communication matrix. Concretely, we will sometimes regard the communication requests σ as inducing a *request graph* $\mathcal{R}(\sigma) = (V(\sigma), E(\sigma))$ over the vertices V; the edges $E(\sigma)$ of $\mathcal{R}(\sigma)$ are annotated with frequency information. (When clear from the context, we will often omit σ in $\mathcal{R}(\sigma)$, $V(\sigma)$, $E(\sigma)$, 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 $E = \{\sigma_t : t \in [0, \ldots, m-1]\}$. 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 form a tree (e.g., a multicast tree), a complete graph, or a set of disconnected components (e.g., describing a clustered communication pattern).

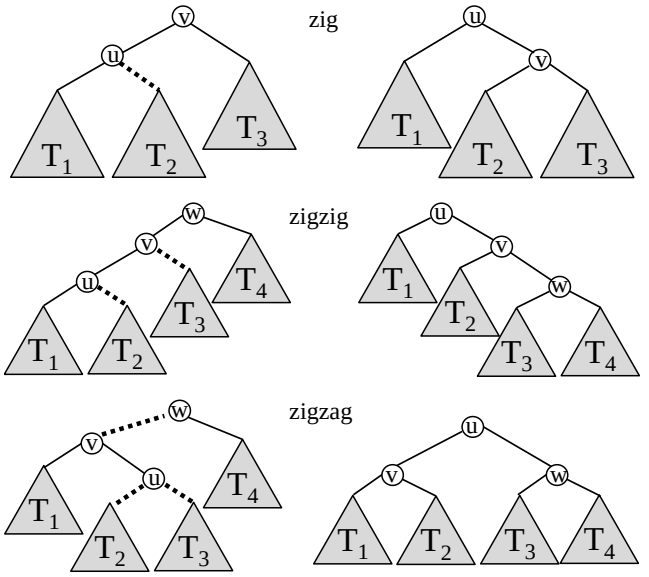


Fig. 2: Basic rotations of splay trees. The dashed bold lines indicate adjacency relationships which are not maintained during the operation.

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.

We are interested in the fundamental tradeoff between the benefits of self-adjusting algorithms (i.e., shorter routing paths) and their costs (namely reconfiguration costs). We introduce a most simple, linear cost model that captures this tradeoff. Concretely, we denote the cost of the network transformations at time t by $\rho(\mathcal{A}, G_t, \sigma_t)$, and we denote 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).

We need the following formal 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 $\sigma = (\sigma_0, \sigma_1 \ldots \sigma_{m-1})$ 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.,* $\max_{G_0,\sigma} \mathrm{Cost}(\mathcal{A}, G_0, \sigma)$.

Similarly to classic splay trees [22], 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.*

A. Entropy and Empirical Entropy

The *entropy* of the communication pattern σ turns out to be a useful parameter to evaluate the performance of self-adjusting *SplayNets*. For a discrete random variable X with possible values $\{x_1, \ldots, x_n\}$, 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 $H(\hat{Y}|\hat{X})$.

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 BSTs.

Theorem 1 ([16])**.** *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 [15] 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.

Knuth [16] first gave an algorithm to find an optimal BST, but Mehlhorn [17] proved that a simple greedy algorithm is near optimal with an explicit bound:

Theorem 3 ([17])**.** *Given s, there is a BST which can be computed using a balancing argument and which has the amortized cost that is at most*

$$\text{Cost}(\perp, T, s) \leq 2 + \frac{H(\hat{Y})}{1 - \log(\sqrt{5} - 1)} \tag{3}$$

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

We can adapt some tools developed for dynamic binary search trees also in our generalized setting. In particular, as we will see, our *SplayNet* algorithm is a natural generalization of the splay tree algorithm: it limits operations to the smallest subtree 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 [22] 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* [22] by Sleator and Tarjan. It is reviewed in the following. In [22], 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 [22])**.** *The amortized time to splay a tree with root r at a node u is at most $3 \cdot (\mathrm{r}(r) - \mathrm{r}(u)) + 1 = O(\log(\mathrm{s}(r)/\mathrm{s}(u)))$.*

Algorithm 1 Algorithm SPLAY

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

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

Theorem 4 (Static Optimality Theorem [22] - rephrased)**.** *Let* SPLAY *denote the* SPLAY *algorithm. Let s be a sequence of lookup requests where each item is requested at least once, then for any initial tree T,* $\text{Cost}(\text{SPLAY}, T, s) = O(H(\hat{Y}))$ *where $H(\hat{Y})$ is the empirical entropy of s.*

III. SPLAY NETWORKS

We first acquaint ourselves with the distributed BST model by studying optimal static topologies. Subsequently, we present the *SplayNet* approach and introduce the simple SPLAYNET algorithm to self-adjust the network.

A. Optimal Static Distributed BST

Given a certain communication pattern or "guest graph", the optimal BST can be computed in polynomial time using a dynamic programming approach. The main insight needed is that the problem for the entire tree can be decomposed into optimal subproblems for smaller trees, and that the demand towards 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.

Concretely, the optimal static tree T which minimizes the sum of the weighted node distances

$$\min \sum_{(u,v) \in \mathcal{R}(\sigma)} \mathrm{d}_T(u, v)$$

can be computed using 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 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 computed as follows:

$$\textit{Cost}(T_I, W_I) \quad = \quad \left[\sum_{u,v \in I} (\mathrm{d}(u, v) + 1) \cdot w(u, v) \right] + D_I \cdot W_I$$

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 two contiguous and adjacent 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):

$$\begin{aligned} \mathit{Cost}(T_I^*, W_I) &= \min_{x \in I = I' \sqcup I''} \mathit{Cost}(T_{I'}^*, W_{I'}) \\ &\quad + \mathit{Cost}(T_{I''}^*, W_{I''}) \\ &\quad + \sum_{v \in I'} W_{I'}(v) + \sum_{v \in I''} W_{I''}(v). \end{aligned}$$

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.)

We have derived the following result.

Theorem 5. *The optimal distributed BST for a sequence σ can be computed in time $O(n^3)$, where n is the number of nodes.*

B. Self-Adjusting Distributed BSTs

Let us now consider *self-adjusting* BST networks. The *SplayNet* algorithm presented in this paper is a natural generalization of the classic splay tree algorithm. It is based on a *double splay* strategy: similarly to classic splay trees, *SplayNet* aggressively moves communicating nodes together; however, rather than splaying nodes to the root of the BST, locality is preserved in the sense that the source and the destination node are only rotated to their *common ancestor* (of the *subtree* covering the 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. When a request (u, v) occurs, *SplayNet* 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 [22] (see Figure 2). 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 *SplayNet* 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.)

The formal algorithm listing of *SplayNet* is shown in Algorithm 2.

Algorithm 2 Algorithm *SplayNet*

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 $T'(u)$

IV. Basic Entropy Bounds

This section provides a formal analysis of self-adjusting distributed splay networks. The amortized costs of *SplayNet* depends on the empirical entropy of the pattern σ, and a simple cost upper bound can be obtained by adapting Lemma 1. The amortized communication cost of *SplayNet* can be upper bounded by the entropy of the sources and destinations of the requests.

Theorem 6. *Let σ be an arbitrary sequence of communication requests, then for any initial tree T_0,*

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

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

Proof: 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 SplayNet separately: first splaying the source to the common ancestor and second splaying the destination to the new common ancestor. The cost $\mathit{Cost}(\textsc{SplayNet}, T_0, \sigma)$ can be computed as

$$\frac{1}{m} \sum_{t=1}^{m} \left(\textbf{splay } \mathrm{src}(\sigma_t) \text{ to } w_t + \textbf{splay } \mathrm{dst}(\sigma_t) \text{ to } w_t' \right)$$

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 $\mathit{Cost}(\textsc{SplayNet}, T_0, \sigma)$ as:

$$\begin{aligned} \mathit{Cost}(\textsc{SplayNet}, T_0, \sigma) &\leq \frac{1}{m} \sum_{t=1}^{m} \big(3 \cdot (r(\mathrm{src}(\sigma_t)) - r(w_t)) \\ &\quad + \; 3 \cdot (r(\mathrm{dst}(\sigma_t)) - r(w_t')) + 2 \big) \\ &= O\Big(\frac{1}{m}(2m + \sum_{i=1}^{n} s(i) \log(\frac{m}{s(i)}) \\ &\quad + \sum_{j=1}^{n} d(j) \log(\frac{m}{d(j)}) \\ &= O(H(\hat{X}) + H(\hat{Y})) \end{aligned}$$

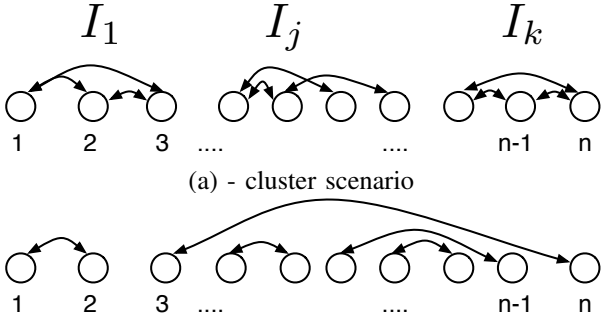
■

A lower bound on the cost can be stated in terms of the *conditional empirical entropy* of the request sequence.

Theorem 7. *Given a request sequence σ, for any optimal (binary search) tree network T:*

$$\text{Cost}(\perp, T, \sigma) = \Omega(H(\hat{Y}|\hat{X}) + H(\hat{X}|\hat{Y})) \tag{4}$$

Proof: For any node $x \in V$, let $\hat{Y}_x$ denote the frequency distribution of the destinations *given that* the source is x.



(a) - cluster scenario

(b) - non-crossing matching scenario

Fig. 3: (a) An example of a request graph, $\mathcal{R}(\sigma)$, for the cluster scenario, (b) an example of a request graph, $\mathcal{R}(\sigma)$, for the non-crossing matching scenario

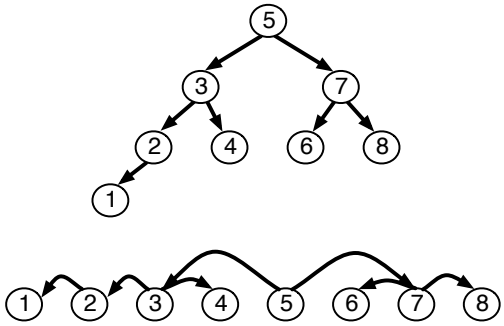


Fig. 4: An example of a request graph, $\mathcal{R}(\sigma)$, for the multicast tree scenario on 8 nodes. The same $\mathcal{R}(\sigma)$ is shown in two layouts.

Consider an optimal tree T with root x. Following Theorem 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

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

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 6 and 7 are relatively general and there remains a gap between upper and lower bound. However, there are interesting problem instances where this simple bound is already tight. For example, observe that *SplayNet* 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. SplayNet *is asymptotically optimal if σ follows a product distribution.*

However, there are also simple examples where the gaps remain open. Figure 6 gives three exemplary request patterns $\mathcal{R}(\sigma)$ which help us to better understand the gap between our upper and lower bound. All three request graphs in Figure 6 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 VI 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.

V. Case Studies: Locality and Optimality

The section provides insights into the properties of our algorithms in different specific settings. In particular, using different case studies, we show that our approach sometimes exhibits a desirable local convergence to the optimal network. The section also serves the purpose of introducing some analytical tools that are useful to study SplayNets.

A. Cluster Scenario

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

Definition 3 (Cluster Scenario)**.** *In a* cluster scenario *the communication pattern σ partitions the nodes into 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: $(u, v) \in \sigma \rightarrow \exists j : \quad u, v \in I_j$. A* maximal *cluster scenario is a cluster scenario where no intervals can be divided into two intervals.*

See Figure 3 (a) for an example of a maximal cluster scenario. For this case we are able to prove the following.

Theorem 8. *In a maximal cluster scenario σ,* SplayNet *features the following two properties:*

1) SplayNet *will eventually construct a tree network 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.*

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

For the (maximal) cluster scenario σ and a BST T, we consider the potential function $\phi = \sum_{j=1}^{k} \mathrm{out}(T(I_j))$. We will prove the theorem by first showing 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: a path which only includes nodes from I_j; ϕ remains zero after such a request. Subsequently, we will show

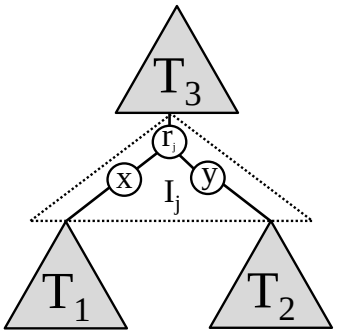


Fig. 5: Illustration for proof of Claim 1 and Theorem 8. $T(r_j)$ is the tree rooted at r_j (and includes T_1 and T_2).

that when $\phi > 0$, *SplayNet* cannot increase ϕ, and there exists a request in σ which will reduce the potential.

We start with the following claim.

Claim 1. *Consider I_j s.t. $\mathrm{out}(T(I_j)) = 0$. Then this property is invariant for all future requests in σ and $\mathrm{out}(T(I_j))$ will always remain 0.*

Proof: A generic situation for $T(I_j)$ is illustrated in Figure 5. Each node not in I_j is in one of 3 possible locations: in one of the (possibly empty) subtrees 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, and 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. Since $\mathrm{out}(T(I_j)) = 0$, there are no requests involving T_3 so all requests remain within $T(I_j)$ and so $\mathrm{out}(T(I_j))$ will remain 0 for all future requests. ■

Now observe that if $\phi = 0$, every $\mathrm{out}(T(I_j)) = 0$, and so ϕ will remain zero also in the future.

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 $\mathrm{out}(T(I_j))$ does not change. If $u, v \in T_i$ for $i = 1, 2$ or 3, then again $\mathrm{out}(T(I_j))$ 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 splaying u and v, $\mathrm{out}(T(I_j))$ will decrease by one. Therefore ϕ can only decrease. Now if $\phi > 0$, there is some j for which $\mathrm{out}(T(I_j)) > 0$. Take the request (u, v) that is a witness; after this request $\mathrm{out}(T(I_j))$ will decrease by one, so ϕ will decrease. So overall given a cluster σ, ϕ will decrease to zero. To conclude the theorem we show that if $\phi = 0$ then 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. Assume by contradiction that this is not the case and there is a j and $(u, v) \in I_j$ (assume w.l.o.g. that $u < v$) s.t. the path between them visits a node $w \in I_i$. Also assume w.l.o.g. that $w < u$. Now it must be the case that either $T(r_j) \subset T(r_i)$ or $T(r_i) \subset T(r_j)$. Let's assume w.l.o.g. that $T(r_j) \subset T(r_i)$ so it must be that case that $r_j < w$. Since it is a maximal cluster scenario there must be a path in σ from a node $x \in I_i$, $x \geq w$ to a node $y \in I_i$, $y \leq r_i$ (otherwise the cluster I_i can be divided into two clusters). Since all nodes in I_i with larger ID than w must be in $T(r_j)$ and since $r_i \notin T(r_j)$ there must be a request along the path from x to y witnessing that $\mathrm{out}(T(I_j)) > 0$. Contradiction. A similar observation can be made for the case $T(r_i) \subset T(r_j)$, to get a contradiction for $\mathrm{out}(T(I_i)) > 0$.

■

B. Non-crossing Matching Scenario

SplayNet sometimes achieves an optimal performance by converging to the optimal static network implied by $\mathcal{R}(\sigma)$. 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 other optimal scenarios in more detail.

For a request (u, v) in σ, let $I_{(u,v)}$ denote the interval $[\min(u, v), \max(u, v)]$.

Definition 4 (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 $\mathcal{R}(\sigma)$ must describe a matching, and for each request (u, v) there are no other requests that enter or leave (i.e., *cross*) the interval $I_{(u,v)}$. Figure 3 (b) provide an example of a non-crossing matching scenario. If $\mathcal{R}(\sigma)$ describes a non-crossing matching scenario, *SplayNet* will converge to an optimal solution. Formally:

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

Proof: To any request pair $p_i = (u_i, v_i)$ we assign a *level* $L(p_i)$ depending on the number of pairs $p_j = (u_j, v_j)$ it is nested in, i.e., $L(p_i) = |\{p_j : I_{p_i} \subsetneq I_{p_j}\}|$. Pairs with level 0 are the outmost pairs, pairs at level 1 are nested in a single other pair and so on. We will prove by induction on the level that all pairs become (and stay) adjacent. Our induction hypothesis is that pairs at level i coverage and we will show that once that happens, pairs at level $i + 1$ will converge next. First we show that pairs at level 0 converge. Notice that pairs do not intersect (i.e., cross), so the pairs at level 0 represent a clustered scenario. By Theorem 8, after the convergence of clusters, requests within each interval (of pair at level 0) will remain only within the interval. After the cluster convergence, we claim that after a pair $p_i = (u_i, v_i)$ of level 0 communicated, it will stay adjacent forever. To see this, consider any other pair $p_j = (u_j, v_j)$. If p_j is outside the cluster of p_i we are done. Let p_j be inside the interval of p_i, i.e., $I_{(u_j,v_j)} \subsetneq I_{(u_i,v_i)}$. By the definition of *SplayNet*, after a request (u_i, v_i), v_i is the right (resp. left) child of u_i if $u_i < v_i$ (resp. $u_1 > v_1$). We will show that this implies that the nodes u_j, v_j must be both in the same subtree, and can hence not change the adjacency relationship of u_i and v_i anymore. The claim follows by case distinction: (1) If $u_i < v_i$, the left

subtree of v_i is the only subtree which can contain nodes w with $u_i < w < v_i$; however, these are exactly the nodes that can fulfill the non crossing property so both u_j and v_j must be in this subtree. If $u_i > v_i$, the right subtree of v_i is the only subtree which can contain nodes w with $u_1 < w < v_1$ so both u_j and v_j must be in this subtree. Therefore the induction hypothesis holds for level 0.

We can now prove the induction step. Assume the links of level up to i are stable. Note that within each cluster the pairs of level $i+1$ are also disjoint and correspond to a clustered scenario. Essentially, then the previous argument still holds, and first each cluster of levels larger than i will converge, by Theorem 8, and then, after a pair from level $i+1$ will communicate, it will never break apart again since all higher level pairs are in the same subtree. ■

C. Multicast Scenario

To give one more example where *SplayNet* is optimal, we consider the multicast tree scenario. For this scenario, we may assume that an overlay network of a binary search tree structure is constructed to facilitate in-network duplication. The same tree may be used by many multicast sources and different receivers, hence the local (hop-by-hop) source and destination pairs (i.e., communication requests) are the endpoints of the tree edges.

Definition 5 (Binary Multicast Tree Scenario)**.** *In a* binary multicast tree scenario*, it holds for the communication pattern σ that the request graph $\mathcal{R}(\sigma)$ forms a rooted and sorted binary tree.*

See Figure 4 for an example of a multicast scenario. For this case as well, we are able to prove optimality.

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

Proof: Let $T(v)$ denote the subtree of v (including v) in the current *SplayNet*, and let $T_l(v)$ and $T_r(v)$ denote v's left and right subtree, respectively. Similarly, let $R(v)$ denote the subtree of v in the multicast tree $\mathcal{R}(\sigma)$, and let $R_l(v)$ resp. $R_r(v)$ denote the left resp. right subtree. Consider the path from the root of $\mathcal{R}(\sigma)$, denoted by v_i, to the maximum node in $\mathcal{R}(\sigma)$, denoted by v_1, and let this path be $P_m = v_i, \ldots, v_2, v_1$, where for $j = 1, \ldots, i$, v_j is the parent of v_{j-1}.

We start with a fundamental invariant.

Claim 2. *Let $v_j \in P_m$ denote a node in the path from the root to the maximum node. Then it holds that once $T(v_j) = R(v_j)$, the parent of v_j in T will be v_{j+1}. These properties will remain invariant during all future requests.*

Proof: We first prove the identical parent property. If $j = i$ the result holds trivially since T and $\mathcal{R}(\sigma)$ are identical. Let $j < i$. For the sake of contradiction, assume that $T(v_j) = R(v_j)$, that the parent of v_j in $\mathcal{R}(\sigma)$ is v_{j+1}, but that the parent of v in T is $x \neq v_{j+1}$. It holds that $v_{j+1} < v_j$. Since both T and R are valid BSTs, x cannot be in the subtree $T(v_j) = R(v_j)$, and it must hold that $v_{j+1} < x < v_j$. However, the fact that v_{j+1} is a parent of v_j in $\mathcal{R}(\sigma)$ implies that x must be a decedent of v_j in $\mathcal{R}(\sigma)$, which contradicts our assumption that $T(v_j) = R(v_j)$.

It remains to show that the trees will also remain the same in the future. Consider any future request (y, z). Since $T(v_j) = R(v_j)$ and there are no requests leaving $R(v_j)$ beside v_j and its parent, v_j cannot be on a path between y to z (which includes their least common ancestor), and hence, $T(v_j)$ will remain unchanged during any splay operation. ■

We can also make the following observation.

Claim 3. *Let $v_j \in P_m$ denote a node in the path from the root to the maximum node for $j < i$. Once the right subtrees $T_r(v_{j-1})$ and $R_r(v_{j-1})$ of the node v_j are the same, then after a request (v_{j+1}, v_j) from v_j's parent in $\mathcal{R}(\sigma)$ the subtree $T_l(v_j)$ will have exactly the same nodes as $R_l(v_j)$. This property will remain invariant during all future requests.*

Proof: Assume $T(v_{j-1}) = R_r(v_{j-1})$. Then by Claim 2, v_j is the parent of v_{j-1} in T, and this still holds after the request (v_{j+1}, v_j). Recall that $v_{j+1} < v_j$ and we are considering the right most tree branch. However, this implies that all the nodes between v_{j+1} and v_j must be situated in the same subtree, the left subtree of v_j So $T_l(v_j)$ will have the exact same nodes as $R_l(v_j)$.

Now consider any future request (y, z). Since $T(v_j)$ and $R(v_j)$ have now the same set of nodes, v_j is not on a path between y to z (which includes their least common ancestor), so $T(v_j)$ will indeed remain unchanged during such a request. ■

We are now ready to prove the theorem by induction over the number of nodes. That is, we will show that for each size k of $\mathcal{R}(\sigma)$, the theorem holds.

Induction Base: The claim trivially holds for the cases where we only have one node ($k = 1$) or one request ($k = 2$).

Induction Step: Assume that the hypothesis is true for $1, \ldots, k-1$; we will now show that it is also true for k nodes. We will prove, again by induction, but now on j, that $T(v_j)$ will become (and stay) $R(v_j)$: one node after the other along this path will stabilize.

Base case, $j = 1$: Since v_1 is the maximum node it has no right child both in T and $\mathcal{R}(\sigma)$. By Claim 3 after the request (v_2, v_1) $T(v_1)$ and $R(v_1)$ will have the same set of nodes. Since $R(v_1)$ has less than k nodes by the main induction hypothesis we will eventually have $T(v_1) = R(v_1)$.

Induction step: Assume the claim is true for $j-1$. If $j < i$ then after the request (v_j, v_{j-1}), by Claim 3 $T_l(v_j)$ will consist of the same nodes as $R_l(v_j)$, and by the main induction hypothesis (since the size of $R_l(v_j) < k$) and Claim 3, $T_l(v_j)$ will converge to $R_l(v_j)$. Then we will have $T(v_j) = R(v_j)$.

To prove the claim also holds for the root, we can simply apply a variant of Claim 3 also to a path from the root to the minimum node. Given that both subtrees of the root stabilize, also the root will be stabilized. ■

VI. Improved Lower Bounds

This section introduces two techniques to derive more powerful lower bounds: one is based on interval cuts and one on edge expansion. These lower bounds also imply the optimality of our approach for a larger class of scenarios.

A. Interval Cuts Bound

The first lower bound leverages an intriguing connection to *request graph cuts*. It is intuitively clear that if the request graph $\mathcal{R}(\sigma)$ 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 6 (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., $\bar{I}_j^\ell = V \backslash I_j^\ell$. 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* $\mathcal{R}(\sigma)$ where a directed edge represent a source-destination pair in σ and edge weights represent the frequency of the communication requests. An interval cut (as defined above) in $\mathcal{R}(\sigma)$ is therefor a set of source-destination pairs, so for an interval cut $\tilde{c}$ the conditional frequency distribution of the sources is denoted $\hat{X}_{\tilde{c}}$ and the one of the destinations is denoted $\hat{Y}_{\tilde{c}}$.

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:*

$$\text{Cost}(\perp, T, \sigma) = \Omega\left(\max_i \min_{j,\ell} \tilde{H}(\text{Cut}_{in}(I_j^\ell, \mathcal{R}(\sigma)))\right) \quad (5)$$
$$\text{Cost}(\perp, T, \sigma) = \Omega\left(\max_i \min_{j,\ell} \tilde{H}(\text{Cut}_{out}(I_j^\ell, \mathcal{R}(\sigma)))\right) \quad (6)$$

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. (6), as Eq. (5) can be proved in an analogous way. Let $\tilde{c}^*$, be a cut that maximizes Eq. (6) 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 5). 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 4. The result now follows from Property 2, since all the requests from inside $I_{j^*}^{\ell^*}$ to outside $\bar{I}_{j^*}^{\ell^*}$ 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* [21] and the *edge expansion* [14] 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 7 (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 $\text{src}(S)$ 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 $\text{dst}(S)$ is defined over the destinations in $E(S,\bar{S})$, $\text{src}(S)$ 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:*

$$\varphi_H(S) = W(S)\left(H(\text{src}(S)) + H(\text{dst}(S))\right) \quad (7)$$

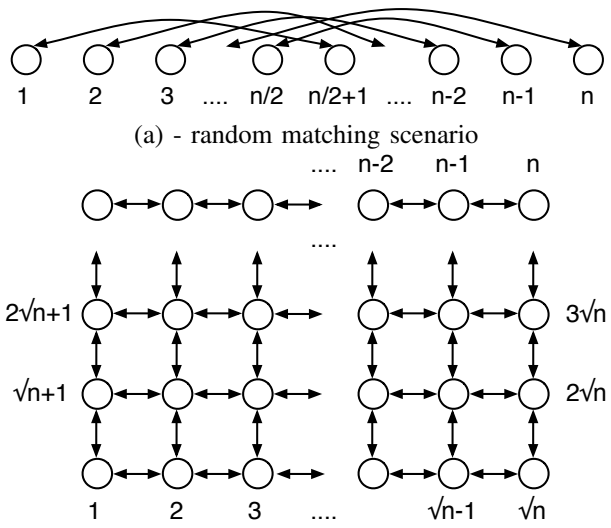
The conductance entropy *of a graph is defined as:*

$$\phi_H(G) = \min_{S\subseteq V} \varphi(S) \quad (8)$$

We can now claim the following:

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

$$\text{Cost}(\perp, T, \sigma) = \Omega(\phi_H(\mathcal{R}(\sigma))) \quad (9)$$



(a) - random matching scenario

(b) - 2-dimensional grid scenario

Fig. 6: (a) An example of a requests graphs, $\mathcal{R}(\sigma)$, for the a random matching scenario (b) An example of a requests graphs, $\mathcal{R}(\sigma)$, for a 2-dimensional grid scenario. Both cases are of bounded degree graphs, i.e., $H(\hat{Y}|\hat{X})$ and $H(\hat{X}|\hat{Y})$ are upper bounded by a constant so the lower bound of Theorem 7 is not tight. Theorems 11, 12 and 13 provide better bounds.

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* [14] of a graph G is defined as:

$$h(G) = \min_{0<|S|\leq \frac{n}{2}} \frac{E(S,\bar{S})}{|S|} \tag{10}$$

For the special case where the request graph $\mathcal{R}(\sigma)$ 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. $\mathcal{R}(\sigma)$ is a d-regular graph with uniform weights and edge expansion α then* $\text{Cost}(\perp, T, \sigma) = \Omega(\log(\alpha n))$.

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 6 and elaborate on their bounds using the above theorems.

C. Examples

Consider a request graph that forms a 2-dimensional grid as in Figure 6 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 6 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 is the case for edge expansion), we can get a tight lower bound of $\Omega(\log n)$ for these cases.

VII. Multiple BSTs

To round off our first study of self-adjusting networks, in the following, we want to initiate the discussion of slightly more complex network topologies. A natural next step towards more scalable networks is to consider *multiple* BSTs. In the following, we will write *SplayNet*(k) to denote an overlay network consisting of k distributed BSTs. Formally, we define:

Definition 8 (*SplayNet*(k))**.** *Consider a set $\{T_1, T_2, \ldots, T_k\}$ of k BSTs.* SplayNet(k) *is an overlay over the peer set V where connections are given by the BST edges, i.e., $E = \bigcup_{i=1}^{k} E(T_i)$.*

Note that our discussions so far hence revolved around *SplayNet*(1). The obvious question we want to address is: Can we improve the amortized costs when using larger k? We will answer this question affirmatively, and highlight another fundamental difference to classic splay trees. In the following, we will focus on the static variant only, and denote the routing cost under a sequence σ by *Cost*(*SplayNet*$(k), \sigma)$.

For communication requests σ let $s(v)$ be the frequency of peer v as a *source* in σ, similarly let $d(v)$ be the frequency of peer i as a *destination* and $f(u,v)$ be the frequency of the request (u,v) in σ. Define $z(v) = (s(v) + d(v))/2$ and note that by definition $\sum_{v\in V} z(v) = 1$. Let $\hat{Z}$ be a random variable (r.v.) with a probability distribution defined by the $z(v)$s. For any k partition of the requests in σ into disjoint sets $S_1, S_2, \ldots, S_k$, let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be the frequency measure of the partition, i.e., $\alpha_i = \sum_{(u,v)\in S_i} f(u,v)$.

First, we can prove a new bound on the optimal static *SplayNet*(1):

Theorem 14. *Given σ, there exists a* SplayNet(1) *such that:*

$$\text{Cost}(\text{SplayNet}(1), \sigma) \leq 4 + \frac{2H(\hat{Z})}{1 - \log(\sqrt{5}-1)}$$

where $H(\hat{Z})$ is the entropy of $\hat{Z}$ as defined earlier.

Proof: The result follows from Theorem 3 with some modifications. Consider a tree T and let $\ell(v)$ denote the distance of node $v \in V$ from the root. We will assume the following non-optimal strategy: each request (u,v) is first routed from u to the root and then from the root to v. Hence, the amortized cost of σ can be bounded as:

$$\begin{aligned} \textit{Cost}(\textit{SplayNet}(1), \sigma) &\leq \sum_{v\in V} s(v)\ell(v) + \sum_{v\in V} d(v)\ell(v) && (11)\\ &= 2\sum_{v\in V} z(v)\ell(v). && (12) \end{aligned}$$

Now given $z(v)$, the problem of finding the tree that minimizes $\sum_{v\in V} z(v)\ell(v)$ is exactly the lookup problem in a single tree and, using Theorem 3, the result follows. ∎

We can extend this upper bound to k BSTs.

Theorem 15. *Given σ, there exists a* SplayNet(k) *such that:*

$$\text{Cost}(\text{SplayNet}(k), \sigma) \leq 4 + \frac{2H(\hat{Z}) - 2H(\alpha_1, \alpha_2, \ldots, \alpha_k)}{1 - \log(\sqrt{5}-1)}$$

where $H(\hat{Z})$ is the entropy of $\hat{Z}$ as defined earlier.

Proof: Assume a non-optimal strategy where we partition σ into k disjoint sets of requests $S_1, S_2, \ldots, S_k$, and each request is routed on its unique BST. In each tree we use the previous method, and the messages are routed from the source to the root and from the root to the destination. For a subset S_i, $1 \leq i \leq k$, let $\hat{Z}_i$ denote the frequency measure defined as $\hat{Z}$, but limited to the requests in S_i. Now:

$$\textit{Cost}(\textit{SplayNet}(k), \sigma) \leq \sum_{i=1}^{k} \alpha_i \left(4 + 2\frac{H(\hat{Z}_i)}{1 - \log(\sqrt{5}-1)}\right) \quad (13)$$

$$= 4 + \frac{2}{1 - \log(\sqrt{5}-1)} \sum_{i=1}^{k} \alpha_i H(\hat{Z}_i) \quad (14)$$

$$= 4 + \frac{2}{1 - \log(\sqrt{5}-1)} \left(H(\hat{Y}) - H(\alpha_1, \alpha_2, \ldots, \alpha_k)\right) \quad (15)$$

where the last step is based on the decomposition property of entropy. ∎

The upper bound in Theorem 15 improves only marginally for larger k, and we will show in the following that this is overly conservative: There are situations where a single additional BST can reduce the optimal communication cost of *SplayNet*(k) from worst possible (e.g., $\Omega(\log n)$) to a constant cost in *SplayNet*$(k+1)$.

Theorem 16. *A single additional BST can reduce the amortized costs from a best possible value of* $\Omega(\log n)$ *to* $O(1)$.

The formal proof appears in Appendix B. Essentially, it follows from the two BSTs $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$ shown in Figure 7: obviously, the two BSTs can be perfectly embedded into *SplayNet*(2) consisting of two BSTs as well. However, embedding the two trees at low cost in one BST is impossible, since there is a large cut in the identifier space. See the proof for details.

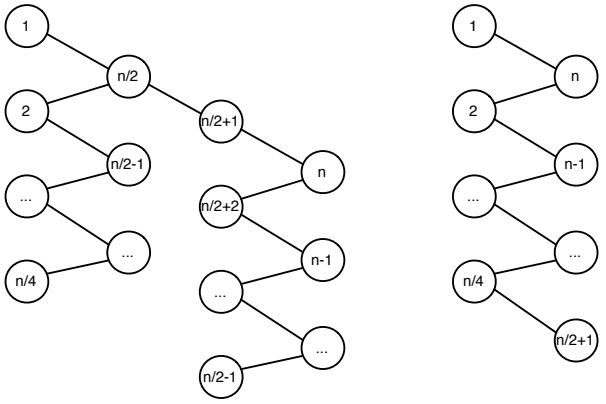


Fig. 7: A request sequence σ originating from these specific two trees can yield high amortized costs on *SplayNet*(1) but low on *SplayNet*(2) .

Interestingly, it turns out that while having multiple BSTs can significantly reduce the routing cost (see Theorem 16), the benefit of having parallel BSTs is rather limited in the context of classical lookup datastructures, i.e., if all requests originate from a single node (the root). Consider a sequence $\sigma = (v_0, v_1, \ldots, v_{m-1})$, $v_i \in V$ of *lookup* requests, and $|V| = n$. Theorem 2 can be generalized to k parallel lookup BSTs.

Theorem 17. *Given* σ*, for any* SplayNet(k)*:*

$$\text{Cost}(\text{SplayNet}(k), \sigma) \geq \frac{H(\hat{Y}) - \log k}{\log 3}, \quad (16)$$

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

Proof: Let $f(i)$ denote the number of times the node v_i appeared in the lookup sequence σ. The empirical frequency distribution is $\hat{p}(i) = f(i)/m$ for all i, and the entropy is given by $H(\hat{Y}) = H(\hat{p}(1), \ldots, \hat{p}(n))$. Since it is sufficient to serve a node by one BST only, we can assume w.l.o.g. each BST $T_i \in$ *SplayNet*(k) is used to serve the lookup requests for a specific subset $V_i \in V$, and that $\bigcap_{i=1}^{k} V_i = \emptyset$ and $\bigcup_{i=1}^{k} V_i = V$.

Let $\hat{Y}_i$ be the empirical measure of the frequency distribution of the nodes in V_i with respect to the lookup sequence σ. Using entropy decomposition property, we can write: $H(\hat{Y}) = H(\hat{p_1}, \hat{p_2}, \ldots, \hat{p_n}) = H(\alpha_1, \alpha_2, \ldots, \alpha_k) + \sum_{i=1}^{k} \alpha_i H(\hat{Y}_i)$, where $\alpha_i = \sum_{v_j \in V_i} \hat{p}_j$.

Always performing lookups on the optimal BST, we get:

$$\begin{aligned} \textit{Cost}(\textit{SplayNet}(k), \sigma) &\geq \sum_{i=1}^{k} \alpha_i \frac{1}{\log 3} H(\hat{Y}_i) \\ &= \frac{H(\hat{Y}) - H(\alpha_1, \alpha_2, \ldots, \alpha_k)}{\log 3} \\ &\geq \frac{H(\hat{Y}) - \log k}{\log 3}. \end{aligned}$$

∎

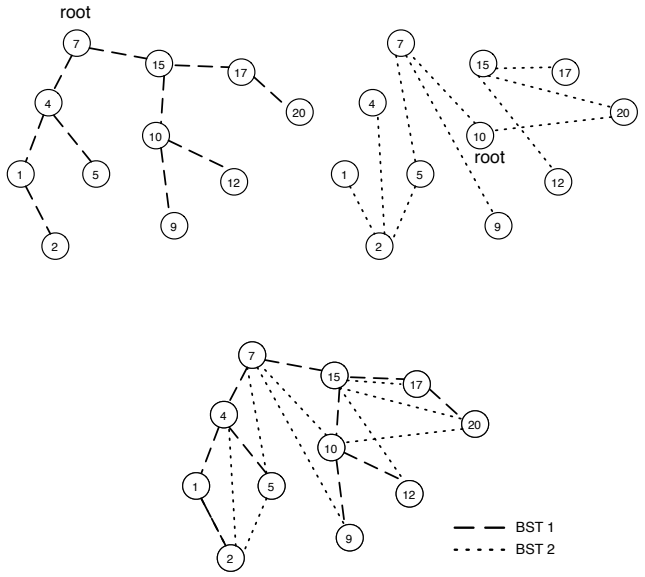


Fig. 8: Example of *SplayNet*(2) consisting of two BSTs. *Top left:* BST 1 (e.g., rooted at peer v_7). *Top right:* BST 2 (e.g., rooted at peer v_{10}). *Bottom:* combined BSTs.

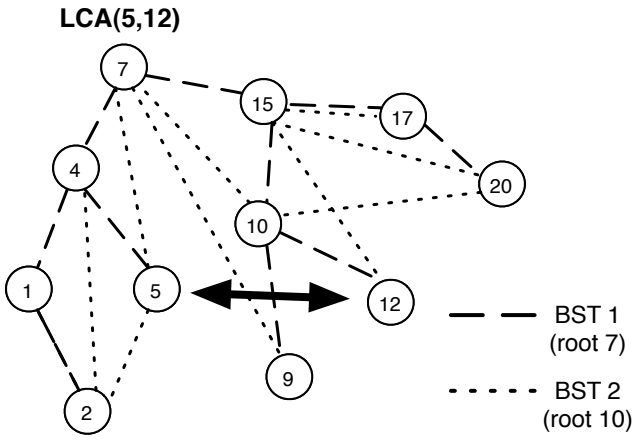


Fig. 9: Example for splay operation in *SplayNet*(2) of Figure 8.

Finally, we sketch an intuitive approach to make *SplayNet*(k) self-adjusting: initially, for each BST, we connect all nodes V at random and independently (see Figure 8 for an example). When communication requests occur, BSTs start to adapt: upon a communication request (u, v), we determine the BST T where u and v are closest (in case multiple trees yield similar cost, an arbitrary one is taken), as well as the least common ancestor w of u and v in T: $w := \alpha_T(u, v)$. Subsequently, u and v are splayed to the root of the subtree of T rooted at w, using our *SplayNet* procedure. Figure 9 gives an example: upon a communication request between peers v_5 and v_{12}, the two peers are splayed to their least common ancestor, peer v_7, in BST T_1.

VIII. Related Work

Our work builds upon classic literature on self-adapting *datastructures*, and in particular upon the work of Sleator and Tarjan on splay trees [22]. Since this seminal work, splay trees and their variants (e.g., *Tango trees* [7] or *multi-splay trees* [23]) have been studied intensively for many years (e.g. [1], [22]). Moreover, the related *dynamic optimality conjecture* [7], [23] is arguably one of the most interesting open problems in Theoretical Computer Science.

In contrast to the classical splay tree datastructures, 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, this paper is situated in the realm of *distributed datastructures* or *networking*.

The static variant of our problem can be seen as a graph embedding [8] or network design [10] problem. In particular, the problem of computing the optimal static network, that is, the binary search tree that minimizes the communication cost, is related to the classic *Minimum Linear Arrangement (MLA)* problem which asks for the embedding of an arbitrary "guest graph" on the line: the nodes of the guest graph must be mapped to the line such that the communication cost, i.e., the sum of the lengths of the projected edges, is minimized. In contrast to the MLA problem, in our case, we need to embed the guest graph on a binary tree which respects a searchable order. MLA [9] was originally introduced by Harper [12] in the context of error-correcting codes with minimal average absolute errors, and later used in many other domains such as for modeling of some nervous activity in the cortex [18] or job scheduling [19]. While there exist many interesting algorithms for this problem already, e.g., with sublogarithmic approximation ratios [11] or polynomial-time executions for special requests graphs [9], no non-trivial results are known about distributed and local solutions.

Bibliographic Note. Preliminary versions of the results presented in this paper appeared at DISC 2012 [20], IEEE IPDPS 2013 [6], and IEEE P2P 2013 [4].

IX. Conclusion

We regard our work as a first step towards the design of novel distributed datastructures and networks which adapt dynamically to the demand. In order to focus on the fundamental tradeoff between benefit and cost of self-adjustments, we purposefully presented our model in a general and abstract form, and many additional and application-specific aspects need to be addressed before our approach can be tested in the wild. The main theoretical simplification made 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 [22], we have focused on the amortized costs of *SplayNets*, and an interesting direction for future research regards the study of the achievable competitive ratio under arbitrary communication patterns.

References

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

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

[3] C. Avin, M. Borokhovich, and S. Schmid. Obst: A self-adjusting peer-to-peer overlay based on multiple bsts. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–5. IEEE, 2013.

[4] C. Avin, M. Borokhovich, and S. Schmid. Obst: A self-adjusting peer-to-peer overlay based on multiple bsts. In *Proc. 13th IEEE International Conference on Peer-to-Peer Computing (P2P)*, September 2013.

[5] C. Avin, B. Haeupler, Z. Lotker, C. Scheideler, and S. Schmid. Locally self-adjusting tree networks. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 395–406. IEEE, 2013.

[6] C. Avin, B. Haeupler, Z. Lotker, C. Scheideler, and S. Schmid. Locally self-adjusting tree networks. In *Proc. 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2013.

[7] 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.

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

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

[10] H. Farvaresh and M. Sepehri. A branch and bound algorithm for bi-level discrete network design problem. *Networks and Spatial Economics*, 13(1):67–106, 2013.

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

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

[13] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. 4th Conference on USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[14] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–562, 2006.
[15] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
[16] D. Knuth. Optimum binary search trees. *Acta informatica*, 1(1):14–25, 1971.
[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. International Colloquium on Automata, Languages and Programming (ICALP)*, 1991.
[20] S. Schmid, C. Avin, C. Scheideler, and B. Haeupler. Brief announcement: Splaynets (towards self-adjusting distributed data structures). In *Proc. 26th International Symposium on Distributed Computing (DISC)*, 2012.
[21] A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.
[22] D. Sleator and R. Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.
[23] 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 node 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 4. *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 5. *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 6. *Sorted binary tree networks facilitate local routing.*

Proof: Basically, a local routing can be achieved by exploiting Claim 4 as follows. Let us consider 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. Proof of Theorem 16

Theorem 16 (restated)**.** *A single additional BST can reduce the amortized costs from a best possible value of $\Omega(\log n)$ to $O(1)$.*

Proof: Consider the two BSTs $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$ in Figure 7. Clearly, the two BSTs can be perfectly embedded into *SplayNet*(2) consisting of two BSTs as well. However, embedding the two trees at low cost in one BST is hard, as we will show now.

Formally, we have that, where $V = \{1, \ldots, n\}$ (for an even n): $E_1 = (\{i, n/2-i+1\} : \forall i \in [1, n/4]) \cup (\{n/2-i, i+2\} : \forall i \in [0, n/4-2]) \cup (\{n/2+i, n-i\} : \forall i \in [0, n/4-1]) \cup (\{n-i, n/2+1+i\} : \forall i \in [0, n/4-1])$, and $E_2 = (\{i, n-i+1\} : \forall i \in [1, n/2]) \cup (\{n-i, i+2\} : \forall i \in [0, n/2-2])$ i.e., BST T_2 is "laminated" over the peer identifier space, and BST T_1 consists of two laminated subtrees over half of the nodes each. Consider a request sequence σ generated from these two trees with a uniform empirical distribution over all source-destination requests. Clearly, optimal *SplayNet*(2) will serve all the requests with cost 2, since all the requests will be neighbors in *SplayNet*(2). In order to show the logarithmic lower bound for the optimal *SplayNet*(1), we leverage the interval cut bound from Theorem 11 in [6]. Concretely, we will show that for any interval I of size $n/8 < \ell < n/4$ an $\Omega(\ell)$ (and hence an $\Omega(n)$) fraction of requests have one endpoint inside I and the other endpoint outside I. In other words, each interval has a linear cut, and the claim follows since the empirical entropy is $\Omega(\log n)$.

The proof is by case analysis. *Case 1*: Consider an interval $I = [x, x + \ell]$ where $x + \ell < n/2$. Then, the claim follows directly from tree T_2, as each node smaller or equal $n/2$ communicates with at least one node larger than $n/2$, so the cut is of size $\Omega(\ell)$. Similarly in *Case 2* for an interval $I = [x, x + \ell]$ where $x \geq n/2$. In *Case 3*, the interval $I = [x, x + \ell]$ crosses the node $n/2$, i.e., $x < n/2$ and $x + \ell > n/2$. Moreover, note that since $n/8 < \ell < n/4$, $n/4 < x$ and $x + \ell < 3n/4$ hold. The lower bound on the cut size then follows from tree T_1: each node $< n/2$ is connected to a node $\leq n/4$ outside the interval, and each node $> n/2$ is connected to a node $\geq n/2$ outside the interval. ∎

Stefan Schmid is a senior research scientist at the Telekom Innovation Laboratories (T-Labs) and at TU Berlin, Germany. He received his MSc and Ph.D. degrees from ETH Zurich, Switzerland, in 2004 and 2008, respectively. 2008-2009 he was a postdoctoral researcher at TU Munich and at the University of Paderborn, Germany, and in 2014, he was also a visiting professor at CNRS Toulouse, France, as well as at the Université catholique de Louvain (UCL), Belgium. Stefan Schmid is interested in fundamental problems in distributed systems. He serves as the editor of the Distributed Computing Column of the Bulletin of the European Association of Theoretical Computer Science (EATCS).

Chen Avin received the B.Sc. degree in Communication Systems Engineering from Ben Gurion University, Israel, in 2000. He received the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles (UCLA) in 2003 and 2006 respectively. He is a senior lecturer in the Department of Communication Systems Engineering at the Ben Gurion University for which he joined in October 2006. His current research interests are: graphs and networks algorithms, modeling and analysis with emphasis on wireless networks, complex systems and randomized algorithms for networking.

Christian Scheideler received his M.S. and Ph.D. degrees in computer science from the University of Paderborn, Germany, in 1993 and 1996. Afterwards, he was a postdoc at the Weizmann Institute, Israel, for a year, an Assistant Professor at the Johns Hopkins University, USA, for five years, and an Associate Professor at the Technical University of Munich, Germany, for three years. He is currently a Full Professor at the Dept. of Computer Science, University of Paderborn. He has co-authored more than 100 publications in refereed conferences and journals and has served on more than 50 conference committees. His main focus is on distributed algorithms and datastructures, network theory (in particular, peer-to-peer systems, mobile ad-hoc networks, and sensor networks), and the design of algorithms and architectures for robust and secure distributed systems.

Michael Borokhovich is currently a Postdoctoral Fellow at the UT Austin, He obtained his B.Sc., M.Sc., and the PhD degrees in Communication Systems Engineering from the Ben-Gurion University of the Negev in Israel. His research interests are: large scale graph analytics, networks algorithms and software defined networks.

Bernhard Haeupler is currently a researcher at Microsoft Research. He obtained his BS, MS and Diploma from the Technical University of Munich in mathematics and his MS and PhD in Computer Science from MIT. His work on distributed information dissemination via network coding and gossip algorithms won best student paper awards at STOC'11 and SODA'13. In the fall of 2014 Haeupler joined CMU as a professor.

Zvi Lotker is an associate professor in the Communication Systems Engineering department at Ben Gurion University in Beer Sheva, Israel. He received a BSc in Mathematics and Computer Science, and a BSc in Industrial Engineering, from Ben Gurion University in 1991. In 1997 he received his MSc in Mathematics and in 2003, he received his PhD in Algorithms both from Tel Aviv University. He was a Postdoctoral Researcher at CWI, in Amsterdam, MPI in Saarbrücken Germany, and Mascotte in Nice France from 2003 to 2006. His main research areas are communication networks, online algorithms, sensor networks and recently, social networks.