Working Set Theorems for Routing in Self-Adjusting Skip List Networks

Chen Avin
School of Electrical and Computer Engineering
Ben Gurion University of the Negev

Iosif Salem
Faculty of Computer Science
University of Vienna

Stefan Schmid
Faculty of Computer Science
University of Vienna

***Abstract*—This paper explores the design of dynamic network topologies which adjust to the workload they serve, in a demand-aware and online manner. Such self-adjusting networks (SANs) are enabled by emerging optical technologies, and can be found, e.g., in datacenters. SANs can be used to reduce routing costs by moving frequently communicating nodes topologically closer. However, such reconfigurations also come at a cost, introducing a need for online algorithms which strike an optimal balance between the benefits and costs of reconfigurations.**

This paper presents SANs which provide, for the first time, provable *working set* guarantees: the routing cost between node pairs is proportional to how recently these nodes communicated last time. Our SANs rely on a *distributed* implementation of skip lists (which serves as the topology) and provide additional interesting properties such as local routing. Our first contribution is $SASL^2$, which is a randomized and sequential SAN algorithm that achieves the working set property. Then we show how $SASL^2$ can be converted to a distributed algorithm that handles concurrent communication requests and maintains $SASL^2$'s properties. Finally, we present deterministic SAN algorithms.

I. Introduction

While traditionally, physical networks such as datacenter networks, are considered a *fixed* infrastructure, emerging technologies (e.g. optical circuit switches, free-space optics) allow to reconfigure the network topology at runtime [1], [2].

Such reconfigurability can be exploited to design self-adjusting networks (SANs) which adapt to the demand (e.g., traffic pattern) they currently serve, in an online manner. In particular, by reconfiguring themselves to move two frequently communicating nodes closer, self-adjusting networks can reduce routing costs (i.e., route lengths), and hence latency or energy costs. This however introduces a tradeoff: while more frequent reconfigurations allow to react to changes in the demand more quickly and hence improve routing costs further, this leads to increased reconfiguration costs.

The problem of designing SAN algorithms is similar in spirit to designing self-adjusting data structures [3], [4], but *with a twist*. Data structures adjust to better serve sequences of lookup requests from a specific node (e.g., the root of a tree or head of a list) towards another node. SANs adjust to better serve *routing* requests, between *arbitrary nodes pairs*.

How can we evaluate the performance of a SAN, or more specifically, of the online algorithm which determines the SAN over time? Online algorithms are often evaluated *by comparison*: how good is the online algorithm compared to an optimal *static* algorithm? If the online algorithm is at most a constant factor worse than the optimal static solution, it is called *statically optimal*. Another frequently considered comparison is to an optimal *offline* algorithm: an online algorithm which is at most a constant factor worse than an optimal offline algorithm is called *constant competitive* or *dynamically optimal*. While statically optimal self-adjusting data structures as well as self-adjusting networks are known [4], [5], the problem of designing dynamically optimal solutions continuous to puzzle researchers, despite major efforts over the last years [6], [7].

However, a self-adjusting network can provide several additional interesting properties beyond static and dynamic optimality. A well-known property in the context of self-adjusting data structures, which lies between static and dynamic optimality, is the *working set property*. While the working set property has been studied intensively in the context of data structures, we are not aware of any work in the context of self-adjusting networks. In this paper, we will fill this gap.

In the context of data structures, the efficiency of an algorithm over a search request sequence σ is usually computed by the average cost over all requests in σ, as well as the *amortized* cost, i.e. the maximum average cost over all input sequences and initial states of the data structure. The working set number of an item x is the number of distinct items searched since the last search for x. An algorithm achieves the working set property when the search cost is asymptotically bounded by the logarithm of the working set number.

In the context of networks, however, the working set property should be defined with respect to communication *pairs*, as explained above. But today, we still lack rigorous definitions: the landscape of metrics for SANs is largely uncharted.

In this paper, we are interested in scalable and robust SANs, e.g., providing not only working set guarantees but also logarithmic diameter, low degree, and connectivity even after a failure. To this end, we consider SAN topologies based on skip lists [13] (cf. Section II). Skip lists are not only interesting for data structures but also for networks as they, e.g., provide *local routing*. This is particularly useful for *dynamic* topologies, which change over time, since we do not have to distribute information about new routing tables. Furthermore, as we will see, we will be able to leverage *distributed* skip list algorithms

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

		Algorithm					
		$SASL$ [8]	BDL [9]	Splay tree [4]	$SASL^2$	BDL^2	SplayNet [10]
property	complexity	SO [8] **WSP**	WSP [9]	SO, WSP, etc. [4]	**pWSP**	implied pWSP	**pWSP**
	robustness	✓	✓	✗	✓	✓	✗
	distributed version	N/A	N/A	N/A	✓	non trivial	[11], [12]

Fig. 1: Summary of related work and our contributions. SO stands for static optimality, (p)WSP for (pairwise) working set property, and N/A for not applicable. In our context, robustness has the meaning of k-connectivity, for $k > 1$. The cells shaded in gray denote our contributions.

to design SANs based on distributed algorithms.

The main question of this work is twofold: (1) what is a meaningful definition of the *pairwise* working set property for routing requests? and (2) how to design efficient SANs based on skip list networks providing such working set property?

Contributions. The main contribution of this paper is the first SAN that achieves the pairwise working set property. To this end, we formally define a natural notion of working set which depends on the number of distinct nodes that participated in communication requests, since the last requests that included the corresponding source and destination nodes. Our algorithms for SANs are based on a straight-forward extension of classic self-adjusting data structures, using "move-to-front" (MTF) data structures as a subroutine.

More specifically, to design SAN algorithms with the pairwise working set property for skip list networks, we extend $SASL$, a self-adjusting skip list algorithm by Ciriani et al. [8], to $SASL^2$, a SAN algorithm (Section III). While so far it was only known that SASL provides static optimality, we in this paper prove that it also has the working set property (Section IV-A). We define a working set property in SANs, i.e., the routing and adjustment cost of a request is bounded asymptotically by the logarithm of the pairwise working set size. We then prove that $SASL^2$, our SAN algorithm for skip list networks, has the working set property (working set theorem, Section IV-B).

We extend our technical results by observing that an existing deterministic self-adjusting skip list algorithm (BDL, [9]) which has the working set property, can be extended to a SAN algorithm (BDL^2) with similar guarantees in networks (Section V-A). We then discuss distributed implementations of both self-adjusting skip list network algorithms (Section V-B). Finally, we observe that a simple MTF-variant of the SplayNet algorithm [10] also has the pairwise working set property (Section V-C). We summarize our contributions on SANs and put them into context in Figure 1. We conclude the paper by presenting related work (Section VI) and discussing future work (Section VII). Before diving into these results, we provide the necessary background (Section II).

II. Background and Model

The networks considered in this paper will be based on a skip list topology, derived from skip list data structures [13]. We hence first present the required background on skip lists, and then introduce our formal model. Throughout the paper whenever we use an interval, e.g. $[a, b)$, we refer only to the integers it includes, i.e. $[a, b) \cap \mathbb{N}$.

Skip lists. The skip list [13] was designed as a search data structure that serves as a probabilistic alternative to balanced trees. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of integer keys (or items or elements) such that each x_i is associated with a node v_i. We also consider two special nodes $head$ and $tail$ (or left and right sentinels), with keys $-\infty$ and $+\infty$, respectively. Given a coin with a fixed probability of heads p, each node decides on the height of its key $h(x_i)$, by starting at height 1 and increasing the height by one for each flip that is heads until the first time the coin flip is tails. The height $H = \max_i h(x_i)$ of the skip list is expected to be in $\mathcal{O}(\log n)$. The depth of an item x_i is $d(x_i) = H - h(x_i)$. The skip list is formed by connecting vertically H doubly-linked lists that contain subsets of $X \cup \{-\infty, +\infty\}$ linked in ascending order. We denote these lists by $\mathcal{L}_1, \ldots, \mathcal{L}_H$, where $|\mathcal{L}_i| = \Theta(2^i)$ and $\mathcal{L}_i \subset \mathcal{L}_{i+1}$, for $i \in \{1, \ldots, H-1\}$. All lists start and end with $-\infty$ and $+\infty$. The bottom list $\mathcal{L}_H$ contains all the keys and list $\mathcal{L}_i$ includes all items of height at least i. We assume that bideractional vertical pointers link occurrences of each node x_i in adjacent lists $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$, $i = 1, \ldots, H-1$. We refer to an item's right neighbor in a list as its successor and to its left neighbor as its predecessor.

Searching a node with key u in a skip list $\mathcal{L}$ occurs by starting from the head item at $\mathcal{L}_1$ and moving to the right if the next item of the current list is smaller than u. If the next item is larger than u, the search moves down one level and continues the rightward search. The search ends successfully upon finding u and fails if it reaches an element x in $\mathcal{L}_H$ such that $x < u$ and $u < y$, where y is the successor of x in $\mathcal{L}_H$. An item u can be added by deciding its height via the coin flip procedure, searching its predecessors and successors, and adding the node in $h(u)$ lists by adjusting the pointers. Deleting a node is done by linking the predecessors and successors in every list the node belongs. The costs of add and delete operations are dominated by the cost of searching the location of the node to be added or deleted, i.e. $O(\log n)$ in expectation.

Computational and cost model. We remark that a skip list can be also viewed as a graph (or skip list *network*), where the node set is $V = \{v_1, v_2, \ldots, v_n\}$ and two nodes are connected with a bidirectional link if their keys are adjacent at some level of the skip list. Each node v_i stores locally the triples

(h, dir, x), for each level $h = 1, 2, \ldots, h(x_i)$, direction $dir \in \{left, right\}$, and adjacent key $x \in X = \{x_1, x_2, \ldots, x_n\}$. Thus, both the data structure and graph point of view are equivalent and we use them interchangeably. Throughout this paper we use the terms network, network topology, and graph interchangeably, as our focus is to adjust such topologies. Later in this section we describe a routing procedure in such graphs.

We proceed on defining the cost of self adjusting algorithms both for search and communication request sequences, based on [10]. A self-adjusting algorithm for a sequence σ of routing (search) requests adjusts the network topology (data structure), i.e., a graph, to minimize routing (search) costs. The cost of a self adjusting algorithm for a single (search or communication) request is the cost of serving the request plus the cost of adjustments on the graph (data structure). In the case of skip lists, the adjustments include the promotion or demotion of items to different levels.

Let G_t be the graph (data structure) at time t, $i = 0, 1, \ldots, m$ and $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ a sequence of m communication (search) requests. G_0 is the initial graph (data structure) and G_i is its state after serving σ_i. Upon a request $\sigma_i \in \sigma$, a self-adjusting algorithm $\mathcal{A}$ serves σ_i in G_{i-1} and then transforms G_{i-1} to G_i. Serving the request can also occur after bringing the communication endpoints closer, however this choice is up to the algorithm designer and does not affect the complexity analysis asymptotically. We denote by $dist_{G_{i-1}}(\sigma_i)$ the routing (search) cost of a request σ_i and by $adjustment(\mathcal{A}, G_{i-1}, \sigma_i)$ the adjustment cost, i.e., the number of unit cost operations to transform G_{i-1} to G_i. In the skip list context, sending a message (following a pointer), and increasing or decreasing the level of an item by 1 are all unit cost operations. The cost of a single communication request σ_i is given by $cost(\mathcal{A}(\sigma_i)) = cost(\mathcal{A}, i, G_{i-1}, \sigma_i) = dist_{G_{i-1}}(\sigma_i) + adjustment(\mathcal{A}, G_{i-1}, \sigma_i)$. The total cost for σ is given by $\sum_{i=1}^{m} cost(\mathcal{A}(\sigma_i))$ and the average cost is the total cost divided by m. The amortized cost is the maximum average cost over all G_0 and σ. We design *online* self-adjusting algorithms that minimize the cost of serving unknown search or communication sequences.

Routing in skip list networks. We explain how a routing request is served in a skip list network. In data structure terms, a routing request is quite similar to finger search [14], i.e. a search request that originates in an item, respectively *node*, u towards another node v, where $u, v \notin \{-\infty, +\infty\}$. In our model, we consider the following procedure. If $u < v$ ($u > v$) then the routing proceeds to the right (left). The routing procedure is split in an up-phase and a down-phase. The routing path starts at the highest level of u with the up-phase. During the up-phase, at the current item the path moves up if the next item is smaller (larger) than v, unless the node's top level is reached, in which case it moves to the right (left) and repeats. When the next item is larger (smaller) than v, then the down-phase begins, which is essentially a standard skip list search for v. That is, at the current item, the path moves to the right if the next item is smaller (larger) than v, otherwise it moves one level down and repeats the rightward (leftward) search, until locating v.

The routing procedure we described requires only local information, however there are corner cases in which there exist shortest paths that route via the left or right sentinels. For example, consider a prefix of a skip list with nodes $1, 2, 3, 4, 5$, such that $h(i) = i$. Then, routing from 1 to 5 requires following 8 pointers, while routing from 1 to $-\infty$, moving up until level 5 and then taking a single pointer to the right to 5 requires following 6 pointers.

Algorithm 1: SASL: Self-adjusting Skip List Algorithm for Search Requests [8]

```
upon search request for u begin
    search u;
    b ← B(u);
    d(u) ← H_1; // promoting u to B_1
    UpdateCountersAfterPromotion(u);
    for i = 1, ..., b − 1 do demotion(i);
demotion(j) begin
    x ← RandomSelect(j);
    d(x) ← H_{j+1};
    UpdateCountersBeforeDemotion(x);
    UpdateCountersAfterDemotion(x);
```

III. $SASL^2$: A Self-Adjusting Algorithm for Skip List Networks

We first present $SASL$, the randomized self-adjusting skip list of Ciriani et al. [8], which is a basic component of our work. Then we present $SASL^2$, a self-adjusting algorithm for skip list networks, which is a straightforward extension of $SASL$. The algorithms presented in this section handle one request at a time. In Section V we discuss how to extend $SASL^2$ to a distributed algorithm, to deal with concurrency.

A. Prior work: Randomized self-adjusting skip lists for search sequences

Ciriani et al. [8] presented $SASL$, an online self-adjusting skip list algorithm for sequences of search requests, that achieves static optimality, i.e. it performs as well as the static offline algorithm. $SASL$ (Algorithm 1) is based on the following three principles: (a) logically partition the levels of a skip list $\mathcal{L}$ in a $\mathcal{O}(\log \log n)$ number of *bands* (sets of consecutive lists) of exponentially increasing size from top to bottom, (b) upon search of an element move it to the top band, and (c) if the searched element was *associated* with the band x, demote an element uniformly at random (using a random walk) for each band $\mathcal{B}_i$ to $\mathcal{B}_{i+1}$, for $i \in [1, x-1]$. For any search sequence $\sigma = (\sigma_1, \ldots, \sigma_m)$ input to $SASL$ the expected average time complexity of these m searches is in $\mathcal{O}(\sum_{i=1}^{n} \frac{n_i}{m} \log \frac{m}{n_i})$, where n_i is the number of times item $x_i \in X$ appears in σ and X is the set of items, and the space complexity is in $\mathcal{O}(n \log n)$. Since the average time complexity of $SASL$ equals the entropy of σ, $SASL$ is

statically optimal. We elaborate on these principles of $SASL$ below.

Partition of consecutive levels in bands. Towards defining the partition of levels to bands, we give the following definitions. Consider $f(x) = \sum_{i=1}^{x} 2^{2^{i-1}}$ to be the maximum number of objects we can insert into x buckets of doubly-exponential size and $b(n)$ to be an integer such that $f(b(n)-1) < n \leq f(b(n))$, for a given integer n. By the definitions above, $b(n) = \Theta(\log\log n)$. We consider skip lists for which $H = \Theta(\log n)$. We now proceed to define the partition of $\{\mathcal{L}_i\}_{i\in[1,H]}$, to bands $\mathcal{B}_j$ of exponentially increasing size, such that every $\mathcal{B}_j$ includes consecutive lists of $\mathcal{L}$. The first (top) band of $\mathcal{L}$, $\mathcal{B}_1$, includes only $\mathcal{L}_1$. The second band, $\mathcal{B}_2$, includes $\mathcal{L}_2$ and $\mathcal{L}_3$, and the i^{th} band, $\mathcal{B}_i$, includes the 2^{i-1} lists $\mathcal{L}_{2^{i-1}}, \ldots, \mathcal{L}_{2^i-1}$. Thus, the number of bands of $\mathcal{L}$ with height $H = \Theta(\log n)$ is $b(n)$. The upper list of $\mathcal{B}_i$ is $\mathcal{L}_{2^{i-1}}$ and the lower list of $\mathcal{B}_i$ is $\mathcal{L}_{2^i-1}$. The height of a band $\mathcal{B}_i$ is the total number of lists belonging to lower bands and denoted by $H_i = 2^{b(n)} - 2^i$ (e.g. $H_1 = H - 1$).

Deterministic and random height. The height of an element s, $h(s)$, consists of a deterministic part, $d(s)$, and a randomized part, $r(s)$. The height of s cannot exceed H, thus $h(s) = \min\{r(s) + d(s), H\}$. When $r(s) + d(s) > H$, $r(s)$ is temporarily set to $H - d(s)$, but its original value is saved and restored when $r(s) + d(s) \leq H$. The deterministic part $d(s)$ is manipulated by $SASL$ to promote or demote s according to demand. $SASL$ ensures that $d(s)$ always ends in the beginning of a band, for any $s \in X$. An item s *resides* in band $\mathcal{B}_i$ if $d(s) = H_i$. The number of items that reside in $\mathcal{B}_j$ is $B_j := 2^{2^{j-1}}$, for all bands except the last one. The size of the last band, $\mathcal{B}_{b(n)}$, is denoted by $B_{b(n)}$ and is at most $2^{2^{b(n)-1}}$. If $h(s) > H_i$, then item s *appears* in band $\mathcal{B}_i$. We denote by $\mathcal{B}(u)$ the band in which u currently resides. The value of $d(s)$ is decided by randomly assigning items to bands upon initialization and by a fair coin upon item addition. Once the value of $d(s)$ is decided (upon initialization or addition), $r(s)$ is decided as if the band it resides is an independent skip list, i.e., s flips a coin to decide if it will increase its level by one until the outcome is negative and $r(s)$ is henceforth a constant.

Search, promotion, demotion. Upon a search request for an item x residing in band $\mathcal{B}_j$, and after serving the request (line 2), $SASL$ promotes x to reside in band $\mathcal{B}_1$ by setting $d(x) = H_1$ (line 4). Then, $SASL$ demotes one item from each $\mathcal{B}_i$, $i \in [1, j)$, to reside in the immediately lower band $\mathcal{B}_{i+1}$ (line 6), thus keeping the number of items in each band fixed. Specifically, the demotion of an item y from band $\mathcal{B}_i$ to $\mathcal{B}_{i+1}$ is done by changing $d(y) = H_i$ to $d(y) = H_{i+1}$ ($r(y)$ remains intact), while keeping the respective lists connected after the removing y from them (line 9). The $j-1$ demotions, i.e., one for each $\mathcal{B}_j$, $j \in [1, j)$, occur uniformly at random as explained below.

Random selection. Let s be an item residing in $\mathcal{B}_j$ and $c_{j+k}(s)$ be the number of items that reside in $\mathcal{B}_{j+k}$, $k \in [0, b(n)-j]$, and are reachable from s by a skip list search, i.e., $c_{j+k}(s) = |\{x \in X \mid d(x) = H_{j+k} \wedge h(s) \geq h(x) \wedge s \leq x\}|$.

Algorithm 2: $SASL^2$: Self-adjusting Skip List Algorithm for Routing Sequences

```
1 upon communication request (u, v) begin
2     route (u, v);
3     adjustSASL(u);
4     adjustSASL(v);
```

Each node maintains these numbers for every band it resides and uses them to drive a random walk that selects the item to be demoted from $\mathcal{B}_j$ ($RandomSelect(j)$ procedure in line 8). That is, after the promotion of the searched item, a random walk starts from the root. From a node s at band $\mathcal{B}_i$, the probability to move forward to a node s' is $c_i(s')/c_i(s)$ and the probability to move downward is $1 - c'_i(s)/c_i(s)$. If s is at $\mathcal{L}_{l(i)}$, i.e., the bottom list in $\mathcal{B}_i$, it is demoted to $\mathcal{B}_{i+1}$. Let z be the item that the random walk selects for demotion. $SASL$ ends by updating the c_i counters of all nodes in two search paths; the ones to z before (line 10) and after demotion (line 11). Similarly, after promoting a searched node u to the first band (line 4), $SASL$ updates all counters of the elements in a skip list search to u after its promotion ($UpdateCountersAfterPromotion(u)$, line 5). We give a concrete example in Figure 2, which serves in showcasing both $SASL$ and $SASL^2$ (see also [8, Section 3.1]).

B. $SASL^2$: Extending from data structures to networks

We present a straightforward extension of $SASL$ to the case of routing in self-adjusting skip list networks. Our algorithm $SASL^2$ (Algorithm 2) uses the promotion and demotion procedures of $SASL$ as a black box. Let $adjustSASL(u)$ be $SASL(u)$ by omitting the search step on input u, i.e. line 2 in Algorithm 1. Upon a communication request (u, v), $SASL^2$ serves the request and then calls $adjustSASL(u)$ and subsequently $adjustSASL(v)$. These calls bring u and v to the top levels of the skip list, by the definition of $SASL$, and thus reduce their distance. We illustrate $SASL^2$ in Figure 2.

Note that the routing cost (recall the routing procedure in Section II) of a request (u, v) is upper bounded by the sum of searching u and searching v. The latter holds, because the up phase starts from the top level of u and possibly reaches a node of maximum height and the down phase starts at most at the highest level of the skip list and proceeds until v is located. The costs of both of these phases are upper bounded by the search costs for searching an element in u and v's bands. Thus, the cost of $SASL^2(u, v)$ is uper bounded by the sum of the costs of $adjustSASL(u)$ and $adjustSASL(v)$. In Section IV-B (Theorem 2) we prove that $SASL^2$ has the pairwise working set property, a rather strong complexity guarantee of self-adjusting algorithms.

IV. Formal analysis: Working Set Theorems

We prove a working set theorem for $SASL^2$, an algorithm that handles a skip list network that self-adjusts to sequences of routing requests, which we presented in Section III-B. We first

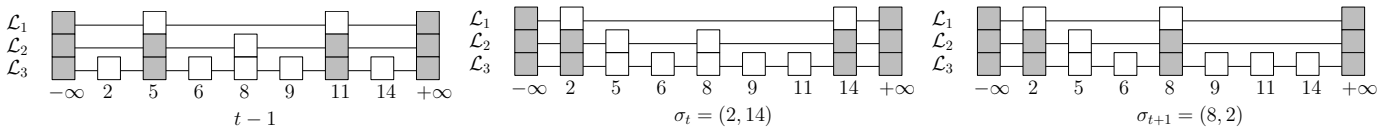

Fig. 2: Example run of $SASL^2$. The shaded boxes denote the deterministic heights and the clear ones the random heights. The first figure shows the state at time $t-1$. The second figure shows the skip list network after time t and request $(2, 14)$. 2 and 14 are promoted to the top, while 5 and 11 are randomly selected for demotion from the first band to the second (their deterministic height is reduced and counters are updated accordingly). Note that 5's random height is set back to 2, since its value was suppressed to 1 when it was in the top band (all heights must be less or equal to the skip list height). Since $SASL^2$ first calls $adjustSASL(2)$ and then $adjustSASL(14)$, starting from the state in $t-1$, 2 is promoted to the top band, $\mathcal{B}_1$, and then a random walk starts for demoting an item from $\mathcal{B}_1$ to $\mathcal{B}_2$, out of 2, 5, and 11. According to Section III-A, the random walk starts from the top of $-\infty$ (the skip list root), and with probability 1 moves to 2, as all items in $\mathcal{B}_1$ are reachable from 2. The random walk then moves to 5 with probability 2/3 and (randomly) decides to stop there, thus demoting 5. However, the probability of moving to 11 is $1/3 > 0$, thus 11 could be demoted in an alternative run. Then, the random walk within $adjustSASL(14)$ decides to demote 11, resulting in the state after serving σ_t (second figure). Finally, the third figure shows the skip list network after time $t+1$ and request $(8, 2)$. 2 is already at the top band, so no promotion or demotions occur. 8 is promoted to the top band and 14 is randomly selected for demotion from the first to the second band. Note that different nodes could be demoted in other runs of the algorithm, since their selection is uniform at random, per band.

prove a novel working set theorem for the $SASL$ algorithm for search request sequences (Theorem 1, Seciton IV-A), which was mentioned as an open question in [8]. We then define the terms working set and working bag in the context of routing request sequences and extend the proof of Theorem 1 to prove a working set theorem for $SASL^2$ (Section IV-B).

A. Working set theorem for search request sequences in $SASL$

We give some necessary definitions. Let σ be a sequence of search requests in a skip list of n elements and $\sigma_i \in \sigma$. We denote by $WS(\sigma_i)$, the *working set* of σ_i, i.e. the set of (distinct) elements since either the last occurrence of σ_i in σ, or the beginning of σ, if σ_i appears for the first time in σ. We refer to the subsequence between two occurrences of σ_i (including both occurrences of σ_i) such that σ_i does not appear again between them as the *working bag* of σ_i and denote it by $WB(\sigma_i)$. Let $|WB(\sigma_i)| = T$, i.e. $WB(\sigma_i) = (\sigma_{i-T+1}, \sigma_{i-T+2}, \ldots, \sigma_i)$, $\sigma_i = \sigma_{i-T+1}$, $T \geq 2$, and there have been $T-2$ requests for other elements between the two requests for σ_i. Recall that by the definition of bands in [8] (Section III-A) the size of band $\mathcal{B}_x$ denoted by B_x is $B_x = 2^{2^{x-1}}$. The time to lookup an item residing in $\mathcal{B}_x$ is $\mathcal{O}(\log B_x)$.

The following definitions are essential for our proofs. Let $\mathcal{B}_t(x) \in [1, b(n)]$ be the band at which element $x \in WB(\sigma_i)$ resides at time $t \in [i-T+1, i]$, where $b(n) = \Theta(\log\log n)$. Moreover, let $\mu_\ell(WB(\sigma_i)) = \max\{\mathcal{B}_{i-T+\ell}(\sigma_j) \mid j \in [i-T+1, i-T+\ell]\}$, for $\ell \in [1, T]$, i.e. the maximum of the bands in which the elements of an ℓ-sized prefix of $WB(\sigma_i)$ reside at time $i-T+\ell$ (after serving the last request in the prefix). Since the proof focuses on an arbitrary σ_i, we will simply write μ_ℓ instead of $\mu_\ell(WB(\sigma_i))$. For example, $\mu_1 = 1$.

Theorem 1. *$SASL$ achieves the working set property: $E[cost(SASL(\sigma_i))] = \mathcal{O}(\log |WS(\sigma_i)|)$.*

Proof. We follow the proof of [8, Theorem 3.7], with some refined calculations. Let $W := |WS(\sigma_i)|$ and $k' := \log\log W$. Recall that μ_{T-1} is the maximum band in which all elements of $(\sigma_{i-T+1}, \ldots, \sigma_{i-1})$ reside at time $i-1$, where $\sigma_{i-T+1} = \sigma_i$. We show that the probability of μ_{T-1} being equal to $k'+j$ falls off doubly exponentially on j.

First, observe that $cost(SASL(\sigma_i)) = \mathcal{O}(cost(SASL(u)))$ upon request σ_i, where $u \in \mathcal{B}_{\mu_{T-1}}$. That is, the cost of request σ_i is upper bounded by the cost of searching an element in the maximum band in which the elements in $(\sigma_{i-T+1}, \ldots, \sigma_{i-1})$ appear at time $i-1$. Thus, we derive $E[cost(SASL(\sigma_i))] = \mathcal{O}(E[cost(SASL(u))]) = \mathcal{O}(\log B_{k'} + \sum_{j=1}^{\infty} \Pr[\mu_{T-1} = k'+j] \cdot \log B_{k'+j})$, since $\mathcal{O}(\log B_{k'})$ is an upper bound for the case where $\mu_{T-1} \leq k'$ and the other element of the sum is an upper bound for the case of $\mu_{T-1} > k'$.

We show that $\Pr[\mu_{T-1} = k'+j] \leq W^2 / B_{k'+j-1}$. First, observe that $\Pr[\mu_{T-1} = k'+j] = \Pr[\mu_{T-1} = k'+j \mid \mu_1 = 1]$. This equality holds by the definition of the SASL algorithm. That is, $\mu_1 = \beta_{i-T+1}(\sigma_{i-T+1}) = 1$, since σ_{i-T+1} is promoted to the top band at time $i-T+1$ and $\Pr[\mu_{T-1} = k'+j]$ is the probability of μ_{T-1} reaching $k'+j$ within the subsequence $(\sigma_{i-T+2}, \ldots, \sigma_{i-1})$ of the working bag, given that $\mu_1 = 1$.

Observe that $\Pr[\mu_{T-1} = k'+j \mid \mu_1 = 1] \leq \Pr[\mu_{T-1} = k'+j \mid \mu_1 = k'+j-1]$. The latter probability describes the event in which at time $i-T+1$, σ_{i-T+1} is promoted to $\mathcal{B}_1$ as in SASL, demotions are done as in SASL, and subsequently σ_{i-T+1} is demoted such that $\beta_{i-T+1}(\sigma_{i-T+1}) = k'+j-1$. The remaining $T-1$ requests of the working bag are handled without any modifications of SASL. For this artificial demotion the following hold: (i) the number of elements that reside in bands $1, \ldots, k'-j-2$ are reduced by 1, (ii) $\mu_1 = \beta_{i-T+1}(\sigma_{i-T+1}) = k'+j-1$, (iii) the probability of increasing μ_τ, $\tau \in [2, T-1]$, from $k'+j-1$ to $k'+j$ within the requests $(\sigma_{i-T+2}, \ldots, \sigma_{i-1})$ is increased compared to $\Pr[\mu_{T-1} = k'+j \mid \mu_1 = 1]$, since the starting maximum band, μ_1, is closer to $k'+j$ ($\mu_1 = k'+j-1$) compared to the event where $\mu_1 = 1$. Figure 3 illustrates this claim.

The probability $\Pr[\mu_{T-1} = k'+j \mid \mu_1 = k'+j-1]$ equals

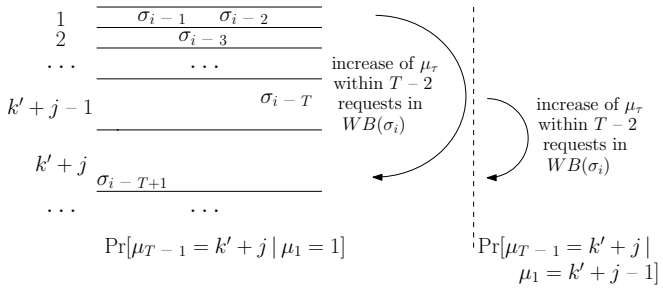

Fig. 3: Illustration of $\Pr[\mu_{T-1} = k' + j \,|\, \mu_1 = 1] \leq \Pr[\mu_{T-1} = k' + j \,|\, \mu_1 = k' + j - 1]$. We depict the first $k' + j$ bands of the skip list and an indication of where the elements in the working bag (possibly) reside at time $i - 1$. The arrow to the left of the dotted line denotes the increase of μ_τ from 1 to $k' + j - 1$ within $\tau \in [2, T - 1]$ in the event of $\Pr[\mu_{T-1} = k' + j \,|\, \mu_1 = 1]$. The arrow to the right of the dotted line denotes the increase of μ_τ from $k' + j - 1$ to $k' + j$ within $\tau \in [2, T - 1]$ in the event of $\Pr[\mu_{T-1} = k' + j \,|\, \mu_1 = k' + j - 1]$.

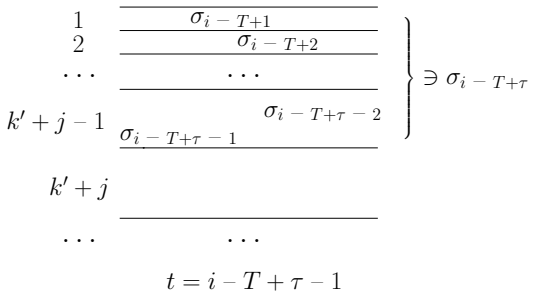

Fig. 4: Illustration of Claim 1. The figure depicts the decomposition in bands and an indication of where the elements of a working set prefix (possibly) reside at time $i - T + \tau - 1$, $\tau \in [2, T - 1]$. Since $\beta_{i-T+\tau-1}(\sigma_{i-T+\tau}) \in \{\beta_{i-T+\tau-1}(\sigma_{i-T+1}), \ldots, \beta_{i-T+\tau-1}(\sigma_{i-T+\tau-1})\}$, $\mu_\tau = \mu_{\tau-1} = k' + j - 1$ holds.

$\sum_{\tau=2}^{T-1}(p_\tau \cdot q_\tau)$, where $p_\tau = \Pr[\mu_\tau = k'+j \,|\, \mu_{\tau-1} = k'+j-1]$ and $q_\tau = \Pr[\mu_{\tau'} = k' + j,\, \tau' \in [\tau + 1, T - 1] \,|\, \mu_\tau = k' + j]$. That is, there are $T - 2$ chances for μ_τ, $\tau \in [2, T - 1]$, to take the value $k' + j$ (denoted by p_τ) and keep it as it's final value (denoted by q_τ) in the sequence of the $T - 2$ requests $(\sigma_{i-T+2}, \ldots, \sigma_{i-1})$. Note that $\mu_\tau \geq k'+j-1$, for $\tau \in [1, T-1]$, because σ_{i-T+1}'s band does not change until time i (at that time it is promoted to $\mathcal{B}_1$). Since $q_\tau \leq 1$, $\Pr[\mu_{T-1} = k' + j \,|\, \mu_1 = k' + j - 1] \leq \sum_{\tau=2}^{T-1} p_\tau$. We need the following two claims to obtain that $\sum_{\tau=2}^{T-1} p_\tau \leq W^2/B_{k'+j-1}$.

Claim 1. *$p_\tau = 0$ if $\sigma_{i-T+\tau} \in \{\sigma_{i-T+1}, \ldots, \sigma_{i-T+\tau-1}\}$, $\tau \in [2, T - 1]$.*

Proof of Claim 1. The claim holds because by the definition of SASL, $\sigma_{i-T+\tau}$ appears in a band within the first $k' + j - 1$ bands and thus after searching for it and after SASL's demotions the maximum band in which the elements in $\{\sigma_{i-T+1}, \ldots, \sigma_{i-T+\tau}\}$ reside cannot increase (but can possibly decrease). □

By Claim 1 we assert that at most $W - 1$ elements of the sum $\sum_{\tau=2}^{T+1} p_\tau$ are non-zero. We illustrate Claim 1 in Figure 4.

Claim 2. *$p_\tau \leq W/B_{k'+j-1}$.*

Proof of Claim 2. $WB(\sigma_i)$ has W distinct elements and at most all of them can appear in $\mathcal{B}_{k'+j-1}$ (or even worse $\mathcal{B}_{k'+j-1} \subseteq WS(\sigma_i)$). Assuming that a random walk is about to choose an element for demotion in $\mathcal{B}_{k'+j-1}$, it can increase the current maximum band by choosing from at most W elements from a total of $B_{k'+j-1}$ that appear in the band, i.e. $p_\tau \leq W/B_{k'+j-1}$. We justify choosing the denominator of the bound to be $B_{k'+j-1}$ as follows. Recall the $c_{s+z}(x)$ variables of SASL that maintain at any time the number of elements that are reachable and reside in band $\mathcal{B}_{s+z}$, $z \in \{1, \ldots, b(n) - s\}$ (where $b(n) = \Theta(\log \log n)$ is the maximum number of bands), starting from an element x residing in band s. These variables are updated by SASL during each random walk, hence the random walk demotes elements uniformly at random. □

By combining claims 1 and 2 we get that $p = \sum_{\tau=2}^{T+1} p_\tau \leq \sum_{\tau=1}^{W-1} W/B_{k'+j-1} \leq W^2/B_{k'+j-1}$. We complete the proof by the following calculations:

$$
\begin{aligned}
&E[cost(SASL(\sigma_i))] = \\
&\mathcal{O}(\log B_{k'} + \textstyle\sum_{j=1}^{\infty} \Pr[\mu_T = k' + j] \cdot \log B_{k'+j}) = \\
&\mathcal{O}(\textstyle\sum_{s=0}^{2} \log B_{k'+s} + \sum_{j=3}^{\infty} \Pr[\mu_T = k' + j] \cdot \log B_{k'+j}) = \\
&\mathcal{O}(\textstyle\sum_{s=0}^{2} \log 2^{2^{k'+s-1}} + \sum_{j=3}^{\infty} (W^2/B_{k'+j-1}) \cdot \log B_{k'+j}) = \\
&\mathcal{O}(\textstyle\sum_{s=0}^{2} 2^{k'+s-1} + \sum_{j=3}^{\infty} (W^2/2^{2^{k'+j-2}}) \cdot \log 2^{2^{k'+j-1}}) = \\
&\mathcal{O}(2^{k'} \textstyle\sum_{s=0}^{2} 2^{s-1} + \sum_{j=3}^{\infty} (W^2/2^{2^{k'} \cdot 2^{j-2}}) \cdot 2^{k'+j-1}) = \\
&\mathcal{O}((\log W) \textstyle\sum_{s=0}^{2} 2^{s-1} + \sum_{j=3}^{\infty} (W^2/(2^{2^{k'}})^{2^{j-2}}) \cdot 2^{k'} 2^{j-1}) = \\
&\mathcal{O}(3.5 \log W + 2^{k'} \textstyle\sum_{j=3}^{\infty} (W^2/W^{2^{j-2}}) \cdot 2^{j-1}) = \\
&\mathcal{O}(3.5 \log W + (\log W) \textstyle\sum_{j=3}^{\infty} 2^{j-1}/W^{2^{j-2}-2}) = \\
&\mathcal{O}((3.5 + \textstyle\sum_{j=3}^{\infty} 2^{j-1}/W^{2^{j-2}-2}) \log W) = \\
&\mathcal{O}((3.5 + 7) \log W) = \mathcal{O}(10.5 \log W) = \mathcal{O}(\log W)
\end{aligned}
$$

The latter line of equalities holds since $\sum_{j=3}^{\infty} 2^{j-1}/W^{2^{j-2}-2} \leq \sum_{j=3}^{\infty} 2^{j-1}/2^{2^{j-2}-2} = \sum_{j=3}^{\infty} 1/2^{2^{j-2}-j-1} \leq 7$, for $W \geq 2$ (the case of $W = 1$ is trivial). □

B. Working set theorem for sequences of communication requests in $SASL^2$

We first define the working bag and working set numbers for routing requests and then prove a working set theorem for $SASL^2$ (Algorithm 2). Let $\circ$ be an operator that joins two sequences. Intuitively, the working bag of a communication request $\sigma_t = (s_t, d_t)$ is $\tau \circ \sigma_t$, where τ is the shortest suffix of the sequence $\sigma = (\sigma_1, \ldots, \sigma_{t-1})$ that includes, possibly individual, requests in which s_t and d_t appear. The size of the working bag is the working bag number. The working set includes all distinct elements in the working bag and the working set number is the size of the working set. Definitions 1 (cf. Figure 5) and 2 give the formal statements.

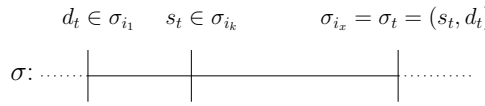

Fig. 5: Example of a working bag for a routing request $\sigma_t = (s_t, d_t)$. In case $d_t \notin \sigma_{i_j}$ for all $j \in \{2, \ldots, x-1\}$, $(\sigma_{i_1}, \ldots, \sigma_{i_x})$ is the minimum sequence that Definition 1 requires, i.e., the working bag of σ_t, and $|WB(\sigma_t)| = x$.

Definition 1 (Working bag and working bag number)**.** *Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ be a sequence of communication requests, where $\sigma_t = (s_t, d_t)$. We define the (pairwise) working bag of a communication request $\sigma_t = (s_t, d_t)$ to be $(\sigma_1, \ldots, \sigma_t)$, if s_t or d_t appear in a request of σ for the first time at time t, otherwise $WB(s_t, d_t) = \min\{\sigma' \sqsubseteq (\sigma_1, \ldots, \sigma_{t-1}) \,|\, \exists_{\sigma_i, \sigma_j \in \sigma'}\, s_t \in \sigma_i \wedge d_t \in \sigma_j\} \circ \sigma_t$, where $\sqsubseteq$ denotes the suffix relation. We denote by $|WB(s_t, d_t)|$ the size of $WB(s_t, d_t)$, i.e., number of requests, and refer to it as (s_t, d_t)'s working bag number.*

Definition 2 (Working set and working set number)**.** *The (pairwise) working set of a communication request $\sigma_t = (s_t, d_t) \in \sigma$ is $WS(\sigma_t) = WS(s_t, d_t) = \{x \in \sigma_i \,|\, \sigma_i \in WB(s_t, d_t)\}$. The working set number of σ_t is the size of $WS(s_t, d_t)$ and we denote it by $|WS(s_t, d_t)|$.*

Our pairwise working bag and set definitions are suitable for topologies that have a top, and it is thus possible to design algorithms that follow a move-to-front/move-to-top principle [3]. For example a linked list's top is the first element, a BST's top is its root, and a skip list's top is the top level. The motivation for these definitions is that pairs of nodes that appear in a lot of searches separately should have a relatively small joint working bag and set. In Theorem 2, we extend Theorem 1 to the case of routing requests. As shown in the proof of Theorem 2, if the sequence of communication requests σ is viewed as a search sequence, then the pairwise and search working set numbers differ by a known constant, which essentially yields the result.

Theorem 2. *$SASL^2$ achieves the pairwise working set property: $E[cost(SASL^2(u, v))] = \mathcal{O}(\log |WS(u, v)|)$.*

Proof sketch. Let $WB(\sigma_t) = ((s_i, d_i), \ldots, (s_t = u, d_t = v))$ be the pairwise working bag of $\sigma_t = (s_t, d_t)$ and $\overline{WB(\sigma_t)} = (s_i, d_i, \ldots, s_t, d_t)$ be the equivalent search request sequence. Let $\overline{WS_{s_t}(\sigma_t)}$ and $\overline{WS_{d_t}(\sigma_t)}$ be the (search) working sets of s_t and d_t in $\overline{WS(\sigma_t)}$, and $\overline{WS_{max}}$ be the working set with the maximum cardinality between the two. Since, $\{s_i, d_i\} \cap \{s_t, d_t\} \neq \emptyset$, $WS(\sigma_t)$ either equals $\overline{WS_{max}}$ or it differs by $\{s_i\}$ (when $s_i \notin \{s_t, d_t\}$). Since the routing path and the adjustment costs are bounded by the lengths of the search paths to s_t and d_t, we observe that (i) $route(s_t, d_t) = \mathcal{O}(\log |\overline{WS_{s_t}(\sigma_t)}| + \log |\overline{WS_{d_t}(\sigma_t)}|) = \mathcal{O}(\log |\overline{WS_{max}}|) = \mathcal{O}(\log |WS(\sigma_t)|)$, and (ii) for $x \in \{s_t, d_t\}$, $E[adjustSASL(x)] = \mathcal{O}(\log |\overline{WS_x(\sigma)}|) = \mathcal{O}(\log |WS(\sigma_t)|)$. By linearity of expectations we get: $E[SASL^2(\sigma_t)] = E[route(s_t, d_t)] + E[adjustSASL(s_t)] + E[adjustSASL(d_t)] = \mathcal{O}(\log |WS(\sigma_t)|)$. □

V. Discussion

We make additional observations regarding the working set property for self-adjusting networks. We discuss an alternative but deterministic self-adjusting algorithm for skip list networks in Section V-A and then provide a feasibility analysis and some challenges for designing distributed versions of the self-adjusting skip list algorithms discussed so far, in Section V-B. We complete the section, by observing that a simple variant of SplayNet [10], a tree-based self-adjusting network, also has the pairwise working set property (Section V-C).

A. Deterministic Self-Adjusting Skip List Networks

We now turn to discuss a deterministic self-adjusting algorithm for skip list networks. Bose et al. presented in [9] a deterministic self-adjusting skip list for search sequences that achieves working set optimality. We give an overview of that work and extend it to routing sequences, where working set optimality is naturally preserved.

BDL algorithm overview. The deterministic self-adjusting skip list of Bose, Douïeb, and Langerman (in short, BDL) [9] builds upon elements from $SASL$, the self-adjusting skip list of [8] (Section III-A), and the biased (a, b)-skip list of Bagchi et al. [14], which is an optimal offline skip list given the access frequencies. The authors assumed a restricted model, such that for a given constant B (chosen by the algorithm designer), a search cannot go forward for more than B pointers in a single level, without taking a downward pointer. Similarly to $SASL$, the authors decompose the levels of the skip list in $2k = 2\lceil \log \log n \rceil$ layers and the skip list height is set to $H = 2^{k+2} - 2$. Namely, the layers are $\ell_1, \ell'_1, \ell_2, \ell'_2, \ldots, \ell_k, \ell'_k$ and for $a \in \{1, \ldots, k\}$, each of ℓ_a and ℓ'_a cover 2^a levels of the skip list. An element x belongs to layer ℓ if x's height is within the levels of ℓ.

Upon a search request for x, the BDL algorithm serves the request, increases the height of x so it belongs to ℓ_1 (promotion step), and then proceeds to a demotion procedure, only in case x did not belong to ℓ_1 before the promotion step. The demotion procedure checks if there is an overflow in ℓ_a ($a = 1$ initially), i.e. when there are more than 2^a elements in ℓ_a. In that case, every element belonging to ℓ'_a is decreased to ℓ_{a+1} and then every element in ℓ_a is decreased by 2^a, so that it ends up belonging to ℓ'_a. The demotion step is applied to ℓ_{a+1} (and so on) in case an overflow occurs in that layer, otherwise the procedure stops. Note that insertions and deletions are handled as in the (a, b)-skip lists [14] by selecting $b = B$ for a number B of our choice and $1 \leq a \leq \lfloor b/2 \rfloor$. A working set theorem for BDL is proved in [9], given that B is a predefined constant.

Extension to sequences of routing requests. A straightforward extension of BDL for sequences of communication requests is the following algorithm, which we refer to as BDL^2. Upon a communication request $\sigma_t = (u, v)$, first $route(u, v)$ (as in Section II), then call $BDL(u)$, and finally call $BDL(v)$, where $BDL(x)$ is the promotion and demotion procedure of BDL upon a search request for element x. Naturally,

the cost of BDL^2 is $\mathcal{O}(\log|\overline{WS_u(\sigma_t)}| + \log|\overline{WS_v(\sigma_t)}|) = \mathcal{O}(\log|WS(u,v)|)$, since as we showed in Section IV (Theorem 2), $|WS(u,v)| = \Theta(\max\{|\overline{WS_u(\sigma_t)}|, |\overline{WS_v(\sigma_t)}|\})$.

B. Towards distributed versions of $SASL^2$ and BDL^2

To obtain a distributed version of $SASL^2$ and BDL^2 we employ tools from concurrent skip list implementations. We show that it is feasible to obtain a distributed $SASL^2$ by using the skip list algorithm of Herlihy et al. [15]. We then discuss the challenges of a distributed version of BDL^2 and how the complexity guarantees of the sequential versions are affected.

Herlihy et al. [15] presented a concurrent lock-based skip list that supports the methods $contains(v)$, $add(v)$, and $remove(v)$, for a node v, which base on a helper function $findNode(v)$. Locks are used only for $add()$ and $remove()$. Moreover, their skip list is single-linked horizontally and doubly-linked vertically. To surpass this restriction for routing from nodes with larger keys to nodes with smaller keys we add a single pointer from the top of the right sentinel to the top of the left sentinel (as in Figure 6). Thus, routing from node u to node v works as follows. If $u < v$, then the procedure is the same with the one described in Section II, since only forward pointers are needed. Otherwise, we route from u to the top of the right sentinel, from there we follow the pointer to the top of the left sentinel, and finally, we follow a standard skip list search route towards v.

We now describe the promotion and demotion procedures using the methods of [15] as a black box. Let x be an element and say that $SASL^2$ decides to change x's height during a promotion or demotion procedure. Denote by x' the desired state (height) of x after changing its current height. A call to $remove(x)$ followed by a call to $add(x')$ after the first method succeeds achieve the height change. The latter approach suffices for the promotion procedure, but not for demotion as it includes the selection of a node via a random walk and updates to the c_i values of the nodes in the search paths before and after the demotion. Thus, the demotion procedure calls $remove(x)$ with the addition of changing the c_i values when each node is locked, and once the latter succeeds, it calls $add(x')$ and again changes the c_i values during the critical section. Each of these procedures can be interpreted by specifically marked messages in the message passing model (a thread taking a forward pointer maps to a message passed to the target node).

It is also possible to follow a white box approach by implementing promotion only via the $add()$ method (instead of random height, we pick max height, and the node is already linked up until its old height) and demotion only via the $remove()$ method (instead of unlinking the node all the way, unlink until the new height). Updates of c_i values in a white box approach can be done by locking single nodes.

$SASL^2$ can thus be run as a distributed algorithm in a self-adjusting network, but even though all operations are linearizable, we do not have any guarantees on how concurrent requests are interleaved. It is possible that promotions and demotions of two different routing requests are interleaved in any possible order in case the requests are concurrent, given the linearized execution [15], e.g. routing of the two requests is followed by the two promotion steps, which are then followed by the two demotion steps. The feasibility of $SASL^2$'s distributed version bases on the design of $SASL$, i.e., it uses only *local* information and its complexity guarantees are not severely affected by a few interleavings, as the ones described above. In the context of [8] this design choice serves the purpose of low overhead in the external memory setting, and in our case it facilitates the design of a distributed version of $SASL$ and $SASL^2$. Note that the working set semantics for $SASL^2$ are preserved for any linearization of an execution of the distributed version of $SASL^2$.

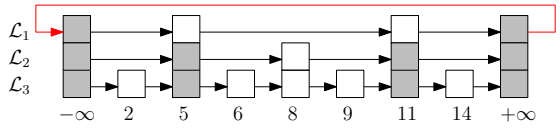

Fig. 6: By adding a link from the top of the right sentinel to the top of the left sentinel, we can route from larger to smaller items via that link. This modification of the concurrent skip list of Herlihy et al. [15] allows us to use it as a black box for implementing $SASL^2$. The shaded boxes denote the deterministic heights.

A distributed version of BDL^2 is more challenging; BDL relies on counting all the elements that are associated with a number of layers (at least one), thus relies less on local information compared to $SASL$. This counting might return inconsistent information in a high congestion setting, thus leading to unnecessary restructuring. In contrast, the order in which $SASL^2$'s operations are performed does not affect its invariants; each promotion is always combined with the correct number of demotions.

C. A variant of SplayNet [10] that has the pairwise working set property

We observe that the working set property can also be achieved in tree-based self-adjusting networks. A simple variant of SplayNet [10] that has the pairwise working set property, is to splay (i.e., perform the tree rotations of splay trees that preserve binary search [4]) the source to the root and the destination to be a child of the source. Recall that SplayNet splays the source to the tree rooted by the lowest common ancestor of the source and destination pair, and then splays the destination to be a child of the source. The pairwise working set property is straightforward from the working set property of splay trees and the fact that for a request $\sigma_t = (u,v)$ we have that $\mathcal{O}(\log|\overline{WS_u(\sigma_t)}|+\log|\overline{WS_v(\sigma_t)}|) = \mathcal{O}(\log|WS(u,v)|)$, since as we showed in Section IV (Theorem 2), $|WS(u,v)| = \Theta(\max\{|\overline{WS_u(\sigma_t)}|, |\overline{WS_v(\sigma_t)}|\})$.

However, there are some benefits of skip list networks compared to tree-based networks. Tree-based networks are not robust to failures, since a single link failure breaks the 1-connectivity. Also, skip list networks have more links (minimum degree is 2 for doubly-linked skip list networks)

than tree-based networks, thus providing more flexibility in adjusting the network to fit the communication demand among nodes. Moreover, there exist challenges that relate to the distributed implementation of SplayNet. The authors of [11], [12] had the limitation of ensuring that the source and destination of a routing request have to be at the top at the same time. In contrast, $SASL^2$ does not have this requirement, as the restructuring for the source and destination is independent, hence no coordination between them is needed.

VI. Related work

The related work to this paper is split between relevant results from self-adjusting networks (SANs) and related tools from self-adjusting data structures.

Self-adjusting Networks. Avin and Schmid positioned the self-adjusting network research in [16]. They characterize SAN algorithms in demand-oblivious (no adjustments occur) and demand-aware, which are either fixed but optimized given the communication frequencies, or reconfigurable, i.e., it is possible to adjust the network topologies over time. A first algorithmic result on SANs is SplayNet, a self-adjusting distributed binary search tree (BST) which generalizes splay trees [10] (Section V-C). Among other results, the authors present lower bound techniques for self-adjusting networks. A distributed implementation of SplayNet was presented in [11], [12] (we commented on this work in Section V-C). In relation to SplayNet, Avin et al. present demand-aware networks (DANs) of bounded degree in [17]. Their goal is to design a graph that minimizes the expected path length for a sequence of communication requests, given the communication pattern and a bound on the node degree. Moreover, [18] presents DANs that minimize congestion and route lengths, and [19] designs guarantees for relating path lengths with communication frequencies (but with no bounded degree). None of the above results have the working set property.

Huq and Gosh [20] have provided a self-adjusting skip graph that costs $\mathcal{O}((\log |\mathtt{WS}(\sigma)|)^2)$, where $\mathtt{WS}(\sigma)$ is the working set number of the communication sequence σ, defined in an alternative manner. Due to the quadratic exponent, [20] does not have the working set property. The working set of [20] is defined as follows. Let $\sigma_i = (u, v)$ be a communication request, σ_j be the last time that u and v communicated, and $G(\sigma_i)$ be a graph formed by all the nodes that were included in requests between σ_j and σ_i, such that the edges of $G(\sigma_i)$ are $\{\sigma_k \,|\, j \leq k < i\}$. Their working set definition for a communication request $\sigma_i = (u, v)$ is the size of the connected component including u and v in $G(\sigma_i)$ and is designed to suit their algorithm and the skip graph data structure. Their working set definition cannot be compared to ours; there are cases where one is larger than the other and vice versa.

For example, let $\sigma_i = \sigma_{i+x} = (u, v)$ and $u, v \notin (s_j, d_j)$, $j \in \{i+1, \ldots, i+x-1\}$, for an arbitrary x. Then $|\mathtt{WS}(\sigma_{\mathtt{i+x}})| = |\{u, v\}| = 2$ and our definition gives $|WS(\sigma_{i+x})| = |\{u, v, s_{i+1}, d_{i+1}, \ldots, s_{i+x-1}, d_{i+x-1}\}| = x$, if $x - 2$ distinct nodes participated in the requests $\sigma_{i+1}, \ldots, \sigma_{i+x-1}$. Conversely, suppose that $\sigma_i = \sigma_{i+x} = (u, v)$, u and v did not communicate in between, $\sigma_{i+x-1} = (s_{i+x-1}, u)$, $\sigma_{i+x-2} = (s_{i+x-2}, v)$, $s_{i+x-1} \neq v$, $s_{i+x-2} \neq u$, and the nodes in $\sigma_{i+1}, \ldots, \sigma_{i+x-1}$ form a single connected component in $G(\sigma_{i+x})$. Then $|\mathtt{WS}(\sigma_{i+x})| = |\{u, v, s_{i+1}, d_{i+1}, \ldots, s_{i+x-1}, d_{i+x-1}\}| = x$, in case $\sigma_{i+1}, \ldots, \sigma_{i+x-1}$ include x distinct nodes, and $|WS(\sigma_{i+x})| = |\{u, v, s_{i+x-2}, s_{i+x-1}\}| \leq 4$.

Self-adjusting Data Structures. Given the fundamental connection between SANs and self-adjusting data structures, we next review works on the latter that motivate or relate with ours. In [3] Sleator and Tarjan presented a dynamically optimal deterministic self-adjusting link list, which yields a 2-approximation by moving the accessed element to the list's front (move-to-front). In [4] they presented the splay tree, which is a self-adjusting binary search tree (BST), achieving static optimality, the working set property, as well as other optimality properties. Upon a search request, splay trees rotate (using unit cost operations and preserving the search properties), such that the searched item moves to the tree's root. Moreover, many other self-adjusting BSTs have been studied that are not pertinent to this work.

Skip lists were designed as a probabilistic alternative to balanced trees [13]. Bagchi et al. [14] designed a *biased* skip list (BSL) that is optimized for a given set of (abstract) weights and also defined finger search, through which we defined our local routing procedure in skip list networks. Moreover, the best known results on self-adjusting skip lists include the static optimality result due to Ciriani et al. [8] ($SASL$, Section III-A) and the dynamic optimality result due to Bose et al. [9] (BDL, Section V-A). Although both have been already presented earlier, we mention that BDL is dynamically optimal in its special class of skip lists, since the authors also prove a matching working set lower bound. For a comparison of different properties of self-adjusting data structures we refer to [4], [21]. For example, we note that the working set property implies static optimality [21].

We remark that a brief announcement of this work appeared in [22]. Moreover, SANs open new opportunities in networking, beyond this paper's scope. For example, [23] studies how to utilize the flexibility of SANs to reduce the cost of multicast traffic, which is frequently observed in datacenters.

VII. Conclusion

This paper presented a robust self-adjusting network based on a skip list topology which adapts itself to the demand, in an online manner, achieving the working set property. We also presented additional insights into working set properties of other networks.

We believe that our work opens several interesting avenues for future research. For example, it will be interesting to study the design of SANs for additional topologies and to explore deterministic algorithms. It will also be interesting to explore stronger properties, beyond working set, such as dynamic optimality. Our work lays the theoretical foundations for self-adjusting skip list-based networks, thus its implementation and realization aspects are a subject of future research.

References

[1] M. Ghobadi, R. Mahajan, A. Phanishayee, N. R. Devanur, J. Kulkarni, G. Ranade, P. Blanche, H. Rastegarfar, M. Glick, and D. C. Kilper, "Projector: Agile reconfigurable data center interconnect," in *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, M. P. Barcellos, J. Crowcroft, A. Vahdat, and S. Katti, Eds. ACM, 2016, pp. 216–229. [Online]. Available: https://doi.org/10.1145/2934872.2934911

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

[3] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, pp. 202–208, 1985. [Online]. Available: http://doi.acm.org/10.1145/2786.2793

[4] ——, "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.

[5] C. Avin and S. Schmid, "Renets: Toward statically optimal self-adjusting networks," *arXiv preprint*, 2019.

[6] C. Avin, K. Mondal, and S. Schmid, "Push-down trees: Optimal self-adjusting complete trees," *arXiv preprint arXiv:1807.04613*, 2018.

[7] J. Iacono, "In pursuit of the dynamic optimality conjecture," in *Space-Efficient Data Structures, Streams, and Algorithms*. Springer, 2013, pp. 236–250.

[8] V. Ciriani, P. Ferragina, F. Luccio, and S. Muthukrishnan, "A data structure for a sequence of string accesses in external memory," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 1, p. 6, 2007.

[9] P. Bose, K. Douïeb, and S. Langerman, "Dynamic optimality for skip lists and b-trees," in *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2008, pp. 1106–1114.

[10] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker, "Splaynet: Towards locally self-adjusting networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 24, no. 3, pp. 1421–1433, 2016.

[11] B. S. Peres, O. Goussevskaia, S. Schmid, and C. Avin, "Brief announcement: Distributed splaynets," in *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, ser. LIPIcs, A. W. Richa, Ed., vol. 91. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, pp. 58:1–58:3. [Online]. Available: https://doi.org/10.4230/LIPIcs.DISC.2017.58

[12] B. Peres, O. A. de Oliveira Souza, O. Goussevskaia, C. Avin, and S. Schmid, "Distributed self-adjusting tree networks," in *39th IEEE International Conference on Computer Communications (INFOCOM)*, April 2019. [Online]. Available: http://eprints.cs.univie.ac.at/5863/

[13] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[14] A. Bagchi, A. L. Buchsbaum, and M. T. Goodrich, "Biased skip lists," *Algorithmica*, vol. 42, no. 1, pp. 31–48, 2005.

[15] M. Herlihy, Y. Lev, V. Luchangco, and N. Shavit, "A simple optimistic skiplist algorithm," in *Structural Information and Communication Complexity, 14th International Colloquium, SIROCCO 2007, Castiglioncello, Italy, June 5-8, 2007, Proceedings*, ser. Lecture Notes in Computer Science, G. Prencipe and S. Zaks, Eds., vol. 4474. Springer, 2007, pp. 124–138. [Online]. Available: https://doi.org/10.1007/978-3-540-72951-8_11

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

[17] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network designs of bounded degree," in *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, ser. LIPIcs, A. W. Richa, Ed., vol. 91. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, pp. 5:1–5:16. [Online]. Available: https://doi.org/10.4230/LIPIcs.DISC.2017.5

[18] ——, "Demand-aware network design with minimal congestion and route lengths," in *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*. IEEE, 2019, pp. 1351–1359. [Online]. Available: https://doi.org/10.1109/INFOCOM.2019.8737431

[19] C. Avin, A. Hercules, A. Loukas, and S. Schmid, "*rDAN*: Toward robust demand-aware network designs," *Inf. Process. Lett.*, vol. 133, pp. 5–9, 2018. [Online]. Available: https://doi.org/10.1016/j.ipl.2017.12.008

[20] S. Huq and S. Ghosh, "Locally self-adjusting skip graphs," in *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*, K. Lee and L. Liu, Eds. IEEE Computer Society, 2017, pp. 805–815. [Online]. Available: https://doi.org/10.1109/ICDCS.2017.249

[21] J. Iacono, "Key-independent optimality," *Algorithmica*, vol. 42, no. 1, pp. 3–10, 2005.

[22] C. Avin, I. Salem, and S. Schmid, "Brief announcement: On self-adjusting skip list networks," in *33rd International Symposium on Distributed Computing*, October 2019. [Online]. Available: http://eprints.cs.univie.ac.at/6098/

[23] L. Luo, K.-T. Foerster, S. Schmid, and H. Yu, "Splitcast: Optimizing multicast flows in reconfigurable datacenter networks," in *INFOCOM*. IEEE, 2020.

[24] A. W. Richa, Ed., *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, ser. LIPIcs, vol. 91. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. [Online]. Available: http://www.dagstuhl.de/dagpub/978-3-95977-053-8