Brief Announcement: On Self-Adjusting Skip List Networks

Chen Avin
Communication Systems Engineering Department,
Ben Gurion University of the Negev, Beersheva, Israel
avin@cse.bgu.ac.il

Iosif Salem
Faculty of Computer Science, University of Vienna, Austria
iosif.salem@univie.ac.at

Stefan Schmid
Faculty of Computer Science, University of Vienna, Austria
stefan_schmid@univie.ac.at

Abstract

This paper explores the design of dynamic network topologies which adjust to the workload they serve, in an 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. 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 skip lists (which serve as the topology) and provide additional interesting properties such as local routing.

2012 ACM Subject Classification Networks → Topology analysis and generation

Keywords and phrases self-adjusting networks, skip lists, working set, online algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.35

1 Introduction

We design scalable and robust self-adjusting networks (SANs) [1, 5], 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 [4]. Skip lists are not only interesting for data structures but also for networks as they, e.g., provide *local routing*. This is particularly useful in the context of dynamic topologies, which change over time, since we do not have to distribute information about new routing tables.

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 algorithm is based on a straight-forward extension of classic self-adjusting data structures, using a "move-to-front" (MTF) data structure as a subroutine.

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

We first provide some background knowledge and modeling details. Then we present our self-adjusting algorithm for skip list networks, $SASL^2$.

Skip lists networks. The skip list [4] 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

33rd International Symposium on Distributed Computing (DISC 2019).
Editor: Jukka Suomela; Article No. 35; pp. 35:1–35:3
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Algorithm 1 $SASL^2$: Self-adjusting Algorithm for Skip List Networks.

```
upon request $(u, v)$
    route $(u, v)$;
    for $x \in \{u, v\}$ do $adjustSASL(x)$;
$adjustSASL(z)$
    $b \leftarrow \mathcal{B}(z)$;
    $promotion(z, \mathcal{B}_1)$;
    $UpdCountersProm(z)$;
    for $i = 1, \dots, b - 1$
        $x \leftarrow RandomSelect(i)$;
        $demotion(x, \mathcal{B}_{i+1})$;
        $UpdCountersDem(x)$;
```

two special nodes *head* and *tail*, 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 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 $\mathcal{L}_1, \dots, \mathcal{L}_H$, where $|\mathcal{L}_i| = \Theta(2^i)$ and $\mathcal{L}_i \subset \mathcal{L}_{i+1}$, for $i \in \{1, \dots, H-1\}$. 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 $\mathcal{L}_{i+1}$, $i = 1, \dots, H-1$. We refer to an item's right neighbor in a list as its successor and to its left neighbor as its predecessor. The search procedure for a node v starts at the top of $-\infty$ and proceeds forward until reaching a node $w \neq v$ such that its successor has a larger key than $key(v)$. Then, the search moves to the list below and continues until the same condition is satisfied. The search ends either (successfully) when the node is found, or when it reaches the bottom list and the condition for moving lower holds.

A skip list can be also viewed as a graph or skip list *network*, where the node set is $V = \{v_1, v_2, \dots, v_n\}$, denoting routers or servers, 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 the triples (h, dir, x), for each level $h = 1, 2, \dots, h(x_i)$, direction $dir \in \{left, right\}$, and adjacent key $x \in \{x_1, x_2, \dots, x_n\}$. The data structure and graph point of view are equivalent.

Routing in skip list networks. In data structure terms, a routing request is quite similar to finger search [2], i.e. a search request that originates in an item resp. *node* $u \neq -\infty$ towards another node v. We consider the following procedure that requires only local information. 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 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 (left) if the next item is smaller (larger) than v, otherwise it moves one level down and repeats the rightward (leftward) search, until locating v.

Prior work: Randomized self-adjusting skip lists for search sequences. Ciriani et al. [3] 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. $SASL$ 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 $i \in [1, x-1]$. To drive the random walk, each node x maintains a set of counters $c_i(x)$ that keeps the number of elements reachable in each band $\mathcal{B}_i$ via a classical skip list search starting from x.

Extending from data structures to networks: $SASL^2$. We present an extension of $SASL$ to the case of routing in self-adjusting skip list networks. Our algorithm $SASL^2$ (Algorithm 11) uses the adjustment part of $SASL$, i.e. the promotion and demotion procedures, as a black box. Let $adjustSASL(z)$ be the adjustment part of $SASL$ with input z [3]. Upon a communication request (u, v), $SASL^2$ serves the request and then calls $adjustSASL(u)$ and subsequently $adjustSASL(v)$. A call to $adjustSASL(z)$ promotes z to the top band, $\mathcal{B}_1$ (line 6), updates the counters of a skip list search to z after its promotion (line 7), and then demotes an element x uniformly at random from $\mathcal{B}_i$ to $\mathcal{B}_{i+1}$, for $i = 1, \dots, b-1$ (lines 8–11), where b (line 5) is the band in which z belonged before its promotion. For a single demotion from band $\mathcal{B}_i$, $adjustSASL(z)$ first selects uniformly at random a node x from band $\mathcal{B}_i$ (line 9), reduces x's height such that x belongs to $\mathcal{B}_{i+1}$ (line 10), and updates the counters in a skip list search to x before and after its demotion to the next band (line 11).

Working set theorem for sequences of communication requests in $SASL^2$. Intuitively, the working bag of a communication request $\sigma_t = (s_t, d_t)$ is the shortest suffix of the sequence $\sigma = (\sigma_1, \dots, \sigma_{t-1})$ that includes 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. Our working bag and set definitions are suitable for topologies that have a front/top, and it is thus possible to design algorithms that follow a move-to-front/move-to-top principle. 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.

▶ **Definition 1** (Working bag and working bag number). *Let $\sigma = (\sigma_t = (s_t, d_t))_{t \in \{1,\dots,m\}}$ be a sequence of communication requests. We define the (pairwise) working bag of a communication request $\sigma_t = (s_t, d_t)$ to be $(\sigma_1, \dots, \sigma_t)$, if s_t or d_t appear in a request of σ for the first time at time t and $WB(s_t, d_t) = \min\{\sigma' \sqsubseteq (\sigma_1, \dots, \sigma_{t-1}) \mid last(\sigma') = \sigma_{t-1} \wedge \exists_{\sigma_i, \sigma_j \in \sigma'}\, s_t \in \sigma_i \wedge d_t \in \sigma_j\}$ otherwise, where $\sqsubseteq$ denotes the suffix relation and $last(\sigma')$ returns the last request of a sequence σ'. We denote by $|WB(s_t, d_t)|$ the size of $WB(s_t, d_t)$ 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 $WS(\sigma_t) = WS(s_t, d_t) = \{x \in \sigma_i \mid \sigma_i \in WB(s_t, d_t)\}$. The working set number of σ_t is the size of $WS(s_t, d_t)$ and we denote it by $|WS(s_t, d_t)|$.*

▶ **Theorem 3.** *For any communication request (u, v), $SASL^2$ achieves the pairwise working set property: $E[cost(SASL^2(u, v))] = \mathcal{O}(\log |WS(u, v)|)$.*

References

1 Chen Avin and Stefan Schmid. Toward demand-aware networking: a theory for self-adjusting networks. *ACM SIGCOMM Computer Communication Review*, 48(5):31–40, 2019.

2 Amitabha Bagchi, Adam L Buchsbaum, and Michael T Goodrich. Biased skip lists. *Algorithmica*, 42(1):31–48, 2005.

3 Valentina Ciriani, Paolo Ferragina, Fabrizio Luccio, and S Muthukrishnan. A data structure for a sequence of string accesses in external memory. *ACM TALG*, 3(1):6, 2007.

4 William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

5 Stefan Schmid et al. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Transactions on Networking (TON)*, 24(3):1421–1433, 2016.