An Evaluation of Dynamic Labeling Schemes for Tree Networks

Noy Rotbart, Marcos Vaz Salles, and Iasonas Zotos

Department of Computer Science, University of Copenhagen (DIKU)
Universitetsparken 5, 2100 Copenhagen

{noyro,vmarcos}@diku.dk, tlc736@alumni.ku.dk

Abstract. We present an implementation and evaluation based on simulation of dynamic labeling schemes for tree networks. Two algorithms are studied: a general scheme that converts static labeling schemes to dynamic, and a specialized dynamic distance labeling scheme. Our study shows that theoretical bounds only partially portray the performance of such dynamic labeling schemes in practice. First, we observe order-of-magnitude differences between the gains in label size when compared to the number of messages passed. Second, we demonstrate a significant bottleneck in the tree network, suggesting that the current practice of counting total messages passed in the whole network is insufficient to properly characterize performance of these distributed algorithms. Finally, our experiments provide intuition on the worst case scenarios for the stated algorithms, in particular path tree networks and fully dynamic schemes permitting both node additions and deletions.

1 Introduction

Labeling schemes are methods used to calculate global properties of a graph based solely on local information present at its nodes. They were introduced in a restricted manner by Bruer and Folkman [5], revisited by Kannan, Naor and Rudich [12], and explored by a wealth of subsequent work [2–4, 10, 19, 20]. Due to their main role in the understanding of graph structures and their practical relevance, trees are central in these studies. In more detail, asymptotically tight labeling schemes for trees were found for the functions adjacency [4], sibling [2], ancestry [10], nearest common ancestor (NCA) [3], routing [20], and distance [19]. The papers mentioned aim to find labeling schemes that minimise the maximum label size, with minor attention to efficient encoding and decoding of the labels.

Korman, Peleg and Rodeh [18] studied labeling schemes in the context of distributed systems. In this context, information about labeling is exclusively communicated via messages, and nodes may join or leave the system dynamically. Since communication is required, maximum label size is not the sole criterion for the quality of such labeling schemes, but also metrics related to the number and total size of messages passing in the network. The authors suggested two algorithms that fit this context: a conversion of static labeling schemes, and a specialized distance labeling scheme. Additional work investigated specialized labeling schemes for routing [15, 16] and various trade-offs for distance [14, 17].

One could attempt to design dynamic labeling schemes without node relabeling at all. Unfortunately, Cohen, Kaplan and Milo [8] showed that if relabeling is not permitted, there is a a tight bound of $\Theta(m)$ bit for labels supporting m insertions. The lower bound shows the inapplicability of dynamic labeling schemes without relabeling for most functions.[1] For the above reasons, we focus on dynamic models with re-labeling. In particular, we focus on the function distance, since distance labels are expressive enough to answer all the queries mentioned above.

Previous experimental papers on the topic focused on the performance of static labeling schemes, emphasizing the label size as the main metric. Caminiti et al. [6] experimented on the influence that different tree decompositions have on label size for NCA labeling schemes. Fischer [9] evaluated the label size expected for a variant of NCA, using various coding algorithms. Kaplan et al. [13] presented experiments to support an ancestry labeling scheme trading slightly larger labels for much faster query time. Finally, Cohen et. al [7] performed experiments on a novel technique for labeling schemes for graphs.

These papers reveal that different families of trees present very different label sizes. A natural question is whether dynamic labeling schemes present the same behavior. In addition, it remains unclear how the various communication and label size trade-offs suggested for dynamic labeling schemes behave in practice. Moreover, one may question whether it is sensible to develop dynamic labeling schemes for individual functions, or whether the trade-offs given by general methods are good enough [18]. Furthermore, the price of allowing for deletions, rather than just insertions, still needs to be characterized in representative scenarios. In short, many questions are relevant and interesting for dynamic labeling schemes, but we are unaware of any experimental papers on the topic.

In this paper, we devise a simulation of a tree network and answer those questions for the dynamic labeling schemes proposed by Korman, Peleg and Rodeh [18]. These labeling schemes can be classified along two dimensions: whether the labeling scheme is specialized for distance or supports general static labelers, and whether the labeling scheme supports both insertion and deletions or only insertions. Our findings suggest three main observations:

1. Generally, the amortized complexity analysis of [18] is verified in our experiments. However, for labeling schemes supporting general static labelers, the observed behavior indicates that the complexity of the static labeler is overshadowed by the overhead introduced by the dynamic scheme.
2. In order to achieve an asymptotically tight label size, the specialized distance labeling scheme uses significantly more communication.
3. The message distribution in labeling schemes that support both insertion and deletion is severely skewed. We show an experiment in which the majority of messages in the system is communicated by less than 0.4% of the nodes.

[1] More specifically: NCA, routing, distance, flow and center. In contrast, siblings, adjacency and connectivity enjoy simple dynamic labeling schemes of size $2\log n$.

1.1 Preliminaries

In the remainder of the paper, we call the collection of all n-node unweighted trees $Trees(n)$. We assume trees to be rooted, and denote $\log_2 n$ as $\log n$. We adopt the terminology in [18], which names dynamic encoding algorithms as *distributed online protocols* or simply *protocols*. From here on, all dynamic labeling schemes discussed are for tree networks. The following types of topology changes are considered: *a*) *Add-Leaf*, where a new degree-one node u is added as a child of an existing node v ; and *b*) *Remove-Leaf*, where a (non-root) leaf v is deleted. Dynamic labeling schemes that support only *Add-Leaf* are denoted *semi-dynamic*, and those that support both operations are denoted *fully-dynamic*.

Metrics for dynamic labeling schemes. In the literature surveyed, labeling schemes aim for smallest possible label. If communication is not accounted for, dynamic labeling schemes can trivially achieve the optimal, static bounds. Therefore, in order to account for the cost of communication, the following metrics are introduced [18]. Let M be a dynamic labeling scheme, with re-labeling allowed, and let $S(n)$ be a sequence of n topological changes.

1. *Label Size* $\mathcal{LS}(M, n)$: the maximum size of a label assigned by M on $S(n)$.
2. *Message Complexity*, $\mathcal{MC}(M, n)$: the maximum number of messages sent in total by M on $S(n)$. Messages are sent exclusively between adjacent nodes.
3. *Bit Complexity*, $\mathcal{BC}(M, n)$: the maximum number of bits sent in total by M on $S(n)$. Note that $\mathcal{BC}(M, n) \leq \mathcal{MC}(M, n) \cdot \mathcal{LS}(M, n)$.

2 Dynamic Labeling Schemes For Tree Networks

Korman et al. [18] presented *semi-dynamic* and *fully-dynamic* labeling schemes that support distance queries and a *semi-dynamic* and *fully-dynamic* labeling scheme that receives a static labeling scheme[2] as input and supports queries of its type. In this section, we give a brief and informal description of both semi-dynamic labeling schemes as well as of the conversion necessary to make these schemes fully-dynamic.

We denote the static distance labeling scheme as *Distance*, its extension to a specialized semi-dynamic mode *SemDL*, and its fully-dynamic specialized extension as *DL*. Korman et al. [18] define the general dynamic labeling schemes *SemGL* and *GL*, which maintain labels on each node of a dynamically changing tree network using a static labeling scheme as a subroutine. The performance of these schemes is tightly coupled to the performance of the static scheme used. Throughout the remainder, we denote the general labeling schemes operating on *Distance* simply as *SemGL* and *GL*, and explicitly mention other distance functions where appropriate. Table 1 reports the different complexities for the aforementioned schemes (the parameter d is explained in Sec. 2.1).

[2] The static labeling scheme must respect a set of conditions as mentioned in [18], which to the best of our knowledge are respected by all labeling schemes in the literature.

Labeling Scheme	Label Size	$\mathcal{MC}$	$\mathcal{BC}$
Distance	$O(\log^2 n)$	$O(n)$	$O(n \log^2 n)$
SemDL	$O(\log^2 n)$	$O(n \log^2 n)$	$O(n \log^2 n \log \log n)$
DL	$O(\log^2 n)$	$O(n \log^2 n)$	$O(n \log^2 n \log \log n)$
SemGL	$O(\frac{d-1}{\log d} \log^3 n)$	$O(n \frac{\log n}{\log d})$	unreported
GL	$O(\log^3 n)$	$O(n \log^2 n)$	unreported

Table 1: Simplified complexity estimates for $Trees(n)$. Bounds for GL are reported with $d = 2$. See [18] for an elaborate complexity report.

2.1 Brief Overview

In this section, we provide an informal description for *Distance*, *SemDL*, *SemGL*, and the approach for semi-dynamic to dynamic conversion.

Encoding *Distance* directly from heavy-light decomposition. For every non-leaf v in a rooted tree T, we denote a child with the heaviest weight[3] as *heavy* and the rest as *light*; we mark the root of the tree as light and its leaves as heavy. The light nodes divide T into disjoint *heavy paths*. For any $v \in T$, the path from the root traverses at most $\log n$ light nodes, and accordingly at most $\log n$ heavy paths. A label of a node can now be defined by interleaving *light sub-labels*, containing the index of a light child, with *heavy sub-labels*, containing only the length of the heavy path. Such labels clearly require $O(\log^2 n)$ bits, which is asymptotically optimal for labels supporting distance on trees. Computing the distance between two nodes can be done by comparing the prefixes of their corresponding labels. Finally, we note that the labels produced by *Distance* can be used to solve most queries in the literature, namely adjacency, sibling, ancestry, routing and NCA.

SemDL. It is not essential for the correctness of the decomposition that the heavy node selected be in fact the heaviest, or say, the 3rd heaviest. It is only essential for the bound on the label size. *semDL* maintains a dynamic version of the *Distance* labels using precisely this observation. Every node maintains an estimate of its weight, and transfers this estimate to its parent, using a previously introduced binning method [1]. When the estimate exceeds a threshold in a node, this implies that a node other than the heavy node is now significantly heavier than the heavy node. Thus, a re-labeling (shuffle, in [18]) is instantiated on the subtree to maintain the $O(\log^2 n)$ label size.

SemGL. The protocol is designed to transform any static labeling scheme *Static* to semi-dynamic, and therefore, does not utilize the heavy-light decomposition directly. Instead, the protocol operates *Static* separately on a cleverly constructed sets of so called bubbles described as follows.

Each node $v \in T$ is included in exactly one induced subtree of T, denoted *bubble* of *order* i ($0 \leq i \leq \log_d n$) that contains at least d^i nodes. The bubbles constitute a *bubble tree*, where the order of a bubble is always less or equal to the order of the parent bubble. In addition, there are no d consecutive bubbles of the same order in any path of the bubble tree. A node added to the tree is

[3] The weight of a node v is the number of its descendants in a tree.

assigned the order 0. If this insertion yields d consecutive bubbles of order 0 in the bubble tree, they are merged to a single bubble of order 1, and the condition is checked again for bubbles of order 1 and so on. We denote the parent of the root of bubble b as b_p and the function that *Static* supports as f. Essentially, the label of a node v in a bubble b is a concatenation of the label of b_p with both a local *Static* label of v in b, and the result of $f(v, b_p)$.

Semi-dynamic to fully-dynamic conversion. Both labeling schemes are converted to their counterpart fully-dynamic labeling schemes, *DL* and GL, using an additional simple protocol. The protocol uses the binning method [1] to maintain local weight estimates for each node. The protocol simply ignores the topological changes up to the point in which their number is large, and then re-labels the entire tree.

3 Experimental Framework

We have used simulation as our performance measurement methodology [11]. There are several reasons for this choice. First, distributed systems of realistic size in the order of millions of nodes are unavailable to us. Second, simulation allows us to remain isolated from effects such as network interference. Third, simulation is a sufficient tool to compare the metrics intended for our purposes. All implementations have been realized in C#. The full package is available over the Internet at the URL: `http://www.diku.dk/~noyro/dynamic-labeling.zip`.

Performance Indicators. In order to match the analysis, we use the metrics defined above, namely $\mathcal{MC}$, $\mathcal{BC}$, and $\mathcal{LS}$, denoted *total network messages*, *total network bandwidth*, and *maximum label length* resp. In particular, we are interested in the tradeoff between $\mathcal{LS}$ and $\mathcal{MC}$ in the different systems. Even though we report the bit complexity, message sizes in our setting are bounded by the label sizes, which tend to be very small. Since in networks the start up costs of sending small messages dominate messaging cost, it is sufficient to focus on message complexity alone. In addition, we study the distribution of messages passing through a single node. The latter provides insight into the congestion for worst-case network performance.

The algorithms present slightly improved bounds if the number of topological events n is known in advance. We simulate labeling schemes that are not aware of n. It is beyond the scope of our paper to account for neither the construction time, nor the performance of the decoder.

Selected static functions. To test *SemGL* and *GL*, we provide, in addition to *Distance*, the static labeling scheme *Ancestry* [12] with labels of size $2 \log n$.

Test Sets. We consider insertions sequences creating the following trees.

- **Complete K-ary Trees.** K-ary trees are trees that have a maximum number of children allowed per node. A complete K-ary tree is a K-ary tree in which all leaf nodes reside on the lowest level and all other nodes have K children. The leaves are inserted starting from the leftmost nodes of the tree. We denote a 2-ary tree as *full binary tree*.

- **Skewed Trees: star and path.** A star is a rooted tree with children adjacent to the root. In a path tree, all nodes have either one or no children.
- **Random Trees.** We implemented random insertion sequences creating trees of bounded depth δ as well as of unbounded depth. While generating trees that are truly random is challenging, we follow a simple iterative procedure. First, select a node with equal probability among the nodes with depth less than δ. Then, insert a new node under the selected node and iterate.

From [6, 9], we conclude that labels constructed using a heavy-light decomposition yield the largest label size when applied on full binary trees. Since we would like to focus on the performance of the dynamic protocols, we use full binary trees as our main test set.

Counted Bits. The labels constructed have polylogarithmic size, and each is composed of a number of components of variable sizes. Therefore, we must account for a realistic bit encoding. Bit accounting matters for both $\mathcal{LS}$ and $\mathcal{BC}$. All labels are accounted by twice their theoretical label size, so that the encoder could find the beginning and the end of each of the components. The same applies for messages. We use unary encoding to represent the number of non-empty elements in each row of the label matrix [18].

4 Experimental Results

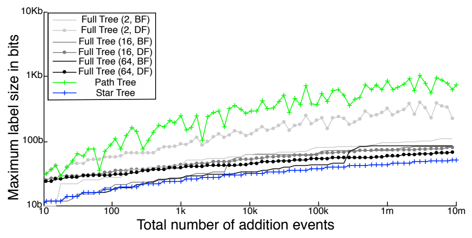
In this section, we summarize our main experimental findings. First, we test *semGL* with two static labeling schemes and outline the role d plays in the protocol (Section 4.1). We then proceed by directly comparing *SemGL* to *SemDL* (Section 4.2). We conclude by observing the performance of *GL* in comparison to *SemGL* and *DL*, as well as comparing the distribution of messages in *GL* and *SemGL* (Section 4.3). All figures presented but Figures 3 and 7 have the number of node additions in their x-axes in a log-scale. Figures describing $\mathcal{MC}$ and $\mathcal{BC}$ have their y-axes in log-scale, and figures describing $\mathcal{LS}$ are in linear scale.

4.1 *SemGL*

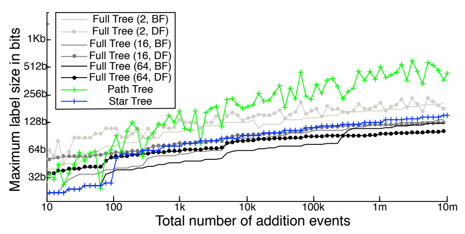
We discuss two experiments dealing exclusively with the performance of *semGL*.

Varying tree type. For the purposes of this section, we compare *SemGL* under two static labeling schemes, namely *Distance* and *Ancestry*. The test is performed on eight different, deterministically constructed trees, where the full K-ary trees are expanded either in a *breadth-first* or a *depth-first* manner. We fix $d = 4$, as it best illustrates the differences observed.

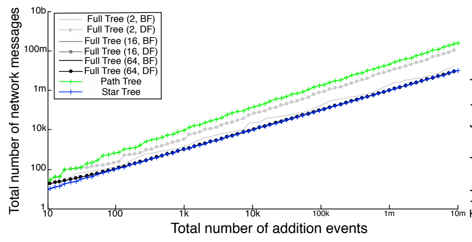
Overall, we observe similar trends for both static labeling schemes (Figures 1a, 1c, and 1e vs. 1b, 1d, and 1f). However, we observe an apparent anomaly on the largest label sizes produced in Figure 1b. While in general labels for *Ancestry* should be more compact than labels for *Distance*, the opposite effect is evident for paths. This effect can be explained when we recall that on a path, the label size of *Distance* is a straightforward $\log n$ bits, whereas the label size of *Ancestry* is fixed to $2 \log n$.



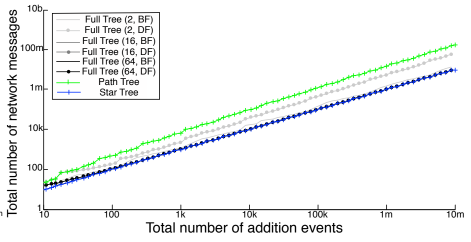
(a) *Ancestry*- $\mathcal{LS}$



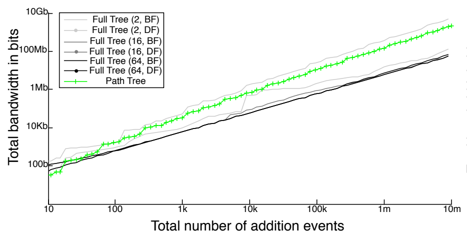
(b) *Distance*- $\mathcal{LS}$



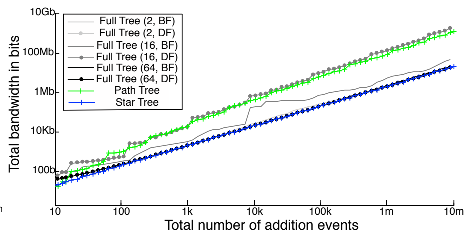
(c) *Ancestry*- $\mathcal{MC}$



(d) *Distance*- $\mathcal{MC}$



(e) *Ancestry*- $\mathcal{BC}$



(f) *Distance*- $\mathcal{BC}$

Fig. 1: *SemGL* on *Distance* and *Ancestry* static labeling schemes, for various trees of different order and node insertion modes, with $d = 4$.

We observe that the maximum label size of *SemGL* is directly related to the depth of the tree, such that trees with higher depth yield larger labels. In addition, trees with higher depth have more variance in their maximum label size. Both of these properties can be attributed to the *bubble tree* structure. Since the bubble order directly relates to the maximum node depth, higher depth leads to larger *bubble order*, which yields a larger label size. The second property, namely higher variance of maximum label size in trees of higher depth, is attributed to more frequent high-level relabeling. At the noticeable decline points, the encoder relabels large parts of the tree to ensure the logarithmic label size.

A clear evidence for the correlation of label size to the depth of the tree is that for random trees of bounded and unbounded depth, the label size grows consistently with depth, as shown in Figure 2. Comparing Figure 2 to Figure 1, the values for random trees lie between the values for path and the remaining trees. We also note that while the total number of messages is essentially iden-

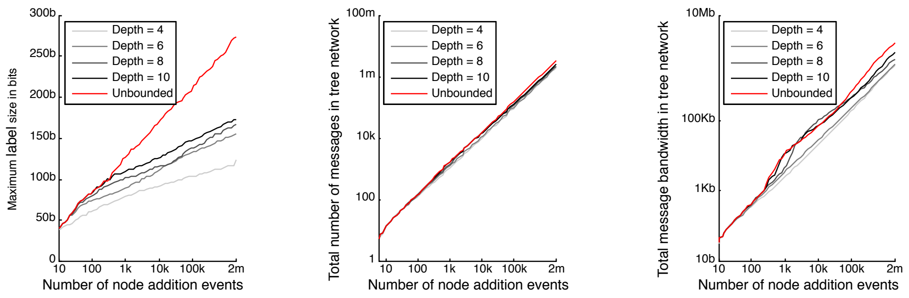


Fig. 2: *SemGL* labels for random trees with variable bounded depth. Each curve represents the average of 10 executions, with $d = 4$.

tical, the bit complexity is higher as the depth is higher. We deduce that the depth influences the size of each message, and not just the number of messages.

Given these results, in the rest of the experiments we focus on full binary trees traversed in depth-first manner. We have observed in separate experiments that these trees generate similar trends as random trees.

Varying the d variable. In Figure 3, we vary the value of d in values between 2 and 16, and analyze its influence on *SemGL*. We choose a set of low d values to trigger larger overhead on the message complexity of *SemGL*, such that performance changes are more noticeable. The results are similar for *SemGL* with the *Ancestry* static labeling scheme, and thus omitted from the figure.

We observe two trends in the figures. First, the number of messages passed in the network drops logarithmically as we increase d. The same drop can be observed for the total bandwidth. In addition, the label size increases as d increases. This trend is in line with the expected $\frac{d-1}{\log d}$ expression reported in Table 1.

In addition, we notice a decay in number of messages in Figure 3 which is also in line with the expected $O((\log d)^{-1})$ curve for $\mathcal{MC}(SemGL, n)$. The total bandwidth consumed by tree network reacts to d in the same manner that the total number of messages does. The values stabilize when $d = 12$ for approx. 5 m (million) nodes and 11 for 1 m nodes. Since both complexities are showing similar behavior throughout all experiments, we omit graphs describing the total bandwidth in the remainder for brevity.

4.2 Comparison of *SemGL* and *SemDL*

Figure 4a shows as expected that *SemGL* produces larger labels than *semDL*. Recall that by Table 1 we expect a log factor in the label size between both methods. For trees with 4m nodes, the latter translates into a factor of 20, in contrast to the factor of four seen in the figure. On the other hand, Figure 4b shows that the distance-specialized *semDL* uses up to 270 times more messages in comparison to *SemGL*, when we would again by Table 1 expect a reverse multiplicative log factor[4]. This suggests that the trade-off between label size and

[4] Note that d is a low constant in our experiment

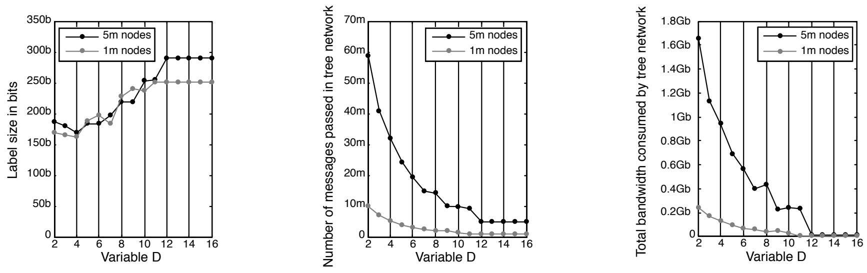
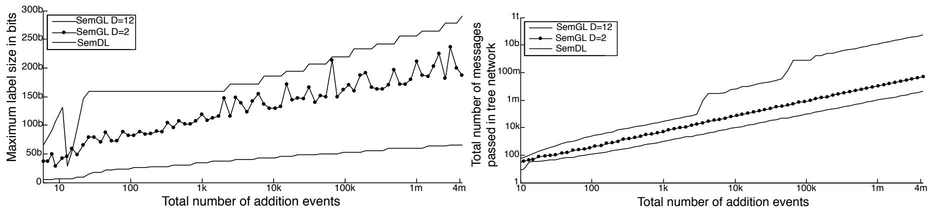


Fig. 3: *SemGL* Variable d experiment for full binary trees expanded in a depth-first manner. Samples are taken for d in [2,16] in trees of size 1m and 5m nodes.



(a) (b)

Fig. 4: *SemGL* versus *SemDL* on full binary trees expanded in a depth-first manner: (a) the maximum label size; (b) the number of messages sent.

communication expected by the reported complexities is in practice overshadowed by significant constants. The ratio of the factor differences amounts to two orders of magnitude.

Comparing Figures 4a and 4b, *semDL* presents a stable increase in the label size in contrast to the two bumps in its message size. The latter is in contrast to the previously explained phenomena in *SemGL*. The two bumps mark a re-label operation done from the root, and contribute to almost an order-of-magnitude difference in the number of messages passed.

4.3 Fully-dynamic labeling schemes

In this section, we analyze the performance of fully-dynamic protocols, and the performance differences between semi-dynamic and dynamic labeling schemes. We use a full binary tree, generated in a depth-first manner.

Comparison of *DL* and *GL*. In this experiment, we compare *DL* and *GL* for tree expansion, i.e., additions of nodes to the tree. We remark that *SemGL* exhibited a definite advantage in $\mathcal{MC}$ over *semDL* (Section 4.2). Similarly, we note in Figures 5a and 5b that *GL* performs significantly better than *DL* in terms of network messages. For 4m nodes, *GL* uses roughly 100 times less messages compared to *DL*. At the same time, as expected *DL* yields labels that are up to five times smaller than *GL*.

Comparison of *semGL* and *GL*. In this experiment, we compare *semGL* and *GL* again over a tree expansion. We emphasise that no deletion operations

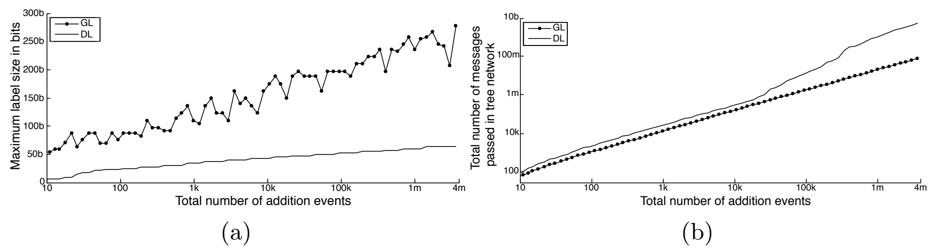


Fig. 5: Comparison of *DL* and *GL* (a) is the maximum label size measured in bits and (b) is the total number of messages passed

are performed. We aim to give an insight on the overhead caused by transforming the semi-dynamic labeling scheme to fully-dynamic.

In Figure 6, we view that the maximum label size is lower for *GL*. However, as we scale on the number of nodes, both curves tend to approach the same value. GL's number of messages is, however, 100 times larger than *semGL* for 4m nodes. This constitutes a significant overhead in the execution of the fully-dynamic labeling scheme. For perspective, 4.3m messages suffice to label 4m nodes using *semGL*, but only 0.23m nodes using *GL*.

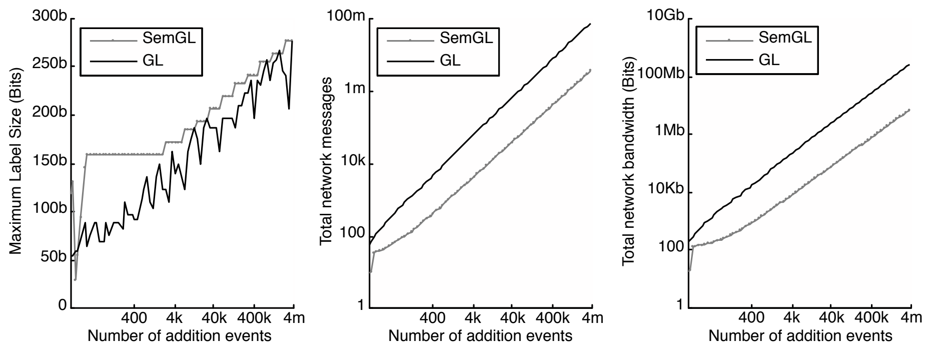


Fig. 6: Comparison of *semGL* ($d = 12$) and *GL*

messages (range)	nodes	Percent of nodes	Percent of messages
0 - 5	500018	50.00%	2.6%
5-50	468725	46.87%	23.4%
51-500	27366	2.74%	15.9%
501-5000	3646	0.36%	20.86%
5001-50000	215	0.02%	15.66%
50001-500000	30	0.00003%	21.55%

Table 2: The distribution of messages per node for *GL*, with $d = 14$.

Message distribution. We group the nodes according to the number of messages that pass through them throughout the *GL* protocol. Table 2 shows that the distribution of messages is remarkably skewed. Interestingly, 0.00003% of the nodes account for 21.55% of the total number of messages passed. Each of the top 30 nodes in the distribution have either sent or received between $50001 - 500000$ messages. The top node, in particular, has either sent or received about 0.5m out of a total of 19m messages. We have verified that the top node in the distribution corresponds to the root, and the top 30 nodes are the shallowest nodes in the tree.

In contrast, Figure 7 shows the distribution of messages for *semGL* on a worst case path network with 1m nodes. *semGL* does not incur similar bottlenecks as *GL*. Instead, the figure shows a distribution that resembles a bell curve: Most nodes either send or receive around 20 messages, and the differences in number of messages among nodes are not stark.

This result suggests that *GL* creates a network bottleneck by concentrating the messages passed. This behavior is undesirable in a distributed system.

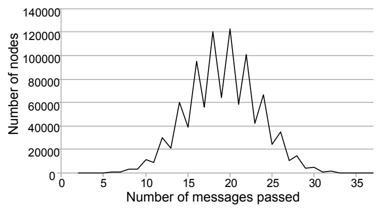


Fig. 7: The distribution of messages per node for *semGL*, with $d = 2$ on a path.

5 Conclusions

Overall, the algorithms perform as expected given the amortized complexity analysis. However, the experiments reveal constant factors that clearly affect performance. We observed that fully-dynamic protocols induce a severe performance overhead when compared to semi-dynamic variants. Moreover, as seen in the distribution of messages passed, fully-dynamic protocols also lead to severe bottlenecks in the tree network.

We affirm Korman et al.'s recommendations, namely to reset the d parameter of *GL* on a restart, and for *semGL*, to predict, if plausible, the target tree length n_f in order to set d to a value close to $\log n_f$. On the other hand, Korman et al. argue for trading network communication for improved label size in *DL* [18]. In light of our results, we suggest that in practice, the opposite trade-off is required, namely larger labeling schemes with decreased message complexity.

Korman et al. have developed *DL* with a metric that accumulates network messages, amortizing this value over a number of insertions. Our experiments show that such a metric allows for major bottlenecks and non-distributed system in practice. We suggest that a revised metric that accounts for the total number of messages passing through a single node be investigated as part of future work.

Bibliography

[1] Afek, Y., Awerbuch, B., Plotkin, S., Saks, M.: Local management of a global resource in a communication network. JACM 43(1), 1–19 (1996)

[2] Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. SIAM J. Discret. Math. 19(2), 448–462 (Jun 2005)

[3] Alstrup, S., Halvorsen, E.B., Larsen, K.G.: Near-optimal labeling schemes for nearest common ancestors. arXiv preprint arXiv:1312.4413 (2013)

[4] Alstrup, S., Rauhe, T.: Small induced-universal graphs and compact implicit graph representations. In: FOCS '02. pp. 53–62. FOCS '02 (2002)

[5] Breuer, M.A., Folkman, J.: An unexpected result in coding the vertices of a graph. J. Mathematical Analysis and Applications 20, 583–600 (1967)

[6] Caminiti, S., Finocchi, I., Petreschi, R.: Engineering tree labeling schemes: A case study on least common ancestors. In: Algorithms-ESA 2008, pp. 234–245. Springer (2008)

[7] Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queries via 2-hop labels. SIAM J. Comp. 32(5), 1338–1355 (2003)

[8] Cohen, E., Kaplan, H., Milo, T.: Labeling dynamic xml trees. SIAM Journal on Computing 39(5), 2048–2074 (2010)

[9] Fischer, J.: Short labels for lowest common ancestors in trees. In: ESA. pp. 752–763 (2009)

[10] Fraigniaud, P., Korman, A.: An optimal ancestry scheme and small universal posets. In: STOC '10. pp. 611–620. STOC '10 (2010)

[11] Jain, R.: The art of computer systems performance analysis, vol. 182. John Wiley & Sons Chichester (1991)

[12] Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. In: SIAM Journal On Discrete Mathematics. pp. 334–343 (1992)

[13] Kaplan, H., Milo, T., Shabo, R.: A comparison of labeling schemes for ancestor queries. In: SODA'02. pp. 954–963 (2002)

[14] Korman, A.: General compact labeling schemes for dynamic trees. Distributed Computing 20(3), 179–193 (2007)

[15] Korman, A.: Improved compact routing schemes for dynamic trees. In: PODC '08. pp. 185–194. ACM (2008)

[16] Korman, A.: Compact routing schemes for dynamic trees in the fixed port model. Distributed Computing and Networking pp. 218–229 (2009)

[17] Korman, A., Peleg, D.: Labeling schemes for weighted dynamic trees. Information and Computation 205(12), 1721–1740 (2007)

[18] Korman, A., Peleg, D., Rodeh, Y.: Labeling schemes for dynamic tree networks. Theory of Computing Systems 37(1), 49–75 (2004)

[19] Peleg, D.: Proximity-preserving labeling schemes. Journal of Graph Theory 33(3), 167–176 (2000)

[20] Thorup, M., Zwick, U.: Compact routing schemes. In: SPAA '01. pp. 1–10. SPAA '01 (2001)