Fast Routing Table Construction Using Small Messages

[Extended Abstract]*

Christoph Lenzen†
Dept. Computer Science & Applied Mathematics
Weizmann Institute of Science
Rehovot 76100, Israel

Boaz Patt-Shamir‡
School of Electrical Engineering
Tel Aviv University
Tel Aviv 69978, Israel

ABSTRACT

We describe a distributed randomized algorithm to construct routing tables. Given $0 < \varepsilon \leq 1/2$, the algorithm runs in time $\tilde{\mathcal{O}}(n^{1/2+\varepsilon} + \mathrm{HD})$, where n is the number of nodes and HD denotes the diameter of the network in hops (i.e., as if the network is unweighted). The weighted length of the produced routes is at most $\mathcal{O}(\varepsilon^{-1} \log \varepsilon^{-1})$ times the optimal weighted length. This is the first algorithm to break the $\Omega(n)$ complexity barrier for computing weighted shortest paths even for a single source. Moreover, the algorithm nearly meets the $\tilde{\Omega}(n^{1/2} + \mathrm{HD})$ lower bound for distributed computation of routing tables and approximate distances (with optimality, up to polylog factors, for $\varepsilon = 1/\log n$). The presented techniques have many applications, including improved distributed approximation algorithms for Generalized Steiner Forest, all-pairs distance estimation, and estimation of the weighted diameter.

Categories and Subject Descriptors

F.1.2 [**Modes of Computation**]: Parallelism and concurrency; G.2.2 [**Graph Theory**]: Graph algorithms

General Terms

Algorithms, Theory

Keywords

routing, small messages, approximate shortest paths

*A full version of this paper is available at [18].

†Supported by the Swiss Society of Friends of the Weizmann Institute of Science and by the Swiss National Science Foundation (SNSF).

‡Supported in part by the Israel Science Foundation (grant 1372/09) and by Israel Ministry of Science and Technology.

STOC'13, June 1-4, 2013, Palo Alto, California, USA.

1. INTRODUCTION

Constructing routing tables is a central task in network operation, the Internet being a prime example. Besides being an end goal on its own (facilitating the transmission of information from a sender to a receiver), efficient routing and distance approximation are critical ingredients in a myriad of other distributed applications.

At the heart of any routing protocol lies the computation of short paths in weighted graphs, where edge weights may reflect properties such as link cost, delay, bandwidth, reliability etc. In the distributed setting, an additional challenge is that the graph whose shortest paths are to be computed serves also as the platform carrying communication between the computing nodes. The result of this double role is a superimposition of two metrics: the given shortest paths metric and the "natural" communication metric of the distributed system. The first metric is used for the definition of shortest paths, where an edge weight represents its contribution to path lengths; the other metric is implicit, controlling the time complexity of the distributed computation: each edge is tagged by the time it takes a message to cross it. If these two metrics happen to be identical, then computing weighted shortest paths is trivial (to a single destination; for the all-pairs problem, see below). In the general case, the standard normalization is that messages cross each link in unit time, regardless of the link weight; this assumption is motivated by network synchronization. On the other hand, the message size must be taken into account as well. In the commonly-accepted CONGEST model of network algorithms [22], it is assumed that all link latencies are one unit and messages have fixed size, typically $\mathcal{O}(\log n)$ bits, where n denotes the number of nodes.

The classic algorithm for computing shortest paths distributedly is the Bellman-Ford algorithm (abbreviated BF), used in many networks, ranging from local to wide area networks. The BF algorithm enjoys many properties that make it an excellent distributed algorithm (locality, simplicity, self-stabilization). However, in weighted graphs, its time complexity, i.e., the number of parallel iterations, may be as high as $\Omega(n)$ for a single destination. This is in sharp contrast with the $\mathcal{O}(\mathrm{HD})$ time needed to compute *unweighted* shortest paths to a single destination, where HD denotes the unweighted "hop-diameter" of the network. The difference between n and HD can be huge; suffices to say that the hop-diameter of the Internet is estimated to be smaller than 50. Intuitively, the problem originates in the fact that the Bellman-Ford algorithm explores paths in a hop-by-hop fashion, and the aforementioned superposition of metrics

may result in a path which is weight-wise short, but consists of $\Omega(n)$ edges. If shortest paths have at most $\mathrm{SD} \in \mathbb{N}$ edges, then it suffices to run the Bellman-Ford algorithm for SD communication rounds. Indeed, the running time of a few distributed algorithms is stated as a function of SD or a similar parameter for exactly this reason (e.g., [7, 15, 16]).

To the best of our knowledge, no distributed algorithm for computing approximate weighted shortest paths in $o(\mathrm{SD})$ time in the CONGEST model was known to date. In this paper we present a distributed algorithm that computes approximate all-pairs shortest paths and distances using small messages, in time that nearly matches the lower bound of $\tilde{\Omega}(\sqrt{n} + \mathrm{HD})$, regardless of the value of SD.

1.1 Contributions and Main Techniques

Our main contribution, presented in Section 4, is an algorithm that, for any $0 < \varepsilon \leq 1/2$, using messages of size $\mathcal{O}(\log n)$, constructs in $\tilde{\mathcal{O}}(n^{1/2+\varepsilon} + \mathrm{HD})$ time routing tables and node labels (addresses) facilitating routing and distance estimation with stretch $\mathcal{O}(\varepsilon^{-1} \log \varepsilon^{-1})$. The labels have size $\mathcal{O}(\log \varepsilon^{-1} \log n)$ bits; we show that relabeling the nodes is necessary by proving that any (randomized) algorithm achieving polylogarithmic (expected) stretch without relabeling must run for $\tilde{\Omega}(n)$ rounds. The running time of our algorithm is close to optimal: it is not difficult to show that computing bounded-stretch routing with small messages must take $\tilde{\Omega}(\sqrt{n} + \mathrm{HD})$ time.

Our algorithm comprises two sub-algorithms that may be of independent interest. One is used for short-range routing (roughly, to the closest $\sqrt{n}$ nodes), and the other for longer distances. The short-range algorithm constructs a hierarchy in the spirit of Thorup-Zwick distance oracles [30]. Our main challenge is to implement the algorithm using small messages; to this end, we introduce a bootstrapping technique that allows us to build a hierarchy that grows in double-exponential speed. Using a restricted variant of the Bellmann-Ford algorithm we call BSP (for "bounded shortest paths") as a building block, we construct low-stretch routing tables for nearby nodes in $\tilde{\mathcal{O}}(\sqrt{n})$ time.

This approach exceeds our target complexity beyond the closest $\mathcal{O}(\sqrt{n})$ nodes, so at that point we switch to the "long-distance" scheme. The basic idea here is to start with about $\sqrt{n}$ random nodes we call the *skeleton* nodes, and compute all distances between them. To this end we introduce a virtual *skeleton graph* over the skeleton nodes, which approximates the distances of G. First, we show that it suffices to consider only paths of at most $h \in \mathcal{O}(\sqrt{n} \log n)$ hops in G. This is not enough, though: if we introduce a skeleton edge for each h-hop shortest path connecting skeleton nodes, we may have $\mathcal{O}(n)$ skeleton edges and excessive congestion during the computation of the skeleton graph. Our solution is to reduce the number of skeleton edges by simulating the spanner algorithm by Baswana and Sen [3] *while constructing the skeleton graph*. Given a parameter k and node set S, the algorithm of [3] constructs, with high probability, a spanner with $\tilde{\mathcal{O}}(|S|^{1+1/k})$ edges and stretch $2k-1$. Choosing $k \in \Theta(\varepsilon^{-1})$ yields the claimed result.

Using variants of our techniques, in Section 5 we derive improved solutions to the following related problems (all statements hold with high probability).

- For the Generalized Steiner Forest (GSF) problem we obtain, for any $0 < \varepsilon \leq 1/2$, an $\mathcal{O}(\varepsilon^{-1})$-approximation within $\tilde{\mathcal{O}}((\sqrt{n} + t)^{1+\varepsilon} + \mathrm{HD})$ rounds, where t is the number of terminals. This should be contrasted with the best known distributed approximation algorithm for GSF [15], which provides $\mathcal{O}(\log n)$-approximation in time $\tilde{\mathcal{O}}(\mathrm{SD} \cdot k)$, where SD is the "shortest paths diameter," namely the maximal number of hops in any shortest path, and k is the number of terminal components in the GSF instance.
- For any $k \in \mathbb{N}$, we obtain an $\tilde{\mathcal{O}}((\sqrt{n})^{1+1/k} + \mathrm{HD})$-time algorithm that constructs labels of size $\mathcal{O}(k \log n)$ and local tables of size $\tilde{\mathcal{O}}(n^{1/(2k)})$, and produces distance estimations with stretch $\mathcal{O}(k^2)$. Compare with the recent distributed algorithm [7] that attains the same local space consumption at running time $\tilde{\mathcal{O}}(\mathrm{SD} \cdot n^{1/(2k)})$ and stretch $4k-1$.
- Given any $0 < \varepsilon \leq 1/2$, we can compute an $\mathcal{O}(\varepsilon^{-1})$-approximation of the weighted diameter in $\tilde{\mathcal{O}}(n^{1/2+\varepsilon} + \mathrm{HD})$ rounds. We show that the $\tilde{\Omega}(\sqrt{n} + \mathrm{HD})$ time lower bound applies to computing approximate weighted diameter.
- Employing a different routing mechanism for the short-range scheme, we can assign the fixed labels of $1, \ldots, n$. This comes at the expense of a larger stretch of $\mathcal{O}(\varepsilon^{-3})$ within $\tilde{\mathcal{O}}(n^{1/2+\varepsilon} + \mathrm{HD})$ rounds, for any $0 < \varepsilon \leq 1/2$.

1.2 Related Work

There are many centralized algorithms for constructing routing tables, usually aimed at minimizing space without incurring a stretch that is considered too large (see below). We comment that a naïve implementation of a centralized algorithm in the CONGEST model requires $\Omega(|E|)$ time in the worst case, since the whole network topology has to be collected at a single node just for computation.

Distributed routing table construction algorithms are usually categorized as either "distance vector" or "link state" algorithms (see, e.g., [27]). Distance-vector algorithms are variants of the Bellman-Ford algorithm [4, 11], whose worst-case time complexity in the CONGEST model is $\Theta(n^2)$. In link-state algorithms [20, 21], each routing node collects the complete graph topology and then solves the single-source shortest path problem locally. This approach has $\Theta(|E|)$ time complexity. None of these algorithms uses relabeling, but it should be noted that the Internet architecture in fact employs relabeling (IP addresses, which are used instead of physical addresses, encode some routing information).

From the theoretical perspective, little progress was made in computing weighted shortest paths beyond the "shortest path diameter" (we denote by SD) even for the single-source case (cf. [7] and references). Unpublished results provide $(1+\varepsilon)$-approximative and exact shortest paths in $\tilde{\mathcal{O}}(n)$ and $\tilde{\mathcal{O}}(n^{3/2})$ rounds, respectively [6]. For the unweighted case, an elegant $\mathcal{O}(n)$-time algorithm for exact all-pairs shortest-paths was recently discovered (independently) in [14] and [23]. These algorithms do not relabel the nodes. In addition, a randomized (3/2)-approximation of HD is given in [23], and a deterministic $(1+\varepsilon)$-approximation is provided by [14]. Combining results, [14] and [23] report a randomized (3/2)-approximation of the hop-diameter in time $\tilde{\mathcal{O}}(n^{3/4})$.

In [8], a lower bound of $\tilde{\Omega}(\sqrt{n})$ on the time to construct a shortest-paths tree of weight within a factor of $n^{\mathcal{O}(1)}$ from the optimum is shown; this implies the same bound on routing (more precisely, on *stateless* routing, where routing decisions depend only on the destination and not on the traversed path). We do not know of other explicit lower bounds on the running time of approximate shortest paths or dis-

tance estimation algorithms, but a lower bound of $\tilde{\Omega}(\sqrt{n})$ can be easily derived using the technique used in [8] or [24]. In [12] it is shown that in the CONGEST model, approximating the diameter of unweighted graphs to within a factor of $3/2-\varepsilon$ requires $\tilde{\Omega}(\sqrt{n})$ rounds, for any constant $\varepsilon > 0$.

In the Generalized Steiner Forest problem (GSF), the input consists of a weighted graph, and a set of *terminal nodes* which is partitioned into subsets called *terminal components*. The task is to find a set of edges of minimum weight so that the terminal components are connected. Historically, the important special case of a minimum spanning tree (all nodes are terminals, single terminal component) has been the target of extensive research in distributed computation. It is known that in the CONGEST model, the time complexity of computing an MST is $\tilde{\Omega}(\sqrt{n}+\mathrm{HD})$ [8, 9, 24]. This bound is essentially matched by an exact deterministic solution [13, 17]. An $\mathcal{O}(\log n)$-approximate MST is presented in [16], whose running time is $\mathcal{O}(\mathrm{SD})$, where SD is the "shortest path diameter" mentioned previously. For the special case of Steiner Trees (arbitrary terminals, single component), [5] presents a 2-approximation algorithm whose time complexity is $\tilde{\mathcal{O}}(n)$ (which can easily be refined to $\tilde{\mathcal{O}}(\mathrm{SD})$). For the general case, [15] presents an $\mathcal{O}(\log n)$-approximation algorithm whose time complexity is $\tilde{\mathcal{O}}(\kappa \cdot \mathrm{SD})$, where κ is the number of terminal components.[1]

We now turn to a (very brief) overview of centralized algorithms. Thorup and Zwick [29] presented an algorithm that achieves, for any $k \in \mathbb{N}$, routes of stretch $2k-1$ using $\tilde{\mathcal{O}}(n^{1/k})$ memory. In terms of memory consumption, it has been established that this is optimal up to a constant factor in (worst-case) stretch w.r.t. routing [26]. This result has been extended to the average stretch, and tightened to be exact up to polylogarithmic factors in memory for the worst-case stretch [1]. For distance approximation, the Thorup-Zwick scheme is known to be optimal for $k = 1, 2, 3, 5$ and conjectured to be optimal for all k (see [31] and references). The algorithm requires relabeling with labels of size $\mathcal{O}(k \log n)$. It is unclear whether stronger lower bounds apply to name-independent routing schemes (which keep the original node identifiers); however, for $k = 1$ trivially $\mathcal{O}(n \log n)$ bits suffice (for $\mathcal{O}(\log n)$-bit identifiers), and Abraham et al. [2] prove a matching upper bound of $\tilde{\mathcal{O}}(\sqrt{n})$ bits for $k = 2$.

A closely related concept is that of *sparse spanners*, introduced by Peleg and Schäffer [25]. A k-spanner of a graph is obtained by deleting edges, without increasing the distances by more than factor k. Similarly to compact routing tables, it is known that a $(2k-1)$-spanner must have $\tilde{\Omega}(n^{1+1/k})$ edges for some values of k, conjectured to hold for all $k \in \mathbb{N}$, and a matching upper bound is obtained by the Thorup-Zwick construction [30]. If an additive term in the distance approximation is permitted, the multiplicative factor can be brought arbitrarily close to 1 [10]. In contrast to routing and distance approximation, there are extremely fast distributed algorithms constructing sparse spanners. Our long-range construction rests on an algorithm by Baswana and Sen [3] that achieves stretch $2k-1$ vs. $\mathcal{O}(n^{1+1/k})$ expected edges within $\mathcal{O}(k)$ rounds in the CONGEST model.

Paper Organization. In Section 2 we formalize the model and define some basic concepts, and in Section 3 we define the problem and state several hardness results. Section 4 describes our main algorithm, and in Section 5 we describe applications of our results. (Missing proofs can be found in [18].) We conclude in Section 6 with a few open problems.

2. THE MODEL & BASIC CONCEPTS

The Computational Model. We follow the CONGEST(B) model as described in [22]. The distributed system is represented by a connected weighted graph $G = (V, E, W)$, where V is the set of nodes, E is the set of edges, and $W : E \to \mathbb{N}$ is the edge weight function. As a convention, we use n to denote the number of nodes. We assume that all edge weights are bounded by some polynomial in n, and that each node $v \in V$ has a unique identifier of $\mathcal{O}(\log n)$ bits (we use v to denote both the node and its identifier).

Execution proceeds in global synchronous rounds, where in each round, each node takes the following three steps: (1) Receive the messages sent by neighbors at the previous round, (2) perform local computation, and (3) send messages to neighbors. Initially, nodes are aware only of their neighbors; input values (if any) are assumed to be fed by the environment at time 0. Output values are placed in special output-registers. In each round, each edge can carry a message of B bits for some given parameter B of the model; we assume that $B \in \Theta(\log n)$ throughout this paper. In this model we may assume, at the price of $\mathcal{O}(\mathrm{HD})$ additional steps, that we have a broadcast facility available, as formalized in the following lemma.

LEMMA 2.1. *Suppose each $v \in V$ holds $k_v \geq 0$ messages of $\mathcal{O}(\log n)$ bits each, for a total of $k \stackrel{\text{def}}{=} \sum_{v \in V} k_v$ messages. Then all nodes in the graph can receive these k messages within $\mathcal{O}(k + \mathrm{HD})$ rounds.*

General Concepts. We extensively use "soft" asymptotic notation that ignores polylogarithmic factors. Formally, we say that $g(n) \in \tilde{\mathcal{O}}(f(n))$ iff there exists a constant $c \in \mathbb{R}_0^+$ such that $f(n) \leq g(n) \log^c(f(n))$ for all but finitely many values of $n \in \mathbb{N}$. $\tilde{\Omega}$, $\tilde{\Theta}$, $\tilde{o}$, and $\tilde{\omega}$ are defined accordingly.

To model probabilistic computation, we assume that each node has access to an infinite string of independent unbiased random bits. When we say that a certain event occurs "with high probability" (abbreviated "w.h.p."), we mean that the probability of the event not occurring can be set to be less than $1/n^c$ for any desired constant c, where the probability is taken over the strings of random bits.

Some Graph-Theoretic Concepts. A *path* p connecting $v, u \in V$ is a sequence of nodes $\langle v = v_0, \ldots, v_k = u \rangle$ such that for all $0 \leq i < k$, $\{v_i, v_{i+1}\}$ is an edge in G. Let $\mathrm{paths}(v, u)$ denote the set of all paths connecting nodes v and u. We use the following unweighted concepts.

- The *hop-length* of a path p, denoted $\ell(p)$, is the number of edges in it.
- The *hop distance* $\mathrm{hd} : V \times V \to \mathbb{N}_0$ is defined as $\mathrm{hd}(v, u) \stackrel{\text{def}}{=} \min\{\ell(p) \mid p \in \mathrm{paths}(v, u)\}$.
- The *hop diameter* of a graph $G = (V, E, W)$ is $\mathrm{HD} \stackrel{\text{def}}{=} \max_{v,u \in V}\{\mathrm{hd}(v, u)\}$.

[1] We note that in [15], time-optimality is claimed, up to factor $\tilde{\mathcal{O}}(\kappa)$. This comes as a consequence of [16], which in turn builds on [9]. However, we comment that the latter construction does not scale beyond the familiar lower bound of $\tilde{\Omega}(\sqrt{n})$, and a more precise statement would thus be that a minimum spanning tree (and thus also a GSF) requires $\tilde{\Omega}(\min\{\mathrm{SD}, \sqrt{n}\})$ rounds to be approximated.

We use the following weighted concepts.
• The *weight* of a path p, denoted $W(p)$, is its total edge weight, i.e., $W(p) \stackrel{\text{def}}{=} \sum_{i=1}^{\ell(p)} W(v_{i-1}, v_i)$.
• The *weighted distance* $\text{wd} : V \times V \to \mathbb{N}$ is defined by $\text{wd}(v,u) \stackrel{\text{def}}{=} \min\{W(p) \mid p \in \text{paths}(v,u)\}$.
• The *weighted diameter* of G is $\text{WD} \stackrel{\text{def}}{=} \max\{\text{wd}(v,u) \mid v,u \in V\}$.

The following concepts mix weighted and unweighted ones.
• Given $h \in \mathbb{N}$ and two nodes $v,u \in V$ with hop distance $\text{hd}(v,u) \leq h$, we define the *h-weighted distance* $\text{wd}_h(u,v)$ to be the weight of the lightest path connecting v and u with at most h hops, i.e.,
$\text{wd}_h(v,w) \stackrel{\text{def}}{=} \min\{W(p) \mid p \in \text{paths}(v,w) \text{ and } \ell(p) \leq h\}$.
If $\text{hd}(v,u) > h$, we define $\text{wd}_h(v,u) \stackrel{\text{def}}{=} \infty$. (Note that wd_h does not satisfy the triangle inequality.)
• The *shortest paths diameter* of a graph, denoted SD, is the maximal number of hops in shortest paths:
$\text{SD} \stackrel{\text{def}}{=} \max_{u,v \in V}\{\min\{\ell(p) \mid p \in \text{paths}(v,u), W(p) = \text{wd}(u,v)\}\}$.
Finally, given $v \in V$ and an integer $i \geq 0$, define $ball_v(i)$ to be the set of the i nodes that are weight-wise closest to v:

$$ball_v(i) \stackrel{\text{def}}{=} \{u : |\{w : \text{wd}(v,u) \leq \text{wd}(v,w)\}| \leq i\} \ .$$

Note that our concept of ball differs from the usual one: we define a ball by its center and *volume*, namely the number of nodes it contains (and not by its center and radius). We have the following immediate property.

LEMMA 2.2. *Let $v,u \in V$. If $u \in ball_v(i)$ for some $i \in \mathbb{N}$ then $\text{wd}(v,u) = \text{wd}_j(v,u)$ for all $j \geq i-1$.*

3. THE PROBLEMS & LOWER BOUNDS

The Routing Table Construction Problem (RTC). In the *routing table construction* problem, the local input at a node is the weight of incident edges, and the output at each node v consists of (i) a unique *label* $\lambda(v)$ and (ii) a function "next_v" that takes a destination label λ and produces a neighbor of v, such that given the label $\lambda(u)$ of any node u, we can reach u from v by following the next pointers in order. Formally, the requirement is as follows. Given a start node v and a destination label $\lambda(u)$, let $v_0 = v$ and define $v_{i+1} = \text{next}_{v_i}(\lambda(u))$ for $i \geq 0$. Then for some i we must have $v_i = u$.

The performance of a solution is measured in terms of its *stretch*: A route is said to have stretch $\rho \geq 1$ if its total weight is no more than ρ times the weighted distance between its endpoints, and a solution to RTC is said to have stretch ρ if all the routes it induces have stretch at most ρ.

Variants. Routing appears in many incarnations. We list a few important variants below.

Name-independent routing. Our definition of RTC allows for node relabeling. This is the case, as mentioned above, in the Internet. The case where no such relabeling is allowed (which can be formalized by requiring λ to be the identity function), is called *name-independent* routing.

Stateful routing. The routing problem as defined above is *stateless* in the sense that routing a packet is done regardless of the path it traversed so far. One may also consider *stateful* routing, where while being routed, a packet may gather information that helps it navigate later (one embodiment of this idea in the Internet routing today is MPLS, where packets are temporarily piggybacked with extra headers). Note that the set of routes to a single destination in stateless routing must constitute a tree, whereas in stateful routing even a single route may contain a cycle. Formally, in stateful routing the label of the destination may change from one node to another: The next_v function outputs both the next hop (a neighbor node), and a new label λ_v used in the next hop.

The Distance Approximation Problem. The *distance approximation* problem is akin to the routing problem. Also here, each node v outputs a label $\lambda(v)$, but now, v needs to construct a function $\text{dist}_v : \lambda(V) \to \mathbb{R}_0^+$ (the table) such that for all $w \in V$ it holds that $\text{dist}_v(w) \geq \text{wd}(v,w)$. The stretch of the approximation for a given node w is $\text{dist}_v(w)/\text{wd}(v,w)$, and the solution has stretch $\rho \geq 1$ if $\text{dist}_v(w) \leq \rho \text{wd}(v,w)$ for all $v,w \in V$.

Similarly to routing, we call a scheme name-independent if λ is the identity function.

3.1 A Few Hardness Results

Name-independent routing. We show that assigning new labels to the nodes is unavoidable by proving that any (randomized) algorithm achieving polylogarithmic (expected) stretch without relabeling must run for $\tilde{\Omega}(n)$ rounds. Formally, we prove the following result.

THEOREM 3.1. *Any RTC algorithm for name-independent stateful routing with (expected) average stretch ρ requires $\Omega(n/(\rho^2 \log n))$ time.*

PROOF SKETCH. Consider a tree of depth 2: the root has n_1 children, and each of them has n_2 children, where $n_1 \in \Theta(\rho)$ and $n_2 = n/n_1$. Assign leaf identifiers as a random permutation. We analyze the expected behavior of a deterministic algorithm. If the RTC algorithm runs for r rounds, the root receives at most $rn_1 \log n$ bits, and hence there are at most $2^{rn_1 \log n}$ different routing tables at the root. We show that if a packet with a random leaf destination is routed correctly at the root with probability more than e^2/n_1 then $r \in \Omega(n_2/(n_1 \log n))$. To deal with statefulness, consider a packet that was routed to a child v. If the destination is not in v's subtree, the packet eventually returns to the root; at this point, we may assume w.l.o.g. that all v's subtree is known to the packet, but not more, since all information regarding other subtrees must have passed through the root. Thus we may apply induction to the tree after eliminating v's subtree, and conclude that with constant probability, at least $n_1/2$ attempts are required. □

We remark that this lower bound was shown independently in [6]. An argument similar in spirit shows that a comparable lower bound applies to distance approximation.

THEOREM 3.2. *Any name-independent distance approximation scheme of (expected) average stretch ρ requires time $\Omega(n/\log n)$ for table construction, even if each edge has either weight 1 or weight $\omega_{\max} \in \mathcal{O}(\rho)$.*

Consequently, in the remainder of the paper we shall consider name-dependent routing schemes only.

Weighted Diameter Estimation. In [12], it is shown that approximating the hop-diameter of a network within a factor smaller than 1.5 cannot be done in the CONGEST model in $\tilde{o}(\sqrt{n})$ time. Here, we prove a hardness result for the weighted diameter, formally stated as follows.

THEOREM 3.3. *There exists a graph family with hop diameter* HD $\in \mathcal{O}(\log n)$ *and edge weights of* 1 *and* $\omega_{\max}$ *only for which any (randomized) algorithm that computes the weighted diameter to within an (expected) approximation factor of* $o(\omega_{\max}/\sqrt{n})$ *requires* $\tilde{\Omega}(\sqrt{n})$ *time.*

Name-dependent routing. A lower bound on name-dependent distance approximation follows from Theorem 3.3.

COROLLARY 3.4. *There exists a graph family with hop diameter* HD $\in \mathcal{O}(\log n)$ *and edge weights in* $\{1, \omega_{\max}\}$ *for which any (randomized) algorithm that constructs labels of size* $\tilde{o}(\sqrt{n})$ *for distance approximations with an (expected) stretch of* $o(\omega_{\max}/\sqrt{n})$ *requires* $\tilde{\Omega}(\sqrt{n})$ *time.*

A variation of the theme shows that stateless routing requires $\tilde{\Omega}(\sqrt{n})$ time.

COROLLARY 3.5. *For any* $\omega_{\max} \in \omega(\sqrt{n})$*, a graph family with hop diameter* HD $\in \mathcal{O}(\log n)$ *and edge weights in* $\{1, \omega_{\max}\}$ *exists for which any (randomized) algorithm that constructs labels of size* $\tilde{o}(\sqrt{n})$ *and stateless routing tables with (expected) stretch* $o(\omega_{\max}/\sqrt{n})$ *requires* $\tilde{\Omega}(\sqrt{n})$ *time.*

In summary, these results imply that neither RTC, nor distance and diameter approximation permit $\tilde{o}(\sqrt{n})$-time algorithms, with the possible exception of stateful name-dependent RTC (whose time complexity remains open).

4. ROUTING ALGORITHM

Overview. The high-level intuition behind our algorithm is as follows. Naïve distributed algorithms explore paths sequentially, adding one edge at a time, leading to potentially linear complexity, since paths which are weight-wise short may contain many hops. Our basic idea is to break hop-wise long paths into small pieces by means of random sampling. Specifically, motivated by the $\tilde{\Omega}(\sqrt{n})$ lower bound from Corollaries 3.4 and 3.5, we select a random subset of $\Theta(\sqrt{n}\log n)$ nodes we call the routing *skeleton*. With high probability, any set of more than $\sqrt{n}$ nodes will contain a skeleton node. In particular, w.h.p., (1) any simple path of hop-length $\sqrt{n}$ contains a skeleton node, and (2) any node has a skeleton node among its closest $\sqrt{n}$ nodes. The route that our scheme will select from a given source to a given destination depends on their mutual distance: If the destination is one of the $\sqrt{n}$ nodes closest to the source, routing will be done using a "short range scheme;" otherwise, the short range scheme is used to route from the source to the nearest skeleton node, from which, using another scheme we call "long distance routing," we route to the skeleton node closest to the destination node, and finally, another application of the short range scheme brings us to the destination. This leaves two non-trivial problems, namely the short-range and the long-distance schemes, whose description occupies most of the remainder of this section. The short range scheme is described in Section 4.2, and the long-distance in Section 4.3. We start by describing the variant of the Bellman-Ford algorithm we use as a basic building block in Section 4.1.

4.1 Algorithm BSP

A basic building block we use throughout the construction is a bounded version of the Bellman-Ford algorithm, which we call Algorithm BSP. The input to Algorithm BSP is a *range* parameter h, an *overlap* parameter Δ, and a set of sources S. The algorithm computes routes from each node to the closest Δ sources in S using only routes of up to h hops. The implementation is by h parallel iterations of slightly modified Bellman-Ford relaxations, where at each iteration, each node v:
(1) receives from its neighbors their current distance estimates to their Δ closest sources, (2) updates its distance estimates and next_v to the sources based on the information it received and its incident edge weights, and (3) sends to all its neighbors its current estimates for the Δ sources with the smallest distance (ties broken by ID).

Let $L_v(t)$ denote the list of the Δ closest sources (with their distances and next_v pointers) sent by v at the end of iteration t above. It is easy to see, by mapping the executions of Algorithm BSP to executions of the standard Bellman-Ford algorithm, that the top Δ entries in the list maintained at node v by the unmodified Bellman-Ford algorithm after iteration t is exactly $L_v(t)$.

Routing. Interestingly, the information provided by $L_v(h)$ (the final list at v) may be insufficient for routing: since the Δ closest source node sets may differ between neighbors, it may be the case that for some source s and two neighbors v and u we have that u is the next node from v to s in $L_v(h)$, but there is no entry for source s in $L_u(h)$! This occurs, for example, if in iteration h, u learns about a source that is closer than s, pushing s out of $L_u(h)$. Using $(L_v(1), \ldots, L_v(h))$ instead of $L_v(h)$ only, we can however still reconstruct the routes. In the example above, $L_u(h-1)$ must contain s. In general, when routing from node v_0 to a node s, arriving at a node v_i after $i \geq 0$ hops, we use $L_{v_i}(h-i)$. (We note that this scheme is not stateless—routing decisions depend on the number of previous routing hops—but it is easy to make it stateless. Details can be found in the full paper.)

We summarize the properties of Algorithm BSP with the following theorem.

THEOREM 4.1. *Algorithm BSP computes the h-weighted distance and next hop of a shortest path of at most h edges from each node to its closest Δ sources. Each node on the corresponding shortest path can determine the next hop on the path out of the number of preceding hops and the output of the algorithm. The time complexity of Algorithm BSP in the* CONGEST *model is $\mathcal{O}(\Delta h)$ rounds.*

4.2 The Short-Range Scheme

Recall that our goal in the short-range scheme is to allow each node to find a route to each of its closest $\sqrt{n}$ neighbors (roughly). A naïve application of Algorithm BSP, where all nodes are sources, would set the overlap parameter to $\Delta := \sqrt{n}$ (this is the number of nodes we want to know about), and the range parameter to $h := \sqrt{n}$ too (by Lemma 2.2). However, Theorem 4.1 tells us that in this case, the time complexity would be $\mathcal{O}(\Delta h) = \mathcal{O}(n)$, a far cry from the $\tilde{\Omega}(\sqrt{n})$ lower bound we are shooting for. Our solution is a hierarchical bootstrapping process, described next.

The Hierarchy. The construction is parametrized by the number of levels, denoted L (to be determined later). Sets of nodes, denoted $S_1, \ldots, S_L$, are sampled uniformly and independently at random with $S_0 \stackrel{\text{def}}{=} V$, and $S_i \subseteq S_{i-1}$ for $1 \leq i \leq L$. In the i^{th} level, each node v finds a route to

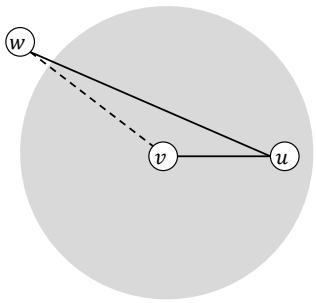

Figure 1: The distance from v to w is at least one third of the length of the route from v to w via u.

$Y_i(v)$, defined to be the node in S_i closest to v, and to all nodes in S_{i-1} that are closer to v than $Y_i(v)$. This property allows us to bound the routing stretch (following [30]). The argument for the first level is a simple application of the triangle inequality (see Fig. 1): Consider a route from node v to node w. If there is a node $u \in S_1$ that is closer to v than w, then the route that consists of shortest paths from v to u to w has stretch at most 3. Therefore, for routing, it suffices for v to determine least-weight routes to nodes in S_{i-1} that are closer to it than $Y_i(v)$ (the interior of the circle in Fig. 1).

To this end, in each level we invoke Algorithm BSP with source set S_{i-1}. Let us first see intuitively what are the appropriate choices of h_i and Δ_i. Let p_i denote the probability of a node to be selected into S_i. W.h.p., each node v has a member of S_i among the $\mathcal{O}(\log n/p_i)$ nodes closest to v. Hence we set $h_i := \Theta(\log n/p_i)$. Next, consider the nodes of S_{i-1} among the h_i nodes closest to a given node. Their expected number is $p_{i-1}h_i$, which means, by Chernoff's bound, that w.h.p., there are at most $\mathcal{O}(p_{i-1}h_i) = \mathcal{O}(\log n \cdot p_{i-1}/p_i)$ sources that need to be detected by each node at level i. We therefore set $\Delta_i := \Theta(\log n \cdot p_{i-1}/p_i)$.

Thus, the running time of Algorithm BSP in level i is $T_i \in \mathcal{O}(h_i\Delta_i) \subseteq \tilde{\mathcal{O}}(p_{i-1}/p_i^2)$. Setting $T_i = \sqrt{n}$, $p_0 = 1$, and ignoring polylog factors, the solution to this recursive equation is $p_i = n^{-(2^i-1)/2^{i+1}}$. Plugging in this value we obtain

$$h_i = \Theta(n^{1/2-1/2^{i+1}} \log n), \text{ and } \Delta_i = \Theta(n^{1/2^{i+1}} \log n)\ . \quad (1)$$

(Note that $L = \log\log n$ levels suffice to ensure that $S_L \in \Theta(\sqrt{n}\log n)$ w.h.p., so we assume that $L \in \mathcal{O}(\log\log n)$ henceforth.) Running Algorithm BSP with parameters as above, we get that w.h.p., after $\tilde{\mathcal{O}}(L\sqrt{n}) = \tilde{\mathcal{O}}(\sqrt{n})$ time, each node knows of the closest Δ_i nodes from S_{i-1}, and how to route to them, for all $1 \le i \le L$.

This facilitates routing in one direction. We now describe how to route from nodes in S_i. Let $C_v(i) \stackrel{\text{def}}{=} \{u \in V \mid Y_i(u) = v\}$, i.e., for each level i, the sets $C_v(i)$ are a Voronoi decomposition of V with centers $Y_i(V) = S_i$. We need to find routes from $Y_i(v)$ to $C_v(i)$. But the sheer size of $C_v(i)$ could be overwhelming: while the depth of the tree rooted at $Y_i(v)$ is bounded by h_i, its size $|C_v(i)|$ could be $\Omega(n)$. To solve this problem we rely on the shortest-paths tree rooted at $Y_i(v)$ that spans $C_v(i)$ (which is already constructed), and employ one of the known techniques for tree routing (e.g., [28]). To minimize space consumption, we use the technique of [29], which constructs in $\mathcal{O}(h_i)$ time routing tables of size $\log^{\mathcal{O}(1)} n$ and node labels of $\mathcal{O}(\log n)$ bits.

We now present a more formal description of the construction for short-range routing. Given $n \in \mathbb{N}$ and $L \le \log\log n$, we define the following for $1 \le i \le L$.

- $p_0 \stackrel{\text{def}}{=} 1$, and $p_i \stackrel{\text{def}}{=} (\sqrt{n})^{-(2^L/(2^L-1))(2^i-1)/2^i}$.
- For each node v, $Y_v(i)$ is the node from S_i closest to v (ties broken by ID).
- For each $u \in S_i$, define

$C_u(i) \stackrel{\text{def}}{=} \{v \mid Y_v(i) = u\}$, and $C_u(0) \stackrel{\text{def}}{=} \{u\}$.

- For each node $v \in V$, define

$H_v(i) \stackrel{\text{def}}{=} \{u \in S_{i-1} \mid \mathrm{wd}(v,u) \le \mathrm{wd}(v, Y_v(i))\}$.

Our construction maintains (w.h.p.) the following properties at stage $i \in \{1,\dots,L\}$.

(1) S_i is a uniformly random subset of S_{i-1}, where $\Pr[v \in S_i] = p_i$, and $\Pr[v \in S_i \mid v \in S_{i-1}] = p_i/p_{i-1}$.
(2) For any node v, it it is possible to route from v to $Y_v(i)$ on a lightest path.
(3) For any node v, it is possible to compute $Y_v(i)$ and $\mathrm{wd}(v, Y_v(i))$ from the label of v.
(4) For each node $u \in S_i$, given the label $\lambda(w)$ of any node $w \in C_u(i)$, we can route from u to w on a lightest path.
(5) For any node v, $H_v(i)$ is locally known at v, and it is possible to route from v to any node $u \in H_v(i)$ on a lightest path (whose weight is known at v).

Suppose that we have such a hierarchy of L stages. Then, given the label of any node $w \in \bigcup_{1\le i\le L, u\in H_v(i)} C_u(i-1)$, v can route a message to w as follows: First, find some $i \in \{1,\dots,L\}$ such that $w \in C_u(i-1)$ for some $u \in H_v(i)$ (cf. Properties (3,5) of the construction). The route from v to w is then defined by the concatenation of two lightest paths: the one from v to u, and the one from u to w (cf. Properties (4,5)). Moreover, the long-range scheme will make sure that we can always route to any destination via the closest skeleton nodes in S_L, which is feasible due to Property (2) and Property (4). By always choosing from the available routes such that the weight of the computed route is minimal (which can be done by Property (3) and Property (5) for the short-range construction, and will also be possible for the long-range scheme), routing becomes stateless.

Stretch Analysis. To bound the weight of the routes constructed by the stated scheme with respect to the weight of the shortest paths, we have the following key lemma.

LEMMA 4.2. *Suppose that for $v, w \in V$ and $1 \le j \le L$ we have that $w \notin \bigcup_{i=1}^{j} \bigcup_{u\in H_v(i)} C_u(i-1)$. Then (a) $\mathrm{wd}(v, Y_v(j)) \le (2j-1)\mathrm{wd}(v,w)$, and (b) $\mathrm{wd}(w, Y_w(j)) \le 2j\mathrm{wd}(v,w)$.*

PROOF. We use induction on i, for a fixed j. More specifically, we show for each $1 \le i \le j$ that (a) $\mathrm{wd}(v, v_i) \le (2i-1)\mathrm{wd}(v,w)$ and (b) $\mathrm{wd}(w, w_i) \le 2i\mathrm{wd}(v,w)$. For the basis of the induction, consider $i = 0$ in Statement (b). In this case, since $S_0 = V$, we have that, $Y_w(0) = w$ and Statement (b) holds because $\mathrm{wd}(v,w) \ge 0 = \mathrm{wd}(w,w)$.

For the inductive step, assume that Statement (b) holds for $0 \le i < j$ and consider $i+1$. Since trivially $w \in C_{Y_w(i)}(i-1)$, the premise of the lemma implies that $Y_w(i) \notin H_v(i+1)$. However, $Y_v(i+1) \in H_v(i+1)$, and hence we obtain

$$\begin{aligned}\mathrm{wd}(v, Y_v(i+1)) \le \mathrm{wd}(v, Y_w(i)) &\le \mathrm{wd}(v,w) + \mathrm{wd}(w, Y_w(i)) \\ &\le (2i+1)\mathrm{wd}(v,w),\end{aligned}$$

where the first inequality follows from the definitions, the second from the triangle inequality, and the last by induction hypothesis. This proves part (a) of the claim. Using the above inequality we also obtain

$$\begin{aligned}\mathrm{wd}(w, Y_w(i+1)) &\leq \mathrm{wd}(w, Y_v(i+1)) \\ &\leq \mathrm{wd}(w,v) + \mathrm{wd}(v, Y_v(i+1)) \\ &\leq (2i+2)\mathrm{wd}(v,w),\end{aligned}$$

where the first inequality holds since $\mathrm{wd}(w, Y_w(i+1)) \leq \mathrm{wd}(w,u)$ for $u \in S_{i+1}$, the second follows from the triangle inequality, and the last from part (a) of the claim. This proves part (b) of the claim, completing the induction. □

Lemma 4.2 gives rise to the following result we shall use.

COROLLARY 4.3. *Let $v, w \in V$, and let $1 \leq i_0 \leq L$ be minimal such that $Y_w(i_0 - 1) \in H_v(i_0)$. Then* $\mathrm{wd}(v, Y_w(i_0 - 1)) + \mathrm{wd}(Y_w(i_0 - 1), w) \leq (4i_0 - 3)\mathrm{wd}(v,w) \in \mathcal{O}(L \cdot \mathrm{wd}(v,w))$.

On the other hand, if there is no i_0 as in the corollary, then by Lemma 4.2, routing via the skeleton nodes closest to source and destination incurs only bounded stretch.

Implementation and Time Complexity. We now explain how to construct the hierarchy efficiently in more detail, and analyze the time complexity of the construction. (Pseudocode is given in the full paper.) The algorithm is parametrized by the total number of nodes n, and by the number of hierarchy levels L.

The algorithm works as follows. First all sets S_i are selected, for $0 \leq i \leq L$ (this is done by local coin tosses). Then L phases are executed. In phase i, BSP is invoked with sources S_{i-1}, and range and overlap parameters as in Eq. (1). Routing information is recorded only for the set $H_v(i)$ (the nodes in S_{i-1} closer to v than $Y_v(i)$, the node closest to v from S_i). To allow for the reverse routing, the last step in a phase i is to invoke the tree-routing algorithm for the trees rooted at the S_i nodes.

We now analyze the algorithm. Property (1) is trivially satisfied by the local coin tosses. In addition, the following properties are easily derived using the Chernoff bound.

LEMMA 4.4. *For appropriate choices of the constants in Eq. (1), for all $1 \leq i \leq L$ it holds w.h.p. that:*
- $|S_i| \in \Theta(p_i n)$ ($|S_0| = n$).
- *For all $v \in V$, $|S_i \cap ball_v(h_i)| \in \Theta(\log n)$.*
- *For all $v \in V$, $H_v(i) \subset ball_v(h_i)$.*
- *For all $v \in V$, $|H_v(i)| \in \Theta(h_i p_{i-1}) = \Theta(p_{i-1} \log n / p_i)$.*
- *For all $v \in V$, $\Delta_i \geq |H_v(i)|$.*

By these properties and Theorem 4.1, Property (5) is satisfied w.h.p., because we invoke Algorithm BSP with sources S_i, depth parameter h_i, and overlap parameter Δ_i: after this invocation, each node v can identify the set $H_v(i)$ and route to any $u \in H_v(i)$ on a path of known weight; since $H_v(i) \subset ball_v(h_i)$ w.h.p., these routing paths are shortest paths. Moreover, the invocation of $\mathrm{BSP}(h, \Delta, S_i)$ allows each node v also to learn what is $Y_v(i)$ and route to it on a shortest path of known weight, establishing Property (2). In order to satisfy Property (3), we simply add $Y_v(i)$ and $\mathrm{wd}(v, Y_v(i)) = d_v(Y_v(i))$ to the label of v for all i. To route to $C_{Y_v(i)}$, as mentioned above, we use the tree routing of [29] that constructs routing tables of size $\log^{\mathcal{O}(1)} n$ and labels of size $(1+o(1)) \log n$ in $\tilde{\mathcal{O}}(h_i)$ time, and we add the respective tree label to v's label to ensure Property (4).

We summarize the complexity of the short-range construction as follows.

LEMMA 4.5. *Given $1 \leq L \leq \log\log n$, constructing the L-levels short-range routing tables and labels can be done in $\mathcal{O}(L(\sqrt{n})^{2^L/(2^L-1)} \log^2 n) \subset \tilde{\mathcal{O}}((\sqrt{n})^{2^L/(2^L-1)})$ rounds, such that node labels have $\mathcal{O}(L \log n)$ bits.*

4.3 Long-Distance Routing

We now explain how to route between the nodes in the top level of the hierarchy created by the short-range scheme. Our central concept is the *skeleton graph*, defined as follows.

DEFINITION 4.1 (SKELETON GRAPH). *Given a weighted graph $G = (V, E, W)$, and* skeleton set $S \subseteq V$, *and $h \in \mathbb{N}$, the* h-hop skeleton-S graph *is a weighted graph $G_{S,h} = (S, E_{S,h}, W_{S,h})$ defined by*
- $E_{S,h} \stackrel{\text{def}}{=} \{\{v,w\} \mid v, w \in S \text{ and } \mathrm{hd}_G(v,w) \leq h\}$, *and*
- *for each $\{v,w\} \in E_{S,h}$, $W_{S,h}(v,w) \stackrel{\text{def}}{=} \mathrm{wd}_h(v,w)$, i.e., $W_{S,h}(v,w)$ is the h-weighted distance between v and w in G.*

The key observation on skeleton graphs is that if the skeleton S is a random set of nodes, and if $h \in \Omega(n \log n / |S|)$, then w.h.p., the distances in $G_{S,h}$ are equal to the corresponding distances in G. This means that it suffices to consider paths of $\mathcal{O}(n \log n / |S|)$ hops in G in order to find the exact distances in G. The following lemma formalizes this idea. (We state it for a skeleton *containing* a random subset; this generality will become useful in Section 5.)

LEMMA 4.6. *Let S_R be a set of uniform random nodes with $\Pr[v \in S_R] = \pi$ for some given π, and Let $S \supseteq S_R$. If $\pi \geq c \log n / h$ for a sufficiently large constant $c > 0$, then w.h.p., $\mathrm{wd}_{S,h}(v,w) = \mathrm{wd}(v,w)$ for all $v, w \in S$.*

Our strategy to solve long-distance routing is to construct $G_{S,h}$ and compute its all-pairs shortest paths. But implementing this approach is not straightforward. First, the edges of the skeleton graph are virtual: each edge represents the shortest path of up to h hops in G; and second, the number of skeleton graph edges may be as large as $\Omega(|S|^2)$. We solve both problems together: While computing the edges of the skeleton graph, we sparsify the graph, bringing the number of edges down to near-linear in the skeleton size. Once we are done, we can afford to let each skeleton node learn the full topology of the sparsified skeleton graph, from which all-pairs routes and distances can be computed locally. Technically, we use the concept of sparse spanners.

DEFINITION 4.2 (SPANNER). *Let $H = (V, E, W)$ be a weighted graph and let $k \geq 1$. A* weighted k-spanner *of H is a weighted graph $H' = (V, E', W')$ where $E' \subseteq E$, $W'(e) = W(e)$ for all $e \in E'$, and $\mathrm{wd}_{H'}(u,v) \leq k \cdot \mathrm{wd}_H(u,v)$ for all $u, v \in V$ (where wd_H and $\mathrm{wd}_{H'}$ denote weighted distances in H and H', respectively).*

We shall compute a spanner of the skeleton graph by simulating the spanner construction algorithm of Baswana and Sen [3] on the implicit skeleton graph. Let us recall the algorithm of [3]. We use a slightly simpler variant that may select some additional edges, albeit without affecting the probabilistic upper bound on the number of spanner edges (cf. Lemma 4.8). The input is a graph $H = (V_H, E_H, W_H)$ and a parameter $k \in \mathbb{N}$.

1. $R_1 := \{\{v\} \mid v \in V_H\}$ (each node is a singleton *cluster*)
2. Repeat $k-1$ times:
   (a) Mark each cluster from R_i independently with probability $|V_H|^{-1/k}$. Let $R_{i+1} \stackrel{\text{def}}{=} \{\text{marked clusters}\}$.
   (b) If v is a node in an unmarked cluster:
      i. Let Q_v be the set of edges that consists of the lightest edge from v to each of the clusters v is adjacent to.
      ii. If v has no adjacent marked cluster, then v adds to the spanner all edges in Q_v.
      iii. Otherwise, let u be the closest neighbor of v in a marked cluster. In this case v *joins* the cluster of u, and it adds to the spanner the edge $\{v,u\}$, as well as all lighter edges from Q_v.
3. Each node v adds, for each cluster $X \in R_k$ it is adjacent to, the lightest edge connecting it to X.

For this algorithm, the following result is proven in [3].

THEOREM 4.7 ([3]). *Given a weighted graph H and an integer $k \geq 1$, the algorithm above computes a $(2k-1)$-spanner of H. It has $\mathcal{O}(k|V(H)|^{1+1/k}\log n)$ edges w.h.p.*[2]

Constructing the Skeleton Graph. In our case, each edge considered in Steps (2b) and (3) of the spanner algorithm corresponds to a shortest path. We implement these steps in our setting by letting each skeleton node find its closest $\mathcal{O}(|S|^{1/k}\log n)$ clusters by running a slightly modeified Algorithm BSP. We now explain how. First, to find the closest cluster, all nodes in a cluster use the same source ID (the one of the cluster leader). Second, we append the ID of the actual destination of the indicated path to each message to allow for routing (without treating nodes in the same cluster as different sources!). We refer to the modified algorithm as BSP$'$. Third, regarding the range parameter, Lemma 4.6 shows that it is sufficient to consider paths of $\mathcal{O}(n\log n/|S|)$ hops only. Finally, the following lemma implies that we may modify the spanner construction algorithm in a way that allows us to use a small overlap parameter.

LEMMA 4.8. *W.h.p., the centralized spanner construction algorithm yields identical results if in Steps 2b and 3, each node considers the lightest edges to the $c|V_H|^{1/k}\log n$ closest clusters only (for a sufficiently large constant $c > 0$).*

PROOF. Fix a node v and an iteration $1 \leq i < k$. If v has at most $c|V_H|^{1/k}\log n$ adjacent clusters, the claim is trivial. So suppose that v has more than $c|V_H|^{1/k}\log n$ adjacent clusters. By Step 2b of the algorithm, we are interested only in the clusters closer than the closest marked cluster. Now, the probability that none of the closest $c|V_H|^{1/k}\log n$ clusters is marked is $(1-|V_H|^{-1/k})^{c|V_H|^{1/k}\log n} \in n^{-\Omega(c)}$. In other words, for c sufficiently large, we are guaranteed that w.h.p., at least one of the closest $c|V_H|^{1/k}\log n$ clusters is marked. Regarding Step 3, observe that a cluster gets marked in all of the first $k-1$ iterations with independent probability $|V_H|^{-(k-1)/k}$. By Chernoff's bound, the probability that more than $c|V_H|^{1/k}\log n$ clusters remain in the last iteration is thus bounded by $2^{-\Omega(c\log n)} = n^{-\Omega(c)}$. Therefore, w.h.p. no node is adjacent to more than $c|V_H|^{1/k}\log n$ clusters in Step 3, and we are done. □

[2]In [3], it is proved that the expected number of edges is $\mathcal{O}(k|V(H)|^{1+1/k})$. The modified bound directly follows from Lemma 4.8.

Algorithm LDC, our implementation for the long-distance construction, therefore follows the modified Baswana-Sen algorithm as described above. The input is the skelton set S and the stretch parameter k. To facilitate Step 2, the set R_{i+1} (determined locally) is broadcast in the beginning of each phase i. Step 2b is implemented by invoking BSP$'$ with sources R_{i+1}, range $\Theta(n\log n/|S|)$ and ovelap $\Theta(|S|^{1/k}\log n)$ (this value of the overlap parameter is justified by Lemma 4.8). The spanner edges are added according to the rule of Step 2b and announced to all nodes. This enables to locally compute the clusters of the next iteration and locally store the routing pointers of the paths corresponding to the added spanner edges. Detailed pseudo-code of our implementation is given in the full paper. To prove the algorithm correct, we show that its executions can be mapped to executions of the centralized algorithm, and then apply Theorem 4.7. We omit details here, and summarize with the following lemma.

LEMMA 4.9. *Suppose the set S input to Algorithm LDC contains a uniformly random subset S_R of V. Then w.h.p., after $\tilde{\mathcal{O}}(n/|S_R|^{1-1/k} + |S|^{1+1/k} + \text{HD})$ rounds:*
(i) The algorithm computes a weighted $(2k-1)$-spanner of the skeleton graph $G_{S,h(S_R)}$ with $\mathcal{O}(|S|^{1+1/k}\log n)$ edges.
(ii) The weighted distances between nodes in S are identical in $G_{S,h(S_R)}$ and G.

Completing the Skeleton Routing Tables. Algorithm LDC constructs a globally-known skeleton-spanner, so each node can locally find a low-stretch route between any two skeleton nodes. However, to use skeleton routes we need to map skeleton edges to graph routes, where we encounter the following subtle issue. Algorithm BSP, used to detect the skeleton edges, marks the route in one way only: if a node s added the edge $\{s,t\}$ to the spanner, then Algorithm BSP sets only the next pointers from s to t, and the route from t to s is not necessarily established. (Similarly, Lemma 4.8 guarantees low *directed* degree.) This is easily resolved as follows. Say edge $\{s,t\}$ was added by node s. We send a message from s to t (we have next pointers for this direction), where the message carries the weight of the s-t route. While progressing, each node on the route records the immediate origin of the message as the next hop to s, and the distance to s; the content of the message is decremented by the weight of the edge it crossed, and then sent to the next node. By Lemma 4.9, there are at most $\tilde{\mathcal{O}}(|S|^{1+1/k})$ edges in the constructed spanner of $G_{S,h(S_R)}$ w.h.p., implying that the maximal number of messages routed over each edge of G is bounded by $\tilde{\mathcal{O}}(|S|^{1+1/k})$ w.h.p. as well. Moreover, no routing path has more than $h(S_R) \in \Theta(n\log n/|S_R|)$ hops. Since the messages traverse shortest h-hop paths, all of them reach their destinations within $\tilde{\mathcal{O}}(n/|S_R| + |S|^{1+1/k})$ rounds [19].

We summarize the properties of the long-distance scheme.

THEOREM 4.10. *Let $S \subseteq V$ be a superset of a uniformly random set of nodes S_R, and let $k \in \{1,\ldots,\log n\}$. Then w.h.p., within $\tilde{\mathcal{O}}(n|S|^{1/k}/|S_R| + |S|^{1+1/k} + \text{HD})$ rounds, Algorithm LDC(S,k) constructs routing tables for routing between nodes in S with stretch $(2k-1)$.*

4.4 Putting the Pieces Together

The overall algorithm is a composition of the short-range and long-distance algorithms, linked by identifying the skeleton set from the long-distance algorithm with the top level

of the hierarchy of the short-range algorithm (we determine the value of the parameters L and k later).

Route Selection. Suppose v wants to send a message to w. The next routing hop is selected by v as follows.
1. If $Y_v(i) = Y_w(i)$ for some i, choose the next routing hop within (the tree on) $C_{Y_v(i)}$ to w and set $d \stackrel{\text{def}}{=} \text{wd}(Y_w(i), w)$.
2. Otherwise, node v computes as follows.
(a) For $i = 1, \ldots, L$, if $Y_w(i-1) \in H_v(i)$, then $d_i := \text{wd}(v, Y_w(i-1)) + \text{wd}(Y_w(i-1), w)$. Else $d_i := \infty$.
(b) Let S_v be the set of skeleton nodes v knows how to route to, let d_s be the route weight for each $s \in S_v$, and let wd^k denote the distance function in the skeleton graph spanner. Set $d_{L+1} := \min_{s \in S_v}\{d_s + \text{wd}^k(s, Y_w(L)) + \text{wd}(Y_w(L), w)\}$.
(c) Finally, set $d := \min_{i \in \{1,\ldots,L+1\}}\{d_i\}$. The next routing hop is selected by the route giving rise to d (for minimal i).

Note that the routing decision is stateless. The following lemma bounds the stretch.

LEMMA 4.11. *Fix L and k. For any node v and label $\lambda(w)$, consider the distance value d and next routing hop* next *computed by v according to the above scheme. Then w.h.p., $d \leq (8kL-1)\text{wd}(v,w)$ and* next *will compute a value $d' \leq d - \text{wd}(v, \text{next})$.*

Finally, we set k and L to state our main result as follows.

THEOREM 4.12. *Let $1/2 \leq \alpha \leq 1$ be given. If $\alpha \geq 1/2 + 1/\log n$ let $k := \lceil 1/(2\alpha - 1) \rceil$, and otherwise let $k := \log n$. Tables of size $\tilde{\mathcal{O}}(n^\alpha)$ for stateless routing and distance approximation with stretch $\rho(\alpha) = 8k\lceil \log(k+1) \rceil - 1$ and label size $\mathcal{O}(\log(k+1)\log n)$ can be constructed in the* CONGEST *model in $\tilde{\mathcal{O}}(n^\alpha + \text{HD})$ rounds.*

Note that $\rho(1/2) \in \mathcal{O}(\log n \log\log n)$ and that $\rho(\alpha) \in \mathcal{O}(1)$ for any constant $\alpha > 1/2$.

5. EXTENSIONS AND APPLICATIONS

Generalized Steiner Forest. The Generalized Steiner Forest problem (GSF) is defined as follows.
Input: A weighted graph $G = (V, E, W)$, a set of *terminals* $T \subseteq V$, and for each terminal $t \in T$ a component number $C(t)$. In the distributed setting, we assume that each node knows whether it is a terminal, and if so, what is its component number.
Output: An edge set $F \subseteq E$ such that for all pairs $s, t \in T$ with $C(s) = C(t)$, s is connected to t in the graph (V, F).
Goal: Minimize $\sum_{e \in F} W(e)$.

To solve GSF, we do the following.
1. Broadcast all terminal labels and compute distance estimates for all terminal-terminal distances.
2. Locally construct the terminal graph $G' = (T, E', W')$, where $E' := \{\{s,t\} \mid s \neq t \in T\}$, and $W'(s,t)$ is the computed estimate of $\text{wd}(s,t)$.
3. Apply any centralized algorithm ALG for GSF to G' with the same terminal components. Obtain solution F'.
4. Identify and output all edges on paths in G that correspond to edges in F'.

It is not difficult to see that (1) solving GSF in the terimnal graph at most doubles the cost of an optimal solution, and (2) using ρ-approximate edge weights increases the optimal cost by a factor of at most ρ. To implement Step 1, we apply the long-range routing scheme with skeleton set $S := T \cup R_S$, where R_S is sampled uniformly and independently at random with probability $n^{-1/2}$ from V (cf. Lemma 4.9). We thus arrive at the following result.

THEOREM 5.1. *Given any integer $k \in [1, \log n]$ and any centralized a-approximation algorithm to* GSF, GSF *can be solved with approximation ratio $2a(2k-1)$ in time $\tilde{\mathcal{O}}((\sqrt{n} + |T|)^{1+1/k} + \text{HD})$, where T denotes the set of terminal nodes.*

Distance Sketches. The problem of distributed distance sketches requires each node to have a label and store a small amount of information (called the *sketch*), so that each node v can estimate the distance to each other node u when given the label of u.[3] Technically speaking, we already solved this problem, since our machinery enables to estimate distances with small stretch. However, since $\alpha \geq 1/2$, the basic construction always consumes $\tilde{\Omega}(\sqrt{n})$ memory.

If we discard the routing information, we can reduce the space requirements of the sketches at the expense of also increasing the stretch. To this end, we tune the sizes of the sets S_i, and change the choice of the skeleton to be $S_{L/2}$. For the remaining sets $S_{L/2+1}, \ldots, S_L$, we can construct the hierarchy locally at each node by approximating distances based on the skeleton spanner, avoiding the need to explore long paths. We arrive at the following result.

THEOREM 5.2. *Given any integer $k \in [1, \ldots, \log n]$, distance sketches with stretch $\rho(k) = 2k(8k-3) \in \mathcal{O}(k^2)$, label size $\mathcal{O}(k \log n)$, and sketch size $\tilde{\mathcal{O}}(n^{1/(2k)})$ can be constructed w.h.p. in $\tilde{\mathcal{O}}(n^{1/2+1/(2k)} + \text{HD})$ rounds.*

Approximate Weighted Diameter. Dropping the short-range scheme, identifying the skeleton set S as one source set, and using Algorithm BSP with parameters $\Delta = 1$ and $h \in \mathcal{O}(n \log n / |S|)$, we can prove the following result.

THEOREM 5.3. *Given $k \in \mathbb{N}$, the weighted diameter* WD *can be approximated w.h.p. to within a factor of $2k+1$ in the* CONGEST *model in $\tilde{\mathcal{O}}(n^{1/2+1/(2k)} + \text{HD})$ rounds.*

Tight Labels. Our routing scheme relabels the nodes according to the Voronoi partition on each level. However, using a different labeling scheme, we can set the labels to be $\{1, \ldots, n\}$, with increased stretch.

THEOREM 5.4. *Given $\alpha \in [1/2, 1]$, let $k = \lceil 1/(2\alpha - 1) \rceil$ if $\alpha > 1/2 + 1/\log n$ and $k = \lfloor 1/\log n \rfloor$ otherwise. Tables for stateless routing and distance approximation with stretch $\rho(\alpha) = 4k \cdot 4^{\lceil \log(k+1) \rceil} + 2k - 1 \in \mathcal{O}(k^3)$ with node labels $1, \ldots, n$ can be constructed in $\tilde{\mathcal{O}}(n^\alpha + \text{HD})$ rounds.*

6. OPEN PROBLEMS

In this paper we have made a first step toward finding good distributed solutions to the routing table construction problem. There still are many hurdles to overcome in this research direction. We list a few below.

[3]The formulation in [7] permits to use *both* sketches to approximate the distance. However, from the distributed point of view it is more appropriate to assume that only a minimal amount of information is exchanged.

- We did not treat the problem of dynamic networks, or more generally fault-tolerant routing. Almost all practical routing algorithms must cope with dynamic changes in the network, or faults. Can our algorithm be extended to such scenarios, and at what cost?
- It is interesting to better understand stateful routing (note that our lower bounds do not cover the case of stateful, name-dependent routing). What can be done with general stateful routing?
- Our algorithm is designed specifically to optimize the worst case. It is very interesting to find a "competitive" algorithm, whose running time is related to the best possible for any given network.

Acknowledgment

We thank David Peleg for valuable discussions.

7. REFERENCES

[1] I. Abraham, C. Gavoille, and D. Malkhi. On Space-Stretch Trade-Offs: Lower Bounds. In *Proc. 18th ACM Symp. on Parallelism in Algorithms and Architectures*, pages 207–216, 2006.

[2] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. *ACM Transactions on Algorithms*, 4(3):37:1–37:12, 2008.

[3] S. Baswana and S. Sen. A Simple and Linear Time Randomized Algorithm for Computing Sparse Spanners in Weighted Graphs. *Random Structures and Algorithms*, 30(4):532–563, 2007.

[4] R. E. Bellman. On a routing problem. *Quart. Appl. Math.*, 16:87–90, 1958.

[5] P. Chalermsook and J. Fakcharoenphol. Simple distributed algorithms for approximating minimum steiner trees. In *Proc. 11th Ann. Conf. on Computing and Combinatorics*, volume 3595 of *LNCS*, pages 380–389, 2005.

[6] J. Chawachat, J. Fakcharoenphol, and D. Nanongkai. Fast shortest-path algorithms on distributed networks. Unpublished manuscript, 2013.

[7] A. Das Sarma, M. Dinitz, and G. Pandurangan. Efficient Computation of Distance Sketches in Distributed Networks. In *Proc. 24th ACM Symp. on Parallelism in Algorithms and Architectures*, 2012.

[8] A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. In *Proc. 43th ACM Symp. on Theory of Computing*, pages 363–372, 2011.

[9] M. Elkin. An Unconditional Lower Bound on the Time-Approximation Tradeoff for the Minimum Spanning Tree Problem. *SIAM Journal on Computing*, 36(2):463–501, 2006.

[10] M. Elkin and D. Peleg. $(1+\epsilon, \beta)$-Spanner Constructions for General Graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.

[11] L. R. Ford. Network flow theory. Technical Report P-923, The Rand Corp., 1956.

[12] S. Frischknecht, S. Holzer, and R. Wattenhofer. Networks Cannot Compute their Diameter in Sublinear Time. In *Proc. 23rd ACM-SIAM Symp. on Discrete Algorithms*, pages 1150–1162, 2012.

[13] J. A. Garay, S. Kutten, and D. Peleg. A Sub-Linear Time Distributed Algorithm for Minimum-Weight Spanning Trees. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 659–668, 1993.

[14] S. Holzer and R. Wattenhofer. Optimal Distributed All Pairs Shortest Paths and Applications. In *Proc. 31st ACM Symp. on Principles of Distributed Computing*, 2012.

[15] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient Distributed Approximation Algorithms via Probabilistic Tree Embeddings. *Distributed Computing*, 25:189–205, 2012.

[16] M. Khan and G. Pandurangan. A Fast Distributed Algorithm for Minimum Spanning Trees. *Distributed Computing*, 20:391–402, 2008.

[17] S. Kutten and D. Peleg. Fast Distributed Construction of Small k-Dominating Sets and Applications. *Journal of Algorithms*, 28(1):40–66, 1998.

[18] C. Lenzen and B. Patt-Shamir. Fast Routing Table Construction Using Small Messages. *ArXiv e-prints*, Oct. 2012.

[19] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *Journal of Algorithms*, 14:449–465, 1993.

[20] J. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the arpanet. *IEEE Trans. on Communications*, COM-28(5):711–719, May 1980.

[21] J. Moy. OSPF version 2. RFC 2328, Network Working Group, 1998.

[22] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[23] D. Peleg, L. Roditty, and E. Tal. Distributed Algorithms for Network Diameter and Girth. In *Proc. 39th Int. Colloquium on Automata, Languages, and Programming*, 2012.

[24] D. Peleg and V. Rubinovich. Near-tight Lower Bound on the Time Complexity of Distributed MST Construction. *SIAM Journal on Computing*, 30:1427–1442, 2000.

[25] D. Peleg and A. A. Schäffer. Graph Spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[26] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, 1989.

[27] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 5th edition, 2011.

[28] N. Santoro and R. Khatib. Labelling and Implicit Routing in Networks. *The Computer Journal*, 28:5–8, 1985.

[29] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. 13th Ann. ACM Symp. on Parallel Algorithms and Architectures*, 2001.

[30] M. Thorup and U. Zwick. Approximate Distance Oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[31] U. Zwick. Exact and Approximate Distances in Graphs - A Survey. In *Proc. 9th Ann. European Symp. on Algorithms*, pages 33–48, 2001.