On the Communication Complexity of Distributed Name-Independent Routing Schemes*

Cyril Gavoille[1], Christian Glacet[2], Nicolas Hanusse[3], and David Ilcinkas[3]

[1] LaBRI - University of Bordeaux, Bordeaux, France
[2] LaBRI - INRIA Bordeaux Sud-Ouest, Bordeaux, France
[3] LaBRI - CNRS, Bordeaux, France

Abstract. We present a distributed asynchronous algorithm that, for every undirected weighted n-node graph G, constructs name-independent routing tables for G. The size of each table is $\tilde{O}(\sqrt{n})$, whereas the length of any route is stretched by a factor of at most 7 w.r.t. the shortest path. At any step, the memory space of each node is $\tilde{O}(\sqrt{n})$. The algorithm terminates in time $O(D)$, where D is the hop-diameter of G. In synchronous scenarios and with uniform weights, it consumes $\tilde{O}(m\sqrt{n}+n^{3/2}\min\{D,\sqrt{n}\})$ messages, where m is the number of edges of G.
In the realistic case of sparse networks of poly-logarithmic diameter, the communication complexity of our scheme, that is $\tilde{O}(n^{3/2})$, improves by a factor of $\sqrt{n}$ the communication complexity of *any* shortest-path routing scheme on the same family of networks. This factor is provable thanks to a new lower bound of independent interest.

Keywords: distributed routing algorithm, name-independent, compact routing, bounded stretch

1 Introduction

Message routing is a central activity in any interconnection network. Route efficiency and memory requirements are two major central parameters in the design of a routing scheme. Routing along short paths is clearly desirable, and the storage of the routing information at each node must also be limited to allow quick routing decision, fast update, and scalability. There is a trade-off between the route efficiency (measured in terms of stretch) and the memory requirements (measured by the size of the routing tables). The shorter the routes, the larger the routing tables. It is also desirable that routing schemes are universal, i.e., they apply to any topology, as the model of large dynamic networks cannot be guaranteed. An additional desirable property of a routing scheme is to use arbitrary routing addresses (say based on processor IDs or MAC addresses), and

* All the authors are supported by the ANR-project DISPLEXITY (ANR-11-BS02-014), and the European STREP7-project EULER. The first author is also member of the "Institut Universitaire de France".
{gavoille,glacet,hanusse,ilcinkas}@labri.fr

thus addresses independent of the topology. Such routing schemes are called name-independent.

This paper focuses on distributed algorithms that can construct universal and name-independent routing schemes for static networks. For practical use, it is essential that such distributed algorithms be as fast as possible (typically linear in the diameter) since the objective is to quickly update routing tables after topological changes in the network. Naturally, to optimize the network troughtput, a distributed algorithm must consume as few messages as possible. We are therefore interested in time and communication complexities of distributed routing schemes. There are well-established trade-offs between the stretch and the memory for centralized routing schemes (see the related works part in Section 1.4). In this paper we show some different trade-offs between the stretch, the memory, and the communication complexity of distributed routing schemes. The fundamental question we address is to determine whether or not theoretical optimal space-stretch trade-offs can be achieved when time and communication complexities are restricted.

1.1 Terminology and Models

We consider undirected weighted graphs with positive edge-weights. The *aspect ratio* of a weighted graph G is the maximum ratio between any two edge-weights in G. A *shortest path* between u and v in G is a path of minimum cost (the weight sum of the path edges) connecting u to v in G, and this cost is the *distance* between u and v. The *hop-distance* between u and v is the minimum number of edges in a shortest path between u and v. The *hop-diameter* is the largest hop-distance in the graph.

In the case of uniform weights, the aspect ratio is 1 and the hop-diameter corresponds to the classical notion of diameter in unweighted graphs. It is well-known that the asynchronous distributed Bellman-Ford algorithm can construct a shortest-path spanning tree rooted at a node u in time $h + 1$, where h is the height of the tree and also the maximum hop-distance between u and its leaves (see [13]). This time is thus at most the hop-diameter of G plus one. The hop-diameter plays an important role, not only in the time for computing a shortest-path tree, but in the running time of all subsequent distributed subroutines using this tree (e.g. for broadcasting).

A *routing scheme* on a family of graphs is an algorithm that produces, for every graph G of the family, a routing algorithm for G. A *routing algorithm* is in charge of delivering any message from every source to every destination node in G. A *name-independent* routing algorithm must deliver messages assuming that the destination names given at the sources are the original names of the input graph.

The *stretch factor* of a routing algorithm is the maximum, over all source-destination pairs (u, v), of the ratio between the cost of the route from u to v, and the distance from u to v in G. So, shortest-path routing algorithms have stretch factor exactly one. The *round-trip stretch factor* is the maximum ratio between the total cost of the route going from u to v and back to u, and the

distance from u to v plus the distance from v to u. This notion is naturally used in the context of directed graphs [23], where the distance from u to v may differ from the one from v to u. In this paper, graphs are undirected though. Note that if the round-trip stretch is bounded above by s, then the average stretch (average over all the source-destination pairs) is at most s.

The *routing tables* are the local data structures used by the routing algorithm to perform routing. The *working memory space* (a.k.a. per-node state or topological memory) is the maximum memory space a node of the graph needs when running the distributed routing scheme. If the working memory space is S, then the routing tables have size at most S as well. The challenge is to design routing schemes with working memory space that is sub-linear in n and not significantly greater than the size of the final routing tables.

We assume a reliable asynchronous network, where a message sent along an edge is received after an unpredictable but finite time. The time complexity of a distributed algorithm A is the worst-case difference of time units between the first emission of a message and the last reception of a message during any execution of A, assuming the slowest message uses one time unit to traverse an edge. The *bit-message* complexity of A is the worst-case total number of bits exchanged along the edges of the graph during any execution of A. As in the standard asynchronous model, processors have no synchronous wake-up: they can either spontaneously wake up, or be activated when receiving a message. We make no assumptions on the number of messages that can be transmitted over a link in one time unit, and so we ignore congestion problems.

As specified by the name-independent model, we do not make any assumption on the distribution of node identifiers, which are chosen by an adversary. However, using hashing technique as explained in [4, 8], we will assume that node identifiers can be represented on $O(\log n)$ bits.

Each message of our distributed algorithm has a poly-logarithmic size. More precisely, messages have size at most $B = O(\log W + \min\{D, \log n\} \cdot \log n)$ bits, where W is the aspect ratio and D is the hop-diameter of the graph. We also assume that each entry of the routing tables is large enough to receive B bits. The *size* of a routing table is the number of its entries. We assume that whenever a node receives a message on some incident edge, it can determine the weight of that edge.

1.2 Our Results

We design a new distributed routing scheme and two lower bounds.

- We propose an asynchronous distributed name-independent routing scheme for weighted n-node graphs of hop-diameter D. The stretch is 7 and the round-trip stretch is 5. The time complexity is $O(D)$, with a small hidden constant (< 10). Moreover, at each time of the algorithm, the working memory space of each node is[4] $\tilde{O}(\sqrt{n}\,)$. In particular, the routing tables have size

[4] The notation $\tilde{O}(f(n))$ stands for a complexity in $O(f(n) \cdot \log^{O(1)} f(n))$.

$\tilde{O}(\sqrt{n}\,)$. In a synchronous scenario, and in the case of uniform weights, the message complexity is $\tilde{O}(m\sqrt{n} + n^{3/2} \min\{D, \sqrt{n}\,\})$.
– For the realistic case of weighted sparse networks of poly-logarithmic hop-diameter, the message complexity is $\tilde{O}(n^{3/2})$. A simple variant of our algorithm shows that, for this same family of networks, we can achieve stretch 5 with sub-linear routing tables and sub-quadratic message complexity. See Table 1 for a summary.

Schemes	Stretch	Memory	#Message	Time	Reference
Distance or Path Vector	1	$\Omega(n)$	$O(n^2)$	$O(D)$	
DistRoute($n^{1/2}$)	7	$\tilde{O}(n^{1/2})$	$\tilde{O}(n^{3/2})$	$O(D)$	Corollary 1
DistRoute$'$($n^{2/3}$)	5	$\tilde{O}(n^{2/3})$	$\tilde{O}(n^{5/3})$	$O(D)$	Corollary 2
Memory lower bound	$< 2k+1$	$\Omega((n \log n)^{1/k})$	any	any	[2]
#messages lower bound	1	any	$\Omega(n^2)$	$o(n)$	Theorem 2
Time lower bound	$\leqslant n/(3D)$	any	any	$\Omega(D)$	Theorem 1

Table 1. Fast distributed name-independent routing schemes for realistic weighted graphs, i.e., with $\tilde{O}(n)$ edges and $\log^{O(1)} n$ hop-diameter. The "Memory" column stands for working memory space and routing table size. Note that lower bounds are given in bits or bit-messages.

Our lower bounds show that time $\Omega(D)$ is indeed required for any constant stretch, and that shortest-path routing requires $\Omega(n^2)$ bit-message complexity even on sparse graphs of logarithmic diameter. More precisely, we prove that:

(1) Every synchronous constant-stretch name-independent distributed routing scheme requires time $\Omega(D)$ on unweighted graphs of diameter D. This bound is independent of the bit-message complexity and the routing table size of the scheme.
(2) There are unweighted n-node graphs of diameter $O(\log n)$ and with maximum degree 3 for which every synchronous distributed shortest-path routing scheme (name-independent or not) of $o(n)$ time complexity requires $\Omega(n^2)$ bit-message complexity.

For these lower bounds, we assume a synchronous scenario which also implies the results for asynchronous scenarios. We also point the fact that we do not make any restriction on the message length.

1.3 Discussion

Our first lower bound may seem trivial at first glance. It is indeed immediate to show that a time $\Omega(D)$ is required for shortest-path routing schemes. Just consider for instance a path of D nodes and a source in the middle of the path. However, this folklore lower bound is less straightforward when *arbitrary* stretched routing schemes are considered. Let us stress that, for paths, the cow-path routing algorithm [12, 18] achieves stretch 9 without any routing tables!

One may also think that the second lower bound is again folklore since clearly a shortest-path routing scheme must send at least one message on each edge. Otherwise subsequent routing queries will not be able to use all the edges of the graph (and so cannot be a shortest-path routing). This gives a communication complexity of $\Omega(n^2)$ for dense graphs. However, this quadratic bound cannot be guaranteed using the same argument for sparse graphs as stated by our lower bound. An option to prove a quadratic bound for sparse graphs might be to show that $\Omega(n)$ bits of information must be transmitted along long paths in the graph, say paths of $\Omega(n)$ edges. Again, this cannot be achieved for poly-logarithmic diameter graphs. Finally, we stress that the arguments of any formal proof must take into account the time complexity of the routing scheme. This is because a 1-bit message can carry more than one bit of information. For instance a 1-bit message can be sent during odd or even clock pulse to carry more information. Senders could also decide to send 1-bit or 2-bit messages, so encoding extra information with the message length.

Our distributed routing scheme, although universal, achieves better performance when realistic networks are considered. By realistic networks we mean sparse and small-diameter graphs, typically graphs with $\tilde{O}(n)$ edges and poly-logarithmic diameter. The classical Distance Vector and Path Vector routing protocols both achieve message complexity of $\Omega(mn) = \Omega(n^2)$ for realistic networks, whereas our scheme consumes at most $\tilde{O}(n^{3/2})$ messages. This good theoretical behavior is confirmed by experiments. We have implemented our routing scheme on a fully distributed routing scheme simulator[5]. For instance, on CAIDA-2004 map[6], our scheme[7] produces an average stretch of 1.75 for 534 entry routing tables on average (maximum size is 1002), and this after exchanging a total of 55M messages (synchronous scenario). Running Distance Vector on the simulator on the same graph generates routing tables of 16K entries after exchanging 1,617M messages. Note that our scheme reduces both the number of messages and the number of entries by a factor close to 30.

Our scheme is widely inspired from the universal name-independent routing scheme [4] that achieves the smallest possible stretch for routing tables of size $\tilde{O}(\sqrt{n}\,)$. Following the work of [4], stretch-3 can be achieved at the price of an extra communication cost factor of roughly $\sqrt{n}$ over our stretch-7 scheme. The communication complexity becomes therefore $\Omega(mn)$, which is as high as the complexity of a shortest-path routing scheme. To implement the stretch-3 scheme of [4], we need to consider the set of vicinity balls touching the vicinity ball of a given node u. Unfortunately, there are small diameter graphs where each node has $\Theta(n)$ different touching vicinity balls, which implies a total volume of $\Omega(n^2)$ routing information to manage in the graph. This translates into a $\Omega(n^2)$ communication complexity. Designing a distributed routing scheme with stretch 3 and $o(n^2)$ message complexity on small diameter graphs, if it exists, requires another approach.

[5] Source code available on demand.
[6] It has 16K nodes and 32K edges.
[7] More precisely, we run DistRoute(k) for $k = 78$, see Section 3.

To conclude the discussion, let us stress that bounding the working memory space of each node considerably reduces the set of standard tricks to decrease communication complexity. For instance, when $o(n)$ working memory space is forced, then a simple broadcast in a spanning tree may cost $O(m)$ messages instead of $O(n)$ messages (since a node cannot store all its children in the tree). More generally, the γ-synchronizer methodology [9] cannot be applied, and the use of sparse spanners (like in [14]) on which subsequent routines consume less messages is problematic.

1.4 Related Works

The theory of name-independent routing schemes has a long history, and started early with Kleinroch's work about routing in the ARPANET. The first provable trade-off between the size of the routing tables and the stretch appeared in [11]. In the line of hierarchical routing schemes initiated by Kleinroch et al. [19], the authors have proposed a name-independent routing scheme of stretch 2^k-1 with routing tables of size $\tilde{O}(n^{1/k})$ on average, where $k \geqslant 1$ is an integral parameter. In [8], better space-stretch trade-offs have been proposed. In particular, the size of the routing tables is bounded by $\tilde{O}(n^{2/k})$ for each node, and not only on average, and the stretch is in $O(k^2)$. However, the schemes assume polynomial aspect ratio. They achieve a stretch 3 with routing tables of size $\tilde{O}(n^{2/3})$, and a stretch 5 for routing tables of size $\tilde{O}(\sqrt{n}\,)$. Finally, [3] proposed a scheme with linear stretch $O(k)$ for routing tables of size $\tilde{O}(n^{1/k})$, and this for arbitrary weighted graphs. According to the best current lower bounds, a linear stretch[8] $\Omega(k)$ is optimal for routing tables of size $\tilde{O}(n^{1/k})$. More precisely, [2] showed that there are weighted depth-1 trees with edge-weights in $\{1, k\}$ such that every name-independent routing scheme of stretch $< 2k+1$ requires $\Omega((n\log n)^{1/k})$-bit routing tables. According to this lower bound, routing schemes of stretch < 5 require routing tables of $\Omega(\sqrt{n\log n}\,)$ bits ($k=2$), and the best possible stretch for $o(n\log n)$-bit routing tables is $\geqslant 3$ ($k=1$). Note that these lower bounds apply to realistic graphs. A scheme with stretch-3 and $\tilde{O}(\sqrt{n}\,)$-bit routing tables has been proposed in [4], which is therefore optimal in space and stretch.

Better stretch-space trade-offs can be achieved for more specific classes of networks. Bounded growth [6] and bounded doubling dimension [1, 20] graphs, trees [21], planar and more generally minor-free unweighted graphs [5], support name-independent routing schemes of constant stretch and poly-logarithmic routing tables.

For practical usage, several distributed routing schemes have been proposed and implemented, and first of all distributed shortest-path routing schemes (stretch 1). Distance Vector and Path Vector protocols are such distributed routing schemes. Based on Bellman-Ford algorithm, they produce after a time $O(D)$ shortest-path routing tables of linear size using $O(mn)$ messages, for small aspect ratio graphs. A variant of Bellman-Ford supporting an aspect ratio $W > 1$ uses $O(mn\log(nW))$ messages while preserving the time complexity. However paths

[8] This holds also for the average stretch.

are no longer shortest paths and may have stretch up to 3. The message complexity of shortest-path routing has been reduced to $O(n^2 \log n)$ in [7], degrading the time complexity to $O(D \log n)$. Actually, $2n^2$ messages are enough [17], but messages can be as large as $\Omega(n \log (nW))$ bits, whereas in Bellman-Ford based routing schemes and in [7], messages have size $O(\log(nW))$ bits.

As proved by the theoretical lower bounds, shortest-path routing has to be scrapped right away if sublinear working memory space and sublinear routing tables are required. In this spirit, [11] proposed a synchronous distributed routing scheme with stretch $2 \cdot 3^k - 1$ and working memory space of $\tilde{O}(d + n^{1/k})$ for a degree-d node. For $k = 2$, the working memory space and routing tables are $\tilde{O}(d + \sqrt{n}\,)$, and the stretch is 17. In [24], a distributed implementation of a stretch-7 routing scheme is presented. Routing tables have size $\tilde{O}(\sqrt{n}\,)$ but the message complexity is not analytically bounded. Moreover, each entry in the tables can be as large as $\Omega(D)$, and the working memory space as large as $\Omega(d\sqrt{n}\,)$ for a degree-d node. [25] proposed a variant of the routing scheme of [4], and show experiments on synthetic power-law graphs and real AS-graphs. For these unweighted graphs, the stretch is asymptotically 2, but it is unbounded for general graphs, even unweighted ones. Techniques using sparse spanners, like in [14, 15], can achieve almost shortest paths with message complexity $\tilde{O}(mn^{\epsilon_1} + n^{2+\epsilon_2})$ where $0 < \epsilon_1, \epsilon_2 < 1$ are constants that can be arbitrarily chosen and influence the stretch of the paths. We observe that for unweighted sparse graphs of small diameter, the scheme requires at least $\Omega(n^2)$ messages and $\Omega(n)$ working memory space.

As far as we know, no distributed name-independent routing scheme is able to guarantee a bounded stretch and a sublinear working memory space.

In the next section, we present our lower bounds on the time and message complexities. In Section 3, we formally present the performance of our distributed routing scheme and give an overview of the scheme. Due to lack of space, details of the proofs and of the distributed algorithm are omitted.

2 Lower Bounds

2.1 Time Lower Bound

We give a formal proof that $\Omega(D)$ time is required for any distributed routing scheme of constant stretch (the result extends to stretch as large as n/D). Our proof is independent of the message and routing table sizes used by the distributed routing scheme. The lower bound holds for single-source routing schemes, a sub-class of routing schemes. A *single-source* routing algorithm can only deliver messages from a fixed source node of the graph. And, a routing scheme is single-source if the routing algorithms it produces are single-source.

A (d, k)-star is a rooted tree with $dk + 1$ nodes obtained by replacing each edge of a $K_{1,d}$ graph, a star of degree d, by a path of k edges. The root is the degree-d node.

Theorem 1. *Every synchronous distributed name-independent routing scheme on the family of unweighted (d, k)-stars, and running in time $t < k$, produces a route of length at least $(2d-1)(k-t)+t$ between the root and some leaf.*

In particular, every synchronous distributed single-source name-independent routing scheme on unweighted n-node graphs of diameter at most $D \in \{2, \ldots, n-1\}$, and of stretch factor at most $\frac{1}{3}n/D$, requires a time $\Omega(D)$.

Note that for $t = 0$ (no pre-processing), the problem stated by Theorem 1 is equivalent to the d-lane cow-path problem in which the distance to the destination, here k, is known at the source. Our bound gives a stretch of $2d-1$ which is known to be optimal if the distance is known and no pre-processing is allowed (cf. [12, 18]).

2.2 Communication Complexity Lower Bound

Next, we prove that the $o(n^2)$ bit-message complexity for sparse graphs, as in Corollary 1, cannot be achieved without degrading the stretch factor. Importantly, the bound holds independently of the compactness of the routing tables, and of the message length.

Theorem 2. *There are a constant $\lambda > 0$, and some unweighted n-node graphs of diameter $O(\log n)$ and maximum degree 3, for which every synchronous distributed shortest-path routing scheme (name-independent or not) of time complexity at most λn requires $\Omega(n^2)$ bit-message complexity.*

3 An Asynchronous Distributed Routing Scheme

Our distributed routing scheme, denoted by DistRoute(k), assumes that each node initially receives a color[9] in $\{1, \ldots, k\}$, where k is an integral parameter of our scheme. In practice, each node picks its color independently at random in $\{1, \ldots, k\}$. However our scheme is deterministic. As we will see in Theorem 3, the correctness of our scheme is independent of the node coloring, which is not the case of the routing scheme of [4].

Theorem 3. *Let G be a connected weighted n-node graph of hop-diameter D. For every k-coloring of G,* DistRoute(k) *is a deterministic asynchronous distributed routing scheme for G. It runs in time $O(D)$. The message complexity is no more than $O(n)$ times the number of messages that a single-source distributed Bellman-Ford consumes in G.*

The routing algorithm it produces has stretch 7, round-trip stretch 5, and uses headers of $O(\min\{D, \log n\} \cdot \log n) \subset O(\log^2 n)$ bits. Each routing decision takes constant time, and the header of each routing message, once created at the source, is modified at most once along the path to the destination.

[9] We do not impose that neighbors get different colors.

Our scheme directly depends on the asynchronous distributed Bellman-Ford that can generate $\Omega(2^n)$ messages in worst-case asynchronous scenarios and for graphs of large aspect ratio (see [10]). So, in some occasions, our scheme may generate an exponential number of messages. However, in a synchronous scenario and for graphs of low aspect ratio, the message complexity is polynomial. Note that it is well-known that the message complexity of the distributed Bellman-Ford algorithm is polynomial on average, and even $O(n^2\Delta^3)$ with overwhelming probability, where Δ is the maximum degree of the graph [26].

The next result (Theorem 4) specifies the size of the routing tables and the message complexity. Both complexities depend on the node coloring, the aspect ratio W of the graph, and on synchrony. The parameters involved in the analysis, namely n, m, D, W, are not known by the nodes when the distributed scheme starts. We will essentially make the two following assumptions:

Random Coloring. The node coloring is uniformly random in $\{1, \ldots, k\}$, and $k = n^\alpha$ for some constant $\alpha \in (0, 1)$. The results claimed under this stochastic hypothesis then hold in expectation or with high probability (w.h.p.)[10], where the probabilities are computed over all k-colorings of the graph.

Synchronous Scenario. The network is synchronous. In that case, the distributed Bellman-Ford algorithm uses a polynomial number of messages.

Hereafter, we define $\xi = 1 + D(1 - 1/W)$. This value appears in the message complexity of our scheme in synchronous scenarios. It corresponds to the maximum number of times a node u changes its state when computing the hop-distance to a node v. At each change, u sends a message to its neighbors. Observe that for uniform weighted graphs $\xi = 1$ as $W = 1$.

Theorem 4. *Let G be a connected weighted n-node graph of hop-diameter D, with m edges, and with aspect ratio W. Under the random coloring hypothesis,* DistRoute(k) *on G produces w.h.p. a working memory space and routing tables of size $O(k \log k + n/k)$. Furthermore if the scenario is also synchronous, the message complexity is, in expectation,*

$$O\left(\xi m \left(k \log k + \frac{n}{k}\right) + \frac{n^2}{k} \cdot \min\{D, k\}\right) .$$

So, for $k = \sqrt{n/\log n}$ the routing tables have $O(\sqrt{n \log n})$ entries, and in the case of uniform weights ($W = \xi = 1$), the message complexity in Theorem 4 even simplifies to

$$\tilde{O}(m\sqrt{n} + n^{3/2} \cdot \min\{D, \sqrt{n}\}) .$$

Another important particular corollary of our analysis is the following:

Corollary 1. *Under random coloring and synchronous hypotheses, and for weighted n-node graphs with $\tilde{O}(n)$ edges and poly-logarithmic hop-diameter, the distributed routing scheme* DistRoute($\sqrt{n}$) *has message complexity $\tilde{O}(n^{3/2})$, produces a stretch-7 routing algorithm, and w.h.p. a working memory space and routing tables of size $\tilde{O}(\sqrt{n})$.*

[10] It means that it holds with probability at least $1 - 1/n^c$ for some constant $c \geqslant 1$.

A simple variant of our algorithm, denoted by DistRoute$'(k)$, fulfills all the statements of Theorem 3 except that it achieves stretch 5. This is done at a price of an extra communication cost of $O(n^3/k^2 \cdot \min\{D, k\})$ messages (under the hypothesis of Theorem 4). We obtain another trade-off which is:

Corollary 2. *Under random coloring and synchronous hypotheses, and for weighted n-node graphs with $\tilde{O}(n)$ edges and poly-logarithmic hop-diameter, the distributed routing scheme* DistRoute$'(n^{2/3})$ *has message complexity $\tilde{O}(n^{5/3})$, produces a stretch-5 routing algorithm, and w.h.p. a working memory space and routing tables of size $\tilde{O}(n^{2/3})$.*

The message complexity that can be achieved by DistRoute or DistRoute$'$ on realistic graphs without the synchronous hypothesis is significantly higher than $\Omega(n^2)$. Observe however that by a slight modification of the algorithms, namely by adding an α-synchronizer (cf. [22]), we can still guarantee a message complexity of respectively $\tilde{O}(n^{3/2})$ and $\tilde{O}(n^{5/3})$ in the asynchronous setting while keeping a time complexity of $O(D)$.

3.1 Overview of the Scheme

Consider an initial uniformly random k-coloring of the nodes of the graph, and denote by $c(u) \in \{1, \ldots, k\}$ the color selected by node u. In parallel of the coloring, nodes are split into groups of size $O(n/k)$ thanks to a fixed balanced hash function h, as in [4], mapping in constant time and w.h.p. the node identifiers to the set $\{1, \ldots, k\}$. A node of color i will be responsible of the routing information for all the nodes of hash value i. Nodes of color 1, called *landmarks*, have a special use in the scheme.

Consider an arbitrary node u. Node u stores three types of routing information. (1) The node u stores in a table B_u the information on how to route along shortest paths to its *vicinity ball*, a set containing $O(k \log k)$ nodes closest to u. More precisely, this ball contains the smallest number of nodes closest to u such that each color has been chosen by at least one node of the ball. (2) For each landmark l, the node u stores a shortest path between l and u. These pieces of information are stored in a table L_u. (3) For each node v such that $h(v) = c(u)$, the node u stores in a table C_u the closest landmark to v, namely l_v, and a shortest path from l_v to v.

All these paths stored in the second and third tables are not arbitrary but are extracted from fixed shortest-path spanning trees T_l rooted at each landmark l. Moreover, paths are stored in a compressed way into *routing labels*, using only $O(\min\{D, \log n\} \cdot \log n)$ bits, thanks to a distributed variant of the technique of [16]. Overall, the routing table of u has size $O(k \log k + n/k)$ since there are $O(n/k)$ landmarks and nodes with the same hash value.

We now describe how the actual routing from a source s to a destination t is performed using these tables. If $t \in B_s$, then the table B_s allows s to transmit the packet along a shortest path to t. Otherwise, node s forwards the packet to the closest node $u \in B_s$ such that $c(u) = h(t)$. This is done by putting u's

identifier in the header of the packet. Also, note that u may be the node s itself. Once in u, the header is replaced by l_t and the compressed path from l_t to t stored in the table C_u. Now, thanks to the header and to the tables L_v of all the intermediate nodes v, the packet will follow the unique path from u to t in the shortest-path spanning tree T_{l_t} rooted in l_t (see Fig. 1).

In practice, the routing algorithm can be improved when routing on the unique path from u to t in T_{l_t}. Each intermediate node v on this path first checks whether node t_i, the i-th nearest ancestor of t on the path from l_t to t, belongs to B_v and is not an ancestor of v in T_{l_t}. In that case, v can route directly to t_i along a shortest path, producing a shortcut in the path from v to t_i in T_{l_t}. These nodes t_i are contained in the header available at v, and they are checked in the order t_0, t_1, t_2, ... where $t_0 = t$. Actually, due to the compressed representation of the path, only $\min\{D, \log n\}$ nodes t_i are available at v.

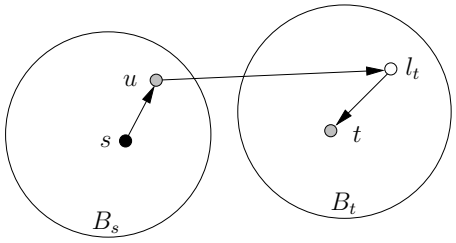

Fig. 1. Routing from s to t where $c(u) = h(t)$.

The stretch analysis of the routing algorithm is as follows. If $t \in B_s$, the stretch is 1. Otherwise, assume that s and t are at distance d. Then the cost of the route $s \rightsquigarrow u$ is at most d, since $t \notin B_s$. The route $l_t \rightsquigarrow t$ is at most $2d$ since the landmark of s (that is in B_s) is at distance at most $2d$ from t, and l_t is the closest landmark to t. It follows that the cost of the route $u \rightsquigarrow l_t$ is bounded by the cost of the route $u \rightsquigarrow s \rightsquigarrow t \rightsquigarrow l_t$ which is at most $4d$. Therefore, the cost of the route $s \rightsquigarrow u \rightsquigarrow l_t \rightsquigarrow t$ is at most $d + 4d + 2d = 7d$. The round-trip stretch analysis is similar and gives an upper bound of 5.

Note that stretch 5 can be achieved if the segment of the route $u \rightsquigarrow t$ would have been done in tree T_{l_s} instead of T_{l_t}. Indeed, the route $u \rightsquigarrow t$ would not be longer that the route $u \rightsquigarrow s \rightsquigarrow l_s \rightsquigarrow s \rightsquigarrow t$ where each of the four segments is a shortest path of length at most d, yielding to a total of $5d$ from s. In other words, u could have store a better landmark tree path in C_u for v. We use this observation for the variant DistRoute$'(k)$ and to prove Corollary 2. Unfortunately, this consumes more messages to construct such enhanced tables C_u.

3.2 Overview of the Distributed Routing Scheme

The goal of the distributed routing scheme is to compute, for every node u, the tables B_u, L_u and C_u. The computation of the table C_u is made after every node v has computed its landmark table L_v. For that we use a weak synchronization

that allows to reduce the number of messages in asynchronous environments since no unreliable information about landmark tables are sent. Thereby our algorithm can be described as two sub-algorithms that run in parallel. The first one computes B_u, and the second one computes L_u, then C_u.

The algorithm to compute vicinity balls is similar to the distributed Bellman-Ford algorithm. The main difference is that to construct B_u, the closest nodes to u start a shortest-path tree spanning u. Importantly, to save messages, tie break between candidates of the last layer for B_u is selected according to the arrival order of their discovery message received at u. This also guarantees that the monotony property of vicinity balls is respected: the next-hop w to reach any $v \in B_u$ from u verifies that $v \in B_w$. In the synchronous scenario, the construction of all vicinity balls takes $O(\xi mk \log k)$ messages.

The algorithm to construct L_u and C_u is subdivised into the following steps, each one runing in time $O(D)$.

Step 1. Each landmark l starts the construction of a shortest-path spanning tree T_l. During this process, node u stores its parents in T_l for all the landmarks, and learns the landmark of smallest identifier, the leader denoted by $l_{\min}$. In a synchronous scenario, Step 1 consumes $O(\xi mn/k)$ messages.

Step 2. After detecting termination of Step 1, the routing label of u in each tree T_l, denoted by $\ell(u, T_l)$, is computed by a process we describe in Section 3.3. After Step 2, L_u is computed, and u can determine its closest landmark denoted by l_u. The termination detection of Step 1 is done by $l_{\min}$ and takes $O(m)$ messages, and Step 2 consumes $O(mn/k)$ messages in total. Note that our bound on the working memory space prevents us from broadcasting in a tree in $O(n)$ messages, because a node cannot store all its children.

The goal of the last two steps is to construct C_u. For that, u needs to retrieve the routing label $\ell(v, T_{l_v})$ for every node v such that $h(v) = c(u)$. For that, every node v of hash value $h(v)$ sends its label to its closest node of color $h(v)$, say w. Node w is then in charge of broadcasting this message to all nodes u of color $h(v)$. It is important to note that we want a more efficient algorithm than a simple broadcast for each node, which would require $\Omega(n^2)$ messages.

Step 3. In this step, we construct an efficient broadcasting scheme composed of k *logical trees*, one for each color. We will use them in Step 4. For every color $i \in \{1, \ldots, k\}$, we build a logical tree $\mathcal{T}_i$ whose node-set is composed only of nodes of color i in G. An edge between w and w' in $\mathcal{T}_i$ represents a path from w to w' in $T_{l_{\min}}$ without any intermediate node of color i.

To construct the edge $\{w, w'\}$ in $\mathcal{T}_i$, w sends to its parent in $T_{l_{\min}}$ a message $\langle i, \ell(w, T_{l_{\min}})\rangle$ to find a potential parent in $\mathcal{T}_i$. (There is a special treatment we do not detail whenever w has no ancestor of color i in $T_{l_{\min}}$.) Such a message is forwarded to the parent of the current node until a node w' of color i is encountered. Whenever node w' learns the existence of w, it knows how to reach w through the routing label $\ell(w, T_{l_{\min}})$. It acknowledges to w by indicating its own routing label $\ell(w', T_{l_{\min}})$.

We can prove that edges of $\mathcal{T}_i$ are composed on average of $t \leqslant 2 \min\{D, k\}$ edges of G. So, to construct $\mathcal{T}_i$ it takes $O(n_i t)$ messages, where n_i is the number

of nodes of color i. For all the k logical trees, this sums to $O(\sum_i n_i t) = O(nt) = O(n \min\{D, k\})$ messages.

Step 4. Node v sends the identifier l_v and its routing label $\ell(v, T_{l_v})$ to w, its closest node of color $i = h(v)$. Node w broadcasts this label to its neighbors in $\mathcal{T}_i$. Eventually, any node u of color $i = c(u) = h(v)$ will receive all such labels to construct its table C_u. Thus v contributes to $O(\min\{D, k \log k\})$ messages for the construction of C_u, the hop-distance between v and w. Then, from w, the cost of broadcasting this label is $O(n/k \cdot t)$ messages, since there are $O(n/k)$ nodes in $\mathcal{T}_i$ connected by paths of at most t edges. Therefore, to construct all the tables C_u, and to complete Step 4, we need $O(n \cdot (\min\{D, k \log k\} + n/k \cdot t)) = O(n^2/k \cdot \min\{D, k\})$ messages since $k \leqslant n$.

The variant DistRoute$'(k)$ is a slight change in Step 4 only. It consists in broadcasting from v the whole collection of routing labels $\ell(v, T_l)$, for each landmarks l, instead of only $\ell(v, T_{l_v})$. These labels are already stored by v in L_v. Then node u, combining with its own routing labels in L_u can select the best landmark tree for each v. This allows u to store an enhanced table C_u producing a stretch at most 5, according to the remark in the stretch analysis of DistRoute(k). The counterpart of this stretch improvement is that node v sends $O(n/k)$ more messages than initially. This is $O(n^3/k^2 \cdot \min\{D, k\})$ messages in total for Step 4, the previous steps being the same.

3.3 Routing Labels

We give in this part some details about routing label computation. Let us consider a shortest-path tree T of G rooted at node r. Note that in T the path between any two nodes contains $O(D)$ edges. Every node can compute a routing label of $O(\min\{D, \log n\} \cdot \log n)$ bits such that routing can be achieved using these labels and headers of the same size. Routing decisions take a constant time. We adapt an algorithm described in [16] which allows to compute in a centralized way routing labels with similar size. However, the solution proposed in [16] would have, in a distributed setting, a time complexity of $O(n)$ due to the computation of a DFS number for every node, this DFS number is part of the routing label. Since we aim at a time complexity $O(D)$, we made some changes to the routing scheme in order to avoid this DFS construction.

In order to compute its routing label $\ell(u, T)$, every node u computes its weight (its number of descendants in T), together with its *heaviest child*. These two metrics can be computed by using a *global function* as described in [22]. Once every node has computed these metrics, node r can initiate the computation of a compact path from r to every other node. A compact path is a sequence of node identifiers in which every identifier that corresponds to an heaviest child identifier is replaced by a star $*$. This computation can be achieved by broadcasting compact paths in T from r. Once a node u has calculated its compact path, namely path^*_u, it can compute locally $\ell(u, T)$ with the following algorithm.

The routing label $\ell(u, T)$ is composed of (1) the cpath_u which is path^*_u where every star sequence is replaced by its own length; (2) and a bit-set b_u that

allows to determine whether an element of cpath_u is a node identifier or a star sequence's length. An example of such a routing label is given in table 2.

path in T	$u_0 = r$	u_1	u_2	u_3	u_4	$u_5 = u$
path^*_u	u_0	u_1	*	*	u_4	*
cpath_u	u_0	u_1	2		u_4	1
b_u	1	1	0		1	0

Table 2. A simple example of $\ell(u, T)$ after computation, considering that u_1 heaviest child is u_2, u_2 heaviest child is u_3 and u_4 heaviest child is u_5.

The routing algorithm at node u with destination v is performed as follows. Node u will use $\ell(u, T)$ and $\ell(v, T)$ to determine the next-hop to v. In short, using these labels, node u can determine an approximate location of v in T. To do so, u has to find the longest matching prefix of $\ell(u, T)$ and $\ell(v, T)$. This actually requires two computations: node u has to find the longest matching prefix of the two bit-sets b_u and b_v, and the longest matching prefix of cpath_u and cpath_v. Once this is done, u can determine whether v is a descendant of u or not (note that the common ancestor of u and v can be v itself). In the latter case, u routes the packet to its parent in T. Conversely, if v is a descendant, then using the first element of the bit-set b_v, node u determines whether the next-hop to v is u's heaviest child or not:

- if it is, then node u knows its heaviest child identifier and can thus route the packet to it;
- if it is not, then the next-hop is part of $\ell(v, T)$, which is contained in the header of the packet and thus, node u can route the packet.

Thus u can route to any node v in T using only $\ell(u, T)$ and $\ell(v, T)$.

References

1. I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi, *Routing in networks with low doubling dimension*, in 26^{th} International Conference on Distributed Computing Systems (ICDCS), IEEE Computer Society Press, July 2006.
2. I. Abraham, C. Gavoille, and D. Malkhi, *On space-stretch trade-offs: Lower bounds*, in 18^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), ACM Press, July 2006, pp. 217–224.
3. I. Abraham, C. Gavoille, and D. Malkhi, *On space-stretch trade-offs: Upper bounds*, in 18^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), ACM Press, July 2006, pp. 207–216.
4. I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup, *Compact name-independent routing with minimum stretch*, ACM Transactions on Algorithms, 3 (2008), Article 37.
5. I. Abraham, C. Gavoille, D. Malkhi, and U. Wieder, *Strong-diameter decompositions of minor free graphs*, Theory of Computing Systems, 47 (2010), pp. 837–855.
6. I. Abraham and D. Malkhi, *Name independent routing for growth bounded networks*, in 17^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), ACM Press, July 2005, pp. 49–55.

7. Y. Afek and M. Ricklin, *Sparser: A paradigm for running distributed algorithms*, Journal of Algorithms, 14 (1993), pp. 316–328.
8. M. Arias, L. J. Cowen, K. A. Laing, R. Rajaraman, and O. Taka, *Compact routing with name independence*, SIAM Journal on Discrete Mathematics, 20 (2006), pp. 705–726.
9. B. Awerbuch, *Complexity of network synchronization*, Journal of the ACM, 32 (1985), pp. 804–823.
10. B. Awerbuch, A. Bar-Noy, and M. Gopal, *Approximate distributed Bellman-Ford algorithms*, IEEE Transactions on Communications, 42 (1994), pp. 2515–2519.
11. B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg, *Improved routing strategies with succinct tables*, Journal of Algorithms, 11 (1990), pp. 307–341.
12. R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins, *Searching in the plane*, Information and Computation, 106 (1993), pp. 234–252.
13. D. P. Bertsekas and R. G. Gallager, *Data Networks (2nd edition) – Chp.5: Routing in Data Networks*, Prentice Hall, 1992.
14. M. Elkin, *Computing almost shortest paths*, ACM Transactions on Algorithms, 1 (2005), pp. 283–323.
15. M. Elkin and J. Zhang, *Efficient algorithms for constructing* $(1+\epsilon, \beta)$*-spanners in the distributed and streaming models*, Distributed Computing, 18 (2006), pp. 375–385.
16. P. Fraigniaud and C. Gavoille, *Routing in trees*, in 28^{th} International Colloquium on Automata, Languages and Programming (ICALP), F. Orejas, P. G. Spirakis, and J. v. Leeuwen, eds., vol. 2076 of Lecture Notes in Computer Science, Springer, July 2001, pp. 757–772.
17. S. Haldar, *An 'all pairs shortest paths' distributed algorithm using* $2n^2$ *messages*, Journal of Algorithms, 24 (1997), pp. 20–36.
18. M.-Y. Kao, J. H. Reif, and S. R. Tate, *Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem*, Information and Computation, 131 (1996), pp. 63–79.
19. L. Kleinrock and F. Kamoun, *Hierarchical routing for large networks; performance evaluation and optimization*, Computer Networks, 1 (1977), pp. 155–174.
20. G. Konjevod, A. W. Richa, and D. Xia, *Optimal-stretch name-independent compact routing in doubling metrics*, in 25^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM Press, July 2006, pp. 198–207.
21. K. A. Laing, *Name-independent compact routing in trees*, Information Processing Letters, 103 (2007), pp. 57–60.
22. D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM Monographs on Discrete Mathematics and Applications, 2000.
23. L. Roditty, M. Thorup, and U. Zwick, *Roundtrip spanners and roundtrip routing in directed graphs*, ACM Transactions on Algorithms, 3 (2008), Article 29.
24. A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy, *Scalable routing on flat names*, in 6^{th} International Conference on emerging Networking EXperiments and Technologies (CoNEXT), ACM Press, Nov. 2010, Article No. 20.
25. M. Tang, G. Zhang, T. Lin, and J. Liu, *HDLBR: A name-independent compact routing scheme for power-law networks*, Computer Communications, 36 (2013), pp. 351–359.
26. J. N. Tsitsiklis and G. D. Stamoulis, *On the average communication complexity of asynchronous distributed algorithms*, Journal of the ACM, 42 (1995), pp. 382–400.