Optimal Scale-free Compact Routing Schemes in Networks of Low Doubling Dimension [*]

Goran Konjevod   Andréa W. Richa   Donglin Xia

Abstract

We present optimal-stretch scale-free compact routing schemes for networks of low doubling dimension, in both the name-independent and name-dependent models. Our name-independent algorithm is the first scale-free name-independent compact routing scheme to achieve asymptotically optimal stretch, closing the gaps left by the work of Abraham et al. (ICDCS'06) and Konjevod et al. (PODC'06). Our name-dependent algorithm is the first scale-free optimal-stretch name-dependent compact routing scheme that uses optimal $\lceil \log n \rceil$-bit routing labels, in spite of the limited routing label information. We define a simple hierarchical decomposition technique based on ball-packings. Our algorithms rely on a novel combination of ball-packings and hierarchical r-nets, which we see as a contribution in its own right.

1 Introduction

A routing scheme is a distributed algorithm that allows any source node to route packets to any destination node in a distributed network. A routing scheme consists of two steps: the *pre-processing*, and the *routing algorithm*. In the pre-processing step, given a network, the scheme renames each node if applicable, and configures the local routing table at each node. In the routing algorithm, given a destination's name, the source node sets up a packet header and sends the packet to one of its neighbors, based on the destination's name and the local routing table. A relay node, upon reception of a packet, decides whether the packet has reached its destination and if not, where to forward it, based on the packet header and the local routing table. The *stretch* of a routing scheme is the maximum ratio of the length of the path by which a packet is delivered, to the length of the shortest source-destination path, over all source-destination pairs. One of the fundamental trade-offs for routing schemes is between the *space* used to store the routing tables at each node (which contain all the information required for routing), and the stretch of the routing scheme. In general, one would like to keep both the storage per node and the packet header size polylogarithmic in the number of nodes, while optimizing stretch. Such a scheme is usually called a *compact routing scheme.*

The *aspect ratio* Δ of a graph is the ratio of the largest to the smallest shortest path distance in the graph. For many routing schemes [8, 24, 10, 21, 18], the routing table size at each node, the packet header size, or the routing label directly depend on a polylogarithmic function on Δ. While those schemes are compact for networks where Δ is polynomial in n, they do not scale well if Δ grows exponentially with n. Hence, one would prefer a compact routing scheme that does not directly depend on Δ. We call a routing scheme *scale-free* if the memory requirements for its routing tables, packet headers, and routing labels are independent of the aspect ratio.

In this paper, we give efficient scale-free compact routing schemes for networks whose shortest path metric has low doubling dimension, where the *doubling dimension* of a metric space is the minimum α such that any ball of radius r can be covered by at most 2^α balls of radius $r/2$.

We consider two variants of the routing problem: the *name-dependent (or labeled)* model, and the *name-independent* model. The former allows the designer of the routing scheme to rename (or label) the nodes so that they contain additional routing (e.g. topological) information. In the latter case, the routing scheme must use the (arbitrary) original naming. A labeled scheme has the advantage of embedding information in the node labels to facilitate routing, but it requires the source node to know the designer-given label of the destination node, which is not always feasible. Thus, given a name-dependent scheme, it remains an issue to determine how (and where) the source will find the label of the destination. Therefore name-independent routing schemes are preferable, especially in applications with intrinsic requirements on node names (e.g. distributed hash tables [7]), that require randomly distributed node names (e.g. Chord [23]), or that perform network operations such as locating nearby copies of replicated objects and tracking of mobile objects [7, 8].

[*]CSE Dept., Arizona State University, Tempe, AZ 85287-8809, USA. {*goran,aricha,dxia*} *at asu.edu.* Supported in part by NSF grant CCR-0209138 and NSF CAREER grant CCR-9985284.

1.1 Our Contributions. We present optimal-stretch scale-free compact routing schemes for networks of low doubling dimension in both the name-independent and name-dependent models. More precisely, our main results are Theorems 1.1 and 1.2.

THEOREM 1.1. *Given any $\epsilon \in (0,1)$ and a weighted undirected graph G with n nodes and doubling dimension α, we present a scale-free name-independent routing scheme for G with $(9+\epsilon)$-stretch, $O(\log^2 n/\log\log n)$-bit packet headers, and $\big((\frac{1}{\epsilon})^{O(\alpha)}\log^3 n\big)$-bit routing information at each node.*

THEOREM 1.2. *Given any $\epsilon \in (0,1)$ and a weighted undirected graph G with n nodes and doubling dimension α, we present a $(1+\epsilon)$-stretch labeled routing scheme for G with $\lceil\log n\rceil$-bit routing labels, $O(\log^2 n/\log\log n)$-bit packet headers, and $\big((\frac{1}{\epsilon})^{O(\alpha)}\log^3 n\big)$-bit routing information at each node.*

We define a network to have *low doubling dimension* if $\alpha = O(\log\log n)$. Hence our algorithm is the *first name-independent scale-free compact routing scheme for networks of low doubling dimension* with (asymptotically) *optimal stretch*, closing the gaps left by the results in [18] (where an optimal-stretch, but not scale-free, name-independent scheme is presented) and in [2] (where a scale-free, but not optimal-stretch, name-independent scheme is presented). By (asymptotically) optimal stretch, we mean that the stretch of our algorithm is $t+\epsilon$, for any fixed $\epsilon > 0$, where t is a lower bound on the best possible stretch for any constant doubling dimension α (in particular, $t \to 9$ as α increases). See [18] for the proof of the lower bound.

Our contributions for the name-dependent model are twofold. First, our algorithm is the first (asymptotically) optimal-stretch *name-dependent scale-free compact routing scheme for networks of low doubling dimension* that uses *optimal $\lceil\log n\rceil$-bit routing labels* (and hence embeds the minimal required amount of network-dependent routing information into the routing labels). Second, our techniques are significantly simpler than the ones used by Abraham et al. in [2], who also present an asymptotically optimal-stretch scale-free name-dependent scheme (they use $2^{O(\alpha)}\log^3 n$-bit routing labels though). Our routing scheme relies on a simple and unifying hierarchical network decomposition technique using a ball-packing, rather than the complex sparse-dense decomposition of [2].

We believe that some of the techniques introduced in this paper are a major contribution on their own. The new techniques and data structures presented in this paper enable us to go beyond the results in both [18] and [2] and obtain an optimal-stretch scale-free name-independent scheme for networks of low doubling dimension. In particular, we believe that our ball-packing decomposition, used in both the name-independent and name-dependent schemes, will have an impact on other problems that also rely on a hierarchical structure of r-nets (See Definition 2.1). In a nutshell, both types of schemes rely on a global hierarchy of r-nets. In order to avoid a dependence on Δ, we cannot store information for all $O(\log\Delta)$ layers of r-nets: We only maintain information about $O(\log n)$ layers at each node, while packing balls are used to account for the layers for which no information exists at a node.

1.2 Related Work. Not surprisingly, there has been a vast amount of research on efficient network routing schemes. General overviews are available in Peleg's book [20] and the surveys by Gavoille [14] and Gavoille and Peleg [16]. For a table summarizing the exact packet header sizes, stretch and storage requirements of most compact routing schemes for networks of low doubling dimension, please refer to [2].

An extreme solution is to keep a complete routing table at each node, providing the next hop information for shortest paths to any other node (note that routing along a shortest path results in stretch 1). However, this requires $\Theta(n\log D)$ bits at each node of degree D. In fact, Gavoille and Hanusse [15] showed that stretch-1 name-independent routing schemes require $\Omega(n\log k)$ bits memory per node for the complete bipartite graph $K_{k,n-k}$.

The trade-off between space and stretch has been further explored in other work. Awerbuch and Peleg [9] pioneered compact routing schemes, designing a name-independent scheme with stretch $O(k^2)$ and $\tilde{O}(n^{1/k}\log\Delta)$ bits of storage per node, where Δ is aspect ratio of the graph (the $\tilde{O}()$ notation denotes complexity similar to $O()$ up to polylogarithmic factors.). The stretch was improved to $O(k)$ with the same space requirement in [3]. In addition, Abraham, Gavoille and Malkhi [5] presented a scale-free name-independent routing scheme with $O(k)$ stretch, and $\tilde{O}(n^{1/k})$ routing tables, asymptotically optimal for general graphs [4].

Constant-stretch name-independent compact routing schemes do exist for restricted classes of graphs. For growth-bounded networks (a subclass of networks of constant doubling dimension), a randomized $(1+\epsilon)$-stretch compact routing scheme is known [6]. For unweighted graphs excluding fixed $K_{r,r}$ minors (including trees and planar graphs), a $(1+\epsilon)$-stretch compact scheme is presented in [1].

For networks with doubling dimension α, Konjevod, Richa and Xia [18] presented an asymptotically optimal $(9+\epsilon)$-stretch name-independent routing scheme with

$(\frac{1}{\epsilon})^{O(\alpha)} \log^2 \Delta \log n$ bits of storage per node, for any $\epsilon > 0$. They also gave a matching lower bound of $(9-\epsilon)$ on the stretch of any routing scheme for networks of low doubling dimension that uses $o(n^{(\epsilon/60)^2})$ bits of routing information per node, for any $\epsilon > 0$. Abraham et al. [2] achieved a scale-free name-independent routing scheme with (large) constant stretch and $(\frac{1}{\epsilon})^{O(\alpha)} \log^4 n$ bits of storage per node.

In the name-dependent (or labeled) model, Eilam et al. [12] achieved stretch 5 with $\tilde{O}(n^{1/2})$ memory and $O(\log n)$-bit labels, while Cowen [11] proposed a stretch 3 labeled routing scheme with $\tilde{O}(n^{2/3})$ memory and $O(\log n)$-bit labels. Furthermore, Thorup and Zwick [26] achieve stretch $2k-1$ using $\tilde{O}(n^{1/k})$ routing tables and $O(k \log^2 n)$-bit labels. For trees, optimal stretch 1 labeled routing schemes with $O(\log^2 n / \log \log n)$ bits of label, packet header and memory were presented in [13, 26]. There are $(1+\epsilon)$-stretch labeled routing schemes with polylogarithmic space for planar graphs [25], graphs excluding a fixed minor [1], and networks of low doubling dimension [22, 2].

Recently, a scale-free constant-stretch name-independent routing scheme with $2^{O(\alpha)} \log^4 n$ memory and $2^{O(\alpha)} \log^3 n$ packet headers was proposed by Abraham et al. [2] for networks of low doubling dimension; their stretch factor, albeit being constant, is very large and of limited pratical interest. They also present a scale-free $(1+\epsilon)$-stretch labeled scheme with $(\frac{1}{\epsilon})^{O(\alpha)} \log^4 n$-bit memory, and $(\frac{1}{\epsilon})^{O(\alpha)} \log^3 n$-bit label and header. Both their schemes rely on a sparse-dense decomposition technique, which differentiates dense and sparse regions of the network. Intuitively, they maintain two sets of routing schemes, one for dense and one for sparse regions, that are applied alternatively depending on the density of the region of the network considered. The sparse-dense decomposition technique is rather involved and does not yield good stretch factors for the name-independent case, nor does it yield optimal stretch if we use optimal-size routing labels in the name-dependent case. In this paper, we define a new unifying and simple $O(\log n)$-level hierarchical decomposition technique based on ball packing, which "efficiently" covers dense and sparse regions of the network alike. We carefully design data structures to combine the two hierarchical network decompositions used by our algorithms, namely r-nets and ball packings, in order to eliminate the storage dependence on Δ. The combined hierarchies also eliminate the need for differentiated treatment of sparse and dense regions, allowing for much simpler routing algorithms, and thus optimal stretch in the name-independent case, and optimal routing label size in the name-dependent case.

2 Preliminaries

Let $G = (V, E)$ be a connected, edge-weighted, undirected graph with n nodes, shortest-path metric d, aspect ratio Δ, and doubling dimension α. Let $B_u(r)$ be the closed ball of radius r around node u in G, i.e. $B_u(r) = \{x \in V \mid d(u,x) \leq r\}$, for any $u \in V$ and $r > 0$. Let $r(u,j)$ be the radius such that $|B_u(r(u,j))| = 2^j$, for any node $u \in V$ and $j \in [\log n]$, where $[x]$ denotes the set $\{0, \cdots, x-1\}$.

Definition 2.1. (r-net) *An r-net of a metric space (V,d) is a subset $Y \subseteq V$ such that any point in V is at distance at most r from Y, and any two points in Y are within distance at least r.*

For any finite metric it is easy to show that such an *r-net* exists and can be constructed greedily. The following is a well-known result about r-nets:

Lemma 2.1. ([17]) *Let Y be an r-net of (V,d). For any $u \in V$ and $r' \geq r$, we have $|B_u(r') \cap Y| \leq \left(\frac{4r'}{r}\right)^{\alpha}$.*

Let $M = \log \Delta + 1$, and for each $i \in [M]$, let Y_i be a 2^i-net of G. W.l.o.g. assume $\min_{u \neq v} d(u,v) = 1$, thereby $\Delta = \max d(u,v)$. Thus we have $Y_0 = V$ and $|Y_{\log \Delta}| = 1$. By connecting each node $u \in Y_i$ to its nearest node in Y_{i+1} for $0 \leq i < \log \Delta$, we naturally define a tree, called the *netting tree* of r-nets $\{Y_i\}$ and denoted by $T(\{Y_i\})$ (Note that some nodes of V may appear multiple times in the *netting tree*). For each $u \in V$, let $\{z(u,i) \in Y_i : i \in [M]\}$ be a sequence of nodes from the leaf $u = z(u,0)$ to the root $z(u, \log \Delta)$ in the netting tree $T(\{Y_i\})$; we call this sequence the *zooming sequence* of u.

Let $N = \log n$, and for each $j \in [N]$, let $\mathscr{B}_j$ be a ball packing as defined in Lemma 2.2.

Lemma 2.2. (Packing Lemma) *For any $j \in [N]$, there exists a ball packing $\mathscr{B}_j$ of G, i.e. a (maximal) set of non-intersecting balls, such that*

1. *For any ball $B \in \mathscr{B}_j$, $|B| = 2^j$.*
2. *For any node u, there exists a ball $B \in \mathscr{B}_j$ centered at c such that the radius of B is at most $r(u,j)$ — i.e. $r(c,j) \leq r(u,j)$ — and $d(u,c) \leq 2r(u,j)$.*

Proof: Consider the set of balls $\{B_u(r(u,j)) \mid u \in V\}$. Greedily select balls from this set in the order of shortest radius to longest to form a maximal set of non-intersecting balls $\mathscr{B}_j$.

First, such a ball packing $\mathscr{B}_j$ has Property (1), since $|B_u(r(u,j))| = 2^j$ for any node $u \in V$. Second, for any node $u \in V$, if $B_u(r(u,j)) \in \mathscr{B}_j$, Property (2) is trivially satisfied. Otherwise, $B_u(r(u,j))$ intersects some $B \in \mathscr{B}_j$ with center c. The radius of B is at most $r(u,j)$, i.e. $r(c,j) \leq r(u,j)$, since the balls are selected from $\mathscr{B}_j$ by increasing radius. Moreover, since B and $B_u(r(u,j))$ intersect and $r(c,j) \leq r(u,j)$, we have $d(u,c) \leq 2r(u,j)$. Thus Property (2) follows. ■

3 Overview

In this section, we give a brief overview of our routing schemes. Our scale-free name-independent routing scheme uses our scale-free name-dependent routing scheme of Theorem 1.2. Thus every time a node u wants to communicate with a node v, u first uses v's name to retrieve the label of v and then routes to v using the labeled routing scheme. Hence the main merit of our scale-free name-independent routing scheme is an efficient method for distributed storage and retrieval of node labels from their names.

3.1 Ball packings for label storage. First, for each $j \leq \log n$ we maintain a packing B_j of balls of size 2^j, as described in Section 2. For each ball $B \in \mathscr{B}_j$, we maintain a *search tree* on B to store the labels of nodes in $B_c(r(c,j+2))$ (where c is the center of B) according to their names. Second, for each $i \leq \log \Delta + 1$, we maintain a 2^i-net Y_i, and a single *netting tree* (defined in Section 2) that stores all the necessary information to route along the nets when possible. Finally, to link the ball packings with the netting tree, each node $u \in Y_i$ keeps the labels of nodes in the ball $B_u(2^i f)$, for a constant f depending on ϵ and α. If there is a packing ball $B \in \mathscr{B}_j$ with center c for some j, such that B is roughly contained in $B_u(2^i f)$ and the search tree on B has already stored the labels of nodes in $B_u(2^i f)$, then u just makes use of the search tree on B to index the labels of nodes in $B_u(2^i f)$ by maintaining a link to the center c, instead of maintaining a search tree on ball $B_u(2^i f)$. Otherwise, we maintain a search tree on $B_u(2^i f)$ to store the labels of nodes in $B_u(2^i f)$. In this way, we avoid storing too many search trees, and achieve scale-free storage. Thus for both cases, we can define a local search procedure at u based on the search tree to find labels of nodes in $B_u(2^i f)$. Finally, the routing algorithm finds the label of the destination node just by recursively calling a local search procedure along the *zooming sequence* of the source node.

3.2 Labeled routing. For our scale-free labeled routing scheme, we define the labels through an enumeration of the leaf set of the netting tree in a depth-first-search traversal, producing labels of $\lceil \log n \rceil$ bits. Moreover, for each node $x \in Y_i$ of the netting tree, we have a label range $Range(x,i)$ of nodes in the leaf set of the subtree rooted at x. Assume we want to route from source u to destination v given the label $l(v)$. If u keeps the range information of nodes in $B_u(2^{i+f}) \cap Y_i$ (the i^{th} ring of u), and there exists $x \in B_u(2^{i+f}) \cap Y_i$ with $l(v) \in Range(x,i)$ and $d(u,x) + d(x,v)$ is roughly equal to $d(u,v)$, then for the minimal such index i, we deliver the packet to the next hop on the shortest path from u to x, and recurse. However, in order to achieve scale-free storage, we only maintain range information for $O(\log n)$ rings at each node. On the other hand, for each $j \leq \log n$ and each packing ball $B \in \mathscr{B}_j$ with center c, we maintain a labeled tree-routing scheme (see Lemma 5.1) for the shortest-path spanning tree on the Voronoi region of c in the Voronoi diagram of centers of balls in $\mathscr{B}_j$. In addition, we use the search tree technique to store the local tree-routing labels and retrieve them according to the global labels. Thus whenever we cannot route to the destination because of lack of some ring information at the current node, we can retrieve the local tree label of v and use the local labeled tree-routing scheme to deliver the packet to the destination.

4 A Scale-Free Name-Independent Routing Scheme

4.1 Data Structures. Our scale-free name-independent routing scheme will use our scale-free name-dependent routing scheme given by Theorem 1.2 and described in Section 5 as the effective underlying labeled routing scheme. For any $u \in V$, let $n(u)$ denote the arbitrary original name, and let $l(u)$ denote the label given by the underlying labeled routing scheme. Thus every time u wants to communicate with a node v given by its name $n(v)$, u uses $n(v)$ to retrieve the label $l(v)$ and then routes to v using the underlying labeled scheme.

In Section 4.1.1, we define a search tree for a ball, and provide procedures to store and retrieve *(key, data)* pairs, where for our name-independent scheme we take the original node name as the *key* and the node label as the *data*. In Section 4.1.2, we define the hierarchical data structures to maintain search trees.

4.1.1 Search Tree

Definition 4.1. (Search Tree) *For any $\epsilon' \in (0,1)$ and any ball $B_c(r)$, let $U_0 = \{c\}$, and for $1 \leq i \leq \lfloor \log(\epsilon' r) \rfloor$ let U_i be a $2^{\lfloor \log(\epsilon' r) \rfloor - i}$-net of $B_c(r) \setminus \bigcup_{0 \leq j < i} U_j$. Then the search tree on $B_c(r)$, denoted by $T(c,r)$, is formed by connecting each node $v \in U_i$ to its closest node in U_{i-1} for $0 < i \leq \lfloor \log(\epsilon' r) \rfloor$, and defining the weight on each edge (u,v) equal to $d(u,v)$ in G.*

From the definition of r-net, we can derive the following bound on the height of the search tree $T(c,r)$:

$$r + \sum_{i=1}^{\lfloor \log(\epsilon' r) \rfloor} 2^{\lfloor \log(\epsilon' r) \rfloor - i} \le (1+\epsilon')r. \tag{4.1}$$

Let the two endpoints of each virtual edge in the search tree keep each other's routing label, so that they can communicate using the underlying labeled scheme. Note that by Lemma 2.1 the root has the maximum degree in the tree, $\left(\frac{4r}{2^{\lfloor \log(\epsilon' r) \rfloor - 1}}\right)^{\alpha} = \left(\frac{1}{\epsilon'}\right)^{O(\alpha)}$. Hence each node keeps $\left(\frac{1}{\epsilon'}\right)^{O(\alpha)}$ labels for the search tree.

Next, given a search tree $T(c, r)$ with m nodes, we show how to store k *(key, data)* pairs in the search tree:

Algorithm 1 Store data in the search tree $T(c, r)$

```
1: sort all pairs according to their keys into a list
2: for each visited node during a depth-first traversal of $T(c,r)$ do
3:     pick $k/m$ new pairs from the list of Step 1, and store them at the current node
4: end for
5: Each node $v$ in $T(c,r)$ stores the range of keys of the pairs stored in $v$ and its descendants in $T(c,r)$; furthermore, $v$ stores the range information of all its children.
```

Finally, we define a procedure that, given a *key*, retrieves the corresponding *data*, or reports an error if such a pair *(key, data)* is not stored in $T(c, r)$.

Algorithm 2 *SearchTree*$(key, T(c, r))$

```
1: $u \leftarrow c$
2: while there exists a child $u'$ of $u$ in $T(c,r)$ such that the range of $u'$ contains key do
3:     go to node $u'$ and $u \leftarrow u'$
4: end while
5: if $u$ stores the data corresponding to the given key then
6:     report the data
7: else
8:     report error: there is no pair with key equal to the given key in $T(c,r)$
9: end if
10: go back from $u$ to $c$ along the tree $T(c,r)$
```

4.1.2 Hierarchical Search Structures

Let Y_{M-1} be an arbitrary 2^{M-1}-net of G, that is, a singleton. Recursively construct a 2^i-net Y_i by greedily expanding Y_{i+1} with nodes to obtain a 2^i-net, for i from $M-2$ to 0. Thus, we have $Y_{M-1} \subseteq Y_{M-2} \subseteq \cdots \subseteq Y_0$. Consider the *netting tree* $T(\{Y_i\})$. For each virtual edge (u, v) of the tree with $u \in Y_i$, $v \in Y_{i+1}$, and $u \neq v$, let u store the label $l(v)$. Since $u \notin Y_{i+1}$ (otherwise $u = v$), we have $u \notin Y_j$ for any $j > i$ and $u \in Y_j$ for any $j \le i$. Thus each node $u \in V$ stores at most one label of its parent in the netting tree. Now each node $u \in V$ can route packets along its *zooming sequence* $\{z(u, i)\}$ to the root.

Let $\mathscr{B}_j$ be a ball packing defined as in Lemma 2.2, for each $j \in [N]$, and let $f(\epsilon) = \frac{80}{\epsilon} + 2$. We maintain search trees for two types of balls:

1. $B \in \mathscr{B}_i$ with center c, for $i \in [N]$. The search tree for B stores the pair $(n(v), l(v))$ for each node v in the ball $B_c(r(c, i+2))$.
2. Any ball $B_u(2^i f(\epsilon))$, for $i \in [M]$ and any $u \in Y_i$, except those balls such that $B_u(2^i(f(\epsilon)+1))$ contains a ball $B \in \mathscr{B}_j$ with center c for some $j \in [N]$ and $B_u(2^i f(\epsilon))$ is contained in the ball $B_c(r(c, j+2))$. The search tree for $B_u(2^i f(\epsilon))$ stores the pair $(n(v), l(v))$ for each node v in the same ball.

We denote by $\mathscr{B}$ the collection of balls of the first type, i.e. $\mathscr{B} = \bigcup_{j=0}^{N} \mathscr{B}_j$, and by $\mathscr{B}'$ the collection of balls of the second type. Let $S(u)$ be the set of indexes i such that $B_u(2^i f(\epsilon)) \notin \mathscr{B}'$, i.e. $S(u) = \{i \in [M] : B_u(2^i f(\epsilon)) \notin \mathscr{B}'\}$.

Finally, for $i \in S(u)$, i.e. $B_u(2^i f(\epsilon)) \notin \mathscr{B}'$, there exists a ball $B \in \mathscr{B}_j$ with center c for some $j \in [N]$ such that $B \subseteq B_u(2^i(f(\epsilon)+1))$ and $B_u(2^i f(\epsilon)) \subseteq B_c(r(c, j+2))$. Without loss of generality, assume such j and then $d(u, c)$ are both minimal, and let $H(u, i)$ denote the ball B. Let u maintain a link to c by storing its routing label $l(c)$.

Therefore for any $i \in [M]$ and any $u \in Y_i$, we can index the routing labels of nodes in $B_u(2^i f(\epsilon))$ either by the search tree for $B_u(2^i f(\epsilon))$ if $B_u(2^i f(\epsilon)) \in \mathscr{B}'$, or by the search tree for the ball $H(u, i)$ if $B_u(2^i f(\epsilon)) \notin \mathscr{B}'$, i.e. $i \in S(u)$. Thus we define a search procedure for any $i \in [M]$ and any $u \in Y_i$ as follows:

Algorithm 3 *Search*$(name, u, i)$

```
1: if $B_u(2^i f(\epsilon)) \in \mathscr{B}'$ then
2:     call SearchTree$(name, T(u, 2^i f(\epsilon)))$;
3: else
4:     let $c$ be the center of $H(u,i)$, and $r$ be the radius;
5:     go to $c$ from $u$ by the labeled scheme;
6:     call SearchTree$(name, T(c,r))$;
7:     go back from $c$ to $u$.
8: end if
```

Note that setting $\epsilon' = \frac{1}{f(\epsilon)+1}$ from Equation 4.1 we have that the cost of *Search*$(name, u, i)$ is at most

$$2 \cdot (1+\epsilon') \cdot 2^i(f(\epsilon)+1) = 2^{i+1}(f(\epsilon)+2), \tag{4.2}$$

where we ignore the cost due to the stretch of the underlying labeled scheme–this will be considered later.

The following two lemmas show the storage at each node is scale-free.

Lemma 4.1. *For any $u \in V$, the number of balls $H(u,i)$ for all $i \in S(u)$ is at most $4 \log n$.*

Proof: We show that for any $j \in [N]$, the number of different balls $H(u,i) \in \mathscr{B}_j$ with $i \in S(u)$ is at most 4.

By contradiction, assume there is an index $j \in [N]$ such that the number of different balls $H(u,i) \in \mathscr{B}_j$ with $i \in S(u)$ is at least 5. Let $i_0 < i_1 < i_2 < i_3 < i_4$ be five of these indices. By definition of $H(u, i_k)$, we have $H(u,i_k) \subseteq B_u(2^{i_k}(f(\epsilon)+1))$. Since $i_k < i_4$ for $k < 4$, we have $B_u(2^{i_k}(f(\epsilon)+1)) \subset B_u(2^{i_4} f(\epsilon))$. Thus $B_u(2^{i_4} f(\epsilon))$ contains balls $H(u,i_k)$ for $k = 0,\ldots,3$. Since all balls $H(u,i_k)$ in $\mathscr{B}_j$ are non-intersecting, we have $|B_u(2^{i_4} f(\epsilon))| > 2^{j+2}$. Thus $B_u(2^{i_4} f(\epsilon)) \nsubseteq B_c(r(c,j+2))$, where c is the center of $H(u,i_4)$. This contradicts the definition of $H(u,i_4)$. ■

Lemma 4.2. *For any $v \in V$, the number of search trees that contain v is at most $(\frac{1}{\epsilon})^{O(\alpha)} \log n$.*

Proof: First, for each $j \in [N]$, the packing balls in $\mathscr{B}_j$ are disjoint, and so at most one of them contains v. Thus the number of search trees for balls in $\mathscr{B}$ that contain v is no more than $\log n$.

Second, consider the search trees for balls in $\mathscr{B}'$. Let $b = \lceil \log(2f(\epsilon)+1) \rceil$. For every $v \in V$, define a sequence of indexes $R(v) = \{i \in [M] : |B_v(2^{i+b})| \geq 2|B_v(2^{i-2})|\}$. The following claim, whose proof we defer to the full version of the paper [19], bounds the size of $R(v)$.

Claim 4.1. $|R(v)| \leq (2+b) \log n$.

The next claim relates $R(v)$ to the search trees for balls in B' that contain v, thereby bounding the number of such trees.

Claim 4.2. *If there is a ball $B_u(2^i f(\epsilon)) \in \mathscr{B}'$ with $u \in Y_i$ that contains v, then $i \in R(v)$.*

For each $i \in R(v)$, the number of $u \in Y_i$ such that $v \in B_u(2^i f(\epsilon))$ is at most $(4f(\epsilon))^\alpha$ by Lemma 2.1. Now by Claim 4.1 and 4.2, the number of search trees for balls in $\mathscr{B}'$ that contain v is at most $(2+b)\log n \cdot (4f(\epsilon))^\alpha = (\frac{1}{\epsilon})^{O(\alpha)} \log n$. The lemma follows. ■

Proof of Claim 4.2: Let j be the index such that $2^j \leq |B_v(2^{i-2})| < 2^{j+1}$. Thus $r(v,j) \leq 2^{i-2}$. We will show that $|B_v(2^{i+b})| \geq 2^{j+2} > 2|B_v(2^{i-2})|$, and therefore $i \in R(v)$.

By Lemma 2.2, there exists a ball $B \in \mathscr{B}_j$ with center c such that $r(c,j) \leq r(v,j) \leq 2^{i-2}$ and $d(v,c) \leq 2r(v,j) \leq 2^{i-1}$. Since $d(u,v) \leq 2^i f(\epsilon)$, we have $d(u,c) + r(c,j) \leq d(u,v) + d(v,c) + r(c,j) \leq 2^i(f(\epsilon)+1)$. Thus $B \subseteq B_u(2^i(f(\epsilon)+1))$. Since $B_u(2^i f(\epsilon)) \in \mathscr{B}'$, we have $B_u(2^i f(\epsilon)) \nsubseteq B_c(r(c,j+2)$ (otherwise we use the search tree on B, instead of maintaining a search tree for $B_u(2^i f(\epsilon))$). Hence $r(c,j+2) \leq d(u,c) + 2^i f(\epsilon)$. Thus $d(v,c) + r(c,j+2) \leq 2^{i-1} + \big(d(u,v) + d(v,c) + 2^i f(\epsilon)\big) \leq 2^{i+b}$, since $b = \lceil \log(2f(\epsilon)+1) \rceil$. Therefore $B_c(r(c,j+2)) \subseteq B_v(2^{i+b})$. Hence $|B_v(2^{i+b})| \geq 2^{j+2} > 2|B_v(2^{i-2})|$. The claim follows. ■

Lemma 4.3. *The routing information at each node has $(\frac{1}{\epsilon})^{O(\alpha)} \log^3 n$ bits.*

Proof: By Theorem 1.2, the underlying labeled routing scheme requires $(\frac{1}{\epsilon})^{O(\alpha)} \log^3 n$ bits at each node.

Each node maintains at most one label of its parent node in the *netting tree*.

By Lemma 4.1, the total storage for maintaining links to the centers of balls $H(u,i)$, for all $i \in S(u)$, is $O(\log^2 n)$ bits.

The storage required to maintain search trees and store routing labels in them can be bounded as follows. Since the maximum degree in any search tree is $(\frac{1}{\epsilon'})^{O(\alpha)} = (\frac{1}{\epsilon})^{O(\alpha)}$, and the size of each range and routing label is $O(\log n)$, each node in a search tree maintains $(\frac{1}{\epsilon})^{O(\alpha)} \log n$ bits of range information and link information. Since each node in a search tree stores at most 4 *(name, label)* pairs, it stores $O(\log n)$ bits of data. Since the number of search trees containing any node is at most $(\frac{1}{\epsilon})^{O(\alpha)} \log n$ by Lemma 4.2, the total storage for maintaining search trees and routing labels at each node is no more than $(\frac{1}{\epsilon})^{O(\alpha)} \log^2 n$.

Thus each node stores $(\frac{1}{\epsilon})^{O(\alpha)} \log^3 n$ bits of routing information. ■

4.2 Routing Algorithm

Now we are ready to describe the scale-free name-independent routing scheme, and prove its performance bounds.

The routing procedure is described in Algorithm 4. Assume that a source node u wants to send a message to a destination node v, given the name $n(v)$ of v.

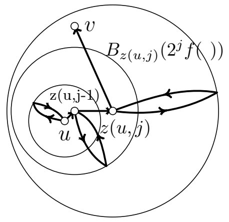

Figure 1: Name-Independent Routing Scheme

Algorithm 4 Name-Independent Routing

```
1: $i \leftarrow 0$
2: **repeat**
3:     go to $z(u,i)$ using the underlying routing scheme
4:     perform a local search at $z(u,i)$ by calling the procedure $Search(n(v), z(u,i), i)$
5:     $i \leftarrow i+1$
6: **until** the routing label $l(v)$ of $v$ is found
7: go to $v$ from $z(u,i)$ using the underlying labeled routing scheme.
```

Figure 1 illustrates an execution of Algorithm 4. The procedure $Search(n(v), z(u,i), i)$ searches for the label of v in the ball $B_{z(u,i)}(2^i f(\epsilon))$ repeatedly, until at level j it finds the routing label of v in the ball $B_{z(u,j)}(2^j f(\epsilon))$. Then we call the underlying labeled routing scheme to route to v from $z(u,j)$. Fix the $(1+\epsilon/20)$-stretch required of the underlying labeled scheme. Then we have the following stretch lemma.

Lemma 4.4. (Stretch) *For any source node u and any destination node v in G, the total routing cost of our algorithm is no more than $(9+\epsilon)d(u,v)$.*

Proof: For simplicity, we factor in the $(1+\epsilon/20)$ stretch of the underlying labeled scheme at the end of the proof. As illustrated in Figure 1, let j be the index of the level at which v's routing label is found. Thus the routing cost from u to v consists of $\sum_{i=1}^{j} d(z(u,i-1), z(u,i))$ for the cost along the zooming sequence, $\sum_{i=0}^{j} 2^{i+1}(f(\epsilon)+2)$ for the search procedures, and $d(z(u,j),v)$ for the cost from $z(u,j)$ to v. Since $\sum_{i=1}^{j} d(z(u,i-1), z(u,i)) \le 2^{j+1}$ by the definition of r-nets and $d(z(u,j),v) \le d(u,v) + \sum_{i=1}^{j} d(z(u,i-1), z(u,i)) \le d(u,v)+2^{j+1}$ by triangle inequality, we find that the total cost is at most $\sum_{i=1}^{j} d(z(u,i-1), z(u,i)) + \sum_{i=0}^{j} 2^{i+1}(f(\epsilon)+2) + d(z(u,j),v) \le 2^{j+2}(f(\epsilon)+3) + d(u,v)$.

Since v's routing label is not found by $Search(n(v), z(u,j-1), j-1)$, we have $d(z(u,j-1),v) > f(\epsilon)2^{j-1}$. Then by triangle inequality, we have

$$\begin{aligned} d(u,v) &\ge d(z(u,j-1),v) - d(z(u,j-1),u) \\ &\ge d(z(u,j-1),v) - \sum_{i=1}^{j-1} d(z(u,i-1), z(u,i)) \\ &\ge 2^{j-1}(f(\epsilon)-2) \end{aligned}$$

and thus,

$$\begin{aligned} 2^{j+2}(f(\epsilon)+3) + d(u,v) &\le \frac{8(f(\epsilon)+3)}{f(\epsilon)-2} d(u,v) + d(u,v) \\ &= \left(9 + \frac{40}{f(\epsilon)-2}\right) d(u,v). \end{aligned}$$

Since $f(\epsilon) = \frac{80}{\epsilon} + 2$, the total routing cost together with the $(1+\epsilon/20)$ of the underlying labeled scheme is no more than

$$\begin{aligned} &\left(9 + \frac{40}{f(\epsilon)-2}\right) d(u,v)(1+\epsilon/20) \\ &= \left(9 + \frac{\epsilon}{2}\right)(1+\epsilon/20)\, d(u,v) \\ &\le (9+\epsilon) d(u,v). \end{aligned}$$

■

Proof of Theorem 1.1: The bounds on stretch and storage requirement at the nodes follow from Lemma 4.4 and 4.3, respectively. The packet header size of our name-independent routing scheme is dominated by the packet header size of the underlying labeled scheme, i.e. $O(\log^2 n / \log\log n)$ bits, by Theorem 1.2. ■

5 A Scale-Free Labeled Routing Scheme

5.1 Data Structures

In this section, we define the data structures needed for our scale-free labeled routing scheme.

First, we have two hierarchical data structures as defined in Section 2: r-nets $\{Y_i : i \in [M]\}$ and ball packings $\{\mathscr{B}_j : j \in [N]\}$.

Second, let $l : V \to [n]$ be the enumeration of the leaves in a depth-first traversal of the *netting tree* $T(\{Y_i\})$. Since the leaf set of $T(\{Y_i\})$ is $Y_0 = V$, we take l as the label function for our scale-free labeled routing scheme. For each $i \in [M]$, and each node $u \in Y_i$, let $Range(u,i) \in [n] \times [n]$ be the range of labels of nodes v such that $z(v,i) = u$, i.e. each v is a leaf of the subtree rooted at u in $T(\{Y_i\})$. Observe that $Range(u,i)$ requires $2\lceil \log n \rceil$ bits, and by the properties of depth-first-search, we have $l(v) \in Range(u,i)$ iff $u = z(v,i)$.

Third, let the i^{th} ring of u be the node set $X_i(u) = B_u(2^{i+f}) \cap Y_i$, for $f = 8 + \log\frac{1}{\epsilon}$, and $R(u) = \{i \in [M] : \exists j \in [N], r(u,j) \cdot 2^{-a} \le 2^i \le r(u,j)\}$, for $a = 12 + \log\frac{1}{\epsilon}$. Then each node u stores the range information $Range(x,i)$ for nodes $x \in X_i(u)$ and $i \in R(u)$, and the $\log n$-bit information to identify which neighbor of u is on the shortest path from u to x. Note that $|R(u)| = O(\frac{\log n}{\epsilon})$, and by Lemma 2.1, we have $|X_i(u)| = (1/\epsilon)^{O(\alpha)}$. Thus the range information stored at each node is $(1/\epsilon)^{O(\alpha)} \log^2 n$ bits.

Finally, for each $j \in [N]$ and each $B \in \mathscr{B}_j$ with center c, let $V(c,j)$ be the Voronoi region of c in the Voronoi diagram of centers of balls in $\mathscr{B}_j$, i.e. $V(c,j) = \{u \in V : d(u,c) \le d(u,c'),$ for the center c' of any ball in $\mathscr{B}_j\}$, and let $T_c(j)$ be a shortest path tree rooted at c and spanning $V(c,j)$. For each tree $T_c(j)$, we maintain a labeled routing scheme as follows, and let $l(v;c,j)$ denote the local routing label of $v \in T_c(j)$:

LEMMA 5.1. ([13, 26]) *For every weighted tree T on n nodes, there exists a labeled routing scheme that, given any destination label, routes optimally on T from any source to the destination. The storage per node, the label size, and header size are $O(\log^2 n/\log\log n)$ bits.*

For each $j \in [N]$, each node $u \in V$ stores the local routing label $l(c; c, j)$ of the center c such that c is the center of a ball $B \in \mathscr{B}_j$ and $u \in V(c, j)$. Note that by Voronoi diagram properties, for each fixed j, the trees $T_c(j)$ are disjoint. Thus the local routing label information at each node is $O(\log^3 n/\log\log n)$ bits.

In addition, we build a search tree $T'(c, r(c, j))$ as in Definition 5.1 to store the $(key, data)$ pairs of nodes $v \in T_c(j) \cap B_c(r(c, j+1))$, where the *key* is the global routing label $l(v)$, and *data* is the local routing label $l(v; c, j)$ of v in the tree $T_c(j)$. Thus given a *key*, i.e. $l(v)$, the $SearchTree(l(v), T'(c, r(c, j)))$ procedure, as defined in Section 4.1.1, retrieves the label $l(v; c, j)$ of v along the shortest path of the search tree.

DEFINITION 5.1. (SEARCH TREE II) *For any ball $B_c(r)$, the search tree II, denoted by $T'(c, r)$, is modified from the search tree $T(c, r)$ given by Definition 4.1, where $\epsilon' = 1/2$:*

(i) Instead of building the tree by iterating i from 1 to $\lfloor \log \frac{r}{2} \rfloor$, we iterate only to $\min(\lceil 2\log n\rceil, \lfloor \log \frac{r}{2} \rfloor)$, and define the weight on each of these edges (u, v) equal to $d(u, v)$ in G.

(ii) If $\lceil 2\log n\rceil < \lfloor \log \frac{r}{2} \rfloor$, for any $u \in U_{\lceil 2\log n\rceil}$, let $V(u)$ be the Voronoi region of u in the Voronoi diagram of a set of sites $U_{\lceil 2\log n\rceil}$ in $B_c(r)$, i.e. $V(u) = \{x \in B_c(r) : d(x, u) \le d(x, u'),$ for any $u' \in U_{\lceil 2\log n\rceil}\}$. Link the nodes in $V(u) \setminus \bigcup_{0\le j\le\lceil 2\log n\rceil} U_j$ into a path, connect it to u and define the weight on these edges equal to $\frac{r}{n^2}$, for each $u \in U_{\lceil 2\log n\rceil}$.

Note that the height of the search tree II $T'(c, r)$ is at most $(1 + \frac{1}{2})r + \frac{r}{n^2} \cdot n < 2r$. The following lemma shows how to link the endpoints of each virtual edge.

LEMMA 5.2. *We can deliver packets along each virtual edge of any search tree II $T'(c, r)$, with cost at most the weight of the edge, by maintaining $(1/2)^{O(\alpha)} \log^2 n$ bits of data per node.*

Proof: First, for any virtual edge (u, v), where $u \in U_{i-1}$, $v \in U_i$ and $0 < i \le \min(\lceil 2\log n\rceil, \lfloor \log \frac{r}{2} \rfloor)$, let each node x on the shortest path from u to v maintain the next hop information. Then u and v can communicate along the shortest path. Now we bound the memory requirement. Since u is the closest node in U_{i-1} to v and x is on the shortest path between u and v, u is also the closest node in U_{i-1} to x. By Lemma 2.1, the number of nodes $v \in U_i$ whose closest node in U_{i-1} is u is 8^α. Since $0 < i \le \min(\lceil 2\log n\rceil, \lfloor \log \frac{r}{2} \rfloor)$ and each next hop information has size no more than $\log n$, each node in $B_c(r)$ requires at most $(1/2)^{O(\alpha)} \log^2 n$ bits of the next hop information.

Second, let a virtual edge link u and each node in $V(u) \setminus \bigcup_{0\le j\le\lceil 2\log n\rceil} U_j$ for each $u \in U_{\lceil 2\log n\rceil}$. Let $T(u)$ be a shortest path tree rooted at u and spanning $V(u)$, for each $u \in U_{\lceil 2\log n\rceil}$. We maintain a local labeled routing scheme given by Lemma 5.1 for the tree $T(u)$. Let the two endpoints of each of these virtual edges keep each other's local label. Since $d(u, v) \le 2^{\lceil \log(r/2)\rceil - \lceil 2\log n\rceil} \le \frac{r}{2n^2}$ for any $v \in V(u) \setminus \bigcup_{0\le j\le\lceil 2\log n\rceil} U_j$, the routing cost along each of these virtual edges is at most $\frac{r}{n^2}$. By the Voronoi diagram properties, the trees $T(u)$ for all nodes $u \in \lceil 2\log n\rceil$ are disjoint. Thus by Lemma 5.1, each node in $B_c(r)$ maintains $O(\log^2 n/\log\log n)$ bits of routing information for the local labeled routing. ■

LEMMA 5.3. (STORAGE) *The routing information at each node is at most $(\frac{1}{\epsilon})^{O(\alpha)} \log^3 n$ bits.*

Proof: Each node maintains $(\frac{1}{\epsilon})^{O(\alpha)} \log^2 n$-bit range information, $O(\log^3 n/\log\log n)$-bit local routing label information, and $(\frac{1}{2})^{O(\alpha)} \log^3 n$-bit data structures and $O(\log^3 n/\log\log n)$-bit data storage for search trees. Hence the routing information at each node is at most $(\frac{1}{\epsilon})^{O(\alpha)} \log^3 n$ bits. ■

5.2 Routing Algorithm Assume that a source node u wants to send a packet to a destination node v given its label $l(v)$. The routing procedure is defined in Algorithm 5.

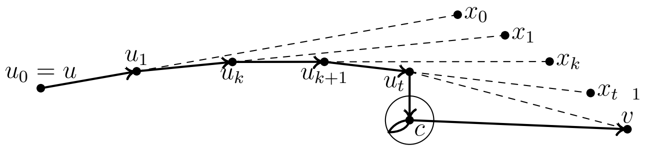

Figure 2: Labeled Routing Scheme

Figure 2 illustrates the routing path from u to v, which consists of the path $u_0 \to u_1 \to \cdots \to u_t$, then the routing path from u_t to c, the search trail of $SearchTree()$ in the ball $B_c(r(c, j))$, and the routing path from c to v. Then the following two lemmas guarantee that the $SearchTree()$ procedure in the ball $B_c(r(c, j))$ retrieves the local label $l(v; c, j)$ sucessfully.

LEMMA 5.4. *Let $t \in [M]$ and $j \in [N]$ be defined as in line 10. Then $\frac{64}{\epsilon} r(u_t, j) < d(u_t, v) < \frac{r(u_t, j+1)}{8}$.*

Proof: Omitted due to page limit. See the full version of the paper [19]. ■

Algorithm 5 Labeled Routing Scheme

```
1: $u_0 \leftarrow u$, and $i_{-1} \leftarrow +\infty$
2: for $k = 0$ to $+\infty$ do
3:   $i_k \leftarrow$ the minimal index in $R(u_k)$ such that there exists $x_k \in X_{i_k}(u_k)$ with $l(v) \in Range(x_k, i_k)$, i.e. $x_k = z(v, i_k)$
4:   if $i_k \leq i_{k-1}$ and $d(u_k, x_k) \geq 2^{i_k+f-1} - 2^{i_k}$ then
5:     $u_{k+1} \leftarrow$ the next hop along the shortest path from $u_k$ to $x_k$, and go to $u_{k+1}$
6:   else
7:     break
8:   end if
9: end for
10: $t \leftarrow k$. $j \leftarrow$ the index in $[N]$ such that $r(u_t, j) \leq 2^{i_t} < r(u_t, j+1)$, and $c \leftarrow$ the center of a ball $B \in \mathscr{B}_j$ such that $u_t \in V(c, j)$
11: route to $c$ using the labeled tree routing on $T_c(j)$ [the label $l(c; c, j)$ is stored at $u_t$]
12: $SearchTree(l(v), T'(c, r(c, j)))$. [By Lemma 5.4 and Lemma 5.5 this retrieves $l(v; c, j)$.]
13: route to $v$ using the labeled tree routing on $T_c(j)$
```

LEMMA 5.5. *For any node u, v and any $j \in [N]$, if $\frac{64}{\epsilon} r(u, j) < d(u, v) < r(u, j+1)/8$, $SearchTree(l(v), T'(c, r(c, j)))$ retrieves the local label $l(v; c, j)$ of v successfully, where c is the center of a ball $B \in \mathscr{B}_j$ and $u \in V(c, j)$.*

Proof: First we show $v \in V(c, j)$, i.e. $v \in T_c(j)$ and then $v \in B_c(r(c, j+1))$ so that the local label $l(v; c, j)$ is stored in the search tree $T'(c, r(c, j))$.

For a contradiction, assume $v \in V(c', j)$ where $c' \neq c$ is a center of a ball $B' \in \mathscr{B}_j$. Thus $d(v, c') \leq d(v, c) \leq d(v, u) + d(u, c)$. Let $B = B_c(r(c, j)) \in \mathscr{B}_j$. Since $B \cap B' = \emptyset$, $r(c, j) + r(c', j) < d(c, c')$. Since $d(c, c') \leq d(c, v) + d(v, c') \leq 2d(v, c)$, we have

$$\begin{aligned} d(u, c') + r(c', j) &\leq (d(u, v) + d(v, c')) + (d(c, c') - r(c, j)) \\ &\leq d(u, v) + 3d(v, c) \\ &\leq 4d(u, v) + 6r(u, j) < \frac{2}{3} r(u, j+1), \end{aligned}$$

where the last inequality follows from $\frac{64}{\epsilon} r(u, j) < d(u, v) < \frac{r(u, j+1)}{8}$. Thus $B' \subset B_u(\frac{2}{3} r(u, j+1))$.

Since $d(u, c) + r(c, j) \leq 3r(u, j) < \frac{2}{3} r(u, j+1)$, $B \subset B_u(\frac{2}{3} r(u, j+1))$. Thus the ball $B_u(\frac{2}{3} r(u, j+1))$ with size less than 2^{j+1} contains two disjoint balls B and B', both of size 2^j, contradicting the assumption and implying $v \in T_c(j)$.

Now we show that $v \in B_c(r(c, j+1))$. Since balls $B_c(r(c, j+1))$ and $B_u(r(u, j+1))$ have the same size, we have $d(u, c) + r(c, j+1) \geq r(u, j+1)$. Hence $d(c, v) \leq d(c, u) + d(u, v) < r(u, j+1) - d(u, c) \leq r(c, j+1)$, i.e. $v \in B_c(r(c, j+1))$. Therefore the tree $T'(c, r(c, j))$ stores the local label $l(v; c, j)$ of v, and $SearchTree(l(v), T'(c, r(c, j)))$ retrieves it. ■

LEMMA 5.6. (STRETCH) *For any source node u and any destination node v in G, the total routing cost of our labeled routing scheme is no more than $(1+\epsilon)d(u, v)$.*

Proof: First we bound the stretch of routing from u_t to v by $1 + \epsilon/4$. By Lemma 5.1, from u_t to c and from c to v we route along the shortest path. Since the cost for $SearchTree(l(v), T'(c, r(c, j)))$ is bounded by $4r(c, j)$, the routing cost from u_t to v is no more than $d(u_t, c) + 4r(c, j) + d(c, v)$. Since $d(u_t, c) \leq 2r(u_t, j)$ and $r(c, j) \leq r(u_t, j)$ by Lemma 2.2 and $\frac{64}{\epsilon} r(u_t, j) < d(u_t, v)$ by Lemma 5.4, we have

$$\begin{aligned} &\frac{d(u_t, c) + 4r(c, j) + d(c, v)}{d(u_t, v)} \\ \leq &\frac{2d(u_t, c) + 4r(c, j) + d(u_t, v)}{d(u_t, v)} \\ \leq &1 + \frac{8r(u_t, j)}{\frac{64}{\epsilon} r(u_t, j)} \leq 1 + \epsilon/4. \end{aligned}$$

Now consider the total stretch. If $t = 0$, the stretch is at most $1 + \epsilon/4$ as above.

If $t > 0$, then $d(u, x_0) > 2^{i_0+f-1} - 2^{i_0}$. By the routing algorithm and Lemma 5.5, the routing cost is at most $\sum_{k=0}^{t-1} d(u_k, u_{k+1}) + (1 + \frac{\epsilon}{4}) d(u_t, v)$. As illustrated in Figure 2, by triangle inequality, we have

$$\begin{aligned} &\sum_{k=0}^{t-1} d(u_k, u_{k+1}) + d(u_t, v) \\ &\leq d(u, x_0) + (\sum_{k=0}^{t-2} d(x_k, x_{k+1}) + d(x_{t-1}, v)) \\ &\leq d(u, x_0) + 2^{i_0+1}. \end{aligned}$$

The last inequality follows since $x_k = z(v, i_k)$. Thus the routing cost is no more than $(1 + \frac{\epsilon}{4}) \left(d(u, x_0) + 2^{i_0+1} \right)$. Since $d(u, v) \geq d(u, x_0) - d(x_0, v) \geq d(u, x_0) - 2^{i_0+1}$ and $f = 8 + \log \frac{1}{\epsilon}$, then

$$\begin{aligned} \frac{d(u, x_0) + 2^{i_0+1}}{d(u, v)} &\leq \frac{d(u, x_0) + 2^{i_0+1}}{d(u, x_0) - 2^{i_0+1}} \\ &\leq 1 + \frac{2^{i_0+2}}{2^{i_0+f-2}} \leq 1 + \epsilon/4. \end{aligned}$$

Therefore the routing cost is no more than $(1 + \epsilon/4)^2 d(u, v) < (1 + \epsilon) d(u, v)$.

■

Proof of Theorem 1.2: The bounds on stretch and storage at the nodes follow from Lemmas 5.6 and

5.3, respectively. The packet header size of our labeled routing scheme is dominated by the packet header size of the underlying labeled tree-routing scheme, i.e. $O(\log^2 n/\log\log n)$ bits, by Lemma 5.1. ■

6 Future work

Since 9 is the asymptotically optimal stretch for name-independent compact routing schemes on networks with low doubling dimension [18], and $(2k+1)$-stretch routing schemes for general graphs require $\Omega((n \log n)^{1/k})$-bit memory per node [4], we see as the main open problem a relaxed version of the routing problem, or routing with *slack*: can an even better stretch be achieved if we allow a small constant fraction of nodes to have large memory, or a small constant fraction of source-destination pairs to incur large routing stretch?

References

[1] I. Abraham and C. Gavoille. Object location using path separators. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, July 2006.

[2] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi. Routing in networks with low doubling dimension. In 26th *ICDCS*. IEEE Computer Society Press, July 2006. To appear.

[3] I. Abraham, C. Gavoille, and D. Malkhi. Routing with improved communication-space trade-off. In *Proceedings of the 18th International Conference on Distributed Computing*, volume 3274 of *Lecture Notes in Computer Science*, pages 305–319, 2004.

[4] I. Abraham, C. Gavoille, and D. Malkhi. On space-stretch trade-offs: Lower bounds. In *Proceedings of the 1ith Annual ACM Symposium on Parallel Algorithms and Architecture*. ACM Press, July 2006.

[5] I. Abraham, C. Gavoille, and D. Malkhi. On space-stretch trade-offs: Upper bounds. In *Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architecture*. ACM Press, July 2006.

[6] I. Abraham and D. Malkhi. Name independent routing for growth bounded networks. In *Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 49–55, 2005.

[7] I. Abraham, D. Malkhi, and O. Dobzinski. Land: stretch $(1+\epsilon)$ locality-aware networks for DHTs. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 550–559, 2004.

[8] B. Awerbuch and D. Peleg. Sparse partitions. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 503–513, 1990.

[9] B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM J. Discret. Math.*, 5(2):151–162, 1992.

[10] H. T.-H. Chan, A. Gupta, B. Maggs, and S. Zhou. On hierarchical routing in doubling metrics. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 762–771, 2005.

[11] L. Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38:170–183, 2001.

[12] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. *Journal of Algorithms*, 46:97–114, 2003.

[13] P. Fraigniaud and C. Gavoille. Routing in trees. In 28th *ICALP*, volume 2076 of LNCS, pages 757–772. Springer, 2001.

[14] C. Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News - Distributed Computing Column*, 32(1):36–52, 2001.

[15] C. Gavoille and N. Hanusse. Compact routing tables for graphs of bounded genus. In 26th *ICALP*, volume 1644 of LNCS, pages 351–360. Springer, July 1999.

[16] C. Gavoille and D. Peleg. Compact and localized distributed data structures. *Journal of Distributed Computing*, 16:111–120, 2003.

[17] A. Gupta, R. Krauthgamer, and J.R.Lee. Bounded geometries, fractals and low-distortion embeddings. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 534–543, 2003.

[18] G. Konjevod, A. Richa, and D. Xia. Optimal-stretch name-independent compact routing in doubling metrics. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*, 2006.

[19] G. Konjevod, A. Richa, and D. Xia. Compact routing in networks of low doubling dimension. Manuscript available at `http://thrackle.eas.asu.edu/papers/compact.ps.gz`, 2007.

[20] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2000.

[21] A. Slivkins. Distance estimation and object location via rings of neighbors. In *Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing*, pages 41–50, 2005.

[22] A. Slivkins. Distributed approaches to triangulation and embedding. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 640–649, 2005.

[23] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *In Proceeding of the 2001 ACM SIGCOMM*, 2001.

[24] K. Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 281–290, 2004.

[25] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

[26] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 1–10, 2001.