Optimal Scale-free Compact Routing Schemes in Doubling Networks

Goran Konjevod, Andréa W. Richa, Donglin Xia
CSE Dept., Arizona State University, Tempe, AZ 85287-8809, USA
{goran,aricha,dxia}@asu.edu

July 5, 2006

Abstract

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

1 Introduction

A routing scheme in a network is a distributed algorithm that allows any source node to route packets to any destination node. 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 the neighbors, based on the destination's name and the local routing table. A relay node, upon the reception of a packet, decides whether the packet has reached the right 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* required to store the routing tables at each node (which contain all the information required by the routing scheme), and the *stretch* of the resulting routing scheme. In general, one would like to keep both the space requirement at the nodes 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, 23, 10, 20, 18], routing table size at each node, packet header size, or 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 doesn't directly depend on Δ. We say a routing scheme is *scale-free* if its memory requirements for the routing table at each node, packet header, and routing label are all independent of the aspect ratio.

In this paper, we give efficient scale-free compact routing schemes for doubling networks. A *doubling network* is a connected, edge-weighted, undirected graph whose shortest path metric is doubling. A *doubling metric* is a metric space with constant 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 target destination node, which is not always feasible. Thus, in most applications, given a name-dependent scheme, it remains an issue to determine how (and where) the source node will find the label of the destination node. Therefore name-independent routing schemes are desirable in many scenarios, for instance, in applications with intrinsic requirements on node names (e.g. distributed hash tables [7]), that require node names be randomly distributed in a segment (e.g., Chord [22]), or that perform network operations such as locating nearby copies by replicated objects and tracking of mobile objects [7, 8].

1.1 Our Contributions

We present optimal-stretch scale-free compact routing schemes for doubling networks 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 $\alpha > 0$, 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* $\left((\frac{1}{\epsilon})^{O(\alpha)}\log^3 n\right)$*-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* $\alpha > 0$*, 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* $\left((\frac{1}{\epsilon})^{O(\alpha)}\log^3 n\right)$*-bit routing information at each node.*

Our algorithm is the *first name-independent scale-free compact routing scheme for doubling metrics* with (asymptotically) *optimal stretch*, closing the gaps left by the results in [18] (where an optimal stretch, but not scale free, scheme is presented) and in [2] (where a scale-free, but not optimal stretch, 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 *name-dependent scale-free compact routing scheme for doubling metrics* 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). In spite of this limitation, we still obtain a scale-free scheme with asymptotically optimal stretch. Second, our techniques are significantly simpler than the ones used by Abraham et al. in [2], who also present an asymptotically optimal scale-free 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 technique 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 doubling metrics. 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 (to be defined later). 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 packed 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 [19] 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 doubling metrics, 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 incurs minimum 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 $\tilde{O}(n^{1/k})$ bits per node, 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 doubling networks), 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 doubling metrics 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 routing scheme with 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. Further, Thorup and Zwick [25] achieve stretch $2k-1$ using $\tilde{O}(n^{1/k})$ memory 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, 25]. There are $(1+\epsilon)$ stretch labeled routing schemes with polylogarithmic space for planar graphs [24], graphs excluding a fixed minor [1], and doubling networks [21, 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 doubling networks; 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 2.2 ([17]) *Let Y be an r-*net *of (V,d). For any $u \in V$ and any radius $r' \geq r$, 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 $T(\{Y_i\})$. 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 $A(u,i) = \sum_{k=1}^{i} d(z(u,k-1), z(u,k))$ for any node u, and any positive $k \in [M]$. The claim below follows directly from Definition 2.1.

Lemma 2.3 *For any node u and any positive $i \in [M]$, $A(u,i) \leq \sum_{k=1}^{i} 2^k < 2^{i+1}$.*

Let $N = \log n$, and for each $j \in [N]$, let $\mathscr{B}_j$ be a ball packing as defined in Lemma 2.4.

Lemma 2.4 (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 A Scale-Free Name-Independent Routing Scheme

3.1 Data Structures

Our scale-free name-independent routing scheme will use our scale-free name-dependent routing scheme as in Theorem 1.2 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 3.1.1, 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 3.1.2, we define the hierarchical data structures to maintain search trees.

3.1.1 Search Tree

Definition 3.1 (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.

By the definition of r-net, the height of the search tree $T(c,r)$ is at most

$$r + \sum_{i=1}^{\lfloor \log(\epsilon' r) \rfloor} 2^{\lfloor \log(\epsilon' r) \rfloor - i} \leq (1+\epsilon')r \tag{3.1}$$

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 2.2 the root has the maximum degree in the tree, $\left(\frac{4r}{2^{\lfloor \log(\epsilon' r) \rfloor - 1}}\right)^{\alpha} = \left(\frac{1}{\epsilon'}\right)^{O(\alpha)}$. Therefore each node keeps $\left(\frac{1}{\epsilon'}\right)^{O(\alpha)}$ number of labels for the search tree.

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

---

1. Sort all pairs according to their keys into a list.
2. Visit $T(c,r)$ from the root in depth-first order, and for each visited node,
   pick k/m new pairs from the list of Step 1, and store them at the current node.
3. 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 $T(c,r)$.

SearchTree (key**,** $T(c,r)$**)**
Begin at the root c and let $u = c$ initially.
WHILE there exists a child u' of u in $T(c,r)$ such that the range of u' contains key.
DO go to node u' and let $u = u'$.
IF u stores the *data* corresponding to the given *key*,
  report the *data*;
ELSE
  report error: there is no pair with key equal to given *key* in $T(c,r)$,
Go back from u to c along the tree $T(c,r)$.

3.1.2 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 $v \neq u$ and v is the nearest one in Y_{i+1} to u, we have $u \notin Y_{i+1}$. Hence $u \notin Y_j$ for any $j > i$. 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 2.4, 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 indices 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 exist 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 $B_u(2^i f(\epsilon)) \in B_c(r(c, j+2))$, we take advantage of the search tree on B to index the routing labels of $B_u(2^i f(\epsilon))$. 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:

Search (name,u,i)
IF $B_u(2^i f(\epsilon)) \in \mathscr{B}'$,
call $SearchTree(name, T(u, 2^i f(\epsilon)))$;
ELSE
let c be the center of $H(u, i)$, and r be the radius;
go to c from u by the labeled scheme, call $SearchTree(name, T(c, r))$,
and go back from c to u.

Note that the cost for $Search(name, u, i)$ is at most $2 \cdot (1+\epsilon') \cdot 2^i(f(\epsilon)+1) = 2^{i+1}(f(\epsilon)+2)$ by setting $\epsilon' = \frac{1}{f(\epsilon)+1}$ for search trees, where we don't count 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 3.2 *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 first show that for any fixed $j \in [N]$, the number of different balls $H(u, i) \in \mathscr{B}_j$ with $i \in S(u)$ is at most 4, and then the lemma follows.

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 greater than or equal to 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 $k = 0, \cdots, 3$. Since all balls $H(u, i_k)$ in $\mathscr{B}_j$ are non-intersecting, then $|B_u(2^{i_4} f(\epsilon))| > 2^{j+2}$. 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 3.3 *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 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 $a = 2$ and $b = \lceil \log(2f(\epsilon) + 1) \rceil$. For every $v \in V$, define a sequence of indexes $R(v) = \{i \in [M] \ : \ |B_v(2^{i+b})| \geq 2|B_v(2^{i-a})|\}$. The following claim, which is proved in Appendix, bounds the size of $R(v)$.

Claim 3.4 $|R(v)| \leq (a + b) \log n$.

Then we have the following claim (see Appendix for the proof), which relates the search trees for balls in B' that contain v to $R(v)$, thereby bounding the number of such trees.

Claim 3.5 *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 nodes $u \in Y_i$ such that $v \in B_u(2^i f(\epsilon))$ is at most $(4f(\epsilon))^\alpha$ by Lemma 2.2. Thus by Claim 3.4 and 3.5, the number of search trees for balls in $\mathscr{B}'$ that contain v is no more than $(a + b) \log n \cdot (4f(\epsilon))^\alpha = (\frac{1}{\epsilon})^{O(\alpha)} \log n$. The lemma follows. ■

Lemma 3.6 *The routing information at each node is no more than $(\frac{1}{\epsilon})^{O(\alpha)} \log^3 n$ 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 3.2, 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 search trees can be bounded as follows. Since the maximum degree of nodes 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 2^2 *(name, label)* pairs, it stores $O(\log n)$ bits of data. Since the number of search trees containing a same node is no more than $(\frac{1}{\epsilon})^{O(\alpha)} \log n$ by Lemma 3.3, 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 there are at most $(\frac{1}{\epsilon})^{O(\alpha)} \log^3 n$ bits of routing information stored at each node. ■

3.2 Routing Algorithm

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

Assume that a source node u wants to send a message to a destination node v, given the name $n(v)$ of v. The routing proceeds in two phases. In Phase 1, we find the routing label $l(v)$ of v by invoking the local search procedure $Search(n(v); z(u, i), i)$ along the *zooming sequence* of u. In Phase 2, we call the underlying labeled routing scheme to route to v. Let $i = 0$ initially.

Phase 1: Perform a local search at $z(u,i)$ by calling the procedure Search$(n(v), z(u,i), i)$.
IF the routing label $l(v)$ of v is not found,
go to $z(u,i+1)$ by the underlying routing scheme, let $i = i+1$, and repeat Phase 1;
ELSE go to Phase 2.
Phase 2: Go to v from $z(u,i)$ using the underlying labeled routing scheme.

Fix the $(1+\epsilon/20)$-stretch required of the underlying labeled scheme. Then we have the following stretch lemma. The proof, which is similar to the one in [18], appears in the Appendix.

Lemma 3.7 *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 of Theorem 1.1: The bounds on stretch and storage requirement at the nodes follow from Lemma 3.7 and 3.6, 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. ∎

4 A Scale-Free Labeled Routing Scheme

4.1 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, traverse the *netting tree* $T(\{Y_i\})$ by a depth-first-search order, and let $l : V \to [n]$ be the enumeration of the leaves during the traversal. 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. the nodes v are the leaves 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 property 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 2.2, 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 4.1 ([13, 25]) *For every weighted tree T with 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 in T, 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 the property of Voronoi diagram, for each fixed j, the trees $T_c(j)$ are disjoint. Thus the local routing label information stored 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 4.2 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 3.1.1, retrieves the label $l(v; c, j)$ of v along the shortest path of the search tree.

Definition 4.2 (Search Tree II) For any ball $B_c(r)$, the search tree II, denoted by $T'(c,r)$, is modified from the search tree $T(c,r)$ with $\epsilon' = 1/2$ as in Definition 3.1:

(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 $u' \in U_{\lceil 2\log n \rceil}\}$. Thus link the nodes in $V(u) \setminus \bigcup_{0 \le j \le \lceil 2\log n \rceil} U_j$ to be a path and 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 no more than $(1+\frac{1}{2})r + \frac{r}{n^2} \cdot n < 2r$. The following lemma shows how to link each virtual edge. We defer its proof to the Appendix.

Lemma 4.3 *For any search tree II $T'(c,r)$, we are able to deliver packets along each virtual edge with cost no more than the weight of the edge, by maintaining $(1/2)^{(\alpha)} \log^2 n$ bits of data structures per node.*

Lemma 4.4 (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. ∎

4.2 Routing Algorithm

Assume that a source node u wants to route to a destination node v given by its label $l(v)$. Let $u_0 = u$, $i_{-1} = M$ and $k = 0$ initially.

---

1. Let $i_k \in R(u_k)$ be the minimal index such that there exists $x_k \in X_{i_k}(u_k)$ with $l(v) \in Range(x_k, i_k)$.
2. **IF** $i_k \le i_{k-1}$ and $d(u_k, x_k) \ge 2^{i_k+f-1} - 2^{i_k}$

   THEN

   Let u_{k+1} be one of u_k's neighbors on the shortest path from u_k to x_k, and go to u_{k+1};
   Let $k = k+1$ and repeat Step 1.

   ELSE

   Let $t = k$. Let $j \in [N]$ be such that $r(u_t, j) \le 2^{i_t} < r(u_t, j+1)$, and
   c be the center of a ball in $\mathscr{B}_j$ such that $u_t \in V(c,j)$.
   By Lemma 4.5, we have $v \in T_c(j)$. Route from u_t to v (with cost less than $(1+\epsilon/4)d(u_t, v)$)
   using the labeled tree-routing scheme on $T_c(j)$.

Lemma 4.5 *Since either $i_t > i_{t-1}$ or $d(u_t, x_t) < 2^{i_t+f-1} - 2^{i_t}$, we have $v \in T_c(j)$, where $j \in [N]$ is the index such that $r(u_t, j) \leq 2^{i_t} < r(u_t, j+1)$ and c is the center of a ball in $\mathscr{B}_j$ such that $u_t \in V(c, j)$. Moreover, we can route from u_t to v with cost less than $(1 + \epsilon/4)d(u_t, v)$.*

Proof: First we have a claim, whose proof we defer to the Appendix.

Claim 4.6 *If $\frac{64}{\epsilon}r(u_t, j) < d(u_t, v) < \frac{r(u_t,j+1)}{8}$, then $v \in T_c(j)$ and we can route from u_t to v with cost less than $(1 + \epsilon/4)d(u_t, v)$.*

Note that $l(v) \in Range(x_t, i_t)$ iff. $x_t = z(v, i_t)$. Let $x' = z(v, i_t - 1)$. We have two cases:

(1) $d(u_t, x_t) < 2^{i_t+f-1} - 2^{i_t}$. Since $d(x', x_t) \leq 2^{i_t}$ by the definition of r-net, we have $d(u_t, x') \leq d(u_t, x_t) + d(x_t, x') \leq (2^{i_t+f-1} - 2^{i_t}) + 2^{i_t} = 2^{i_t+f-1}$.

(2) $i_t > i_{t-1}$. We have $d(u_t, x') \leq d(u_t, x_{t-1}) + d(x_{t-1}, x') \leq 2^{i_{t-1}+f} + d(x_{t-1}, x')$. If $i_{t-1} = i_t - 1$, then we have $x_{t-1} = x'$ and $d(u_t, x') < 2^{i_t+f-1}$. Otherwise we have $2^{i_{t-1}+f} \leq 2^{i_t-2+f}$ and $d(x_{t-1}, x') \leq 2^{i_t}$ by Lemma 2.3. Hence $d(u_t, x') < 2^{i_t+f-1}$.

Therefore $x' \in X_{i_t-1}(u_t)$. Hence by the minimality of i_t at Step 1, we have $i_t - 1 \notin R(u_t)$. Since $r(u_t, j) \leq 2^{i_t} < r(u_t, j+1)$, by the definition of $R(u)$ we have $r(u_t, j) < 2^{i_t-1} < r(u_t, j+1)2^{-a}$. Hence $d(u_t, v) \leq d(u_t, x') + d(x', v) < 2^{i_t+f-1} + 2^{(i_t-1)+1} < r(u_t, j+1)2^{-a+f+1} = r(u_t, j+1)/8$.

Let $i' = \lceil \log r(u_t, j) \rceil \in R(u_t)$. Since $r(u_t, j) < 2^{i_t-1}$ as shown above, we have $i' < i_t$. By the minimality of i_t, we have $z(v, i') \notin X_{i'}(u_t)$, i.e. $d(u_t, z(v, i')) > 2^{i'+f} \geq r(u, j)2^{f-1}$. Thus $d(u_t, v) \geq d(u_t, z(v, i')) - d(z(v, i'), v) \geq r(u, j)2^{f-1} - 2^{i'+1} > r(u, j)2^{f-2} = \frac{64}{\epsilon}r(u, j)$.

Therefore we have $\frac{64}{\epsilon}r(u_t, j) < d(u_t, v) < r(u_t, j+1)/8$ and by Claim 4.6, the lemma follows. ■

Lemma 4.7 (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 have a claim, whose proof we defer to the Appendix.

Claim 4.8 $\sum_{k=0}^{t-1} d(u_k, u_{k+1}) + d(u_t, v) \leq d(u_0, x_0) + 2^{i_0+1}$

If $t = 0$, by Claim 4.6 the stretch is at most $1 + \epsilon/4$.

If $t > 0$, then $d(u, x_0) > 2^{i_0+f-1} - 2^{i_0}$. By the routing algorithm and Lemma 4.5, the routing cost is at most $\sum_{k=0}^{t-1} d(u_k, u_{k+1}) + (1 + \frac{\epsilon}{4})d(u_t, v)$. By Claim 4.8, we have $\sum_{k=0}^{t-1} d(u_k, u_{k+1}) + (1 + \frac{\epsilon}{4})d(u_t, v) \leq (1 + \frac{\epsilon}{4})\left(d(u_0, x_0) + 2^{i_0+1}\right)$. 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

$$\frac{d(u_0, x_0) + 2^{i_0+1}}{d(u, v)} \leq \frac{d(u_0, x_0) + 2^{i_0+1}}{d(u, x_0) - 2^{i_0+1}} \leq 1 + \frac{2^{i_0+2}}{2^{i_0+f-2}} \leq 1 + \epsilon/4 \tag{4.2}$$

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 4.7 and 4.4, 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 4.1. ■

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 26^{th} *International Conference on Distributed Computing Systems (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 F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, 28^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of Lecture Notes in Computer Science, 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 J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, 26^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of Lecture Notes in Computer Science, 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] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2000.

[20] 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.

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

[22] 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.

[23] 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.

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

[25] 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.

5 Appendix

Proof of Claim 3.4: We bound the size of $R(v)$ by its maximal subsequence $\tilde{R}(v) = \{i_0, i_1, \cdots, i_{m-1}\}$ such that (i) $i_0 = \min\{R(v)\}$, and (ii) for $0 < j < m$, i_j is the minimal element in $R(v)$ such that $i_j \geq i_{j-1} + (a+b)$. Thus $|R(v)| \leq (a+b)|\tilde{R}(v)|$.

We show that $|\tilde{R}(v)| \leq \log n$. By the definition of $R(v)$, we have $|B_v(2^{i_j+b})| \geq 2|B_v(2^{i_j-a})|$ for $j \in [m]$. Since $i_j \geq i_{j-1} + (a+b)$, then $|B_v(2^{i_j+b})| \geq 2|B_v(2^{i_j-a})| \geq 2|B_v(2^{i_{j-1}+b})|$ for positive $j \in [m]$. Since $|B_v(2^{i_0+b})| \geq 2|B_v(2^{i_0-a})| \geq 2$, then $n \geq |B_v(2^{i_{m-1}+b})| \geq 2^{m-1}|B_v(2^{i_0+b})| \geq 2^m$. Thus $m \leq \log n$ and $|R(v)| \leq (a+b)\log n$.

■

Proof of Claim 3.5: Let j be the index such that $2^j \leq |B_v(2^{i-a})| < 2^{j+1}$. Thus $r(v,j) \leq 2^{i-a} = 2^{i-2}$. We will show that $|B_v(2^{i+b})| \geq 2^{j+2} > 2|B_v(2^{i-a})|$, and therefore $i \in R(v)$.

By Lemma 2.4, 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 will make use of 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 $|B_v(2^{i+b})| \geq 2^{j+2} > 2|B_v(2^{i-a})|$. The claim follows.

■

Proof of Lemma 3.7: Let j be the index of the level at which the procedure $Search(z(u, j), j, v)$ finds the routing label of v in Phase 1.

Note that the routing cost of $Search(z(u, i), i, v)$ is $2^{i+1}(f(\epsilon) + 2) \cdot (1 + \epsilon/20)$. Then the total routing cost is no more than $\Big(A(u, j) + \sum_{i=0}^{j} 2^{i+1}(f(\epsilon) + 2) + d(z(u, j), v)\Big)\,(1 + \epsilon/20)$. First we have

$$\begin{aligned} &A(u, j) + \sum_{i=0}^{j} 2^{i+1}(f(\epsilon) + 2) + d(z(u, j), v) \\ &\leq A(u, j) + \sum_{i=0}^{j} 2^{i+1}(f(\epsilon) + 2) + d(u, v) + A(u, j) \\ &\leq 2^{j+2}(f(\epsilon) + 3) + d(u, v) \end{aligned} \tag{5.3}$$

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

$$\begin{aligned} d(u, v) &\geq d(z(u, j - 1), v) - d(z(u, j - 1), u) \\ &\geq d(z(u, j - 1), v) - A(u, j - 1) \\ &\geq f(\epsilon)2^{j-1} - 2^j \\ &= 2^{j-1}(f(\epsilon) - 2) \end{aligned} \tag{5.4}$$

and thus,

$$2^{j+2}(f(\epsilon) + 3) + d(u, v) \leq \frac{8(f(\epsilon) + 3)}{f(\epsilon) - 2} d(u, v) + d(u, v) = \left(9 + \frac{40}{f(\epsilon) - 2}\right) d(u, v). \tag{5.5}$$

Since $f(\epsilon) = \frac{80}{\epsilon} + 2$, the total routing cost is no more than

$$\left(9 + \frac{40}{f(\epsilon) - 2}\right) d(u, v)(1 + \epsilon/20) = \left(9 + \frac{\epsilon}{2}\right)(1 + \epsilon/20)\, d(u, v) \leq (9 + \epsilon)d(u, v). \tag{5.6}$$

■

Proof of Lemma 4.3: First, for any virtual edge (u, v), where $u \in U_{i-1}$, $v \in U_i$ and $0 < i \leq \min(\lceil 2 \log n \rceil, \lfloor \log \frac{r}{2} \rfloor)$, let each node x on the shortest path from u to v maintain the next hop information and then u and v can communicate along the shortest path. Now 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 2.2, the number of nodes $v \in U_i$ whose closest node in U_{i-1} is u is 8^α. Since $0 < i \leq \min(\lceil 2 \log n \rceil, \lfloor \log \frac{r}{2} \rfloor)$ 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, consider the virtual edges to link u and the nodes in $V(u) \setminus \bigcup_{0 \leq j \leq \lceil 2 \log n \rceil} U_j$ 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 as in Lemma 4.1 for the tree $T(u)$. Let the two endpoints of each of these virtual edges keep each other's local label. Since $d(u, v) \leq 2^{\lceil \log(r/2) \rceil - \lceil 2 \log n \rceil} \leq \frac{r}{2n^2}$ for any

$v \in V(u) \setminus \bigcup_{0 \le j \le \lceil 2 \log n \rceil} U_j$, then the routing cost along each of these virtual edge is no more than $\frac{r}{n^2}$. By the property of Voronoi diagram, the trees $T(u)$ for all nodes $u \in \lceil 2 \log n \rceil$ are disjoint. Thus by Lemma 4.1, each node in $B_c(r)$ maintains $O(\log^2 n / \log \log n)$ bits of routing information for the local labeled routing. ■

Proof of Claim 4.6: First we show $v \in V(c, j)$, i.e. $v \in T_c(j)$. For a contradiction, assume $v \notin V(c, j)$, and let c' be a center of a ball $B' \in \mathscr{B}_j$ such that $v \in V(c', j)$. Thus $d(v, c') \le d(v, c) \le d(v, u_t) + d(u_t, c)$. Since $B \cap B' = \emptyset$, $r(c, j) + r(c', j) < d(c, c')$. Since $d(c, c') \le d(c, v) + d(v, c') \le 2d(v, c)$, then we have

$$\begin{aligned} d(u_t, c') + r(c', j) &\le (d(u_t, v) + d(v, c')) + (d(c, c') - r(c, j)) \\ &\le d(u_t, v) + 3d(v, c) \\ &\le 4d(u_t, v) + 6r(u_t, j) \\ &< \frac{2}{3} r(u_t, j+1) \end{aligned} \tag{5.7}$$

, where the last inequality follows from $\frac{64}{\epsilon} r(u_t, j) < d(u_t, v) < \frac{r(u_t, j+1)}{8}$. Thus $B' \subset B_{u_t}(\frac{2}{3} r(u_t, j+1))$. Since $d(u_t, c) + r(c, j) \le 3r(u_t, j) < \frac{2}{3} r(u_t, j+1)$, then $B \subset B_{u_t}(\frac{2}{3} r(u_t, j+1))$. Since $|B| = |B'| = 2^j$ and $B \cap B' = \emptyset$, we have $|B_{u_t}(\frac{2}{3} r(u_t, j+1))| > 2^{j+1}$, contradiction with the definition of $r(u_t, j+1)$. Therefore the assumption is not right and we have $v \in T_c(j)$.

Now we show that $v \in B_c(r(c, j+1))$ so that the local label $l(v; c, j)$ is stored in the search tree $T'(c, r(c, j))$. Since $|B_c(r(c, j+1))| = |B_{u_t}(r(u_t, j+1))|$, we have $d(u_t, c) + r(c, j+1) \ge r(u_t, j+1)$ (Otherwise, $B_c(r(c, j+1)) \subset B_{u_t}(r(u_t, j+1))$, contradiction). Hence $d(c, v) \le d(c, u_t) + d(u_t, v) < r(u_t, j+1) - d(u_t, c) \le r(c, j+1)$. Therefore $v \in B_c(r(c, j+1))$.

Now we can (1) route from u_t to c by the labeled tree-routing scheme on $T_c(j)$ since u_t has the local label $l(c; c, j)$; (2) call $SearchTree(l(v), T'(c, r(c, j)))$ to retrieve the local label $l(v; c, j)$; and (3) route from c to v by the labeled tree-routing scheme on $T_c(j)$.

By Lemma 4.1, 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 $4r(c, j)$. Thus the routing cost from u_t to v is no more than $d(u_t, c) + 4r(c, j) + d(c, v)$. Since $d(u_t, c) \le 2r(u_t, j)$ and $r(c, j) \le r(u_t, j)$ by Lemma 2.4, we have

$$\frac{d(u_t, c) + 4r(c, j) + d(c, v)}{d(u_t, v)} \le \frac{2d(u_t, c) + 4r(c, j) + d(u_t, v)}{d(u_t, v)} \le 1 + \frac{8r(u_t, j)}{\frac{64}{\epsilon} r(u_t, j)} \le 1 + \epsilon/4 \tag{5.8}$$

■

Proof of Claim 4.8: Since u_{k+1} is on the shortest path from u_k to x_k, we have $d(u_k, u_{k+1}) + d(u_{k+1}, x_k) = d(u_k, x_k)$. Since $d(u_{k+1}, x_{k+1}) \le d(u_{k+1}, x_k) + d(x_k, x_{k+1})$, we have

$$d(u_k, u_{k+1}) + d(u_{k+1}, x_{k+1}) \le d(u_k, x_k) + d(x_k, x_{k+1}). \tag{5.9}$$

Since $d(u_t, v) \le d(u_t, x_{t-1}) + d(x_{t-1}, v)$, we have

$$d(u_{t-1}, u_t) + d(u_t, v) \le d(u_{t-1}, x_{t-1}) + d(x_{t-1}, v). \tag{5.10}$$

Thus by adding Equations 5.9 for all $0 \le k < t-1$ and Equations 5.10, we have

$$\sum_{k=0}^{t-1} d(u_k, u_{k+1}) + d(u_t, v) \le d(u_0, x_0) + \sum_{k=0}^{t-2} d(x_k, x_{k+1}) + d(x_{t-1}, v) \tag{5.11}$$

Since $x_k = z(v, i_k)$ and $i_k \ge i_{k+1}$ for $k < t$, then $\sum_{k=0}^{t-2} d(x_k, x_{k+1}) + d(x_{t-1}, v) \le \sum_{j=0}^{i_0} 2^j < 2^{i_0+1}$. Therefore $\sum_{k=0}^{t-1} d(u_k, u_{k+1}) + d(u_t, v) \le d(u_0, x_0) + 2^{i_0+1}$ ■