# Compact Routing with Slack in Low Doubling Dimension

Goran Konjevod [*]
CSE Department
Arizona State University
Tempe, AZ 85287, USA
goran@asu.edu

Andréa W. Richa [†]
CSE Department
Arizona State University
Tempe, AZ 85287, USA
aricha@asu.edu

Donglin Xia [‡]
CSE Department
Arizona State University
Tempe, AZ 85287, USA
dxia@asu.edu

Hai Yu
CS Department
Duke University
Durham, NC 27708, USA
fishhai@cs.duke.edu

## ABSTRACT

We consider the problem of compact routing with slack in networks of low doubling dimension. Namely, we seek name-independent routing schemes with $(1+\epsilon)$ stretch and polylogarithmic storage at each node: since existing lower bound precludes such a scheme, we relax our guarantees to allow for (i) a small fraction of nodes to have large storage, say size of $O(n \log n)$ bits, or (ii) a small fraction of source-destination pairs to have larger, but still constant, stretch.

In this paper, given any constant $\epsilon \in (0,1)$, any $\delta \in \Theta(1/\operatorname{polylog} n)$ and any connected edge-weighted undirected graph $G$ with doubling dimension $\alpha \in O(\log \log n)$ and arbitrary node names, we present

1. a $(1+\epsilon)$-stretch name-independent routing scheme for $G$ with polylogarithmic packet header size, and with $(1-\delta)n$ nodes storing polylogarithmic size routing tables each and the remaining $\delta n$ nodes storing $O(n \log n)$-bit routing tables each.

2. a name-independent routing scheme for $G$ with polylogarithmic storage and packet header size, and with stretch $(1+\epsilon)$ for $(1-\delta)n$ source nodes and $(9+\epsilon)$ for the remaining $\delta n$ source nodes.

These results are to be contrasted with our lower bound from PODC 2006, where we showed that stretch 9 is asymptotically optimal for name-independent compact routing schemes in networks of constant doubling dimension.

[*]Supported in part by NSF grant CCR-0209138.

[†]Supported in part by NSF CAREER grant CCR-9985284.

[‡]Supported in part by NSF grant CCR-0209138, NSF CAREER grant CCR-9985284, and ASU GPSA research grant.


PODC'07, August 12–15, 2007, Portland, Oregon, USA.


## Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols - Routing protocols; E.1 [**Data Structures**]: Distributed data structures; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems - Computations on discrete structures, Routing and Layout; G.2.2 [**Discrete Mathematics**]: Graph Theory - Graph algorithms, Graph labeling, Network problems

## General Terms

Algorithms, Performance, Theory

## Keywords

Compact Routing, Doubling Dimension, Name-independent

## 1. INTRODUCTION

A routing scheme is a distributed algorithm that allows any source node to deliver packets to any destination node in a network. In this paper, we model a network as an undirected graph with arbitrary given node names and edge weights. A *name-independent* routing scheme uses only the given node names to route, while a *name-dependent* (or labeled) routing scheme allows the designer to label (or rename) the nodes to embed additional information that will aid in routing, such as topological information.

One of the fundamental trade-offs in routing is between stretch and space. The *stretch* of a routing scheme is the maximum ratio of the length of a routing path between a pair of nodes $s$ and $t$ according to the scheme, to the length of a shortest $s$-$t$ path, over all pairs $s, t$. The *space* requirement of a scheme refers to the size of routing tables maintained at each node and the size of the packet headers used by the scheme. A routing scheme is *compact* if the routing table at each node and every packet header have size at most a polylogarithmic function of the number of nodes.

Compact routing schemes are desirable for scalability. For general graphs, however, any (name-independent or name-dependent) routing scheme with $\tilde{O}(n^{1/k})$[1]-bit routing tables requires $\Omega(k)$ stretch [16, 19, 5], and hence no com-

[1]$\tilde{O}()$ denotes complexity similar to $O()$ up to polylog factors.

pact routing scheme of constant stretch exists for general graphs. Moreover, every name-independent routing scheme with less than $(n/(9k))^{1/k}$ bits of space requirement per node has *average stretch* at least $\Theta(k)$ [5]. On the other hand, optimal $(9+\epsilon)$-stretch compact name-independent routing schemes exist for networks of low doubling dimension [14]. In fact, we showed that stretch 9 is asymptotically optimal for name-independent compact routing in networks with constant doubling dimension [13]. The *doubling dimension* of a network is the least value $\alpha$ such that any ball of radius $r$ can be covered by at most $2^\alpha$ balls of radius $r/2$ (the ball of radius $r$ centered at $c$ is the set of nodes within distance at most $r$ of $c$).

In this paper, we consider the problem of achieving $(1+\epsilon)$-stretch name-independent compact routing schemes for networks of low doubling dimension if we are allowed slightly *relaxed guarantees* either in memory or in stretch—i.e., we either allow a $1/\operatorname{polylog} n$ fraction of the nodes to have larger than polylogarithmic storage size, or allow the stretch between a $1/\operatorname{polylog} n$ fraction of all pairs of nodes to be larger than $(1+\epsilon)$ (albeit still constant). If we compare our results with the lower bound of 9 for stretch in networks of constant doubling dimension, we have managed to improve the stretch of our scheme considerably by allowing a small relaxation of the guarantees. Relaxed guarantees have also been recently considered in the context of metric embeddings [12, 1, 2, 3]. Our main results are described in the following two theorems:

**Theorem 1.1** *Given any $\delta \in \Theta(1/\operatorname{polylog} n)$, any constant $\epsilon \in (0,1)$, and an edge-weighted undirected graph $G$ with $n$ nodes and doubling dimension $\alpha \in O(\operatorname{loglog} n)$, we present a $(1+\epsilon)$-stretch name-independent routing scheme for $G$ with packet headers of $O(\log^2 n/\operatorname{loglog} n)$ bits, where $(1-\delta)n$ nodes store $\left(\log^2 n/\epsilon^{O(\alpha)}(1/\delta+\log n)\right)$-bit routing tables each, and the remaining $\delta n$ nodes store $O(n\log n)$-bit routing tables each.*

**Theorem 1.2** *Given any $\delta \in \Theta(1/\operatorname{polylog} n)$, any constant $\epsilon \in (0,1)$, and an edge-weighted undirected graph $G$ with $n$ nodes and doubling dimension $\alpha \in O(\operatorname{loglog} n)$, we present a name-independent compact routing scheme for $G$ with routing tables at each node of $\left(\log^4 n/(\delta\epsilon^{O(\alpha)})\right)$ bits, and packet header size of $O(\log^2 n/\operatorname{loglog} n)$ bits, and with stretch $(1+\epsilon)$ for $(1-\delta)n$ source nodes, and $(9+\epsilon)$ for the remaining $\delta n$ source nodes.*

## 1.1 Other related work

In recent years, there has been a great amount of work on efficient network routing schemes. For a general review of this topic, we refer to the book of Peleg [15] and the surveys by Gavoille [9] and Gavoille and Peleg [10]. More recent references can be found in [14].

In particular, there has been a lot of recent progress in designing compact routing schemes for networks of low doubling dimension. Abraham et al. [4] achieved a compact routing scheme with (large) constant stretch and storage polylogarithmic in $n$, while Konjevod et al. [13] obtained an asymptotically optimal $(9+\epsilon)$-stretch routing scheme with storage of size polylogarithmic in $n$ and the normalized network diameter. In [14], Konjevod et al. closed the gap left by the previous two papers and presented a compact routing scheme with $(9+\epsilon)$-stretch and polylogarithmic storage that is scale-free (independent of the normalized network diameter).

There are several lower bound results on compact routing schemes [16, 19, 5, 13]. The strongest lower bound result for name-independent compact routing schemes in networks of low doubling dimension [13] shows that any name-independent scheme with $o(n^{(\epsilon/60)^2})$-bit storage has stretch at least $9-\epsilon$ for a tree with doubling dimension no more than $6-\log\epsilon$ and normalized diameter $O(2^{1/\epsilon}n)$, for any $\epsilon \in (0,8)$.

# 2. OVERVIEW

In this section, we give a brief overview of our compact name-independent routing schemes with slack.

First, our compact name-independent routing schemes use a $(1+\epsilon)$-stretch compact name-dependent routing scheme as their underlying labeled routing scheme. Thus given the name of the destination, a source node first retrieves the label of the destination according to its name, and then routes to the destination using the labeled routing scheme. Hence the main merit of our name-independent routing schemes is an efficient method for distributed storage and retrieval of node labels from their names.

Second, we maintain a hierarchy of $r$-nets (see Section 3 for the formal definition), which has been widely used in context of compact routing schemes [7, 17, 4, 13]. For our schemes, in order to achieve $1+\epsilon$ stretch, a natural idea is that, for any node $u$ in the $2^i$-net, a search tree is maintained on the ball $B_u(2^i f)$ to store routing information of nodes in the ball $B_u(2^i f/\epsilon)$, where $f$ can be any fixed constant and $B_u(r)$ denotes the set of nodes within $r$ distance to the center $u$. By using a typical search tree technique, we provide a search procedure on ball $B_u(2^i f)$ with cost $2^{i+1}f$ to either retrieve the routing label of a given node if the node is in the ball $B_u(2^i f/\epsilon)$, or report error if it is not in the ball $B_u(2^i f/\epsilon)$. Thus the routing algorithm starts at the source node and repeatedly applies search procedure from level $i=0$ until the routing label of the destination is founded. Then it uses the underlying labeled routing scheme to route the message to the destination. The overall stretch is $1+O(\epsilon)$, since the cost for the retrieval of the routing label is at most $O(\epsilon)$ times the source-destination distance.

However, the ball $B_u(2^i f/\epsilon)$ might contain so many nodes that the search tree on $B_u(2^i f)$ with polylog-size storage per node cannot store the routing information of all nodes in $B_u(2^i f/\epsilon)$. We present a novel counting lemma (See Lemma 4.2) to deal with the case of too many nodes in $B_u(2^i f/\epsilon)$. The counting lemma guarantees the number of $2^i$-net nodes $u$ with $|B_u(2^i f/\epsilon^2) \setminus B_c(2^{i+2})|/|B_u(2^i f)| = \Omega(\operatorname{polylog} n)$ is limited, where $c$ is a center node of the nearest packing ball to $u$ (see Section 3 for a formal definition of ball packings); in some sense, $B_c(2^{i+2})$ is a local dense area within $2^i f/\epsilon$ distance to $u$ , and $|B_u(2^i f/\epsilon^2) \setminus B_c(2^{i+2})|/|B_u(2^i f)| = \Omega(\operatorname{polylog} n)$ implies that there is more than one dense area in the ball $B_u(2^i f/\epsilon^2)$. For those $r$-net nodes $u$ with more than one dense area in $B_u(2^i f/\epsilon^2)$, we just relax guarantees either on storage or on stretch. On the other hand,if $|B_u(2^i f/\epsilon^2) \setminus B_c(2^{i+2})|/|B_u(2^i f)| = O(\operatorname{polylog} n)$, let the search tree on $B_u(2^i f)$ store the routing information of nodes in $B_u(2^i f/\epsilon^2) \setminus B_c(2^{i+2})$. Thus

if the search procedure on $B_u(2^i f)$ cannot find the destination's label, the routing algorithm will go to node $c$ to search the routing information in the local dense area of $B_c(2^{i+2})$. In that case, the $1+O(\epsilon)$ stretch is still guaranteed, since the distance between $u$ and $c$ is basically at most $2^i f/\epsilon$, and since the destination is either in the local area of $c$, i.e. $B_c(2^{i+2})$, or outside of $B_u(2^i f/\epsilon^2)$.

Finally the counting lemma relies on the combination of hierarchical $r$-nets and ball packings together with a novel entropy analysis. We believe that this novel data structure and technique may be of independent interest. In addition, for the routing scheme with slack on stretch, we use the techniques of our previous work in [14] to guarantee (i) $9+\epsilon$ stretch for all source-destination pairs and (ii) scale-free storage (independent of the normalized network diameter).

# 3. PRELIMINARIES

Let $G=(V,E)$ be a connected, edge-weighted, undirected graph with $n$ nodes, shortest-path metric $d$, doubling dimension $\alpha \in O(\text{loglog}\, n)$, arbitrary $\log n$-bit node id (name), and arbitrary normalized diameter $\Delta$, i.e. $\Delta = \max d(u,v)/\min_{u\neq v} d(u,v)$. W.l.o.g., assume that the minimum weight of an edge is normalized to be one, and assume that both $n$ and $\Delta$ are some power of 2. Otherwise just let $n$ (and $\Delta$) be the least power of 2 that is at least $n$ (and $\Delta$), and the results of Theorem 1.1 and 1.2 still hold.

## 3.1 Labeled routing

Our name-independent routing schemes for $G$ will use the labeled routing scheme of [14] as the effective underlying labeled routing scheme. For reference, we list the main results achieved by this labeled routing scheme:

**Lemma 3.1 (Labeled Routing [14])** *Given any weighted undirected graph with $n$ nodes, and doubling dimension $\alpha$, for any $\epsilon \in (0,1)$, there exists a $(1+\epsilon)$-stretch labeled routing scheme with $\lceil \log n \rceil$-bit routing labels, $\left((\frac{1}{\epsilon})^{O(\alpha)} \log^3 n\right)$-bit storage at each node and $O(\log^2 n/\log\log n)$-bit packet header.*

For any $u \in V$, let $id(u)$ denote the arbitrary original node id of $u$, and let $l(u)$ denote the label assigned to $u$ by the underlying labeled routing scheme. When a node $u$ wants to send a message to another node $v$ in the network, it will initiate a route request for $id(v)$. The routing algorithm uses $id(u)$ to retrieve the label $l(v)$ and then routes to $v$ using $l(v)$ via the underlying labeled scheme. Hence the main merit of our name-independent routing scheme is to present an efficient distributed method for storing and retrieving the label of a node given its id. Since $(1+\epsilon)(1+O(\epsilon)) = 1+O(\epsilon)$ for $\epsilon < 1$, we will omit the $(1+\epsilon)$ stretch caused by the underlying labeled routing when analyzing our name-independent schemes. Moreover, for simplicity, we will prove Theorem 1.1 and 1.2 with stretch in terms of big-$O$ of $\epsilon$, i.e. with stretch $1+O(\epsilon)$ or $9+O(\epsilon)$.

## 3.2 $r$-nets and ball packing.

We now introduce two hierarchical data structures that will be used in our routing schemes: $r$-nets and ball packings. $R$-nets have been widely used in the context of routing schemes and metric embeddings [18, 7, 17, 4, 13, 14], while ball packings were used in the context of routing schemes [14, 8] and metric embeddings [6].

**Definition 3.2 ($r$-net)** *An $r$-net of a metric space $(V,d)$ is a subset $Y \subseteq V$ such that any point in $V$ is within distance at most $r$ from $Y$, and any two points in $Y$ are within distance at least $r$.*

For example, $V$ is a 1-net, and a single node $v$ is a $\Delta$-net. Note that for any finite metric such an *$r$-net* exists and can be constructed greedily. 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 $r > 0$. Then the following is a well-known result about the number of $r$-net nodes in a ball:

**Lemma 3.3 ([11])** *Let $Y$ be an $r$-net of $(V,d)$. For any $u \in V$ and $r' \geq r$, we have $|B_u(r') \cap Y| \leq \left(\frac{4r'}{r}\right)^{\alpha}$.*

We now construct $2^i$-nets $Y_i$ for all $i \in [\log \Delta]$[1] as follows:

1. The $\Delta$-net $Y_{\log\Delta}$ is a singleton consisting of an arbitrary node in $V$.
2. The $2^i$-net $Y_i$ is constructed recursively by greedily expanding $Y_{i+1}$ with nodes to obtain a $2^i$-net, for $i = \log\Delta - 1, \log\Delta - 2, \ldots, 0$.

Following this construction, we have

$$Y_{\log\Delta} \subseteq Y_{\log\Delta - 1} \subseteq \cdots \subseteq Y_0 = V. \tag{1}$$

For any node $u \in V$, we define its *zooming sequence* as follows: (i) Let $u(0) = u$; (ii) recursively define $u(i)$ be the nearest node to $u(i-1)$ in $Y_i$ for $i$ from 1 to $\log\Delta$ (if there are several such nearest nodes, use some arbitrary tie-breaking mechanism, e.g., the least node id). By the definition of $r$-net, for any $u \in V$ and any $i \in [\log\Delta]$, we have

$$\sum_{k=1}^{i} d(u(k-1), u(k)) \leq \sum_{k=1}^{i} 2^k < 2^{i+1}. \tag{2}$$

We can now form a tree $T$ of $V$ by building a path along the zooming sequence $\langle u(0), u(1), \cdots, u(\log\Delta)\rangle$, for each node $u \in V$. For each virtual edge $(u(i), u(i+1))$ of the tree $T$ with $u(i) \neq u(i+1)$, let $u(i)$ store the label of $u(i+1)$. By Eqn. (1) and the fact that $u(i) \neq u(i+1)$, we have that $u(i) \in Y_k$ for all $k \leq i$ and $u(i) \notin Y_k$ for all $k > i$. Thus the node $u(i)$ stores only one zooming sequence label, namely that of its parent $u(i+1)$ in the tree $T$. Each node $u \in V$ can now route packets along its zooming sequence by using the underlying labeled routing scheme.

We next introduce the concept of *ball packing*. Let $r_u(s)$ denote the radius of the ball centered at $u$ with size $s$, i.e. $|B_u(r_u(s))| = s$. The following lemma is proved in [14]:

**Lemma 3.4 (Ball Packing [14])** *For any positive integer $s \leq n$, there exists an $s$-size ball packing $\mathcal{B}$ of $G$, i.e. a (maximal) set of non-intersecting balls, such that*

1. *For any ball $B \in \mathcal{B}$, $|B| = s$.*
2. *For any node $u$, there exists a ball $B \in \mathcal{B}$ centered at $c$ such that the radius of $B$ is at most $r_u(s)$ (i.e., $r_c(s) \leq r_u(s)$) and $d(u,c) \leq 2r_u(s)$.*

[1]For any integer $x > 0$, let $[x]$ denote the set $\{0, 1, \cdots, x\}$.

Let $\mathcal{B}_j$ be a $2^j$-size ball packing, for all $j \in [\log n]$, and $\mathcal{C}_j$ be the set of centers of all balls in $\mathcal{B}_j$. For each node $u \in V$ and $j \in [\log n]$, let $B(u,j)$ denote the nearest ball in $\mathcal{B}_j$ to $u$, i.e. $\max_{v \in B(u,j)} d(u,v)$ is minimized; let $c(u,j)$ denote the center of ball $B(u,j)$. Node $u$ maintains a link to $c(u,j)$ by storing its routing label, for all $j \in [\log n]$. In addition, the following lemma states that there is a packing ball near a node.

**Lemma 3.5** *For any $u \in V$, any real numbers $r' > r > 0$, and any positive integer $j \leq \log \frac{|B_u(r')|}{(4r'/r)^\alpha}$, there exists a packing ball $B \in \mathcal{B}_j$ of radius at most $r$ and such that $B \subseteq B_u(r'+3r)$. Moreover $B(u,j) \subseteq B_u(r'+3r)$.*

**Proof:** Let $Y$ be an *$r$-net* of $B_u(r')$. By the definition of $r$-net, the union of balls $B_x(r)$ for all $x \in Y$ covers the ball $B_u(r')$. By Lemma 3.3, we have $|B_u(r') \cap Y| \leq (4r'/r)^\alpha$. Thus by the pigeonhole principle, there exists a ball $B_x(r)$, for some $x \in Y$, such that $|B_x(r)| \geq |B_u(r')|/(4r'/r)^\alpha$. Thus by Lemma 3.4, for any $j \leq \log \frac{|B_u(r')|}{(4r'/r)^\alpha}$, there exists a ball $B \in \mathcal{B}_j$ with center $c$ such that $d(x,c) \leq 2r$ and the radius of $B$ is no more than $r$. Since $d(u,x) \leq r'$, we have $B \subseteq B_u(r'+3r)$. Moreover since $B(u,j)$ is the nearest ball to $u$ in $\mathcal{B}_j$ , we have $B(u,j) \subseteq B_u(r'+3r)$. ■

## 3.3 Search tree.

Finally we give the definition of a search tree on any fixed ball, used to store and retrieve information.

**Definition 3.6 (Search Tree)** *For any $\epsilon \in (0,1)$ and any ball $B_c(r)$, let $X_0 = \{c\}$, and for $1 \leq i \leq \log(\epsilon r)$ let $X_i$ be an $(\epsilon r/2^i)$-net of $B_c(r) \setminus \bigcup_{0 \leq j < i} X_j$. Then the search tree on $B_c(r)$, denoted by $T(c,r)$, is formed by connecting each node $v \in X_i$ to its closest node in $X_{i-1}$ for $0 < i \leq \log(\epsilon r)$, and defining the weight on each edge $(u,v)$ to be $d(u,v)$.*

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

$$r + \sum_{i=1}^{\log(\epsilon r)} 2^{\log(\epsilon r) - i} \leq (1+\epsilon) r. \tag{3}$$

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 $\{X_i\}$ is a partition of $B_c(r)$, and by Lemma 3.3 the root has the maximum degree in the tree, $\left(\frac{4r}{2^{\log(\epsilon r)-1}}\right)^\alpha = \left(\frac{1}{\epsilon}\right)^{O(\alpha)}$. Hence each node keeps $\left(\frac{1}{\epsilon}\right)^{O(\alpha)}$ labels for the search tree.

With typical search tree techniques, we can use our search tree to store a collection of $(key, data)$ pairs evenly in the nodes of the tree, and provide a search procedure that starts from and reports back to the root to search the *data* by the given *key* with cost $2(1+\epsilon)r$. For simplicity, we use $SearchTree(key, B_c(r))$ denote the search procedure if there is a search tree on $B_c(r)$. For detailed description of the *store* and *search* procedures in search trees, please refer to our previous paper [14].

For our name-independent schemes we take the node id as the *key* and the node label as the *data*.

# 4. COUNTING LEMMA

In this section we present a counting lemma that combines the properties of both $r$-nets and ball packings and is used to limit number of nodes with slack on either storage or stretch. To prove the counting lemma, we need the following result whose proof is in the Appendix.

**Lemma 4.1** *Let $(V,d)$ be an $n$-point metric space with minimum distance 1, and $G = (V,E)$ be an undirected graph on $V$. If for every node $x \in V$ and every integer $i \geq 0$, the number of edges $(x,y) \in E$ incident to $x$ with $2^i \leq d(x,y) < 2^{i+1}$ is no more than $c$, then we have $|E| \leq 3cn \log n$.*

**Lemma 4.2 (Counting)** *For any $j \in [\log n]$ and any $i \in [\log \Delta]$, let $\mathcal{D}_{i,j}$ be the set of nodes $u \in Y_i$ with $|B_u(2^i f) \setminus B_c(2^{i+2})| \geq (4f)^\alpha 2^j$, where $f > 1$ is a constant and $c = c(u,j)$. We have*

$$\sum_{i=0}^{\log \Delta} |\mathcal{D}_{i,j}| = (O(f))^{3\alpha} \cdot (n \log n / 2^j).$$

**Proof:** By Lemma 3.5 with $r' = 2^i f$, $r = 2^i$ and $|B_u(2^i f) \setminus B_c(2^{i+2})| \geq (4f)^\alpha 2^j$, there exists a packing ball $B' \in \mathcal{B}_j$ with center $c'$ such that $B' \subseteq B_u(2^i(f+3))$, the radius of $B'$ is no more than $2^i$, and $B(u,j) \subseteq B_u(2^i(f+3))$. Thus we have

$$d(c,c') \leq d(u,c) + d(u,c') \leq 2^{i+1}(f+3). \tag{4}$$

Moreover, since we applied Lemma 3.5 while excluding the nodes of $B_c(2^{i+2})$, by a more careful argument, we have

$$d(c',c) \geq 2^{i+2} - 2 \cdot 2^i = 2^{i+1}. \tag{5}$$

Fix any such $c' \in \mathcal{C}_j$, and denote it by $\phi(u,j)$.

Let $E' = \{(c(u,j), \phi(u,j)) \mid \forall u \in \mathcal{D}_{i,j}, \forall i \in [\log \Delta]\}$, and define a graph $G' = (\mathcal{C}_j, E')$. We now show that $G'$ satisfies the condition of Lemma 4.1, i.e. for any node $x \in \mathcal{C}_j$, the number of edges $(x,y) \in E'$ incident on $x$ with $2^k \leq d(x,y) < 2^{k+1}$, for every integer $k \geq 0$, is a constant $O(\log f)O(f)^\alpha$. First for a fixed $k$, the number of indices $i \in [\log \Delta]$ such that there exists $u \in Y_i$ with $2^k \leq d(c(u,j), \phi(u,j)) < 2^{k+1}$ is a constant $O(\log f)$, since $2^{i+1} \leq d(c(u,j), \phi(u,j)) \leq 2^{i+1}(f+3)$ by Eqn. (4) and (5). Second, for a fixed $x \in \mathcal{C}_j$ and a fixed $i \in [\log \Delta]$, the number of $u \in Y_i$ with either $c(u,j) = x$ or $\phi(u,j) = x$ is a constant $O(f)^\alpha$, by Lemma 3.3 with Eqn. (4) and $u \in Y_i$. Hence the number of edges $(x,y) \in E'$ incident on $x$ with $2^k \leq d(x,y) < 2^{k+1}$ is a constant $O(\log f)(O(f))^\alpha$. Therefore, by Lemma 4.1 we have $|E'| = O(\log f)(O(f))^\alpha |\mathcal{C}_j| \log |\mathcal{C}_j|$.

Furthermore, by a similar argument, one can show that for a fixed edge $(x,y) \in E'$, the number of $u \in \sum_{i=0}^{\log \Delta} \mathcal{D}_{i,j}$ with $(c(u,j), \phi(u,j)) = (x,y)$ is at most $O(\log f)(O(f))^\alpha$. Thus we obtain

$$\sum_{i=0}^{\log \Delta} |\mathcal{D}_{i,j}| \leq O(\log f)(O(f))^\alpha |E'| < (O(f))^{3\alpha} |\mathcal{C}_j| \log |\mathcal{C}_j|.$$

Finally, since $\mathcal{B}_j$ is a $2^j$-size packing, we have $|\mathcal{C}_j| = |\mathcal{B}_j| = n/2^j$. The lemma then follows. ■

# 5. ROUTING SCHEME WITH SLACK ON STORAGE

In this section, we present our name-independent routing scheme with relaxed storage guarantees.

## 5.1 Data structure

First, as defined in Section 3, we have $Y_i$, a $2^i$-net, for $i \in [\log \Delta]$, and each node $u$ can route packets along its zooming sequence.

Second, let $j_0 = \log(\log n/(\delta\epsilon^{c_1\alpha}))$, for a constant $c_1 = 7$, and only maintain the $j_0$-th level of ball packings $\mathcal{B}_{j_0}$; for simplicity, denote $\mathcal{B} = \mathcal{B}_{j_0}$, $\mathcal{C} = \mathcal{C}_{j_0}$, and $c(u) = c(u, j_0)$ for all $u \in V$. Let each node in $\mathcal{C}$ have large memory and store the $(id, label)$ pairs of all the nodes in $V$. Since $\mathcal{B}$ is a $(\log n/(\delta\epsilon^{c_1\alpha}))$-size ball packing, we have

$$|\mathcal{B}| = |\mathcal{C}| \le n\delta\epsilon^{c_1\alpha}/\log n. \tag{6}$$

Third, let each node $u \in V$ store the $(id, label)$ pairs of nodes in the ball $B_u(r_u)$, where $r_u$ is the radius of the ball $B_u(r_u)$ with size $\log n/(\delta\epsilon^{c_2\alpha})$ for a constant $c_2 = 9$, i.e. $r_u$ is an abbreviation of $r_u(\log n/(\delta\epsilon^{c_2\alpha}))$.

Finally, for any node $u \in V$ and $i = \log(\epsilon r_u)$, if $u \in Y_i$, let $u$ store the $(id, label)$ pairs of nodes in $B_u(2^i/\epsilon^2) \setminus B_{c(u)}(2^{i+2})$. Let $\mathcal{D}_{i,j_0}$ as defined in Lemma 4.2 with $f = 1/\epsilon^2$, i.e. $\mathcal{D}_{i,j_0}$ is the set of nodes $u \in Y_i$ with $|B_u(2^i/\epsilon^2) \setminus B_{c(u)}(2^{i+2})| \ge (4/\epsilon^2)^\alpha \log n/(\delta\epsilon^{c_1\alpha})$. Let $\mathcal{D} = \sum_{i=0}^{\log\Delta} \mathcal{D}_{i,j_0}$. By Lemma 4.2, we have

$$|\mathcal{D}| = (O(1/\epsilon^2))^{3\alpha} \cdot n \log n/2^{j_0} \le \delta n/2, \tag{7}$$

where the last inequality follows for $j_0 = \log(\log n/(\delta\epsilon^{c_1\alpha}))$ and $c_1 = 7$.

**Lemma 5.1 (Storage)** *In the above scheme, there are $\delta n$ nodes with $O(n \log n)$-bit storage each, and $(1-\delta)n$ nodes with $\left(\log^2 n/\epsilon^{O(\alpha)}(1/\delta + \log n)\right)$-bit storage each.*

**Proof:** There are two kinds of nodes with large storage: the nodes in $\mathcal{C}$ storing the $(id, label)$ pairs of all nodes in the network, and the nodes $u \in \mathcal{D}$ storing the $(id, label)$ pairs of more than $\log n/(\delta\epsilon^{O(\alpha)})$ nodes. By Eqn. (6) and (7), the number of nodes in $\mathcal{C} \cup \mathcal{D}$ is no more than $\delta n$.

All the remaining nodes, the nodes in $V \setminus (\mathcal{C} \cup \mathcal{D})$, store at most $\log n/(\delta\epsilon^{O(\alpha)})$ pairs $(id, label)$ each.

Since both the node id and label have size $\log n$ bits, we have $\delta n$ nodes with $O(n \log n)$-bit storage each, and $(1-\delta)n$ nodes with $\left(\log^2 n/(\delta\epsilon^{O(\alpha)})\right)$-bit storage each. Thus together with the storage requirement in Lemma 3.1, the lemma follows. ■

## 5.2 Routing algorithm

Now we present our routing algorithm in full. Assume that a source node $u$ wants to send a packet to a destination node $v$, given the id $id(v)$ of $v$.

The routing procedure is described in Algorithm 1. The algorithm searches for the label $l(v)$ according to the id $id(v)$ along the zooming sequence of $u$, until it finds the label $l(v)$ of $v$ stored at the current node $u(i)$ or $i \ge \log(\epsilon r_{u(i)})$. In the latter case, the algorithm goes to $c(u(i))$, which stores the $(id, label)$ pairs of all nodes in the network, and then it goes to $v$ from $c(u(i))$ using the underlying labeled routing scheme, and terminates.

**Algorithm 1** Name-independent routing with relaxed guarantees on storage

1: set $i \leftarrow 0$
2: **if** $u(i)$ has the routing label $l(v)$ of $v$ **then**
3: go to $v$ from $u(i)$ using the underlying labeled routing scheme, and terminate
4: **end if**
5: **if** $i \ge \log(\epsilon r_{u(i)})$ **then**
6: go to $c(u(i))$ from $u(i)$ using the underlying labeled routing scheme
7: get the label $l(v)$ at $c(u(i))$ according to the id $id(v)$
8: go to $v$ from $c(u(i))$ using the underlying labeled routing scheme, and terminate
9: **end if**
10: go to $u(i+1)$ from $u(i)$ using the underlying labeled routing scheme
11: increase $i$ by one, and go back to Step 2

**Lemma 5.2 (Stretch)** *For any source node $u$ and any destination node $v$ in $G$, the total routing cost of Algorithm 1 is no more than $(1 + O(\epsilon))d(u, v)$.*

**Proof:** Let $t$ be the index of the level at which $v$'s routing label $l(v)$ is found either at $u(t)$ in Step 2, or at $c(u(t))$ in Step 7.

Since the label of $v$ is not found at the $(t-1)$-th iteration, we have $t - 1 < \log(\epsilon r_{u(t-1)})$; otherwise it would have been found in Step 7 at Iteration $t-1$. Hence by $v \notin B_{u(t-1)}(r_{u(t-1)})$, we have $d(u(t-1), v) \ge 2^{t-1}/\epsilon$. Thus by Eqn. (2) and the triangle inequality, we have

$$\begin{aligned} d(u, v) &\ge d(u(t-1), v) - \sum_{i=1}^{t-1} d(u(i-1), u(i)) \\ &\ge 2^{t-1}(1/\epsilon - O(1)). \end{aligned} \tag{8}$$

Now consider the routing cost. First, if $l(v)$ is found at $u(t)$ in Step 2, by Eqn. (2) and the triangle inequality, the routing cost is at most

$$\begin{aligned} &\sum_{i=1}^{t} d(u(i-1), u(i)) + d(u(t), v) \\ &\le d(u, v) + 2\sum_{i=1}^{t} d(u(i-1), u(i)) \\ &\le d(u, v) + 2^{t+2} \\ &\le (1 + O(\epsilon))d(u, v), \end{aligned} \tag{9}$$

where the last inequality follows from Eqn. (8).

Second, consider the case that $l(v)$ is found at $c = c(u(t))$ in Step 7. Since the *if* condition of Step 5 is satisfied, we have $t \ge i_0$, for $i_0 = \log(\epsilon r_{u(t)})$. Thus $|B_{u(t)}(2^{i_0}/\epsilon)|/(4/\epsilon)^\alpha = \log n/(\delta\epsilon^{c_2\alpha})/(4/\epsilon)^\alpha \ge \log n/(\delta\epsilon^{c_1\alpha}) = 2^{j_0}$ for $c_2 - c_1 = 2$. Hence by Lemma 3.5 with $r' = 2^{i_0}/\epsilon$ and $r = 2^{i_0}$, we have

$$d(u(t), c) \le 2^{i_0}(1/\epsilon + 3). \tag{10}$$

Since $l(v)$ is not found at $u(t)$ in Step 2, we have $v \notin B_{u(t)}(2^{i_0}/\epsilon^2) \setminus B_c(2^{i_0+2})$. Hence if $v \in B_c(2^{i_0+2})$, by Eqn. (2)

and $t \geq i_0$, the routing cost is at most

$$\begin{aligned}
&\sum_{i=1}^{t} d(u(i-1),u(i)) + d(u(t),c) + d(c,v) \\
&\leq d(u,v) + 2\left(\sum_{i=1}^{t} d(u(i-1),u(i)) + d(c,v)\right) \qquad (11)\\
&\leq d(u,v) + 3\cdot 2^{t+2} \\
&\leq (1+O(\epsilon))d(u,v),
\end{aligned}$$

where the last inequality follows by Eqn. (8). If $v$ is not in the ball $B_{u(t)}(2^{i_0}/\epsilon^2)$, by Eqn. (2) and the triangle inequality, we have

$$\begin{aligned}
d(u,v) &\geq d(u(t),v) - \sum_{i=1}^{t} d(u(i-1),u(i)) \qquad (12)\\
&\geq 2^{i_0}/\epsilon^2 - 2^{t+1}.
\end{aligned}$$

Thus by Eqn. (2), (10) and the triangle inequality, the routing cost is at most

$$\begin{aligned}
&\sum_{i=1}^{t} d(u(i-1),u(i)) + d(u(t),c) + d(c,v) \\
&\leq d(u,v) + 2\left(\sum_{i=1}^{t} d(u(i-1),u(i)) + d(u(t),c)\right) \qquad (13)\\
&\leq d(u,v) + 2^{t+2} + 2^{i_0+1}(1/\epsilon + O(1)) \\
&\leq (1+O(\epsilon))d(u,v),
\end{aligned}$$

where the last inequality follows from Eqn. (8) and (12).

In conclusion, by Eqn. (9), (11) and (13), the lemma follows. ■

**Proof of Theorem 1.1:** The bounds on stretch follow from Lemma 5.2 and the bounds on storage follow from Lemma 5.1 and 3.1. ■

# 6. ROUTING SCHEME WITH SLACK ON STRETCH

In this section, we present our name-independent routing scheme with relaxed guarantees on stretch.

## 6.1 Data structure

Let $g_1 = \log^2 n/(\delta\epsilon^{c_1\alpha})$ and $g_2 = \log^2 n/(\delta\epsilon^{c_2\alpha})$ for constants $c_1 = 14$ and $c_2 = 10$.

First as defined in Section 3, we have $Y_i$, a $2^i$-net, for $i \in [\log\Delta]$, and each node $u$ can route packets along its zooming sequence.

Second for each $j \in [\log n]$, we maintain a search tree on each ball $B \in \mathcal{B}_j$ with center $c$, storing the $(id, label)$ pairs of nodes in $B_c(r_c(2^j g_1))$.

Third, for each $u \in Y_i$, $i \in [\log\Delta]$, if there is an index $j \in [\log n]$ such that $B(u,j) \subseteq B_u(2^i(1/\epsilon+3))$ and $B_u(2^i/\epsilon^2) \subseteq B_c(r_c(2^j g_1))$ as illustrated in Fig. 1(a), where $c = c(u,j)$ is the center of $B(u,j)$, we use the search tree on $B(u,j)$ to index the routing labels of nodes in $B_u(2^i/\epsilon^2)$, and w.l.o.g. assume such an index $j$ is minimized, denoted by $j(u,i) = j$. Otherwise set $j(u,i) = null$, and as in Fig. 1(b) we maintain a search tree on $B_u(2^i/\epsilon)$, storing the $(id, label)$ pairs of nodes in $B_u(2^i/\epsilon)$ and nodes in $B_u(r(u,i))\setminus B_{c'}(2^{i+2})$, where $c' = c(u,j')$ for $j' = \log(|B_u(2^i/\epsilon)|\cdot g_2)$, and $r(u,i)$ is the radius of $B_u(r(u,i))$ such that $|B_u(r(u,i)) \setminus B_{c'}(2^{i+2})| = |B_u(2^i/\epsilon)|\cdot g_1$. A *Search* procedure at $u$ based on the *SearchTree* procedure is described in Algorithm 2. Since

**Algorithm 2** Search(id,u,i)

```
if j(u,i) ≠ null then
    go to c(u,j(u,i)); SearchTree(id, B(u,j(u,i))); and report back to u
else
    SearchTree(id, B_u(2^i/ε))
end if
```

$B(u,j(u,i)) \subseteq B_u(2^i(1/\epsilon+3))$ if $j(u,i) \neq null$, the cost of the *Search* procedure is

$$\max(2\cdot 2^i(1/\epsilon+3), 2\cdot 2^i/\epsilon) = 2^{i+1}(1/\epsilon + O(1)). \qquad (14)$$

Finally, for each $i \in [\log\Delta]$ and each $j \in [\log n]$, let $\mathcal{D}_{i,j}$ be as defined in Lemma 4.2 with $f = 1/\epsilon^3$, i.e. $\mathcal{D}_{i,j}$ is the set of nodes $u \in Y_i$ with $|B_u(2^i/\epsilon^3)\setminus B_c(2^{i+2})| \geq (4/\epsilon^3)^\alpha 2^j$. Now we define the set of source nodes that we allow larger stretch, i.e. $9+O(\epsilon)$, by $\mathcal{D} = \{u \mid \exists i \in [\log\Delta] \text{ s.t. } u(i) \in \mathcal{D}_{i,j'} \text{ for } j' = \log(|B_{u(i)}(2^i/\epsilon)|\cdot g_2)\}$.

**Lemma 6.1** *The number of nodes in $\mathcal{D}$ is at most $\delta n$.*

**Proof:** First we have $\sum_{i=0}^{\log\Delta}|\mathcal{D}_{i,j}| \leq (O(1/\epsilon^3))^{3\alpha}\cdot n\log n/2^j$ by Lemma 4.2, for each $j$.

Second, for each $j \in [\log n]$, let $\mathcal{D}_j = \{u \mid \exists i \text{ s.t. } u(i) \in \mathcal{D}_{i,j} \text{ for } \log(|B_{u(i)}(2^i/\epsilon)|\cdot g_2) = j\}$. Since $d(u,u(i)) \leq 2^{i+1} < 2^i/\epsilon$ by Eqn. (2), for any fixed node $x \in Y_i$, we have $|\{u \mid u(i) = x\}| < |B_x(2^i/\epsilon)|$. Hence $|\mathcal{D}_j| \leq 2^j/g_2 \cdot \sum_{i=0}^{\log\Delta}|\mathcal{D}_{i,j}|$. Thus

$$\begin{aligned}
|\mathcal{D}_j| &\leq 2^j/g_2\cdot(O(1/\epsilon^3))^{3\alpha}\cdot n\log n/2^j \qquad (15)\\
&\leq \delta n/\log n,
\end{aligned}$$

where the last inequality follows from $g_2 = \log^2 n/(\delta\epsilon^{c_2\alpha})$ and $c_2 = 10$.

Since $\mathcal{D} = \bigcup_{j=0}^{\log n}\mathcal{D}_j$, we have $|\mathcal{D}| \leq \delta n$. ■

**Lemma 6.2** *For each node $v \in V$, the number of search trees that contain $v$ is $\log n/\epsilon^{O(\alpha)}$.*

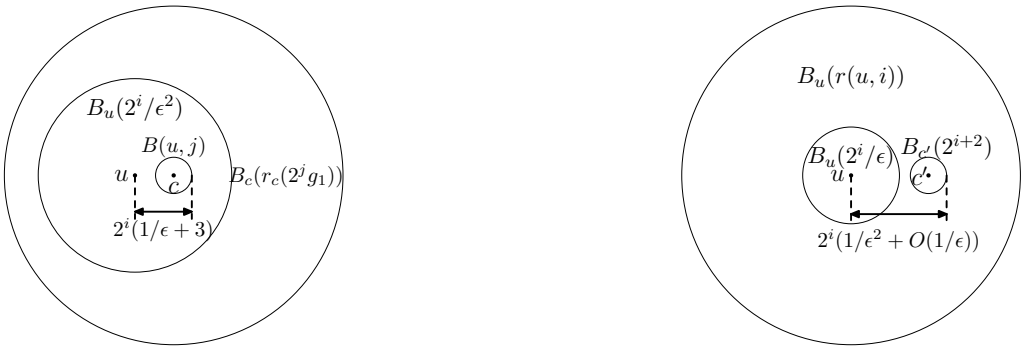
**Proof:** First, for each $j \in [\log n]$, the packing balls in $\mathcal{B}_j$ are disjoint, so at most one of them contains $v$. Thus the number of search trees for balls in $\mathcal{B}_j$, for all $j$, that contain $v$ is no more than $O(\log n)$.

Second, consider the search trees for the ball $B_u(2^i/\epsilon)$ that contain $v$, for $u \in Y_i$ and $j(u,i) = null$. Let $j$ be an integer such that $2^j \leq B_v(2^i) < 2^{j+1}$. By Lemma 3.4, there exists a ball $B \in \mathcal{B}_j$ such that $B \subseteq B_v(3\cdot 2^i)$. Since $v \in B_u(2^i/\epsilon)$, we have $B \subseteq B_u(2^i(1/\epsilon+3))$, and therefore $B(u,j) \subseteq B_u(2^i(1/\epsilon+3))$. Since we maintain a search tree for $B_u(2^i/\epsilon)$, we have $B_u(2^i/\epsilon^2) \nsubseteq B_c(r_c(2^j g_1))$, where $c = c(u,j)$. Thus $r_c(2^j g_1) \leq d(u,c) + 2^i/\epsilon^2 = 2^i(1/\epsilon^2 + O(1/\epsilon))$; otherwise, $B_u(2^i/\epsilon^2) \subseteq B_c(r_c(2^j g_1))$. Hence

$$d(v,c) + r_c(2^j g_1) \leq 2^i(1/\epsilon^2 + O(1/\epsilon)), \qquad (16)$$

which implies $B_c(r_c(2^j g_1)) \subseteq B_v(2^i(1/\epsilon^2 + O(1/\epsilon)))$. Therefore we have

$$\begin{aligned}
\frac{|B_v(2^i(1/\epsilon^2+O(1/\epsilon)))|}{|B_v(2^i)|} &\geq \frac{|B_c(r_c(2^j g_1))|}{|B_v(2^i)|} \qquad (17)\\
&\geq g_1/2 \\
&= \log^2 n/(\delta\epsilon^{O(\alpha)}).
\end{aligned}$$




(a) $j(u,i) \neq null$ (b) $j(u,i) = null$

**Figure 1.** (a) Node $u$ uses the search tree on $B(u,j)$ to index the routing information of nodes in $B_u(2^i/\epsilon^2)$, since $B(u,j) \subseteq B_u(2^i(1/\epsilon+3))$ and $B_u(2^i/\epsilon^2) \subseteq B_c(r_c(2^j g_1))$, where $j = j(u,i)$ and $c = c(u,j)$. (b) Node $u$ uses the search tree on $B_u(2^i/\epsilon)$ to index the routing information of nodes in $B_u(r(u,i)) \setminus B_{c'}(2^{i+2})$, where $j' = \log(|B_u(2^i/\epsilon)|g_2)$, $c' = c(u,j')$, and $|B_u(r(u,i)) \setminus B_{c'}(2^{i+2})| = |B_u(2^i/\epsilon)|g_1$.

By a simple calculation, for a fixed node $v$, the number of indices $i$ that satisfy the above inequality is at most $(1/\epsilon^2 + O(1/\epsilon)) \cdot \frac{\log n}{\log(g_1/2)} < O(\log n/\epsilon^2)$. Furthermore, for a fixed $i$, the number of nodes $u \in Y_i$ with $B_u(2^i/\epsilon)$ containing $v$ is no more than $(1/\epsilon)^{O(\alpha)}$ by Lemma 3.3. Therefore the number of search trees on balls $B_u(2^i/\epsilon)$ containing $v$, for all $u \in Y_i$ and all $i \in [\log \Delta]$, is no more than $\log n/\epsilon^{O(\alpha)}$. ■

**Lemma 6.3 (Storage)** *In the above scheme, each node has a* $\left(\log^4 n/(\delta\epsilon^{O(\alpha)})\right)$*-bit routing table.*

**Proof:** Note that each node in any search tree stores $g_1 = \log^2 n/(\delta\epsilon^{O(\alpha)})$ pairs $(id, label)$. By Lemma 6.2, the number of search trees containing a fixed node is $\log n/\epsilon^{O(\alpha)}$. Since both the node id and label have size $\log n$ bits, each node stores $\left(\log^4 n/(\delta\epsilon^{O(\alpha)})\right)$-bit routing information. ■

## 6.2 Routing algorithm

Now we present our routing algorithm in full. Assume that a source node $u$ wants to send a packet to a destination node $v$, given the id $id(v)$ of $v$.

The routing procedure is described in Algorithm 3 and illustrated in Fig. 2. The algorithm searches for the label $l(v)$ according to the id $id(v)$ along the zooming sequence of $u$, until it finds the label of $v$ either in Step 3 or in Step 9. Then it routes the packet to $v$ using the underlying labeled scheme by $l(v)$.

**Lemma 6.4 (Stretch)** *Algorithm 3 guarantees* $(1+O(\epsilon))$ *stretch for* $(1-\delta n)n$ *source nodes, and* $(9+O(\epsilon))$ *stretch for the remaining* $\delta n$ *source nodes.*

**Proof:** First we bound the cost of each step (the Claim is proved latter):

**Claim 6.5** *For each iteration* $i$*, we have*

$$\begin{aligned} \textit{The cost of Step 3} &= 2^{i+1}(1/\epsilon + O(1)), && (18)\\ \textit{The cost of Step 9} &= O(2^i), && (19)\\ \textit{The cost of Step 8} &\le 2^i(1/\epsilon^2 + O(1/\epsilon)) && (20)\\ \textit{The cost of Step 13} &\le 2^i(1/\epsilon^2 + O(1/\epsilon)). && (21) \end{aligned}$$

**Algorithm 3** Name-independent routing with relaxed guarantees on stretch

1: set $i \leftarrow 0$ {*Initial Step*}
2: set $\ell \leftarrow 1$ {*Set the step increment*}
3: *Search*$(id(v), u(i), i)$ {*Search at* $u(i)$ *with cost* $2^i(1/\epsilon + O(1))$ *as in Fig. 2*}
4: **if** the label $l(v)$ is found **then**
5: go to $v$ from $u(i)$; and terminate the algorithm
6: **end if**
7: **if** $j(u(i), i) = null$ and $u(i) \notin \mathcal{D}_{i,j'}$, for $j' = \log(|B_{u(i)}(2^i/\epsilon)| \cdot g_2)$ **then**
8: set $c' \leftarrow c(u(i), j')$, and $i' \leftarrow i - (\log(1/\epsilon) - 3)$; go to $c'$ from $u(i)$
9: go to $c'(i')$ along the zooming sequence of $c'$; *Search*$(id(v), c'(i'), i')$; and report back to $c'$ {*Search in* $B_{c'}(2^{i+2})$ *with cost* $O(2^i)$ *as in Fig. 2(b)*}
10: **if** the label $l(v)$ is found **then**
11: go to $v$ from $c'$; and terminate the algorithm
12: **end if**
13: go back to $u(i)$ from $c'$; and set $\ell \leftarrow \log(1/\epsilon) + 1$ {*Set the step increment to* $\log(1/\epsilon) + 1$}
14: **end if**
15: go to $u(i+\ell)$ from $u(i)$ along the zooming sequence
16: set $i \leftarrow i + \ell$; and repeat Step 2

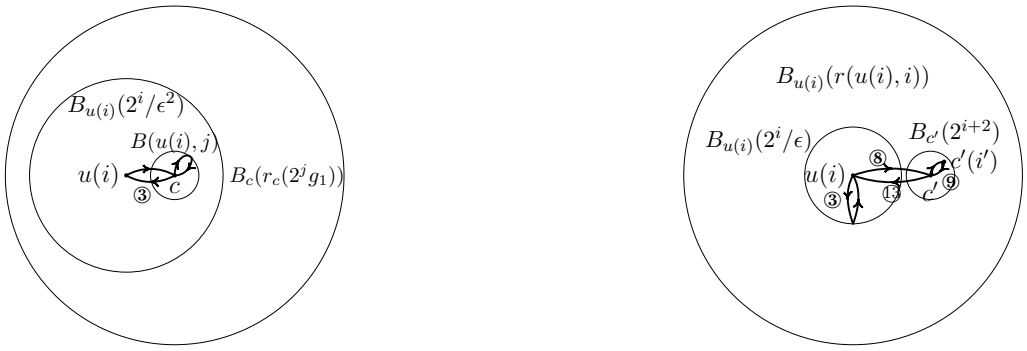
Moreover since $(20)+(19)+(21) = 2^{i+1}(1/\epsilon^2 + O(1/\epsilon)) = 2^{(i+\log(1/\epsilon))+1}(1/\epsilon + O(1))$, we can consider the cost for Step 8, 9 and 13 at Iteration $i$ as the cost of Eqn. (18) at Iteration $i + \log(1/\epsilon)$, which we skip by setting the step increment $\ell = \log(1/\epsilon) + 1$ in Step 13.

Now let $t$ be the index of the level at which $v$'s label $l(v)$ is found. Since $l(v)$ is not found before the $t$-th iteration, we have the following claim, whose proof is provided latter.

**Claim 6.6** *If the label of* $v$ *is not found before the* $t$*-th iteration, we have*

$$d(u,v) \ge \begin{cases} 2^{t-1}(1/\epsilon^2 - O(1)) & \text{if } u \notin \mathcal{D} \\ 2^{t-1}(1/\epsilon - O(1)) & \text{for all } u. \end{cases} \quad (22)$$

Note that the label of $v$ can be found either at $u(t)$ in Step 3, or at $c'$ in Step 9, where $c' = c(u(t), j')$ and $j' = \log(|B_{u(t)}(2^t/\epsilon)| \cdot g_2)$.




(a) $j(u(i),i) \neq null$ (b) $j(u(i),i) = null$

**Figure 2.** (a) If $j(u(i),i) \neq null$, Step ③ is executed with cost $2^{i+1}(1/\epsilon + O(1))$, where $j = j(u(i),i)$ and $c = c(u(i),j)$. (b) If $j(u(i),i) = null$, Step ③ is executed with cost $2^{i+1}(1/\epsilon + O(1))$. If the label $l(v)$ is not found and $u(i) \notin \mathcal{D}_{i,j'}$, execute Step ⑧ with cost at most $2^i(1/\epsilon^2 + O(1/\epsilon))$ and Step ⑨ with cost $O(2^i)$. If the label $l(v)$ is still not found, Step ⑬ is executed with cost at most $2^i(1/\epsilon^2 + O(1/\epsilon))$. Here $j' = \log(|B_{u(i)}(2^i/\epsilon)| \cdot g_2)$, $c' = c(u(i),j')$, and $i' = i - (\log(1/\epsilon) - 3)$.

If $l(v)$ is found in Step 3, by Eqn. (2), (18) and the triangle inequality, the routing cost is at most

$$\begin{aligned}
&\sum_{i=0}^{t} 2^{i+1}(1/\epsilon + O(1)) + d(u(t),v) \\
&\leq \sum_{i=0}^{t} 2^{i+1}(1/\epsilon + O(1)) + d(u,v) + \sum_{i=1}^{t} d(u(i),u(i-1)) \\
&= d(u,v) + 2^{t+2}(1/\epsilon + O(1)),
\end{aligned} \tag{23}$$

which implies $(1 + O(\epsilon))$ stretch for source nodes $u \notin \mathcal{D}$ and $(9 + O(\epsilon))$ stretch for the remaining source nodes $u$ by Claim 6.6.

Consider the case that $l(v)$ is found at $c'$ in Step 9. Since $j(u(t),t) = null$ and $l(v)$ is not found in Step 3, we have $v \notin B_{u(t)}(r(u(t),t)) \setminus B_{c'}(2^{t+2})$. If $v \in B_{c'}(2^{t+2})$, by Eqn. (2), (18) and (19), the routing cost is at most

$$\begin{aligned}
&\sum_{i=0}^{t} 2^{i+1}(1/\epsilon + O(1)) + d(u(t),c') + O(2^t) + d(c',v) \\
&\leq 2^{t+2}(1/\epsilon + O(1)) + d(u,v),
\end{aligned} \tag{24}$$

which implies $(1 + O(\epsilon))$ stretch for source nodes $u \notin \mathcal{D}$ and $(9 + O(\epsilon))$ stretch for the remaining source nodes $u$ by Claim 6.6. In addition, since $i' = t - (\log(1/\epsilon) - 3)$ as in Step 8, we have $d(c', c'(i')) + 2^{t+2} \leq 2^{i'}/\epsilon$, i.e. $B_{c'}(2^{t+2}) \subseteq B_{c'(i')}(2^{i'}/\epsilon)$. Thus if $v \in B_{c'}(2^{t+2})$, the $Search(id(v), c'(i'), i')$ subprocedure in Step 9 returns the label $l(v)$.

Now consider the case of $v \notin B_{u(t)}(r(u(t),t))$. Note that the *if* condition in Step (7) is satisfied in order to run Step 9. Thus $u(t) \notin \mathcal{D}_{t,j'}$, i.e. we have $|B_{u(t)}(2^t/\epsilon^3) \setminus B_{c'}(2^{t+2})| \leq (4/\epsilon^3)^\alpha 2^{j'} \leq (4/\epsilon^3)^\alpha |B_{u(t)}(2^t/\epsilon)| \cdot g_2 \leq |B_{u(t)}(2^t/\epsilon)| \cdot g_1$ for $c_1 - c_2 = 4$, which implies $r(u(t),t) \geq 2^t/\epsilon^3$. Thus by Eqn. (2) and the triangle inequality, we have

$$\begin{aligned}
d(u,v) &\geq d(u(t),v) - 2^{t+1} \\
&\geq r(u(t),t) - 2^{t+1} \\
&\geq 2^t(1/\epsilon^3 - 2).
\end{aligned} \tag{25}$$

Hence since $d(c',v) \leq d(u,v) + 2^{t+1} + d(u(t),c')$, by Eqn. (18), (19), (20) and (21), the routing cost is at most

$$\begin{aligned}
&\sum_{i=0}^{t} 2^{i+1}(1/\epsilon + O(1)) + d(u(t),c') + O(2^t) + d(c',v) \\
&\leq d(u,v) + 2^{t+1}(1/\epsilon^2 + O(1/\epsilon)) \\
&\leq (1 + O(\epsilon))d(u,v),
\end{aligned} \tag{26}$$

where the last inequality follows from Eqn. (25).

Since $|\mathcal{D}| \leq \delta n$ by Lemma 6.1, the lemma follows from Eqn. (23), (24) and (26). ■

**Proof of Claim 6.5:** The cost of Step 3 follows directly from Eqn. (14).

By Eqn. (2), (14) and $i' = i - (\log(1/\epsilon) - 3)$ in Step 8, the cost of Step 9 is at most

$$2\sum_{k=1}^{i'} d(c'(k-1), c'(k)) + 2^{i'+1}(1/\epsilon + O(1)) = O(2^i). \tag{27}$$

Now consider the cost of Step 8 and 13, i.e. $d(u(i),c')$. Since the *if* condition at Step 7 is satisfied, we have $j(u(i),i) = null$, which implies that $B_{u(i)}(2^i/\epsilon^2)$ is large relative to $B_{u(i)}(2^i/\epsilon)$. Specifically, by the pigeonhole principle and a similar argument as in Lemma 3.5, we have

$$|B_{u(i)}(2^i(1/\epsilon^2 + O(1/\epsilon)))| \geq \frac{|B_{u(i)}(2^i/\epsilon)|}{(4/\epsilon)^\alpha} g_1. \tag{28}$$

Thus $2^{j'} = |B_{u(i)}(2^i/\epsilon)| \cdot g_2 \leq \frac{|B_{u(i)}(2^i/\epsilon)| \cdot g_1}{(4/\epsilon)^\alpha \cdot (4(1/\epsilon^2 + O(1/\epsilon)))^\alpha} \leq |B_{u(i)}(2^i(1/\epsilon^2 + O(1/\epsilon)))|/(4(1/\epsilon^2 + O(1/\epsilon)))^\alpha$, for $c_1 - c_2 = 4$. Hence by applying Lemma 3.5 with $r' = 2^i(1/\epsilon^2 + O(1/\epsilon))$ and $r = 2^i$, we have $B(u(i),j') \subseteq B_{u(i)}(2^i(1/\epsilon^2 + O(1/\epsilon)))$. Thus it follows Eqn. (20) and (21). ■

**Proof of Claim 6.6:** First, if the $(t-1)$-th iteration is skipped because of setting $\ell = \log(1/\epsilon) + 1$ at Step 13, the last iteration run before the $t$-th iteration is Iteration $t - (\log(1/\epsilon) + 1)$. By Eqn. (25), we have

$$d(u,v) \geq 2^{t-(\log(1/\epsilon)+1)}(1/\epsilon^3 - 2) = 2^{t-1}(1/\epsilon^2 - O(1/\epsilon)). \tag{29}$$

Second, consider the case where the $(t-1)$-th iteration is executed and the label $l(v)$ is not found at this iteration. If $j(u(t-1), t-1) \neq null$, the *Search* procedure in Step 3 guarantees $v \notin B_{u(t-1)}(2^{t-1}/\epsilon^2)$ by running the Step 2 of Algorithm 2, which implies

$$d(u,v) \geq d(u(t-1),v) - O(2^t) \geq 2^{t-1}(1/\epsilon^2 - O(1)). \quad (30)$$

If $j(u(t-1), t-1) = null$, the *Search* procedure in Step 3 guarantees $v \notin B_{u(t-1)}(2^{t-1}/\epsilon)$ by running the Step 4 of Algorithm 2, which implies

$$d(u,v) \geq d(u(t-1),v) - O(2^t) \geq 2^{t-1}(1/\epsilon - O(1)). \quad (31)$$

Note that the *if* condition in Step 7 is not satisfied; otherwise Step 13 would be executed and Iteration $t$ would be skipped. Thus if $j(u(t-1), t-1) = null$, we have $u(t-1) \in \mathcal{D}_{t-1,j'}$ for $j' = \log(|B_{u(t-1)}(2^{t-1}/\epsilon)| \cdot g_2)$, which implies $u \in \mathcal{D}$. Therefore the claim follows from Eqn. (29), (30) and (31). ■

**Proof of Theorem 1.2:** The bounds on stretch follow from Lemma 6.4 and the bounds on storage follow from Lemma 6.3 and 3.1. ■

## 7. FUTURE WORK

Abraham et. al [5] show that any name-independent routing scheme for general graph with $o((n/(9k))^{1/k})$-bit storage at each node has *average stretch* $k/4 + 7/8$. Hence, an interesting question is whether a constant-stretch name-independent compact routing scheme for general graphs with relaxed guarantees exists. Furthermore, in the labeled routing model, it may be interesting to investigate whether we can achieve a $(1+\epsilon)$-stretch labeled routing scheme for general graphs with relaxed guarantees. The strongest lower bound result for labeled routing in general graphs states that a labeled scheme with stretch $< 3$ requires $O(n^2)$-bit storage at some nodes [19].

## 8. REFERENCES


[1] I. Abraham, Y. Bartal, J. Kleinberg, T.-H. Chan, O. Neiman, K. Dhamdhere, A. Slivkins, and A. Gupta. Metric embeddings with relaxed guarantees. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 83–100, 2005.

[2] I. Abraham, Y. Bartal, and O. Neiman. Advances in metric embedding theory. In *Proceedings of the 38th ACM Symposium on Theory of Computing*, pages 271–286, 2006.

[3] I. Abraham, Y. Bartal, and O. Neiman. Embedding metrics into ultrametrics and graphs into spanning trees with constant average distortion. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, pages 502–511, 2007.

[4] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi. Routing in networks with low doubling dimension. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 75, 2006.

[5] I. Abraham, C. Gavoille, and D. Malkhi. On space-stretch trade-offs: Lower bounds. In *Proceedings of the 18th ACM Symposium on Parallel Algorithms and Architecture*, pages 207–216, 2006.

[6] H. T.-H. Chan, M. Dinitz, and A. Gupta. Spanners with slack. In *Proceedings of the 14th European Symposium on Algorithms*, volume 4168 of *Lecture Notes in Computer Science*, pages 196–207, 2006.

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

[8] M. Dinitz. Compact routing with slack. In *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing*, 2007.

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

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

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

[12] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 444–453, 2004.

[13] G. Konjevod, A. Richa, and D. Xia. Optimal-stretch name-independent compact routing in doubling metrics. In *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing*, pages 198–207, 2006.

[14] G. Konjevod, A. Richa, and D. Xia. Optimal scale-free compact routing schemes in networks of low doubling dimension. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, pages 939–948, 2007.

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

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

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

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

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


## 9. APPENDIX

**Proof of Lemma 4.1:** Partition $E$ by defining $E_i = \{(x,y) \in E \mid 2^i \leq d(x,y) < 2^{i+1}\}$ for all $i > 0$. Now either $\sum_i |E_{3i}| \geq |E|/3$, or $\sum_i |E_{3i+1}| \geq |E|/3$, or $\sum_i |E_{3i+2}| \geq |E|/3$. W.l.o.g., assume the first inequality is true.

A *partition* $\mathcal{P}$ of $V$ is a collection of nonempty disjoint subsets of $V$ with $\bigcup_{S\in\mathcal{P}} S = V$. We now construct a sequence of partitions $\mathcal{P}_{-1}, \mathcal{P}_0, \mathcal{P}_1, \cdots$ of $V$, so that the following is true:

($\star$) For any $\mathcal{P}_i$ in the sequence, any $S \in \mathcal{P}_i$, and any $x, y \in S$, we have $d(x, y) < 2^{3(i+1)}$.

To start with, let $\mathcal{P}_{-1} = \big\{\{x\} \mid x \in V\big\}$. Given the partition $\mathcal{P}_{i-1}$, we construct $\mathcal{P}_i$ as follows. Set $\mathcal{P} = \mathcal{P}_{i-1}$. Let $S \in \mathcal{P}$ be the largest set in $\mathcal{P}$ (if there are multiple such sets, choose an arbitrary one from them as $S$). Let

$$\mathcal{N}(S) = \{K \in \mathcal{P} \mid \exists \text{ an edge } (x, y) \in E_{3i} \text{ s.t. } x \in S \text{ and } y \in K\}.$$

We define $\widetilde{S}$ as the union of $S$ and the sets in $\mathcal{N}(S)$, i.e., $\widetilde{S} = S \cup \big(\bigcup_{K\in\mathcal{N}(S)} K\big)$, and put $\widetilde{S}$ into the set $\mathcal{P}_i$. We say that $\widetilde{S} \in \mathcal{P}_i$ is *induced by* $S \in \mathcal{P}_{i-1}$, and the sets in $\mathcal{N}(S)$ together with $S$ are the *ancestors* of $\widetilde{S}$. We repeat the above process in the leftover set $\mathcal{P} = \mathcal{P}\backslash\{S\}\backslash\mathcal{N}(S)$ until $\mathcal{P}$ becomes empty. At the end of the process, we obtain the set $\mathcal{P}_i$. See Figure 3 for an illustration.

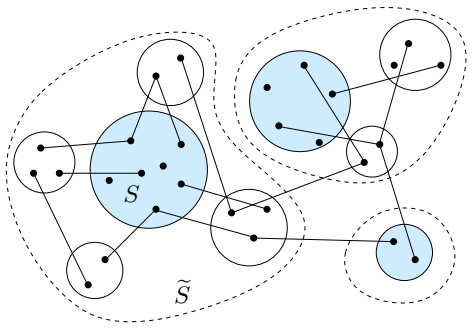



**Figure 3.** Constructing $\mathcal{P}_i$ from $\mathcal{P}_{i-1}$. Solid circles represent sets in $\mathcal{P}_{i-1}$. Regions bounded by dotted curves represent sets in $\mathcal{P}_i$. Line segments represent edges in $E_{3i}$. The sets in $\mathcal{P}_{i-1}$ that induce sets in $\mathcal{P}_i$ are shaded.

Clearly, $\mathcal{P}_i$ is a valid partition of $V$. It remains to prove that $\mathcal{P}_i$ satisfies ($\star$), assuming $\mathcal{P}_{i-1}$ satisfies ($\star$). Let $\widetilde{S} \in \mathcal{P}_i$ and $x, y \in \widetilde{S}$. We need to show that $d(x, y) < 2^{3(i+1)}$. Assume $\widetilde{S}$ is induced by $S \in \mathcal{P}_{i-1}$. There are three simple cases to consider:

(a) $x, y \in K$ for some ancestor $K$ of $\widetilde{S}$. By ($\star$) for $\mathcal{P}_{i-1}$, we have $d(x, y) < 2^{3i} < 2^{3(i+1)}$.

(b) $x \in S$ and $y \in K$ for some $K \in \mathcal{N}(S)$. Let $(x_0, y_0)$ be the edge in $E_{3i}$ with $x_0 \in S$ and $y_0 \in K$. Then $d(x, y) \le d(x, x_0) + d(x_0, y_0) + d(y_0, y) < 2^{3i} + 2^{3i+1} + 2^{3i} < 2^{3(i+1)}$,

(c) $x \in K_1$ and $y \in K_2$ for some $K_1, K_2 \in \mathcal{N}(S)$. Similarly, let $(x_1, y_1), (x_2, y_2)$ be the two edges in $E_{3i}$ with $x_1, x_2 \in S$, and $y_1 \in K_1$ and $y_2 \in K_2$ respectively. We have $d(x, y) \le d(x, y_1) + d(y_1, x_1) + d(x_1, x_2) + d(x_2, y_2) + d(y_2, y) < 3 \cdot 2^{3i} + 2 \cdot 2^{3i+1} < 2^{3(i+1)}$.

Hence we can conclude that $\mathcal{P}_i$ satisfies ($\star$).

We define the *entropy* of a partition $\mathcal{P}$ as

$$\mathcal{E}(\mathcal{P}) = \sum_{S\in\mathcal{P}} |S| \cdot \log(n/|S|).$$

Here all log's are in base 2. Next we show that the entropy decreases significantly when we move from $\mathcal{P}_{i-1}$ to $\mathcal{P}_i$. For every $\widetilde{S} \in \mathcal{P}_i$, let $\mathcal{D}(\widetilde{S}) \subseteq \mathcal{P}_{i-1}$ denote the set of ancestors of $\widetilde{S}$. We define

$$\Delta(\widetilde{S}) = \sum_{S\in\mathcal{D}(\widetilde{S})} |S| \cdot \log(n/|S|) - |\widetilde{S}| \cdot \log(n/|\widetilde{S}|).$$

It is easy to verify the following equality:

$$\mathcal{E}(\mathcal{P}_{i-1}) - \mathcal{E}(\mathcal{P}_i) = \sum_{\widetilde{S}\in\mathcal{P}_i} \Delta(\widetilde{S}). \tag{32}$$

Furthermore, we can write

$$\begin{aligned}\Delta(\widetilde{S}) &= |S| \cdot \log(|\widetilde{S}|/|S|) + \sum_{K\in\mathcal{N}(S)} |K| \cdot \log(|\widetilde{S}|/|K|) \\ &\ge \sum_{K\in\mathcal{N}(S)} |K| \cdot \log(|\widetilde{S}|/|K|) \\ &\ge \sum_{K\in\mathcal{N}(S)} |K|, \end{aligned} \tag{33}$$

where the last inequality follows from the fact that $|\widetilde{S}| \ge |S| + |K| \ge 2|K|$ for each $K \in \mathcal{N}(S)$ as $|S| \ge |K|$ by the choice of $S$.

Let $\sigma_i(\widetilde{S}) \subseteq E_{3i}$ be the subset of edges in $E_{3i}$ that are incident to points in $\bigcup_{K\in\mathcal{N}(S)} K = \widetilde{S} \setminus S$, where $S \in \mathcal{P}_{i-1}$ is the set that induces $\widetilde{S}$. Because for any node in $V$, the number of edges in $E_{3i}$ that are incident to it is at most $c$, we have $c \cdot \sum_{K\in\mathcal{N}(S)} |K| \ge |\sigma_i(\widetilde{S})|$. Combined with (33), we then obtain

$$\Delta(\widetilde{S}) \ge |\sigma_i(\widetilde{S})|/c. \tag{34}$$

On the other hand, we show that any edge in $E_{3i}$ must be incident to a point in $\widetilde{S} \setminus S$, for some $\widetilde{S} \in \mathcal{P}_i$ induced by some $S \in \mathcal{P}_{i-1}$. That is,

$$\bigcup_{\widetilde{S}\in\mathcal{P}_i} \sigma_i(\widetilde{S}) = E_{3i}. \tag{35}$$

Otherwise, there exists an edge $(x, y) \in E_{3i}$ for which one of the two cases has to happen:

(a) $x, y \in S$ for an $S \in \mathcal{P}_{i-1}$ that induces some $\widetilde{S} \in \mathcal{P}_i$; or

(b) $x \in S_1$ for an $S_1 \in \mathcal{P}_{i-1}$ that induces some $\widetilde{S}_1 \in \mathcal{P}_i$, and $y \in S_2$ for an $S_2 \in \mathcal{P}_{i-1}$ that induces some $\widetilde{S}_2 \in \mathcal{P}_i$.

The first case cannot happen because otherwise by ($\star$) we have $d(x, y) < 2^{3i}$, contradicting with the fact $(x, y) \in E_{3i}$. The second case cannot happen either, because otherwise we will have $S_1 \in \mathcal{N}(S_2)$ (in which case $S_1$ cannot induce $\widetilde{S}_1$) or $S_2 \in \mathcal{N}(S_1)$ (in which case $S_2$ cannot induce $\widetilde{S}_2$).

It follows from (32), (34) and (35) that

$$\mathcal{E}(\mathcal{P}_{i-1}) - \mathcal{E}(\mathcal{P}_i) \ge \sum_{\widetilde{S}\in\mathcal{P}_i} \sigma_i(\widetilde{S})/c \ge |E_{3i}|/c.$$

Because $\mathcal{E}(\mathcal{P}_{-1}) = n \log n$ and $\mathcal{E}(\mathcal{P}) \ge 0$ for any partition $\mathcal{P}$, summing up the above inequality for all $i \ge 0$, we then obtain

$$\sum_i |E_{3i}|/c \le \mathcal{E}(\mathcal{P}_{-1}) = n \log n.$$

Recall that $\sum_i |E_{3i}| \ge |E|/3$. This implies $|E| \le 3cn \log n$, as desired. ∎