Laboratoire Bordelais de Recherche en Informatique
UMR 5800 - Université Bordeaux I, 351, cours de la Libération,
33405 Talence CEDEX, France

Research Report RR-1330-04

Routing with Improved Communication-Space Trade-Off

by Ittai Abraham, Cyril Gavoille, Dahlia Malkhi

July, 2004

Routing with Improved Communication-Space Trade-Off

Ittai Abraham[*] Cyril Gavoille[†] Dahlia Malkhi[*]

30th July 2004

Abstract

Given a weighted undirected network with arbitrary node names, we present a family of routing schemes characterized by an integral parameter $\kappa \geq 1$. The scheme uses $\widetilde{O}(n^{1/\kappa} \log D)$ space routing table at each node, and routes along paths of linear stretch $O(\kappa)$, where D is the normalized diameter of the network. When D is polynomial in n, the scheme has asymptotically optimal stretch factor. With the same memory bound, the best previous results obtained stretch $O(\kappa^2)$.

Of independent interest, we also construct a single-source name-independent routing scheme for uniform weighted graphs with $O(1)$ stretch and $\widetilde{O}(1)$ bits of storage. With the same stretch, the best previous results obtained memory $\widetilde{O}(n^{1/9})$.

1 Introduction

The ability to route messages to specific destinations is one of the basic building blocks of any networked distributed system. Consider a weighted undirected network $G = (V, E, \omega)$ with n nodes having arbitrary unique network identifiers in $\{1, \dots, n\}$. A *name-independent routing scheme* is a distributed algorithm that allows any source node to route messages to any destination node, given the destination's network identifier.

Several measures characterize the efficiency and feasibility of a routing scheme.

Memory: The amount of memory bits stored by each node for purposes of routing.

Headers: The size of message headers that are written by nodes along the route.

Stretch: The maximum ratio, over all pairs, of the length of the routing path produced by the routing scheme by routing from s to t and the shortest path distance from s to t in G.

[*]School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem, Israel.
[†]Laboratoire Bordelais de Recherche en Informatique, University of Bordeaux, Bordeaux, France.

Our aim is to devise *compact* routing schemes with poly-logarithmic headers that have improved tradeoffs between the memory consumption and the stretch factor.

Our contributions. We first present in Section 3 a family of routing schemes parameterized by an integer $\kappa > 0$, that has the complexity measures below. The $\widetilde{O}()$ notation denotes complexity similar to $O()$ up to poly-logarithmic factors. Concrete constants are provided in the body of the paper.

Each node keeps $\widetilde{O}(n^{1/\kappa} \log D)$ bits of storage, where D is the normalized diameter of the graph. Message headers are of size $\widetilde{O}(1)$, and each route has stretch $O(\kappa)$

When D is polynomial in n, the scheme has asymptotically optimal stretch factor, as proven by [19]. With the same memory bound, the best previous results obtained stretch $O(\kappa^2)$ [6, 3].

Then, in Section 4, we consider the problem of routing messages from a distinguished node, the *source*, to all the other nodes. Single source routing problem with small local storage can also be seen as a searching problem through DHT or distributed dictionaries, or as locating keys in peer-to-peer systems. Efficient solution to these problems is interesting on its own right, and might be of practical interests. We show prove that uniform weighted graphs have a single-source name-independent routing scheme with $O(1)$ stretch and $\widetilde{O}(1)$ bits of storage, the first constant stretch routing scheme with poly-logarithmic memory. The best previous bound with similar stretch was $\widetilde{O}(n^{1/9})$ bits of storage [16].

Previous results. There are known lower bounds on the compact routing problem in general graphs. These are summarized in the first rows of Table 1. For lower bounds, the table makes the subtle distinction between a *designer port* model and a *fixed port* model. In the *fixed port* model (also known as the adversarial port model) the names of outgoing links, or ports, from each node may be arbitrarily chosen by an adversary from the set $\{1, \ldots, n\}$. In the *designer port* model they may be determined by the designer of the routing scheme. In particular, Gavoille and Gengler [13] indicate at least stretch-3 when each node has memory $o(n)$. For stretch-k routing scheme Peleg and Upfal [19] prove that a total of $\Omega(n^{1+1/(2k+4)})$ routing information bits is required. Thorup and Zwick refine this bound and show in [22] that the stretch is at least $2k+1$ when each node has memory $o(n^{1/k})$, proved for $k = 1, 2, 3, 5$ and conjectured for other values of k. For comprehensive surveys on compact routing and compact network data structures, see [12, 14].

Initial results in [5] provide name-independent routing with $\widetilde{O}(n^{3/2})$ total memory. Awerbuch and Peleg [6] presented a scheme that for any k, requires $\widetilde{O}(k^2 n^{1/k} \log D)$ bits per node and routes on paths of stretch $O(k^2)$. Arias et al. [3] present a slight improvement that uses the same memory bounds but improves the constant in the $O(k^2)$ stretch by a factor of 4.

All known name-independent schemes that are "combinatorial" and do not rely on the normalized diameter, D, in their storage bound have exponential stretch factor. Awerbuch et al. [4] achieve with $\widetilde{O}(n^{1/k})$ memory stretch $O(9^k)$, and [3] improved to stretch $O(2^k)$

stretch	graphs	port model	memory	address	reference
ALL-TO-ALL					
$s < 1.4$	any unweighted	designer	$n \log n$	$\log n$	[15][1]
$s < 3$	any unweighted	designer	n	n	[13]
$s < 2k+1$	any weighted	fixed	$n^{1/k}$	$n^{1/k}$	[22][2]
SINGLE-SOURCE					
$s = 1$	unweighted tree	designer	$\sqrt{n}$	$\lceil \log n \rceil$	[10][3]
$s = 1$	unweighted tree	fixed	n	$\lceil \log n \rceil$	[10]
$s = 1$	unweighted tree	fixed	$\log^2 n / \log\log n$	$\log^2 n / \log\log n$	[11]
$s < 3$	unweighted tree	designer	$n \log n$	Name-Indep.	[2]

Table 1: Lower bounds on the local memory requirements. Bounds of Line 1,4,5,6,7 are known to be optimal up to a constant factor, Line 2 and 3 are optimal up to a poly-logarithmic factor.

with the same memory bound. For $\widetilde{O}(\sqrt{n})$ memory Arias et al. provide stretch 5. Recently, Abraham et al. [2], achieve optimal stretch 3 with $\widetilde{O}(\sqrt{n})$.

A weaker variant of the routing problem, *labeled routing*, was initiated in [4]. In this problem model, the algorithm's designer can choose the network addresses of nodes (and of course, use node names to store information about their location in the graph). This paradigm does not provide for a realistic network design, however, the tools devised for its solution have proven useful as building blocks of full routing schemes (in fact, we make use here of certain building blocks devised in the context of labeled routing schemes).

Indeed, optimal compact schemes for labeled routing are known. The first non trivial stretch-3 scheme was given by Cowen [9] with $\widetilde{O}(n^{2/3})$ memory. Later, Thorup and Zwick [21, 22] improved the memory bound to only $\widetilde{O}(\sqrt{n})$ bits. They also gave an elegant generalization of their scheme, achieving stretch $4k-5$ (and even $2k-1$ with handshaking) using only $\widetilde{O}(n^{1/k})$ bits. Additionally, there exist various labeled routing schemes suitable only for certain restricted forms of graphs. For example, routing in a tree is explored, e.g., in [10, 22], achieving optimal routing. It requires $\widetilde{O}(1)$ bits for local tables and $\widetilde{O}(1)$ bits for headers.

Due to space limitation, some proofs have been moved in [1].

[1] For stretch-1, a stronger lower bound of $\Omega(n \log d)$ holds for every maximum degree d graphs, $3 \leq d \leq \epsilon n$ [15].

[2] The bound relies to a conjecture proved for $k = 1, 2, 3, 5$. The bound holds only for fixed port model and weighted graphs with polynomial weights.

[3] If slightly larger than $\lceil \log n \rceil$ bit addresses are allowed, namely $(1 + o(1)) \log n$, then $O(\log n)$ bits suffice for memory [22, 10].

2 Preliminaries

We denote an undirected weighted graph by $G = (V, E, \omega)$, where V is the set of nodes, E the set of links, and $\omega : E \rightarrow \mathbb{R}^+$ a link-cost function. For any two nodes $u, v \in V$ let $d_G(u, v)$ be the cost of a minimum cost path from u to v, where a cost of a path is the sum of weights of its edges. Define the *normalized diameter* of G, $D = \frac{\max_{u,v} d_G(u,v)}{\min_{u \neq v} d_G(u,v)}$.

Define $B(v, r) = \{u \in V \mid d_G(v, u) \leq r\}$ as the set of nodes whose distance is at most r from v.

We denote a rooted weighted tree by $T = (V, r, E, \omega)$, and define for every node $u \in V$ its *parent* $p(u)$ and for the root $p(r) = r$. The *children* of a node u are defined as $\mathrm{child}(u) = \{v \mid p(v) = u\}$. The *weight* of a node u denoted $w(u)$ is the number of nodes in u's subtree not including u itself. Define the *radius* of T as maximum distance from the root, $\mathrm{rad}(T) = \max_u \{d_T(r, u)\}$.

Define the *maximum edge weight* of a weighted tree $T = (V, E, \omega)$ as $\mathrm{maxE}(T) = \max_{e \in E} \{\omega(e)\}$.

For $u \in V$, let $N(u) = \{v \mid (u, v) \in E\}$ denote u's neighbors. For every node u, let $\mathrm{port}(u, v)$ for every $v \in N(u)$ be a unique port name in $\{1, \ldots, n\}$. If node u wants to forward a message to node $v \in N(u)$ it does so by sending the message on port $\mathrm{port}(u, v)$. In the *fixed port* model (also known as the adversarial port model) the values $\{\mathrm{port}(u, v) \mid v \in N(u)\} \subseteq \{1, \ldots, n\}$ are arbitrarily chosen.

3 Linear Communication-Space Trade-Off

Let $G = (V, E, \omega)$ be a graph, where $|V| = n$. In this section, we provide a family of name-independent routing schemes for G parameterized by κ, in which each node keeps $\widetilde{O}(n^{1/\kappa} \log D)$ storage, where D is the normalized diameter of the graph, and each route has stretch $O(\kappa)$. When D is polynomial in n, the scheme has asymptotically optimal stretch factor, as proven by [19].

The construction makes use of two building blocks. The first one is a new tree cover based on Sparse Partitions, the second is a novel tree-routing scheme we devise. Below, we first state these building blocks, then make a black-box use of them for our full solution, and finally go back to provide the details of our novel tree-routing scheme.

3.1 Tree cover based on Sparse Partitions

Lemma 3.1 [6, 7, 18] *For every weighted graph $G = (V, E, \omega)$, $|V| = n$ and integers $\kappa, \rho \geq 1$, there exists a polynomial algorithm that constructs a collection of rooted trees $\mathcal{TC}_{\kappa,\rho}$ such that:*

1. *(Cover) For all $v \in V$, there exists $T \in \mathcal{TC}_{\kappa,\rho}$ such that $B(v, \rho) \subseteq T$.*
2. *(Sparse) For all $v \in V$, $|\{T \in \mathcal{TC}_{\kappa,\rho} \mid v \in T\}| \leq 2\kappa n^{1/\kappa}$.*

3. *(Small radius) For all* $T \in \mathcal{TC}_{\kappa,\rho}$, $\mathrm{rad}(T) \leq (2\kappa - 1)\rho$.
4. *(Small edges) For all* $T \in \mathcal{TC}_{\kappa,\rho}$, $\mathrm{maxE}(T) \leq 2\rho$.

Note that property (4) is a novel property that does to appear in the tree covers of [6, 7, 18]. However, it is crucial for our construction and its proof is a simple consequence of the manner in which the cover algorithm works: in each iteration, any cluster S added to a cover Y has $\mathrm{rad}(S) \leq \rho$. The end result is a set of covers $\mathcal{R}$ that has properties (1),(2), and (3). For every cover $Y \in \mathcal{R}$ define $r(Y)$ as the initial node that started that cover, and $G[Y]$ as the subgraph containing Y and all the edges connecting nodes in Y whose cost is at most 2ρ. $G[Y]$ spans Y because Y is formed by a connected union of clusters whose radius is at most ρ. The set $\mathcal{TC}_{\kappa,\rho}$ is defined by taking every $Y \in \mathcal{R}$ and setting $T_Y \in \mathcal{TC}_{\kappa,\rho}$ to be a minimum cost path tree spanning $G[Y]$ whose root is $r(Y)$.

W.l.o.g. assume that the minimum cost edge is 1. We define an index set $I = \{1, \ldots, \lceil \log D \rceil\}$. For all $i \in I$, we build a tree cover $\mathcal{TC}_{\kappa,2^i}$ according to Lemma 3.1 above. For all $v \in V$ and $i \in I$, let $\mathrm{Tree}_v[i]$ be a tree $T \in \mathcal{TC}_{\kappa,2^i}$ such that $B(v, 2^i) \subseteq T$.

3.2 Bounded cost name-independent tree-routing

Having built a hierarchy of tree covers, any source v would like to perform name-independent routing on $\mathrm{Tree}_v[i]$, for $i \in I$ in increasing order, until the target is found. Our second building block addresses this need using a novel and efficient construction. This construction provides a name-independent *error-reporting* routing scheme in which the cost of routing to a destination in the tree or learning that the name does not exist is bounded by a function of the tree's radius, the maximum edge cost, and a parameter κ.

Theorem 3.2 *For every tree* $T = (U, E, \omega)$, $|U| = m$, $U \subset V$, $|V| = n$, *and integer* κ *there exists a name-independent routing scheme on* T *with error-reporting that routes on paths of length bounded by* $4\mathrm{rad}(T) + 2\kappa\mathrm{maxE}(T)$, *each node requires* $O(\kappa n^{1/\kappa} \log^2 n)$ *memory, and headers are of length* $O(\log^2 n)$. *(And routing for a non-existent name in* T *also incurs a path of length* $4\mathrm{rad}(T) + 2\kappa\mathrm{maxE}(T)$ *until a negative result is reported back to the source.)*

The proof of Theorem 3.2 is deferred until Section 3.4.

For a tree T containing a node v, we let $\phi(T, v)$ denote the routing information of node v as required from Theorem 3.2.

3.3 The name-independent routing scheme

We now combine Theorem 3.2 with Lemma 3.1 in a manner similar to the hierarchical routing scheme of Awerbuch and Peleg [7].

Storage. For all $v \in V$, $i \in I$, and $T \in \mathcal{TC}_{\kappa,2^i}$ such that $v \in T$ node v stores $\phi(T,v)$. According to Lemma 3.1 and Theorem 3.2 above, the total storage of each node is $O(\kappa^2 n^{2/\kappa} \log D \log^2 n)$.

Routing. The sender s looks for destination t in the tree $\mathrm{Tree}_s[i]$ successively for $i = 1, 2, \ldots, \lceil \log D \rceil$ using the construction in Theorem 3.2.

Stretch analysis. From Lemma 3.1 for $T \in \mathcal{TC}_{\kappa,\rho}$ we have that the cost $4\mathrm{rad}(T) + 2\kappa \mathrm{maxE}(T)$ is bounded by $4(2\kappa - 1)\rho + 2\kappa 2\rho \leq 12\kappa\rho$. Hence, for any source s, integer $i \in I$, the cost of searching for any target t in $\mathrm{Tree}_s[i]$ is at most $12\kappa 2^i$.

For the index $j \in I$ such that $2^{j-1} < d(s,t) \leq 2^j$ we have $t \in B(v, 2^j) \subseteq \mathrm{Tree}_v[j]$ and therefore t will be found in the jth phase. The total cost will be

$$\sum_{1 \leq i \leq j} 12\kappa 2^i \leq 12\kappa 2^{j+1} < 48\kappa d(s,t).$$

Hence, using $\hat{\kappa} = 2\kappa$ instead of κ in the above construction, we proved the following.

Theorem 3.3 *For every weighted graph $G = (V, E, \omega)$ whose normalized diameter is D and integer $\kappa \geq 1$, there is a polynomial time constructible name-independent routing scheme with stretch $O(\kappa)$ and memory $O(\kappa^2 n^{1/\kappa} \log D \log^2 n)$.*

In the remainder of this section, we provide the construction that proves Theorem 3.2 above.

3.4 Bounded-cost name-independent tree-routing

Consider a set V of n nodes in which every node $u \in V$ has a unique name $n(u) \in \{1, \ldots, n\}$. (We can remove this assumption using hash functions given by Lemma 4.4. Let $T = (U, r, E, \omega)$ be a rooted tree with $r \in U \subseteq V$ and $|U| = m$.

Sorting the nodes in U by their unique name $n()$, we denote $U[i]$ as the ith largest node in U, $U[1] = \max_{v \in U}\{n(v)\}$ and for $1 < i \leq m$ define $U[i] = \max_{v \in U}\{n(v) \mid n(v) < U[i-1]\}$.

In addition to their given name $n(v)$, we give each node $v \in T$ three more names.

First, we give v its name in the labeled tree-routing of Thorup & Zwick [22] and Fraigniaud & Gavoille [10]:

Lemma 3.4 [10, 22] *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 the header size are $O(\log^2 n / \log\log n)$ bits. Given the information of a node and the label of the destination, routing decisions take constant time.*

For a tree T containing a node v, we let $\mu(T, v)$ denote the routing information of node v and $\lambda(T, v)$ denote the destination label of v in T as required from Lemma 3.4. Thus, the first name we assign with v is $\ell(v) = \lambda(T, v)$.

Secondly, $d(v)$ denotes the depth-first-search (DFS) preorder enumeration of the rooted tree, note that $\{d(u)|u \in U\} = \{1, \ldots, m\}$. Finally every node has a name $s(v)$ which will be defined as a function of its own subtree size relative to its siblings' subtree sizes. In some sense this reflects its rank among its siblings. The formal value of $s(v)$ will be defined later.

In our construction a node whose DFS enumeration is i is responsible to the ith largest node in U. Formally, for any $x \in T$ we define its *responsibility* as

$$o(x) = U[d(x)].$$

Given a target u the idea is first to route to the node y such that $o(y) = n(u)$ and then use labeled tree-routing to reach u.

We begin by presenting a simple name-independent scheme in which the storage requirements on any node v is $\widetilde{O}(|\mathrm{child}(v)| + 1)$ and the total cost of routing will be at most $4\mathrm{rad}(T)$.

Storage. Every node $x \in T$ stores the following:

1. Let $y \in T$ be such that $o(x) = n(y)$. Node x stores the tuple $(y,\ n(y),\ \ell(y))$.
2. Node x stores $A(x) = \{o(y) \mid y \in \mathrm{child}(x)\}$ together with a map from any $o(y) \in A(x)$ to the corresponding port name $\mathrm{port}(x, y)$ to reach the child y.
3. x stores $\mu(T, x)$, its tree-routing label as required from Lemma 3.4.

Routing. Given a target $u \in U$, first route to the root r.

1. On a node x
   (a) If $o(x) = n(u)$ then use $\ell(u)$ to reach u.
   (b) If there is no child $y \in \mathrm{child}(x)$ such that $o(y) \leq n(u)$ then report back that $u \notin T$.
   (c) Route to the child $y \in \mathrm{child}(x)$ with the maximum $o(y)$ such that $o(y) \leq n(u)$. Set $x := y$ and goto 1.

This procedure is similar to the interval routing of [20, 24]. If the label $\ell(u)$ is found, routing proceeds using the labeled tree-routing scheme of Lemma 3.4. In the simple scheme presented above, the cost of reaching root is at most $\mathrm{rad}(T)$, cost of reaching the node storing the required label is bounded by $\mathrm{rad}(T)$ and reaching the target (or reporting an error to the source) requires at most another $2\mathrm{rad}(T)$. In the fixed port model the storage per node is unbounded $\widetilde{O}(|\mathrm{child}(v)| + 1) = \widetilde{O}(n)$.

Bounding storage. We proceed to show how, at the cost of adding at most κ length-2 cycles to the routing path, we can reduce the storage of each node to only $\widetilde{O}(n^{1/\kappa})$ bits even in the fixed port model. The idea is to spread the information about v's children in a directory among v and its children child(v) in a load balanced manner that will ensure that at most κ probes to directories are performed in the whole routing path until the target is found.

First, for determining $d(v)$ we use a DFS enumeration that always prefers heavy children first (when faced with a choice, it explores a child with the maximum weight among the unexplored children).

Second, for every node u, we now define its child name $s(u)$. For any node v, we enumerate its children child(v) in their weighted order from large to small using words of the alphabet $\Sigma = \{0, 1, 2, \ldots n^{1/\kappa} - 1\}$. Specifically, for any node, given a list of its children sorted by their weight (from large to small), we name each of the first $n^{1/\kappa}$ nodes in non-increasing order of their weights by a child name which consists of one digit in Σ in increasing order $(0), (1), \ldots, (n^{1/\kappa} - 1)$. Then we name each of the next $n^{2/\kappa}$ nodes in order of their weights by a child name in Σ^2 in increasing lexicographic order, $(0,0), (0,1), \ldots, (0, n^{1/\kappa} - 1), (1,0), (1,1), \ldots, (1, n^{1/\kappa} - 1), \ldots, (n^{1/\kappa} - 1, 0), \ldots, (n^{1/\kappa} - 1, n^{1/\kappa} - 1)$. We continue this naming process until all nodes in child(v) are exhausted, up to at most a κ-digit child name in Σ^κ.

The central property of our naming is as follows. Let u be a child of v with a child name $s(u)$ consisting of $j > 1$ digits. Then $w(u) \le w(v)/n^{(j-1)/\kappa}$. The reason this property holds is that v must have $n^{(j-1)/\kappa}$ children that are at least as heavy as u. Since each one weights at least $w(u)$ their total weight would be larger than $w(v)$, a contradiction.

Storage. For every $x \in T$, we define $S(x)$ as follows:

$$S(x) = \left\{ \begin{array}{cccc} (0) & (1) & \ldots & (n^{1/\kappa} - 1) \\ (0,0) & (1,0) & \ldots & (n^{1/\kappa} - 1, 0) \\ \vdots & & & \vdots \\ (0, \underbrace{0, \ldots, 0}_{\kappa-1}) & (1, \underbrace{0, \ldots, 0}_{\kappa-1}) & \ldots & (n^{1/\kappa} - 1, \underbrace{0, \ldots, 0}_{\kappa-1}) \end{array} \right\}$$

For each child y of x such that $s(y) \in S(x)$, node x stores $o(y)$ and a map from $o(y)$ to the corresponding port name port(x, y) to reach child y.

We now define the storage held by x's children to assist in lookup. Let y be in child(x) and assume y has a length-j child name, $s(y)$, with with $j - i$ trailing zeros, $s(y) = (a_1, \ldots, a_i, \underbrace{0, \ldots, 0}_{j-i})$ for some $i \le j$. We define a subset $S'(y)$ of the enumerated set of v's children as follows:

$$S'(y) = \left\{ \begin{array}{cccc} (a_1,\ldots,a_i,\underbrace{0,\ldots,0}_{j-i-1},0) & (a_1,\ldots,a_i,\underbrace{0,\ldots,0}_{j-i-1},1) & \ldots & (a_1,\ldots,a_i,\underbrace{0,\ldots,0}_{j-i-1},n^{1/\kappa}-1) \\ (a_1,\ldots,a_i,\underbrace{0,\ldots,0}_{j-i-2},0,0) & (a_1,\ldots,a_i,\underbrace{0,\ldots,0}_{j-i-2},1,0) & \ldots & (a_1,\ldots,a_i,\underbrace{0,\ldots,0}_{j-i-2},n^{1/\kappa}-1,0) \\ \vdots & & & \vdots \\ (a_1,\ldots,a_i,0,\underbrace{0,\ldots,0}_{j-i-1}) & (a_1,\ldots,a_i,1,\underbrace{0,\ldots,0}_{j-i-1}) & \ldots & (a_1,\ldots,a_i,n^{1/\kappa}-1,\underbrace{0,\ldots,0}_{j-i-1}) \end{array} \right\}$$

The child node y of x stores the following information. For each $z \in \text{child}(x)$ such that $s(z) \in S'(y)$, y stores $o(z)$ and a map from $o(z)$ to the corresponding port name $\text{port}(x, z)$ to reach child z from parent x.

Intuitively, here is how this directory scheme works. Suppose the current node is x and the target node is u. The child-name enumeration of x's children is consistent with their responsibility enumeration order. That is, let v be the child of x whose sub-tree has responsibility for the value $n(u)$. Denote the child name of v by $s(v) = (a_1, \ldots, a_j)$. Then because of our DFS ordering, given any child $y \in \text{child}(x)$:

- If $s(y)$ has more than j digits then $o(v) \leq n(u) < o(y)$;
- If $s(y)$ has less than j digits then $o(y) < o(v) \leq n(u)$;
- If $s(y)$ has j digits, and according to lexicographical order $s(y) < s(v)$, then $o(y) < o(v) \leq n(u)$;
- If $s(y)$ has j digits, and according to lexicographical order $s(v) < s(y)$, then $o(v) \leq n(u) < o(y)$;

Given a target u, node x would like to find the appropriate child v such that $o(v)$ is the maximum value out of all $\{o(y) \leq n(u) \mid y \in \text{child}(x)\}$. Since x does not maintain $o(y)$ of all of its children $y \in \text{child}(x)$, the highest $o()$ value it maintains that is no greater than the target $n(u)$ belongs to the node y_1 with child name $s(y_1) = (a_1, \underbrace{0, \ldots, 0}_{j-1})$. Continuing from y_1, it too maintains only partial information about x's children. Here, the highest $o()$ value it maintains that is no greater than the target $n(u)$ belongs to the node y_2 with child name $s(y_2) = (a_1, \underbrace{0, \ldots, 0}_{i}, a_{i+2}, \underbrace{0, \ldots, 0}_{j-i-2})$ where $i \geq 0$ is the number of consecutive zeros that $s(v)$ has starting from its second digit a_2. And so on. With each such step, we reach a child of x whose child name matches the target's child name $s(v)$ in one more digit at least (and zero's in v's child name are matched without further steps). After at most j such steps, we reach v, and continue to search for u within the sub-tree it roots.

More precisely, the routing algorithm is as follows.

Routing algorithm. Given a target $u \in U$, first route to the root r. Then, on any node x there are three cases:

1. if $o(x) = n(u)$ then use $\ell(u)$ to reach u.
2. if x is a leaf or if $n(u) < o(y)$ for all y such that $s(y) \in S(x)$, then report back that $u \notin T$.
3. Otherwise, we would like to route to the child $y \in \text{child}(x)$ with the maximum $o(y)$ value out of all y such that $o(y) \leq n(u)$. Since x does not store $o(y)$ for all $y \in \text{child}(x)$ performing this case is done using the following directory algorithm.

Directory algorithm.

1. Route to the child y with maximum $o(y)$ value out of all y such that $o(y) \leq n(u)$ and $s(y) \in S(x)$.
2. On node y,
   (a) If $n(u) < o(z)$ for all z such that $s(z) \in S'(y)$ then the directory algorithm has reached the required child and the routing algorithm can proceed from node y.
   (b) Otherwise, route to the sibling z such that $o(z)$ has maximum value out of all z such that $o(z) \leq n(u)$ and $s(z) \in S'(y)$.
   Set $y := z$ and goto 2.

3.5 Analysis

Lemma 3.5 *Given a parameter κ, the name-independent error-reporting tree-routing scheme requires $O(\kappa n^{1/\kappa} \log^2 n)$ bits of storage per node in the tree.*

Proof. Each node v stores $\widetilde{O}(1)$ information for each child $u \in \text{child}(v)$ such that $s(u) \in S(v)$. By definition of $S(v)$, it contains at most $\kappa n^{1/\kappa}$ members, hence the storage is $\widetilde{O}(\kappa n^{1/\kappa})$. In addition node v maintains information to assist its parent node $p(v)$. This includes $\widetilde{O}(1)$ storage per each member in $S'(v)$, which, by definition, also requires $\widetilde{O}(\kappa n^{1/\kappa})$ bits. Finally, each node v stores the routing information $\mu(T, v)$ according to Lemma 3.4, requiring $\widetilde{O}(1)$ storage. Thus, the total storage of this scheme is $\widetilde{O}(\kappa n^{1/\kappa})$ items per node, each of $O(\log^2 n)$ bits at most. $\square$

Lemma 3.6 *Given a parameter κ, the name-independent error-reporting tree-routing scheme routs on paths whose cost is at most*

$$4\text{rad}(T) + 2\kappa\text{maxE}(T)$$

until either the destination is reached or the source receives notification that the destination does not exists in the tree

Proof. We now bound the total cost of searching for a target u on a tree T. Reaching the root takes at most $\text{rad}(T)$, reaching the node v such that $o(v) = n(u)$ (or getting a negative result) takes $\mathrm{rad}(T) + 2j\mathrm{maxE}(T)$ where j is the number of times the directory service had to probe other children along the path to node u. Once node u is reached, routing to t or reporting a negative result back to the source takes at most $2\mathrm{rad}(T)$.

Therefore, we are left to show that $j \leq \kappa$. The directory structure above guarantees that if appropriate next hop child has a length-i child name then it will reached in at most $i-1$ intermediate queries. Specifically, let $s(y)$ denote a length-i child name of x's child, whose sub-tree stores information on a target $n(u)$. Given a target name $n(u)$, node v finds $o(u_1)$, the maximum name stored by v that is at most $n(u)$. Then v routes to u_1, a child with length-i child-name whose first digit is the same as the child covering $n(u)$. Node u_1 is either the actual child y, or it finds $o(u_2)$, the maximum name stored in u_1 that is at most $n(u)$. Then u_1 routes up to v and down to u_2, which has a length-i child name that matches $s(y)$ in at least the first two digits. This process continues until the correct child y is reached after at most $i-1$ intermediate steps from v to a child and back.

A crucial property maintained by the storage hierarchy is that if v has weight $w(v)$, then a child with a length-i child name with $i > 1$ has weight at most $w(v)/n^{(i-1)/\kappa}$. This is due to the weighted sorting: Otherwise the $n^{(i-1)/\kappa}$ children with length $i-1$ child names would each have at least $w(v)/n^{(i-1)/\kappa}$ children, and their total weight would be larger than $w(v)$ which is a contradiction.

Following a path from the root r to the node containing the label takes at most distance $\mathrm{rad}(T)$. Along the path, every node with child name of length $i > 1$ may cost additional $i-1$ double-steps from its parent to a child and back to the parent. Since every node with a length-i id reduces the weight of the tree by a factor of at least $n^{(i-1)/\kappa}$, there are at most $j \leq \kappa$ such extra double-steps along the whole path. Each double-step costs at most $2\mathrm{maxE}(T)$. Therefore, the total distance of the path is bounded by $4\mathrm{rad}(T) + 2\kappa\mathrm{maxE}(T)$. □

4 Single-Source Name-Independent Routing Scheme

In this part we consider the problem of routing messages from a distinguished node, the *source*, to all the other nodes, while keeping the name-independent constraint. The Single-source routing problem with small local storage can also be seen as a searching problem through a distributed information system, e.g., a distributed dictionary or a hash table. Efficient solutions to these problems are interesting in their own right.

We restrict our attention to single-source routing schemes in trees rooted at the source, i.e., the single-source shortest path tree rooted at the source in the graph. We assume that node names of the tree are taken from some universe $\mathcal{U}$ with $|\mathcal{U}| \geq n$, the number of nodes of the tree. The names and the port numbers are assumed to be fixed by an adversary (fixed port model) after the given tree and before the design of the routing scheme.

A single-source routing scheme on a tree T with source s is *L-reporting* if, for every $v \in \mathcal{U}$, the routing from s to v reports to s a *failure* mark in the header if $v \notin T$ after a loop of cost at most L. And, if $v \in T$, then the route from s to v has cost at most

$L + d_T(s,v)$. Note that the stretch constraint of an L-reporting routing scheme concerns restriction on route length to destinations $v \in T$ only. In the following $d(T)$ denotes the *depth* of the tree T, i.e., $d(T) = \max_{v \in T} d_T(s,v)$.

Theorem 4.1 *Every unweighted rooted tree T with n nodes taken from $\mathcal{U}$ has a single-source name-independent routing scheme of stretch 17 that is $12d(T)$-reporting, and using $O(\log^5 n/(\log\log n)^2 + \log|\mathcal{U}|\log^3 n/\log\log n)$ bits per node, that is $o(\log^5 n)$ if $|\mathcal{U}| \le n^{o(\log n \log\log n)}$.*

The best previous scheme, due to [16] and for $|\mathcal{U}| = n$, was using $\widetilde{O}(n^{1/k})$ bits for a stretch factor of $2k-1$, i.e., $\widetilde{O}(n^{1/9})$ bits for stretch 17. However our scheme works only for uniform weights.

The next lemma reduces the problem to one of designing efficient L-reporting schemes on trees without any specification of the stretch. Observe that there is no straightforward relationship between the L-reporting property and the stretch factor property of a routing scheme. This reduction can be seen as the specialization of the Awerbuch-Peleg's sparse cover for trees [6, 18].

Lemma 4.2 *Assume that there exists $\alpha \ge 1$ such that every unweighted rooted tree T with at most n nodes has (in the fixed port model) a single-source name-independent routing scheme that is $\alpha d(T)$-reporting and that uses at most M bits per node. Then, every rooted tree T with n nodes has a single-source name-independent routing scheme (also in the fixed port model) of stretch $4\alpha+1$ that is $3\alpha d(T)$-reporting, and using at most $M(\lceil \log d(T) \rceil + 1)$ bits per node.*

Proof. Let $\alpha \ge 1$ be a number satisfying the hypothesis of Lemma 4.2. We consider a tree T rooted at s. For every integer $d \in \{1, \dots, d(T)\}$, T_d denotes the subtree of T composed of the nodes at a distance at most d from s, and let S_d denote a single-source name-independent routing scheme for T_d that is αd-reporting and that uses at most M bits per node.

Recall that for every L-reporting routing scheme on T, the length of the route from s to $v \in T$ is at most $L + d_T(s,v)$. So if $d(T) \le 1$, every $\alpha d(T)$-reporting routing scheme on T is of stretch at most $\alpha + 1$ which satisfies Lemma 4.2. So assume that $d(T) > 1$.

The scheme we propose for T is defined as follows. The storage for each node v of T is composed of the storage of v in all the schemes $S_{2^p}, S_{2^{p+1}}, \dots, S_{2^q}, S_{d(T)}$, where p is the smallest integer ≥ 0 such that $v \in T_{2^p}$ and where q is the greatest integer ≥ 0 such that $2^q < d(T)$. Note that $q = \lceil \log d(T) \rceil - 1$ exists since $d(T) > 1$. The space requirement for v is at most $M(q - p + 2) \le M(\lceil \log d(T) \rceil + 1)$ bits as claimed. We emphasize here that each routing scheme S_d should be constructed in the fixed port model since otherwise they might be not superposed.

The routing algorithm for sending a message to any $v \in \mathcal{U}$ consists in searching v from s successively with the routing schemes $S_1, S_2, S_4, \dots, S_{2^q}, S_{d(T)}$. If $v \notin T$, then $v \notin T_{2^i}$ for all i's and $v \notin T_{d(T)}$ as well. Thus the scheme will successively report to s a failure mark. Such a route is of length at most

$$\sum_{i=0}^{q}(\alpha 2^i) + \alpha d(T) \;=\; \alpha(2^{q+1}-1) + \alpha d(T) \;<\; \alpha\left(2^{\lceil \log d(T) \rceil} + d(T)\right) \;<\; 3\alpha d(T) \;.$$

Now, assume that $v \in T$. As we seen previously, if $d_T(s,v) \leq 1$, then the stretch is at most $\alpha+1$. Assume $d_T(s,v) > 1$. The scheme reports a failure mark for all the schemes S_{2^i} with $2^i < d_T(s,v)$, and then routes to v along a path of length bounded by $\alpha d(T) + d_T(s,v)$ by the uses of the scheme $S_{d(T)}$ in the worst-case. So the total length of this route is at most

$$\sum_{i=0}^{q}(\alpha 2^i) + \alpha d(T) + d_T(s,v) \;<\; 3\alpha d(T) + d_T(s,v)$$

reusing the previous equation. So we have proved that the scheme is $3\alpha d(T)$-reporting.

In order to bound the stretch, let k denote the greatest integer such that $2^k < d_T(s,v)$. Note that $d_T(s,v) \leq 2^{k+1}$. From the previous discussion, the length of the route from s to $v \in T$ is bounded by

$$\begin{aligned}\sum_{i=0}^{k}(\alpha 2^i) + \alpha 2^{k+1} + d_T(s,v) &\;\leq\; \alpha\left(2^{k+1}-1\right) + \alpha 2^{k+1} + d_T(s,v)\\ &\;<\; 4\alpha 2^k + d_T(s,v) \;<\; 4\alpha d_T(s,v) + d_T(s,v) \;.\end{aligned}$$

Therefore, the scheme is of stretch at most $4\alpha + 1$ that completes the proof. □

According Lemma 4.2, to prove Theorem 4.1 it suffices to prove that we can set $\alpha = 4$ with a suitable memory bound M. More precisely:

Theorem 4.3 *Every unweighted rooted tree with n nodes taken from $\mathcal{U}$ has a single-source name-independent routing scheme (in the fixed port model) that is $4d(T)$-reporting, and using $O(\log^4 n/(\log\log n)^2 + \log|\mathcal{U}|\log^2 n/\log\log n)$ bits per node. Moreover, the first header construction takes $O(\log n)$ time at the source, and all the other routing decisions $O(\log\log n)$ time.*

Before proving Theorem 4.3 we need some basic results about hash functions (see [8, 17]). W.l.o.g. we assume that $\mathcal{U} = \{0,\ldots,|\mathcal{U}|-1\}$.

Lemma 4.4 [**8**] *Let $\mathcal{P} = \{0,\ldots,p-1\}$ for some prime number $p = \Theta(n)$. There exists a family of hash functions $\mathcal{H} = \{h : \mathcal{U} \to \mathcal{P}\}$ such that for every set $V \subseteq \mathcal{U}$ with $|V| = n$, there exists a function $h \in \mathcal{H}$ such that:*

1. *h is a degree-$O(\log n)$ polynomial of the field $\mathbb{Z}_p$;*
2. *$|\{v \in V \mid h(v) = k\}| = O(\log n)$ for every $k \in \mathcal{P}$;*

The first point of Lemma 4.4 implies that each function h can be stored with $O(\log^2 n)$ bits and have time complexity $O(\log n)$, whereas the second point states that there are at most $O(\log n)$ collisions for each $v \in V$.

The proof of Theorem 4.3. From now, we consider a tree T with source s. The node set of T is denoted by V, $n = |V|$, and p is a prime number such that $n \le p < 2n$. (Such p always exists from Bertrand's Postulate and Dirichlet's Theorem). Let $\mathcal{P} = \{0, \ldots, p-1\}$. Each value $k \in \mathcal{P}$ is called hereafter a *key*. We consider the hash function $h \in \mathcal{H}$ for V as given by Lemma 4.4.

For every $v \in V$, we denote by $\ell_T(v)$ the tree-routing label of v in T, which is used for the routing in T from source s to destination v. The length of each of these labels is $O(\log^2 n/\log\log n)$ bits [22, 10].

Overview of the scheme. The basic idea of the scheme is to use *indirection*: the keys of $\mathcal{P}$ are mapped to the nodes of T in a balanced way, typically with no more than $\widetilde{O}(1)$ keys per node. Then the node on which the key k is mapped is in charge of the tree-routing label of all names $u \in \mathcal{U}$ such that $h(u) = k$. First we route from s to the node in charge of k, and then to the destination.

More precisely, consider the routing from the source s to an arbitrary name $v \in \mathcal{U}$. First s hashes v into the key $k = h(v) \in \mathcal{P}$. Then we use a label-based routing scheme (i.e., a name-dependant routing scheme) to find a route in T from s to the node labeled k in this routing scheme, say node w. Roughly speaking, this labeled scheme is similar to Interval Routing Scheme [20, 24] which is based on a DFS numbering of the nodes. Locally w is aware of the tree-routing labels $\ell_T(s)$, $\ell_T(w)$, and $\ell_T(u)$ for all $u \in V$ such that $h(u) = k$. Node w also stores the corresponding list of names, i.e., the u's of V with $h(u) = k$. Our scheme ensures that each possible key of $\mathcal{P}$ is mapped to exactly one node of T. So that once node w is attained, we only need to check whether v belongs or not to the list of names stored by w. If it does not belong to, then we can conclude that $v \notin V$, and then w reports to s a failure mark thanks to the tree-routing labels $\ell_T(s)$ and $\ell_T(w)$. If v is found in the w's name list, then w directly routes to v thanks to $\ell_T(v)$ and $\ell_T(w)$. Such a scheme is therefore $2d(T)$-reporting.

However, in the scheme sketched above, the routing from s to k cannot be done via a standard implementation of Interval Routing Scheme for several reasons: 1) the set of keys, $\mathcal{P}$, is in general larger than V; 2) the memory requirements of node w for interval routing is $O(\deg(w)\log n)$ bits whereas we expect $\widetilde{O}(1)$ bits of storage for every node. More fundamentally, even if $p = n$ and even if the designer port model is allowed, then every scheme that bijectively assign labels to nodes require $\Omega(\sqrt{n}\,)$ bits of routing information at the source in the worst-case [10]. This lower bound forces us to map keys over the nodes in a non fully-balanced way.

The remainder of the proof consists in constructing the mapping from $\mathcal{P}$ to V, and the compact encoding of routing information.

The header of the message at any step of the routing from s to v is composed at most of the following fields: a type of message on a constant number of bits, a key of $\mathcal{P}$, a name of $\mathcal{U}$, and possibly a tree-routing label. The second and the third fields never change and are initialized to $h(v)$ and v respectively. The length of the header is no more than

$O(\log n + \log |\mathcal{U}|)$ bits.

Simulating designer port model via double-step routing. An important hypothesis to apply Lemma 4.2 is that the ports of each node x of the tree are arbitrarily permuted (the fixed port model). However, according to the next remark we will assume that the routing from s to the key k is done in the designer port model (i.e., the ports of each node have been permuted with a desirable permutation). Nevertheless it should be clear that once k is attained, then the routing to v (if $v \in V$) or to s (if $v \notin V$) is done thanks to the label $\ell_T(v)$ or $\ell_T(s)$ that have been computed in the fixed port model.

Indeed, during the routing from s to the key of v, one can apply the following *routing simulation*: Let $\mathrm{port}_d(x, y)$ (resp. $\mathrm{port}_f(x, y)$) be the port number between x and y in the designer port model (resp. in the fixed port model). For the simulation, every node y with parent x stores the numbers $p_1 = \mathrm{port}_d(y, x)$, $p_2 = \mathrm{port}_f(y, x)$, and $p_3 = \mathrm{port}_f(x, z)$ where z is the child of x such that $\mathrm{port}_d(x, z) = \mathrm{port}_f(x, y)$. In y, if the routing scheme outputs p_1, then the answer is converted to port p_2. If in x, the answer p of the routing scheme is different from port number of its parent (x knows it), x sends the message on port number p with a mark m_1 attached to the header. If y receives a message with mark m_1, it forwards to its parent, on port p_2, the message with mark m_2 and the value p_3 attached to its header. Finally, in x, if the routing scheme receives a header with a m_2 mark, then it extracts from the header the value p, and forward the message on port number p. To summarize the routing from y toward its parent is done as previously, whereas the routing from x toward its child z is done by a route of length 3. So if $v \notin V$, the routing will report to s a fail mark after a route of length $3d(T) + d(T) = 4d(T)$ instead of $2d(T)$. And if $v \in V$, the route length is at most $3d(T) + d_T(k, v) \leq 4d(T) + d_T(s, v)$. This leads to a $4d(T)$-reporting scheme with an $O(\log n)$ additive factor on the memory requirements and on the header size. So, simulating and superposing the $O(\log d(T)) = O(\log n)$ designer port schemes raise the overall memory requirement of a node to an $O(\log^2 n)$ additive factor, for headers the overhead is only $O(\log \log n)$ bits since the schemes are used successively and so only the index $i \leq O(\log n)$ of the current scheme needs to be specified.

Routing in the designer port model. So we restrict our attention to the routing from s to the key of v in the designer port model. From now we assume that the children $x_1, x_2, \ldots$ of every node x are ordered according to their increasing number of descendants and that $\mathrm{port}_d(x, x_i) = i$. (We fix $\mathrm{port}_d(x_i, x) = 0$.) Let $w(x)$ be the *weight* of x defined by the number of descendents of x in T (x included).

The scheme is parametrized by the integer $t = \lceil \log n \rceil$. We partition the nodes of T in *heavy* and *light*. The t heaviest children of x are heavy and the others (if any) are light. The root is heavy. Clearly, if the child x_i of x is light, then $w(x_i) < w(x)/t$, so that the number of light ancestors of x_i is at most $O(\log_t n)$.

The routing scheme is based on two numbers, $c(x)$ and $q(x)$, we assign to each node x. The first, called the *charge of* x, represents the total number of keys that must be mapped on the nodes of T_x, the subtree of root x. (So for the root, $c(s) = p$). The second one denotes the number of keys assigned to x. These two numbers must satisfy that, for every x,

$$c(x) \;=\; \sum_{y \in T_x} q(y) \;. \tag{1}$$

The heart of our scheme is the way we compute and encode $c(x)$ while balancing the charge of x over its descendants, i.e., guaranteeing $q(y) = \widetilde{O}(1)$ for every y. Given the numbers $c(x)$ and $q(x)$ one can then route through a *modified* DFS number $f(x)$ associated with each x and defined by: $f(s) := 0$, and $f(x_i) := f(x) + q(x) + \sum_{j<i} c(x_j)$, where x_i is the ith child of x. (This matches to the standard definition if $q(x) = 1$ for every x.)

Now the routing is done similarly to Interval Routing Scheme. Let w be the node in charge of $h(v)$, the key of v. Assume that w is a descendant of some node x, initially $x = s$. It is easy to see that:

1. either $h(v) \in [f(x), f(x) + q(x))$, and $w = x$, i.e., the key of v is stored by x;
2. or w is a descendant of x_i where $h(v) \in [f(x_i), f(x_{i+1}))$, and thus the routing in x must answer port i.

So the routing from x to $h(v)$ is well defined if x is aware of $f(x)$, $q(x)$, and of the vector $\vec{c}(x) = (c(x_1), c(x_2), \dots)$ of charges of the children of x. Indeed the numbers $f(x_i)$ and $f(x_{i+1})$ can be computed from $f(x)$, $q(x)$, and from $\vec{c}(x)$. We are now left with the description of $c(x)$, $q(x)$, and the compact encoding of $\vec{c}(x)$.

For that, let W be the function defined by $W(k, q, m) = 2^k \cdot (1 + 1/q)^m$, where $k, m \geq 0$ and $q \geq 1$ are all integers. Function W satisfies the following properties:

1. $q \cdot W(k, q, m+1) = (q+1) \cdot W(k, q, m)$.
2. $W(k, q, m) < W(k, q, m+1)$, since $1 + 1/q > 1$.
3. $W(k, q, q) \geq W(k+1, q, 0)$, because simple analysis shows that $(1 + 1/q)^q \geq 2$ for $q \geq 1$.

In particular Property 3 implies that for every $w \geq 1$, there exists $m \in \{0, \dots, q-1\}$ such that $W(k, q, m) \leq w < W(k, q, m+1)$ where $k = \lfloor \log w \rfloor$ (clearly $W(k, q, 0) = 2^k \leq w < 2^{k+1} = W(k+1, q, 0)$), and by strictness of inequality of Property 2, m is unique.

Computing $c(x)$ and $q(x)$. The numbers $c(x)$ and $q(x)$ are computed through a DFS with priority to lightest children. At any step, if $c(x) = 0$, then we skip x in the search. We start from the source by setting:

0. $c(s) := p$ and $q(s) := \lceil p/n \rceil \in \{1, 2\}$.

Then, for the ith child of x such that $c(x) > 0$:

1. Let $q = q(x)$ and $k = \lfloor \log w(x_i) \rfloor$.
2. If x_i is heavy, then $c(x_i) := q \cdot w(x_i)$ and $q(x_i) := q$.

3. If x_i is light, then $c(x_i) := \lceil (q+1)\cdot W(k,q,m) \rceil$ and $q(x_i) := q+1$ where m is such that $w(x_i) \in [W(k,q,m), W(k,q,m+1))$.
4. If $\sum_{j\leq i} c(x_j) > c(x) - q(x)$, then correct $c(x_i) := \max\{c(x) - q(x) - \sum_{j<i} c(x_j), 0\}$.
5. If $q(x_i) > c(x_i)$, then correct $q(x_i) := c(x_i)$.

By induction on the depth of x, $q(x)$ is at most $q(s)$ plus the number of x's light ancestors. Since $q(s) \leq 2$, it follows that $q(x) = O(\log_t n) = O(\log n / \log\log n)$.

In order to validate our routing algorithm, based on $f(x)$, $q(x)$ and $\vec{c}(x)$, we need to show that $c(x)$ and $q(x)$ numbers satisfy Eq. (1), i.e.,

Lemma 4.5 *For every x, $c(x) = \sum_{y \in T_x} q(y)$.*

Proof. Let us first show that, for every x,

$$c(x) \;\leq\; q(x)w(x) \tag{2}$$

and for every child x_i of x for which Line 4 does not apply,

$$q(x)w(x_i) \;\leq\; c(x_i)\ . \tag{3}$$

Note that $c(s) = p \leq \lceil p/n \rceil\, n = q(s)w(s)$, so Ineq. (2) holds for $x = s$. If x_i is heavy, then $c(x_i) = q(x)w(x_i)$ and Ineq. (3) holds, and since $q(x_i) = q(x)$, Ineq. (2) holds as well. So assume that x_i is light. We have $w(x_i) \geq W(k, q(x), m)$ that implies $(q(x)+1)w(x_i) \geq (q(x)+1)W(k,q(x),m)$. Note that $q(x_i) = q(x)+1$, and $(q(x)+1)w(x_i)$ is an integer, so $q(x_i)w(x_i) \geq \lceil (q(x)+1)W(k,q(x),m) \rceil \geq c(x_i)$ (there is equality if Line 4 does not apply). Since we have seen that $c(s) \leq q(s)w(s)$, it follows, setting $x = x_i$, that $c(x) \leq q(x)w(x)$ for every x, proving therefore Ineq. (2). To prove Ineq. (3) we assume that Line 4 does not apply to x_i. We have $w(x_i) < W(k,q(x),m+1)$ implying $q(x)w(x_i) < q(x)W(k,q(x),m+1)$. By Property 1 of function W, $q(x)w(x_i) < (q(x)+1)W(k,q(x),m) \leq \lceil (q(x)+1)W(k,q(x),m) \rceil = c(x_i)$. This completes the proof of Ineq. (3).

Let us denote by $Q(x) = \sum_{y \in T_x} q(y)$, and let us prove by induction on $w(x)$, that $c(x) = Q(x)$ for every x. If $w(x) = 1$, then $Q(x) = q(x)$. By Line 5, $q(x) \leq c(x)$, and by Ineq. (2), $c(x) \leq q(x)w(x)$. So, for $w(x) = 1$, $c(x) = Q(x)$ as claimed.

Now assume that $c(x_i) = Q(x_i)$ holds for every child x_i of x. We want to prove that $c(x) = Q(x)$. By definition $Q(x) = q(x) + \sum_i Q(x_i)$ that is $Q(x) = q(x) + \sum_i c(x_i)$ applying the induction hypothesis.

If Line 4 applies to some child of x, then $c(x) - q(x) = \sum_i c(x_i)$, and thus $c(x) = Q(x)$ and the result holds. So assume that Line 4 does not apply to any child of x. So, $\sum_i c(x_i) \leq c(x) - q(x)$, i.e., $Q(x) \leq c(x)$. Ineq. (3) applied to x_i gives $c(x_i) \geq q(x)w(x_i)$. Thus $\sum_i c(x_i) \geq q(x) \sum_i w(x_i)$. Observing that $w(x) = 1 + \sum_i w(x_i)$, it follows that

$$Q(x) \;=\; q(x) + \sum_i c(x_i) \;\geq\; q(x)\left(1 + \sum_i w(x_i)\right) \;=\; q(x)w(x) \;\geq c(x)$$

by Ineq. (2). So we can conclude that $Q(x) = c(x)$ for every x, completing the proof of Lemma 4.5. □

A *range query* on a sequence of integers $(c_1, \ldots, c_r)$ consists in finding, for every input z, the index i such that $z \in [\sum_{j \le i} c_j, \sum_{j \le i+1} c_j)$. Clearly, the routing algorithm as described above reduces to the range query $z = h(v) - f(x) - q(x)$ on the sequence $\vec{c}(x)$. Remark: range queries can be solved in $O(\log\log n)$ time with the $O(r)$ space van Emde Boas's data structure [23]. We show here that one can obtain the same time complexity while working on a very compact representation of the sequence. Compact representation of $\vec{c}(x)$ is possible because of the special choice of $c(x_i)$ values.

Lemma 4.6 *For every x, $\vec{c}(x)$ can be coded with a data structure of $O(\log^3 n/\log\log n)$ bits supporting range queries in $O(\log\log n)$ worst-case time.*

Proof. Let $\vec{c}(x) = (c(x_1), c(x_2), \ldots)$. Lemma 4.5 implies that $c(x) \le c(s) < 2n$. If x_i is heavy, then we store in extenso $c(x_i)$ with $O(\log n)$ bits since $c(x_i) < 2n$. There are at most t heavy children so it sums to $O(t \log n)$ bits.

Consider the value of $c(x_i)$ before applying Line 4. Since q is fixed for all the children, so the properties of function W implies that $c(x_i)$ is nondecreasing with k, and then is nondecreasing with m for each fixed k (Property 2). It follows that $c(x_i)$ is nondecreasing with i. Therefore, the children on which Line 4 does not apply have consecutive indices starting from 1. Such children, say $x_1, \ldots, x_r$, are called *standard children*. The charge of the other children can be stored with $O(\log n)$ bits since only x_{r+1} might have a non null charge, all the other children with larger indices having a null charge (so it suffices to store their number).

We are left with the coding of standard children. Observe that given q, each integer $c(x_i)$ is entirely determined by the pair (k, m), using function W. We have $k \in \{0, \ldots, \lfloor \log p \rfloor\}$, and from Property 3 on W, $m \in \{0, \ldots, q-1\}$. There is at most $kq = O(\log n \log_t n)$ different pairs (k, m), thus as many different values of charge.

Since $c(x_i)$ is nondecreasing with i, the sub-sequence $S = (c(x_1), \ldots, c(x_r))$ can be efficiently represented as two integer sequences $(c_1, \ldots, c_p)$ and $(a_1, \ldots, a_p)$ such that a_i is the number of occurrences of c_i in S. For instance, if $S = (1, 1, 1, 4, 8, 8, 8, 8)$, then S is coded by $(1, 4, 8)$ and $(3, 1, 4)$. Clearly, in this representation all the c_i's are different, so $p \le kq$. Therefore, S can be coded with $O(kq \log n)$ bits since $c_i, a_i \le O(n)$.

So the number of bits to represent $\vec{c}(x)$ is at most:

$$O(t \log n + kq \log n) \;=\; O(t \log n + \log^2 n \log_t n) \;=\; O(\log^3 n/\log\log n)\ .$$

We now examine the time complexity of a range query z on $\vec{c}(x)$. The problem reduces to a query on S, the standard children sub-sequence. We have seen that $c(x_i)$ is nondecreasing with i, so do the c_i's. Let $s_i = \sum_{j \le i} c_j$ be the ith partial sum of S. Storing the s_i's instead of the c_i's one can then answer to any range query z on S in $O(\log(kq)) = O(\log\log n)$ time by the use of binary search. For each c_i of S we also store the least index l_i such that

$c(x_{l_i}) = c_i$. Note that the storage of the s_i's and l_i's does not increase the space complexity. We can check that the answer to the range query z on S is i_0 given by:

$$i_0 \ := \ l_i + \left\lfloor \frac{z - s_i}{a_i} \right\rfloor \ .$$

that completes the proof of Lemma 4.6. □

The time complexity of the routing in $x \neq s$ is bounded by a range query in $\vec{c}(x)$, since the other tasks consist in search in tables of size $O(\log n)$ (so in $O(\log\log n)$ time using binary search), or consist in routing with tree-routing label that takes constant time. The source however spends $O(\log n)$ time to initialize the header with $h(v)$. To complete the proof of Theorem 4.3, we show:

Lemma 4.7 *The memory requirement for x is $O(\log^4 n/(\log\log n)^2 + \log|\mathcal{U}|\log^2 n/\log\log n)$ bits.*

Proof. In our scheme, every node $x \neq s$ stores $\ell_T(x)$, $\ell_T(s)$, $f(x)$, $q(x)$, $\vec{c}(x)$, and a list of $q(x)$ keys. At each key k of this list we correspond a table composed of all the names of $\mathcal{U}$ that collide in k, i.e., all $u \in \mathcal{U}$ such that $h(u) = k$, and their tree-routing label $\ell_T(u)$. From Lemma 4.4, there are at most $O(\log n)$ such collisions. All together, and using Lemma 4.6, it sums to:

$$\begin{aligned} &O(\log^3 n/\log\log n) + q(x)O(\log n)(\log|\mathcal{U}| + O(\log^2 n/\log\log n)) \ = \\ &O(\log^4 n/(\log\log n)^2 + \log|\mathcal{U}|\log^2 n/\log\log n) \end{aligned}$$

since we have seen that $q(x) = O(\log n/\log\log n)$. The source s stores an extra $O(\log^2 n)$ bits for the hash function h given by Lemma 4.4, which does not change the above complexity. □

References

[1] I. Abraham, C. Gavoille, and D. Malkhi, *Routing with improved communication-space trade-off*, Tech. Rep. RR-1330-04, LaBRI, University of Bordeaux 1, 351, cours de la Libération, 33405 Talence Cedex, France, July 2004.

[2] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup, *Compact name-independent routing with minimum stretch*, in 16^{th} Annual ACM Symposium on Parallel Algorithms and Architecture (SPAA), ACM Press, 2004, pp. 20–24.

[3] M. Arias, L. Cowen, K. Laing, R. Rajaraman, and O. Taka, *Compact routing with name independence*, in 15^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), ACM Press, June 2003, pp. 184–192.

[4] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg, *Compact distributed data structures for adaptive routing*, in 21^{st} Annual ACM Symposium on Theory of Computing (STOC), ACM Press, May 1989, pp. 479–489.

[5] B. Awerbuch, A. B. Noy, N. Linial, and D. Peleg, *Improved routing strategies with succinct tables*, Journal of Algorithms, 11 (1990), pp. 307–341.

[6] B. Awerbuch and D. Peleg, *Sparse partitions*, in 31^{th} Annual IEEE Symposium on Foundations of Computer Science (FOCS), Oct. 1990, pp. 503–513.

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

[8] J. L. Carter and M. N. Wegman, *Universal hash functions*, Journal of Computer and System Sciences, 18 (1979), pp. 143–154.

[9] L. J. Cowen, *Compact routing with minimum stretch*, Journal of Algorithms, 38 (2001), pp. 170–183.

[10] P. Fraigniaud and C. Gavoille, *Routing in trees*, in 28^{th} International Colloquium on Automata, Languages and Programming (ICALP), vol. 2076 of Lecture Notes in Computer Science, Springer, July 2001, pp. 757–772.

[11] P. Fraigniaud and C. Gavoille, *A space lower bound for routing in trees*, in 19^{th} Annual Symposium on Theoretical Aspects of Computer Science (STACS), vol. 2285 of Lecture Notes in Computer Science, Springer, Mar. 2002, pp. 65–75.

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

[13] C. Gavoille and M. Gengler, *Space-efficiency of routing schemes of stretch factor three*, J. of Parallel and Distributed Computing, 61 (2001), pp. 679–687.

[14] C. Gavoille and D. Peleg, *Compact and localized distributed data structures*, J. of Distributed Computing, 16 (2003), pp. 111–120. PODC 20-Year Special Issue.

[15] C. Gavoille and S. Pérennès, *Memory requirement for routing in distributed networks*, in 15^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM Press, May 1996, pp. 125–133.

[16] K. Laing, *Name-independent compact routing in trees*, Tech. Rep. 2003-02, Tufts Univ. Dep. of Comp. Science, Nov. 2003. Also in PODC '04 as brief announcements.

[17] R. Motwani and P. Raghavan, *Randomized Algorithms*, Camb Univ Press, 1995.

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

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

[20] N. Santoro and R. Khatib, *Labelling and implicit routing in networks*, The Computer Journal, 28 (1985), pp. 5–8.

[21] M. Thorup and U. Zwick, *Approximate distance oracles*, in 33^{rd} Annual ACM Symposium on Theory of Computing (STOC), July 2001, pp. 183–192.

[22] ——, *Compact routing schemes*, in 13^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), ACM Press, July 2001, pp. 1–10.

[23] P. van Emde Boas, *Preserving order in a forest in less than logarithmic time and linear space*, Information Processing Letters, 6 (1977), pp. 80–82.

[24] J. van Leeuwen and R. B. Tan, *Computer networks with compact routing tables*, in The Book of L, Springer-Verlag, 1986, pp. 259–273.