

# Compact roundtrip routing with topology-independent node names

Marta Arias<sup>a,1</sup>, Lenore J. Cowen<sup>b,2</sup>, Kofi A. Laing<sup>b,\*,3</sup>

<sup>a</sup> Center for Computational Learning Systems, Columbia University, New York, NY 10115, USA

<sup>b</sup> Department of Computer Science, Tufts University, Medford, MA 02155, USA

Received 9 November 2004; received in revised form 24 January 2007

Available online 14 September 2007

## Abstract

Consider a strongly connected directed weighted network with  $n$  nodes. This paper presents compact roundtrip routing schemes with  $\tilde{O}(\sqrt{n})$  sized local tables<sup>4</sup> and stretch 6 for any strongly connected directed network with arbitrary edge weights. A scheme with local tables of size  $\tilde{O}(\epsilon^{-1}n^{2/k})$  and stretch  $\min((2^{k/2} - 1)(k + \epsilon), 8k^2 + 4k - 4)$ , for any  $\epsilon > 0$  is also presented in the case where edge weights are restricted to be polynomially-sized. Both results are for the topology-independent node-name model. These are the first topology-independent results that apply to routing in directed networks.

© 2007 Elsevier Inc. All rights reserved.

**Keywords:** Compact routing; Roundtrip; Name-independence; Digraphs; Stretch; Distributed lookup tables

## 1. Introduction

This paper concerns compact roundtrip routing for strongly connected directed graphs in the *topology-independent* node-name model first introduced by Awerbuch and Peleg in 1989. Most recent results in compact routing (and all previous results on compact roundtrip routing in directed networks) (see, e.g., [13,14,17,18,24,25,35,39] and the surveys of [19,44]) have been in a model where the routing scheme designer may assign his/her own  $O(\log n)$ -bit or sometimes  $O(\log^2 n)$ -bit node labels, dependent on network topology. That is, “ $i$ ” has been renamed, not by some arbitrary permutation  $P$  but by the routing scheme designer, in order to give maximum information about the underlying topology of the network, and the packet destined for  $i$  arrives instead with its informative rename. An alternate but equivalent formulation is that a packet destined for  $i$  arrives also with a short  $O(\log n)$ -bit address in its header, chosen by the compact routing scheme designer, dependent on network topology. This is equivalent because the (short address, original name) pair could be assigned to rename a node. For example, if the underlying network was a planar grid, in the topology-dependent model, the algorithm designer could require a packet destined for a node to come addressed (or renamed) with its  $(x, y)$  coordinates.

\* Corresponding author. Fax: +1 617 627 3220.

E-mail addresses: [marta@cs.columbia.edu](mailto:marta@cs.columbia.edu) (M. Arias), [cowen@cs.tufts.edu](mailto:cowen@cs.tufts.edu) (L.J. Cowen), [laing@cs.tufts.edu](mailto:laing@cs.tufts.edu) (K.A. Laing).

<sup>1</sup> Supported in part by NSF grant IIS-0099446 while at Tufts University.

<sup>2</sup> Supported in part by NSF grant CCR-0208629.

<sup>3</sup> Supported in part by NSF grant EHR-0227879.

<sup>4</sup> The  $\tilde{O}(R)$  notation denotes  $O(R \log^{(1)} R)$ .

In [6], Awerbuch et al. argue that while topology-dependent node labels might be fine for static networks, they make less sense in a dynamic network, where the network topology is changing over time. There are serious consistency and continuity issues if the identifying label of a node changes as network topology evolves. In such a model, a node's identifying label needs to be decoupled from network topology. In fact, network nodes should be allowed to choose arbitrary names (subject to the condition that node names are unique), and packets destined for a particular node enter the network with the node name only, with no additional topological address information. In the grid example above, the packet would come with a destination name independent of its  $(x, y)$  coordinates, and would have to learn how to associate its  $(x, y)$  coordinates with its name from the local routing tables as it wandered the network. Below, we call this the TINN model (for topology independent node names).

Awerbuch et al., in the same paper where they introduced the TINN model, produced the first TINN compact routing schemes for undirected networks. It achieved a stretch of  $O(k^2 \cdot 9^k)$  using  $O(kn^{1/k} \log n)$  space in each node [6]. A paper of Awerbuch and Peleg in the following year [8], presented an alternate scheme with a polynomial space/stretch tradeoff, achieving a stretch of  $O(k^2)$  using  $O(kn^{1/k} \log n \log D)$  space in each node, where  $D$  is the diameter of the network.

In 2003, joint with Rajaraman and Taka [3,4], we presented TINN compact routing schemes that use local routing tables of size  $\tilde{O}(n^{1/2})$ ,  $O(\log^2 n)$ -sized packet headers, and obtained a stretch of 5. For smaller table-size requirements, the ideas in these schemes were generalized to a scheme that uses  $O(\log^2 n)$ -sized headers and  $\tilde{O}(k^2 n^{2/k})$ -sized tables, and achieved a stretch of  $\min\{1 + (k - 1)(2^{k/2} - 2), 16k^2 - 8k\}$ . The following year Abraham et al. improved the stretch of the Arias et al. scheme to 3, which is optimal [2]. Additionally, for the special case of tree networks, Laing [27,28] presented a TINN compact routing algorithm that uses  $\tilde{O}(n^{1/k})$  space,  $O(\log n)$  headers, and achieves stretch  $2^k - 1$ . Abraham, Gavoille and Malkhi [23] recently obtained  $O(k)$  stretch with  $\tilde{O}(n^{1/k})$  table size and arbitrary edge weights which is asymptotically optimal [22].

All the low-stretch schemes cited above, as well as the schemes with exponential tradeoffs are highly dependent on small dominating set landmark selection schemes first pioneered by Awerbuch and Peleg [6]; the polynomial tradeoff schemes are based on ideas from their sparse partitions data structures [8], both developed for compact routing in the name-dependent model. All previous papers in the TINN model, as do our current results, depend heavily on the distributed dictionary ideas pioneered by [6,30]. Arias et al. [3,4] also introduced a novel coloring idea that was subsequently improved and extended by Laing [27,28] and Abraham et al. [2]. However, all previous work in the TINN model has been only for undirected networks.

In fact, no results are known for constructing (one-way, topology-dependent) compact routing schemes on directed networks; and it appears that it is hard to design “compact” routing schemes when the network is directed. For example, it is shown in [16] that distinguishing between pairs of vertices at distances 2 and  $\infty$  even in *unweighted* directed graphs is at least as hard as Boolean matrix multiplication. Roditty et al. [35] observe that sparse spanners do not exist for all digraphs, and there is further discussion in [13,41]. Cowen and Wagner [11,13] made the observation that in directed graphs, instead of bounding the length of a one-way path from node  $x$  to node  $y$  in terms of the shortest distance  $d(x, y)$ , we could bound the length of a roundtrip from node  $x$  through node  $y$  in terms of a shortest cycle between the two nodes, which is of length  $d(x, y) + d(y, x)$ . This would account for a packet and its acknowledgment, for example. As observed by Cowen and Wagner [11,13], sparse roundtrip spanners do exist in this model, and can be used as a basis for compact roundtrip routing schemes in directed networks where they presented the first sublinear space universal compact routing schemes for directed networks, obtaining a stretch of  $2^{k+1}$  for tables of size  $\tilde{O}(kn^{3^{k+1}} \cdot 2 \cdot 3^k)$ . The (name-dependent) roundtrip routing scheme of [11,13] was subsequently improved by Roditty et al. [35] to a stretch of  $4k + \epsilon$  for  $\tilde{O}(\frac{k^2}{\epsilon} n^{1/k})$  using ideas of Awerbuch et al. [7], Cohen [10] and Thorup and Zwick [38].

Here we present the first universal compact roundtrip routing schemes for (positive weighted) directed networks in the TINN model. A preliminary version of this paper appeared in PODC 2003 [5].

### 1.1. Model and definitions

**Definitions.** Let  $G = (V, E)$  be a strongly connected (meaning that there exists a directed path between every source and every destination) directed graph with  $n$  nodes and  $m$  edges, and positive real weights  $w(i, j)$  on each directed edge  $(i, j) \in E$ . In Sections 3 and 4, we further assume that the weights fall in the range  $[1, W]$ , where  $W$  is of size at most a polynomial in  $n$ , that is  $W = O(n^{O(1)})$ . Let  $d(i, j)$  denote the length of a minimum-weight path from  $i$  to  $j$ ;

the distance from  $i$  to  $j$ . Let  $p(u, v)$  be the length of a path  $p$  from  $u$  to  $v$ . The *stretch* of  $p$  is defined as the ratio  $p(u, v)/d(u, v)$ . We define the *roundtrip distance* between  $i$  and  $j$  as  $r(i, j) = d(i, j) + d(j, i)$ , the minimum cost of a directed tour beginning and ending at  $i$ , and passing through  $j$ . Notice that  $r(i, j) = r(j, i)$ .

### 1.1.1. The routing scheme

A *roundtrip routing scheme* is a distributed algorithm defined on a network where a packet  $P$  that is labeled with destination “ $t$ ” arrives at a source node  $s$ , the local routing algorithms route the packet successfully to  $t$ , and an acknowledgment or reply packet is routed back to  $s$  using only information stored in each node’s local routing table. Given an input strongly connected directed network the routing scheme must specify: (1) the construction of the routing table at each node  $v$ , (2) a forwarding function  $F(\text{table}(x), \text{header}(P))$  computed locally at each node  $x$  which specifies the outgoing edge on which to route  $P$ , and the new header of  $P$ . (Note that in the TINN model defined below, packets always arrive with  $\text{header}(P)$  consisting only of the name of the packet’s destination.)

Given a source node  $u$  and a destination node  $v$  and a packet  $P$ , the sequence of nodes  $\langle u = v_0, v_1, \dots, v_k \rangle$  obtained by successive applications of  $F(\text{table}(v_i), \text{header}(P))$  must be such that  $k$  is finite, and  $v_k = v$ . We refer to this  $(u, v)$ -path as the route  $P_{uv}$  from  $u$  to  $v$ . A roundtrip  $(u, v)$  path consists of  $P_{uv}$  followed by  $P_{vu}$ , but where the header of the packet on the return trip path may contain topological information about the network discovered on the forward path. A roundtrip routing scheme is called *compact* if packet headers are polylogarithmic in size, and all local routing tables are sublinear in size. A compact roundtrip routing scheme has stretch  $\alpha$  if and only if, the length of the path to route from  $s$  to  $t$ , and route the acknowledgment back, is of length at most  $\alpha \times r(s, t)$ .

### 1.1.2. Node names

In this paper, we study compact roundtrip routing in the TINN model where node names are *topology-independent*. In particular, we assume node names of the vertices of  $V$  are an arbitrary permutation of  $\{0, \dots, n - 1\}$ . At first glance, this is too weak a model: that is, what we ultimately want for practical application in distributed networks is a model where nodes with only local network knowledge can choose their own  $O(\log n)$ -bit names; requiring them to be exactly a permutation of the integers  $\{0, \dots, n - 1\}$  seems to require their assignment by a centralized entity that, among other things, knows exactly the number of nodes in the network. However, a reduction in [4] shows that, if nodes choose their own names from a range space sufficiently large, they will be unique with high probability, and that these names can be hashed to the values  $\{0, \dots, n - 1\}$  with small numbers of collisions. It is straightforward to adapt our protocols to this setting with only a constant blowup in the size of the routing tables.<sup>5</sup> For details see [4]; the generalization of this hashing scheme to roundtrip routing in directed graphs is entirely straightforward. So in this paper, we assume the labels are a permutation of  $\{0, \dots, n - 1\}$  and implicitly apply the reduction at the end to handle the more general case.

### 1.1.3. Edge names

Each node  $v$  is also assumed to have a unique name from a set of size  $O(n)$  assigned to each outgoing edge; again these names are assumed to be assigned by an adversary, with no global consistency. As an illustrative example, suppose  $u$  and  $v$  are adjacent, and say  $u$  is assigned the unique node name 1 and  $v$  is assigned the unique node name 5. However,  $u$ ’s link to  $v$  may be labeled port 200 while  $v$ ’s link to  $u$  may be labeled port 1080, where these numbers have no relation to 1 and 5. In addition,  $v$  may have another link called port 200, but this might go to a different vertex  $y$ ! This is equivalent to the model that Fraigniaud and Gavoille call the *fixed-port* model [18]; in contrast, the *designer-port* model considered by [18] allows the network designer to specify port names dependent on global network topology.

### 1.1.4. Headers

Some recent topology-dependent compact routing schemes have been able to “wire-in” the routing information to a short packet header that arrives with the packet, and so intermediate nodes do not have to modify packet headers. In the TINN model, as routing information is discovered, it will be written into the header of the packet before it is

<sup>5</sup> In fact this reduction will continue to work in a model where an adversary chooses the node names, provided they are required to be unique, and the adversary chooses the node names before the protocol selects the particular hash function from the family of universal hash functions to map the node names, or else the adversary could force too many collisions.

	Table size	Roundtrip	Name independent	Stretch
[39]	$\tilde{O}(n^{1/2})$	X	X	3
[35]	$\tilde{O}(n^{1/2})$	✓	X	3
[2]	$\tilde{O}(n^{1/2})$	X	✓	3
<b>This Paper</b>	$\tilde{O}(n^{1/2})$	✓	✓	<b>6</b>
[4]	$\tilde{O}(n^{2/k})$	X	✓	$1 + (k - 1)(2^{k/2} - 2)$
[1]	$\tilde{O}(n^{2/k})$	X	✓	$O(k)$
<b>This Paper</b>	$\tilde{O}(n^{2/k})$	✓	✓	$\min \begin{cases} (2^{k/2} - 1)(k + \epsilon), \\ 8k^2 + 4k - 4 \end{cases}$

Fig. 1. Best stretch routing schemes known that use sublinear-sized local routing tables in the various models. This paper gives the first roundtrip routing scheme with topology-independent node names.

forwarded. All TINN schemes therefore require writable packet headers. While packets initially arrive with  $O(\log n)$ -bit names, in our schemes we will write up to  $O(\log^2 n)$  bits of learned routing information into the packet header.

### 1.2. Our results

A key idea in all our schemes is that of a distributed dictionary, first introduced by Peleg [30]. We also incorporate ideas introduced by Awerbuch and Peleg [9]. In each of our schemes, we assign blocks of dictionary entries to nodes in a balanced way, while ensuring that the entire address space is covered in some neighborhood structure (or recursive neighborhood structure for the later schemes). We also make use of the (topology-dependent) roundtrip routing scheme of Roditty et al. [35]. In designing roundtrip schemes, we note that the main difficulty usually comes from the following: while our measure, roundtrip distance, averages the distances  $d(u, v)$  and  $d(v, u)$ , we still have to route along one-way edges. Thus the cost of return from a lookup must be appropriately amortized, even though we cannot in general retrace the same path back.

In Section 2, we present the first compact roundtrip routing scheme in the TINN model. The algorithm uses local routing tables of size  $\tilde{O}(\sqrt{n})$ , packet headers of size  $O(\log^2 n)$ , and achieves stretch 6. In Sections 3 and 4, we generalize this scheme to achieve stretch/space tradeoffs: with tables of size bounded by  $\tilde{O}(\epsilon^{-1}n^{2/k})$ , one scheme achieves stretch  $k + 2^{k/2}(k + \epsilon)$ , and the second scheme achieves stretch  $8k^2 + 4k - 4$  (where  $k$  and  $\epsilon$  are constants; both tradeoff schemes require that edge weights be bounded by a polynomial in  $n$ ). The first generalized scheme follows easily from the definition of roundtrip routing and a result in [4], the second general scheme involves some new ideas (i.e., using double tree covers that subsume the roundtrip distance of a node and its entire neighborhood in the same set) and produces the best space/stretch tradeoff for larger  $k$ . Putting these last two results together yields the result in Fig. 1.

Finally, some work has been done on lower bounds for (one-way) compact routing in undirected graphs, that applies to the TINN model. In particular, a construction of Gavaille and Gengler [20] implies that any compact routing scheme that uses  $o(n)$ -sized tables at every node in the TINN model, must have stretch  $\geq 3$ . (In fact, the result of [20] is stronger; the lower bound holds even when the packet arrives with up to  $\log_2 n$  bits of topology-dependent routing information.) We show below that this result implies a stretch lower bound of 2 for compact roundtrip routing in the TINN model.

## 2. Stretch 6 scheme

In this section, we construct a TINN compact roundtrip routing scheme with  $\tilde{O}(n^{1/2})$ -sized routing tables,  $O(\log^2 n)$ -sized routing headers, while achieving stretch 6.

Following [12] and [35], we define the roundtrip distance metric as follows: let  $G = (V, E)$  be a positive edge-weighted directed graph. Recall that  $d(u, v)$  denotes the shortest path from  $u$  to  $v$ , and  $r(u, v) = d(u, v) + d(v, u)$ . Two nodes  $u$  and  $w$  are related by  $u <_v w$  (read:  $u$  is closer to  $v$  than  $w$  is, by the roundtrip metric) if and only if one of the following is true ( $ID_u$  refers to the index of  $u$  in a listing of  $V$ , and could simply be  $u$  when  $V = [n]$ ):

- (1)  $r(v, u) < r(v, w)$ ,

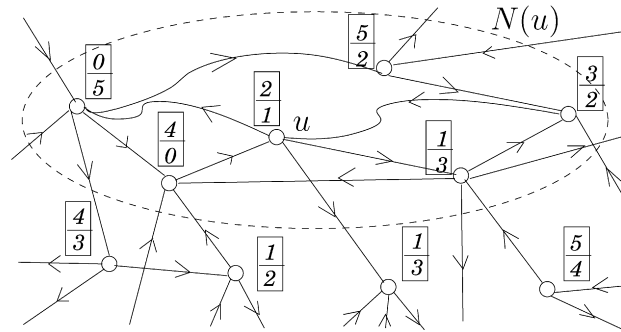


Fig. 2. A block distribution example showing a section of a 36 node directed graph, with a node  $u$  and its neighborhood  $N(u)$ . The example is unweighted, but the lemma works for positive weighted graphs as well. In this case the possible block labels are  $\{0, \dots, 5\}$ . The sets  $S_v$  are indicated as vertical stacks of block labels. Notice that in the first (lower) “layer,” there is no “4” in  $N(u)$ , but on the second layer there is a “4.” The block distribution lemma guarantees that we never need more than  $O(\log n)$  blocks per node (for constant  $k$ ) to ensure that each neighborhood contains each type of block.

- (2)  $r(v, u) = r(v, w)$  and  $d(u, v) < d(w, v)$ ,
- (3)  $r(v, u) = r(v, w)$  and  $d(u, v) = d(w, v)$  and  $ID_u < ID_w$ .

This produces a total order of  $V$  for each node  $v$ :  $v <_v u_1 <_v u_2 <_v \dots <_v u_{n-1}$ . We call this sequence  $Init_v$ . Additionally, we define  $u \preceq_v w$  to mean  $u <_v w$  or  $u = w$ .

Given a positive weighted directed graph  $G$ , we determine for each node  $u$ , a neighborhood ball  $N(u)$  of the first  $n^{1/2}$  nodes in  $Init_u$ .

We assume for simplicity that  $n$  is a perfect square, and divide the address space  $\{0, \dots, n - 1\}$  into  $\sqrt{n}$ -sized blocks  $B_i$ , for  $i = 0, \dots, \sqrt{n} - 1$ , such that block  $B_i$  contains information about the nodes labeled  $i\sqrt{n}$  to  $(i + 1)\sqrt{n} - 1$  (where there is no ambiguity we sometimes simply say the block  $B_i$  is the set of labels). Each node  $i$  will store a particular set of blocks  $S_i$ , such that every node is close enough to a node which stores each type of block. This is illustrated in Fig. 2.

**Lemma 1.** *Let  $G$  be a directed graph on  $n$  nodes, and let  $N(v)$  denote a set of the first  $\sqrt{n}$  nodes of  $Init_v$ . Let  $\{B_i \mid 0 \leq i < \sqrt{n}\}$  denote a set of blocks. There exists an assignment of sets  $S_v$  of blocks to nodes  $v$ , so that*

- $\forall v \in G, \forall i, 0 \leq i < \sqrt{n}$ , there exists an  $j \in N(v)$  with  $B_i \in S_j$ ,
- $\forall v \in G, |S_v| = O(\log n)$ .

**Proof.** This is a restatement of the lemma proved in [4], the only difference being  $N(v)$  is now defined in terms of  $Init_v$  and roundtrip distance. The proof is identical.  $\square$

We also make use of the following result of Roditty et al. [35] in the topology-dependent model (note that a proof appears in [34]):

**Lemma 2.** *(See [35].) There exists a name-dependent compact roundtrip routing algorithm for strongly connected directed graphs with stretch 3, which uses  $\tilde{O}(n^{1/2})$  space. The path taken by a packet when routed from  $u$  to  $v$  in this scheme satisfies  $p(u, v) \leq r(u, v) + d(u, v)$ .*

In the following, let  $Tab_3(x)$  refer to the storage table at node  $x$ , and let  $R_3(x)$  be the topology-dependent address of node  $x$ , according to a stretch three roundtrip routing scheme as described in Lemma 2.

### 2.1. Storage requirements

#### 2.1.1. Storage

Each node  $u$  stores the following in its local routing table:

- (1) For every node  $v$  in  $N(u)$ ,  $(v, R_3(v))$ .
- (2) For every  $i$ ,  $0 \leq i < \sqrt{n}$ ,  $(i, t)$ , where  $t \in N(u)$  satisfies  $B_i \in S_t$  (such a node  $t$  exists by our construction of  $S_u$  in Lemma 1).
- (3) For every block  $B_k$  in  $S_u$ , and for each node  $j$  in  $B_k$ , the dictionary entry  $(j, R_3(j))$ .
- (4)  $u$  stores the routing table  $Tab_3(u)$ .

### 2.1.2. Analysis of space requirements

It is easy to verify that these entries take  $\tilde{O}(n^{1/2})$  space. (1) Takes  $\tilde{O}(\sqrt{n})$  space by definition of  $N(u)$ . (2) Consists of  $\tilde{O}(\sqrt{n})$  entries each of constant size. For (3) we note that since we are storing  $\tilde{O}(1)$  information for each of the  $\sqrt{n}$  nodes in each block, it takes  $\tilde{O}(\sqrt{n})$  space per block times the number of blocks that are stored at a node (which is  $O(\log n)$ ), for a total of  $\tilde{O}(\sqrt{n})$  space per node. Finally, (4) takes  $\tilde{O}(\sqrt{n})$  space by Lemma 2.

### 2.2. Algorithm and stretch analysis

The local routing algorithm is presented as pseudocode in Fig. 3 on page 781. When a packet enters the network at source node  $s$  it is labeled with its topology-independent destination node name  $t$ , and a *Mode* variable which is set to *NewPacket*. Other header fields are empty.

When a reply packet is sent, *Mode* is set to *ReturnPacket* before the routing algorithm receives it. Other *Mode* values are self-explanatory from the pseudocode. The distinction from a *NewPacket* is that some topology-dependent information learned in the original direction may now appear in the header of packet that is being acknowledged. Functions *GetR3Label()* and *GetRTZNextEdge()* both look up information stored in the local routing table.

The stretch analysis proceeds as follows: Let  $s$  be the source node and  $t$  the destination node. There are two cases to consider.

- (1)  $t \in N(s)$ : Then the entry  $(t, R_3(t))$  is stored at node  $s$ , by (1) above. So we can route to  $t$  and back to  $s$  with a stretch of 3, by Lemma 2, using the tables stored in (4).
- (2)  $t \notin N(s)$ : If  $(t, R_3(t))$  is stored at node  $s$ , this is the same as case (1). If we fail to find  $(t, R_3(t))$  stored at  $s$ , it must be that  $t \notin N(s)$ . We then compute the index  $i$  for which  $t \in B_i$ , and look up the node  $w \in N(s)$  that stores entries for all nodes in  $B_i$ . Now we route to node  $w$ , where we look up  $R_3(t)$ , and then route to  $t$  using Lemma 2. The return trip to  $s$  is accomplished using  $R_3(s)$ , which is contained in the packet header.

**Lemma 3.** *Given a strongly connected directed graph with arbitrary edge weights, the compact roundtrip routing algorithm above uses  $\tilde{O}(\sqrt{n})$  space,  $\tilde{O}(1)$  sized packet headers, and achieves stretch 6.*

**Proof.** If  $t \in N(s)$ , the algorithm costs stretch 3, by Lemma 2. Otherwise, the length  $\tilde{r}(s, t)$  of the roundtrip route taken from  $s$  to  $w$  to  $t$  and back to  $s$  is given by

$$\begin{aligned} \tilde{r}(s, t) &\leq p(s, w) + p(w, t) + p(t, s) \\ &\leq r(s, w) + d(s, w) + r(w, t) + d(w, t) + r(t, s) + d(t, s) \end{aligned} \quad (2)$$

$$\leq r(s, w) + d(s, w) + r(w, s) + r(s, t) + d(w, t) + r(t, s) + d(t, s) \quad (3)$$

$$\leq 4r(s, t) + d(s, w) + d(w, t) + d(t, s) \quad (4)$$

$$\leq 4r(s, t) + d(s, w) + d(w, s) + d(s, t) + d(t, s) \quad (5)$$

$$\leq 6r(s, t). \quad (6)$$

Equation (2) follows by Lemma 2, and thus certainly (3) follows, by the triangle inequality for roundtrip distances. But since  $w \in N(s)$  but  $t \notin N(s)$ , it must be that  $r(s, w) \leq r(s, t)$  so this gives Eq. (4), from which (5) follows by the triangle inequality. Using again the fact that  $r(s, w) \leq r(s, t)$ , we obtain (6), whence the result.  $\square$

We also note that the algorithm could operate by routing from  $s$  to  $w$  and back to  $s$ , before routing to  $t$  and back. This would be slightly simpler to analyze and would result in the same worst-case stretch. However it can result in longer paths since it always routes back through  $s$  when routing from  $w$  to  $t$ .

**Algorithm StretchSix():**

```

upon receipt of packet P at node s:

    // initialize local variables from the packet header
    ReadPacketHeader()

    if (Mode = NewPacket):
        Mode ← Outbound
        SrcID ← MyNodeID
        SrcLabel ← GetR3Label(SrcID)
        if (DestID is within neighborhood according to local table)
            NextID ← DestID
        else: // remote dictionary lookup is needed
            DictID ← GetLookupNodeID(MyNodeID, DestID)
            NextID ← DictID
            NextLabel ← GetR3Label(NextID)

    else if (Mode = ReturnPacket):
        Mode ← Inbound
        NextID ← SrcID
        NextLabel ← SrcLabel

    else if (Mode = Outbound and MyNodeID = DictID):
        DestLabel ← GetR3Label(DestID)
        NextID ← DestID           NextLabel ← DestLabel

    else if ((Mode = Outbound and MyNodeID = DestID) or
              (Mode = Inbound and MyNodeID = SrcID)):
        Deliver the packet to the host node
        exit(Success)

    endif

    // write modified local variables back into packet header
    WritePacketHeader()
    // forward packet
    NextEdge ← GetRTZNextEdge(NextLabel)
    Forward the packet P on NextEdge
    exit(Success)

```

Fig. 3. Distributed pseudocode for stretch six algorithm.

**3. A generalized routing scheme**

Our first generalized routing scheme is a straightforward generalization of the exponential scheme of Arias et al. [3,4]. Unfortunately, everything needs to be redefined and proved because the underlying neighborhood structure changes in the roundtrip metric.

*3.1. Preliminaries*

We use a randomized distribution of blocks of lookup information first introduced by Arias et al. [4]. The following description is nearly syntactically identical to the one in [4] for the undirected case, but the neighborhoods themselves will be different (since they are based on roundtrip distance), and thus the actual information that is stored will be different as well.

Given a directed graph  $G$  with  $n$  nodes, we assume for simplicity that  $n$  is a  $k$ th power, and define the alphabet  $\Sigma = \{0, \dots, n^{1/k} - 1\}$ . For each  $0 \leq i \leq k$ ,  $\Sigma^i$  is the set of words over  $\Sigma$  of length  $i$ . Let  $\langle u \rangle \in \Sigma^k$  be the base

$n^{1/k}$  representation of  $u$ , padded with leading zeros so it is of length exactly  $k$ . For each  $0 \leq i \leq k$ , we also define functions  $\sigma^i: \Sigma^k \rightarrow \Sigma^i$ , such that  $\sigma^i((a_0, \dots, a_{k-1})) = (a_0, \dots, a_{i-1})$ . That is,  $\sigma^i$  extracts the prefix of length  $i$  from a string  $\alpha \in \Sigma^k$ .

For each  $\alpha \in \Sigma^{k-1}$ , define the sets  $B_\alpha = \{u \in V \mid \sigma^{k-1}(\langle u \rangle) = \alpha\}$ . We will call these sets *blocks*. Clearly  $|B_\alpha| = n^{1/k}$ . We abuse notation slightly by defining  $\sigma^i(B_\alpha) = \sigma^i(\alpha 0)$ , where  $\alpha 0$  is the word in  $\Sigma^k$  obtained by appending a 0 to  $\alpha$ . Note that by this definition,  $\sigma^{k-1}(B_\alpha) = \sigma^{k-1}(\langle u \rangle)$  if and only if  $u \in B_\alpha$ .

For every node  $u$ , we define the neighborhoods  $N^i(u)$  as the set of the first  $n^{i/k}$  nodes in  $Init_u$ . Now we can state the following lemma, which is originally proved in [4].

**Lemma 4.** (See [4].) *Given a directed graph  $G$ , there exists an assignment of sets of blocks  $S_v$  to nodes  $v$ , so that*

- $\forall v \in G, \forall i, 0 \leq i < k, \forall \tau \in \Sigma^i$ , there exists a node  $w \in N^i(v)$  with  $B_\alpha \in S_w$  such that  $\sigma^i(B_\alpha) = \tau$ ,
- $\forall v \in G, |S_v| = O(\log n)$ .

The proof of the lemma is by the probabilistic method, and it yields a simple randomized procedure for generating the desired assignments of sets of blocks to nodes. Originally it was applied to undirected graphs, but the proof for directed graphs is no different, because the result is a general one on sets of neighborhoods; only the definition of the neighborhoods has changed. Lemma 1, used in Section 2, is a special case of this preceding lemma, given by setting  $k = 2$ .

### 3.2. Required results

Our algorithm uses the roundtrip spanner construction of Roditty et al. [35]. First we need the following definitions. Let  $C$  be a strongly connected set of vertices, and  $v = RTCenter(C)$  its center. We define  $OutTree(C)$  as a shortest paths tree rooted at  $v$  that spans all the vertices in  $C$ . Let  $InTree(C)$  be the tree that consists of a shortest path from every node in  $C$  to the root  $v$ . Let  $DoubleTree(C)$  be the union of the two trees  $InTree(C)$  and  $OutTree(C)$ . Define  $RTHeight(T)$  where  $T$  is a double-tree as the maximum roundtrip distance from the root of  $T$  to any vertex in  $T$ .

**Lemma 5.** (See [35].) *Let  $G$  be a positive weighted strongly connected directed graph on  $n$  nodes with edge weights in the range  $[1, W]$ . For every integer  $k \geq 1$  and every  $\epsilon > 0$ , there exists a  $(2k + \epsilon)$ -roundtrip spanner of  $G$  which includes each node  $v$  in at most  $O(\frac{k^2}{\epsilon} n^{1/k} (\log n)^{1-1/k} \log(nW))$  double-trees.*

Roditty et al. [35] use the spanner of Lemma 5 to construct a routing scheme that achieves stretch  $4k + \epsilon$ . However, they note that there exists for any pair of nodes  $u$  and  $v$ , a  $o(\log^2 n)$  bit “handshake,” consisting of the name of a double-tree that contains both  $u$  and  $v$ , whose knowledge at node  $u$  would allow routing with stretch  $2k + \epsilon$ . We incorporate handshake information into our routing tables and thus can use the  $2k + \epsilon$  scheme of [35] as a subroutine in our exponential algorithm.

### 3.3. Storage

Let  $Tab(x)$  refer to the storage table at node  $x$  in the algorithm of Roditty et al. [35], and let  $R_2(u, v)$  (which is of size  $o(\log^2 n)$  bits) be the minimum of routing information required to route from node  $u$  to node  $v$  and back, in a  $(2k + \epsilon)$ -roundtrip spanner. This information consists of the name of the most convenient double tree  $T$  in the tree cover for routing from  $u$  to  $v$ , as well as the topology-dependent routing addresses of nodes  $u$  and  $v$  within that tree  $T$ . Therefore the routing address  $R_2(u, v)$  does not work from all the nodes in the directed graph. Note that  $R_2(u, v)$  is not exactly the same as the labels assigned by the algorithm of Roditty et al. Instead, they deduce  $R_2(\cdot)$  from their node labels. Their labels, which are of size  $o(\frac{k}{\epsilon} \log^2 n \log(nW))$  bits, are globally valid and can therefore be used for routing to node  $v$  from any node  $u$  in the digraph, with stretch  $4k + \epsilon$ .

Let  $\{S_u \mid u \in V\}$  be a collection of sets of blocks that satisfies Lemma 4. For each node  $u$ , let  $S'_u = S_u \cup \{B_\beta\}$ , where  $u \in B_\beta$  (that is, each node always stores the block its own address belongs to).

Given these definitions, we specify the memory contents of each node  $u$  as follows:

**Algorithm ExStretch():**  
**for**  $i \leftarrow 0$  **upto**  $k - 1$  **step** 1:  
  **if**  $(i + 1 < k)$ :  $v_{i+1} \leftarrow$  closest node to  $v_i$  in the set  
    $N^{i+1}(v_i) \cap \{v \mid \exists B_\beta \in S_v: \sigma^{i+1}(B_\beta) = \sigma^{i+1}((t))\}$   
  **else**:  $v_k \leftarrow t$   
  **if**  $(v_i \neq v_{i+1})$ :  
   push  $R_2(v_i, v_{i+1})$  onto header  
   route to  $v_{i+1}$  along  $Hop(v_i, v_{i+1})$  using  $R_2(v_i, v_{i+1})$   
**for**  $i \leftarrow k$  **downto** 1 **step**  $-1$ :  
  pop  $R_2(v_i, v_{i+1})$  from header  
  route back to  $v_i$  along  $Hop(v_{i+1}, v_i)$  using  $R_2(v_i, v_{i+1})$

Fig. 4. Roundtrip routing algorithm with exponential stretch.

- (1)  $Tab(u)$ .
- (2) For every  $v \in N^1(u)$ , the pair  $(v, R_2(u, v))$ .
- (3) The set  $S'_u$  of  $O(\log n)$  blocks  $B_\alpha$ , and for each block  $B_\alpha \in S'_u$ , the following:
  - (a) For every  $0 \leq i < k - 1$ , and for every  $\tau \in \Sigma$ , we store the routing address  $R_2(u, v)$ , where  $v$  is the nearest node by the roundtrip distance metric which contains a block  $B_\beta$  such that  $\sigma^i(B_\beta) = \sigma^i(B_\alpha)$  and the  $(i + 1)$ st element of  $\sigma^{k-1}(B_\beta)$  is  $\tau$ .
  - (b) For every  $\tau \in \Sigma$ , we store the routing address  $R_2(u, v)$ , where the node  $v$  satisfies  $\sigma^{k-1}(B_\beta) = \sigma^{k-1}(v)$  and the  $k$ th element of  $\sigma^k(v)$  is  $\tau$ .

**Lemma 6.** *The storage requirement of our algorithm is  $\tilde{O}(n^{1/k})$  for fixed  $k$ .*

**Proof.** We need  $\tilde{O}(n^{1/k})$  space for (1). Since  $|N^1(u)| = n^{1/k}$  for all  $u$ , it is clear that (2) also requires  $\tilde{O}(n^{1/k})$  space. For (3) we note that  $|S'_u| = O(\log n)$  blocks. For each block, we store  $kn^{1/k}$  values  $R_2(u, v)$ , where the size of  $R_2(u, v)$  in bits is  $\tilde{O}(1)$ . Therefore the space requirement for (3) is  $\tilde{O}(kn^{1/k})$ . The total of all these space requirements is clearly  $\tilde{O}(n^{1/k} \log(nW))$ , which is  $\tilde{O}(n^{1/k})$  for fixed  $k$  when  $W$  is bounded by  $O(n^{O(1)})$ .  $\square$

### 3.4. Routing algorithm

Given a source node  $s$  and destination node  $t$ , our roundtrip routing algorithm visits a sequence of nodes  $s = v_0, \dots, v_k = t$  (not necessarily distinct) to reach  $t$ , and  $v_{k-1}, \dots, v_0$  to return to  $s$ . Just like in [4], the sequence  $s = v_0, \dots, v_k = t$  has the property that each  $v_i$  (except  $v_k$ ) contains a block  $B_{\beta_i}$  for which  $\sigma^i(B_{\beta_i}) = \sigma^i(t)$ . When  $v_i \neq v_{i+1}$  we route from  $v_i$  to  $v_{i+1}$  along route  $Hop(v_i, v_{i+1})$ , which is the route in the  $(2k + \epsilon)$ -roundtrip-spanner from  $u$  to  $v$ . This is possible because  $R_2(u, v)$  is stored at  $u$ . On the return trip, we route from each  $v_{i+1}$  to  $v_i$  using  $R_2(v_i, v_{i+1})$  which is appended to the header during the outbound phase. Algorithm ExStretch is presented in Fig. 4 and illustrated in Fig. 5. The idea of matching increasing prefixes of node names is well known in the parallel algorithms literature for multidimensional array routing (see [29]); it has also been used more recently in the context of peer to peer systems for locating replicated objects [21,26,31,40], and also for compact routing in undirected graphs [4].

**Lemma 7.** *Algorithm ExStretch correctly delivers packets from any source node  $s$ , to any destination  $t$  and back.*

**Proof.** At each  $v_i$  we read the name-dependent routing information  $R_2(v_i, v_{i+1})$  for routing to the node  $v_{i+1}$ . Delivery to node  $v_{i+1}$  is assured by the correctness of the algorithm of Roditty et al. [35]. The algorithm is guaranteed to find node  $t$ , because in the worst case we have stored information for routing to a node  $v$  in  $N^k(v_{k-1}) = V$  such that  $\sigma^k(v) = \sigma^k(t)$ , and the latter condition implies  $v = t$ . The return trip is successful because we store all the  $R_2(v_i, v_{i+1})$  labels in the header and each one suffices for returning to  $v_i$ .  $\square$

### 3.5. Stretch analysis

We now proceed to analyze the stretch of Algorithm ExStretch in Fig. 4:

**Lemma 8.** *For  $0 \leq i \leq k - 1$ ,  $r(v_i, v_{i+1}) \leq 2^i r(s, t)$ .*

**Proof.** Recall that  $v_i$  is the  $i$ th node visited by the routing algorithm, as defined above. For each  $0 \leq i \leq k$ , let  $v_i^*$  be the closest node to node  $s$  by the roundtrip distance metric  $Init_s$ , such that  $\sigma^i(v_i^*) = \sigma^i(t)$ . The proof is by induction.

For the basis case, we note that based on the algorithm  $r(s, v_1) = r(v_0, v_1) \leq 2^0 r(s, t)$ , since  $t$  itself is a candidate to be  $v_1$ . If  $r(s, t) < r(s, v_1)$ , then  $t$  would have been chosen to be node  $v_1$ , because  $t$  contains a block  $B_\beta$  such that  $\sigma^1(B_\beta) = \sigma^1(t)$ .

The inductive hypothesis is that for all  $i$  such that  $0 \leq i \leq l-1 < k-1$ , we have  $r(v_i, v_{i+1}) \leq 2^i r(s, t)$ . We bound  $r(v_l, v_{l+1})$  as follows:

$$r(v_l, v_{l+1}) \leq r(v_l, v_{l+1}^*) \quad (1)$$

$$\leq r(v_l, s) + r(s, v_{l+1}^*) \quad (2)$$

$$\leq r(s, t) + r(v_l, s) \quad (3)$$

$$\leq r(s, t) + r(s, v_l) \quad (4)$$

$$\leq r(s, t) + \sum_{i=0}^{l-1} r(v_i, v_{i+1}) \quad (5)$$

$$\leq r(s, t) \left[ 1 + \sum_{i=0}^{l-1} 2^i \right] \quad (6)$$

$$\leq 2^l r(s, t)$$

where (1) follows by definition of  $v_{l+1}$  and  $v_{l+1}^*$ ; (2) because  $r(v_l, v_{l+1}^*)$  is a shortest roundtrip distance; (3) follows by commutativity, and because  $t$  is candidate for  $v_{l+1}^*$ ; (4) follows by symmetry; (5) is true because  $r(s, v_l)$  is the shortest distance, and (6) follows by the induction hypothesis.  $\square$

**Theorem 9.** Algorithm *ExStretch* uses space  $\tilde{O}(n^{1/k})$  for fixed  $k$ , headers of size  $o(k \log^2 n)$  and delivers packets correctly with stretch  $(2^k - 1)(2k + \epsilon)$ .

**Proof.** We have already established space requirements and correctness of the algorithm in Lemma 6 and Lemma 7 respectively. The header size of  $o(k \log^2 n)$  is obtained from the  $k$  push operations each of which pushes a  $o(\log^2 n)$  routing label. It only remains to prove the stretch bound. Let  $\tilde{r}(u, v)$  be the roundtrip path taken by our algorithm while routing from  $u$  to  $v$  and back, and let  $\tilde{d}(u, v)$  be the one-way path taken by our algorithm from node  $u$  to node  $v$ :

$$\tilde{r}(s, t) = \sum_{i=0}^{k-1} \tilde{d}(v_i, v_{i+1}) + \sum_{i=0}^{k-1} \tilde{d}(v_{i+1}, v_i) \quad (1)$$

$$\leq \sum_{i=0}^{k-1} \tilde{r}(v_i, v_{i+1}) \quad (2)$$

$$\leq \sum_{i=0}^{k-1} (2k + \epsilon) r(v_i, v_{i+1}) \quad (3)$$

$$\leq (2k + \epsilon) \sum_{i=0}^{k-1} r(v_i, v_{i+1}) \quad (4)$$

$$\leq (2k + \epsilon) \sum_{i=0}^{k-1} 2^i r(s, t) \quad (5)$$

$$\leq (2k + \epsilon) (2^k - 1) r(s, t). \quad (6)$$

Step (1) simply expresses the roundtrip path in terms of its  $2k$  segments. Step (3) uses the stretch of  $(2k + \epsilon)$  of the roundtrip spanner construction of Roditty et al. [35]. Step (5) results from applying Lemma 8.  $\square$

The apparently more sensible approach of routing from  $t$  directly back to  $s$  (without going back through the lookup nodes  $v_{k-1}, \dots, v_1$ ) requires that we also implement the full compact roundtrip routing  $4k + \epsilon$  scheme of Roditty

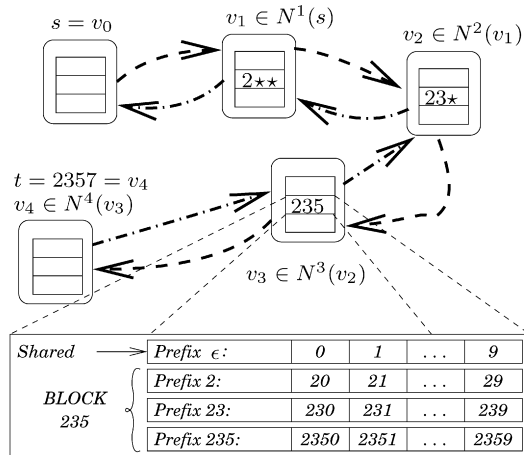


Fig. 5. A schematic of how the prefix-matching algorithm of Theorem 9 works. The figure only includes the sequence of nodes where the distributed dictionary is read—the other nodes in the path are not shown. Each node contains 3 blocks in this example, and the contents of each block are illustrated in the magnified table. Asterisks stand for irrelevant (or arbitrary) digits in block labels. Notice that the blocks that are actually consulted (shown labeled) have prefixes that increasingly match the destination 2357.

et al., and that we include a header of size  $o(\frac{k}{\epsilon} \log^2 n \log(nW))$  for the return trip to  $s$  using the scheme of Roditty et al. This is necessary so that at node  $t$  we can determine a best double-tree for routing directly back to  $s$ . Using this approach we obtain a worse stretch of  $2k + 2^k(2k + \epsilon)$ , in addition to having longer headers and two sets of routing tables.

### 3.6. Distributed implementation notes

In this subsection we provide more detail on the implementation of the *ExStretch* algorithm presented in Fig. 4. A distributed pseudocode version of the algorithm is presented in Fig. 6 on page 786.

Recall that when a packet arrives in the network at a source node  $s$  it is labeled solely with its topology-independent node name *DestinationID*. Other header fields are empty. The task of the local routing node is to update the header fields which include the following: *SourceID*, *NextWaypointID*, *SourceLabel*, *DestinationLabel*, *NextWaypointLabel*, and *Mode*. The subroutine *ReadPacketHeader()* parses the packet header and initializes all local variables from the packet header. The corresponding subroutine *WritePacketHeader()* writes all the local variables into the packet header, so they can be used by the next node.

We also use a *Mode* header field, with the values *NewPacket* for new packets before they are seen by the routing system, *Outbound* for packets which have not yet been delivered to their destination  $t$ , and *Inbound* for packets on the way back to their source  $s$ . When a packet is inserted into the network at the destination for the return trip, the *Mode* is *ReturnPacket*. Note that returning a packet is easier than sending a new packet from *DestinationID* to *SourceID* – because a returned packet is assumed to know some routing information about its source and possibly the destination. The algorithm *ExStretch* uses a stack called *WaypointStack* to store all intermediate lookup nodes (and their routing labels) which it uses to retrace its steps back to the source node. Finally the *Mode* is set to *Delivery* when the packet is passed from a router to its host either at the source or destination.

The function *NextStop()* function is defined in Section 3. We lookup the topology-dependent label associated with a particular node ID using the  $R_2()$  function. By convention,  $R_2()$  returns two topology-dependent labels, one for each endpoint.

After updating a packet header, the packet is forwarded along the edge labeled *NextEdge*. The *NextEdge* variable is assigned a value using the function *RTZNextEdge()* defined in the topology-dependent routing subsystem of Roditty et al. [35].

**Algorithm ExStretch():****upon receipt of packet  $P$  at node  $s$ :**

```

ReadPacketHeader()

if ( $Mode = NewPacket$ ):
     $SourceID \leftarrow MyNodeID$ 
     $Hop \leftarrow 1$ 
     $WaypointStack.initialize\_empty()$ 
     $Mode \leftarrow Outbound$ 
     $NextWaypointID \leftarrow NextStop(SourceID, 1, DestinationID)$ 
     $(NextWaypointLabel, MyWaypointLabel) \leftarrow R\_2(SourceID, NextWaypointID)$ 
     $WaypointStack.push(MyWaypointLabel, MyNodeID)$ 

else if ( $Mode = ReturnPacket$ ):
     $Mode \leftarrow Inbound$ 
     $(NextWaypointLabel, NextWaypointID) \leftarrow WaypointStack.pop()$ 

else if ( $Mode = Outbound$  and  $MyNodeID = DestinationID$ ):
     $Mode \leftarrow Delivery$ 

else if ( $Mode = Inbound$  and  $MyNodeID = SourceID$ ):
     $Mode \leftarrow Delivery$ 

else if ( $Mode = Outbound$  and  $MyNodeID = NextWaypointID$ ):
     $Hop \leftarrow Hop + 1$ 
    if ( $Hop = k$ ):
         $NextWaypointID \leftarrow DestinationID$ 
    else:
         $NextWaypointID \leftarrow NextStop(MyNodeID, Hop, DestinationID)$ 
         $(NextWaypointLabel, MyWaypointLabel) \leftarrow R\_2(MyNodeID, NextWaypointID)$ 

else: // ( $MyNodeID \neq NextWaypointID$ ):
    // do nothing to  $NextWaypointLabel$ 

if ( $Mode = Delivery$ ):
    Deliver the packet to the host node
else:
    WritePacketHeader()
     $NextEdge \leftarrow RTZNextEdge(NextWaypointLabel)$ 
    Forward the packet on  $NextEdge$ 

```

Fig. 6. Pseudocode for distributed exponential stretch algorithm.

**4. A generalized routing scheme with a polynomial tradeoff**

In this section we present a compact roundtrip routing scheme that achieves a polynomial tradeoff between stretch and maximum storage. Unlike the scheme in Theorem 9, the second scheme we present does not follow directly from any of the results in [4]. It uses a new underlying roundtrip cover construction, then prefix matches addresses in a hierarchy of neighborhood covers. The first scheme has an exponential tradeoff between space and stretch; while the stretch/space tradeoff of the second scheme is polynomial. However, for small values of  $k$  ( $k \leq 12$ ), the first scheme gives a better tradeoff than the second; putting the two results together gives the bound claimed in the abstract.

All edge weights in this section are assumed to be of polynomial size. We further assume that edge weights are normalized so that the smallest edge weight in the graph is 1. Our construction is based on a hierarchical sparse double-tree construction similar to the sparse tree construction used in [8] but adapted to directed positive-weighted graphs and roundtrip distance. We need the following definitions:

**Algorithm PartialCover**( $\mathcal{R}, k$ ) from [8]:

```

1.  $\mathcal{U} \leftarrow \mathcal{R}; \mathcal{DT} \leftarrow \emptyset; \mathcal{DR} \leftarrow \emptyset$ 
2. while ( $\mathcal{U} \neq \emptyset$ ):
3.   Select an arbitrary cluster  $S_0 \in \mathcal{U}$ 
4.    $\mathcal{Z} \leftarrow \{S_0\}; Z \leftarrow S_0$ 
5.   repeat:
6.      $\mathcal{Y} \leftarrow \mathcal{Z}; Y \leftarrow Z$ 
7.      $\mathcal{Z} \leftarrow \{S \mid S \in \mathcal{U}, S \cap Y \neq \emptyset\}$ 
8.      $Z \leftarrow \cup_{S \in \mathcal{Z}} S$ 
9.   until ( $|\mathcal{Z}| \leq |\mathcal{R}|^{1/k} |\mathcal{Y}|$ )
10.   $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{Z}$ 
11.   $\mathcal{DT} \leftarrow \mathcal{DT} \cup \{Y\}$ 
12.   $\mathcal{DR} \leftarrow \mathcal{DR} \cup \mathcal{Y}$ 
13. output ( $\mathcal{DR}, \mathcal{DT}$ )

```

Fig. 7. The partial cover algorithm used as a subroutine by the main cover construction algorithm.

Given a positive-weighted strongly-connected directed graph  $G = (V, E)$  with  $|V| = n$ , and a distance metric  $DM$  between pairs of nodes, let  $Diam_{DM}(G)$  be the maximum  $DM$ -distance between any pair of nodes in  $G$ , i.e.,  $Diam_{DM}(G) = \max\{DM(u, v) \mid u, v \in G\}$ .  $Rad_{DM}(v, G)$  is the maximum  $DM$ -distance between  $v$  and any node in  $G$ ,  $Rad_{DM}(G)$  is  $\min\{RTRad(v, G) \mid v \in V\}$ , and  $Center_{DM}(G)$  is any vertex  $v \in V$  such that  $Rad_{DM}(v, G) = Rad_{DM}(G)$ . Our terminology is borrowed from Thorup and Zwick [38] and Roditty et al. [35], though it is important to note that our underlying RT covers will be constructed to satisfy a stronger property (i.e. requirement 1 of Theorem 10) than the RT covers of Roditty et al. [35]. Using the weaker covers of [35] unchanged would result only in a constant blowup in the stretch guarantee.

A cluster  $C$  is a subset of the nodes in the graph which induces a strongly connected subgraph, and a cover is a collection of clusters  $\mathcal{C} = \{C_i\}_i$  covering all the vertices of  $G$ , that is, such that  $\bigcup_i C_i = V$ . We extend our definition of  $Diam_{DM}()$ ,  $Rad_{DM}()$ , and  $Center_{DM}()$  to clusters  $C$  by considering the subgraph induced by the vertices in  $C$ . Finally, these definitions are extended to covers  $\mathcal{C}$  by taking the maximum over the values of every cluster in the cover, e.g.,  $Rad_{DM}(\mathcal{C}) = \max\{Rad_{DM}(C) \mid C \in \mathcal{C}\}$ .

When the distance metric considered is the roundtrip distance  $r(\cdot, \cdot)$  as defined in Section 1.1 we will refer to these quantities as  $RTDiam()$ ,  $RTRad()$ , and  $RTCenter()$ . Notice that by construction, we have  $RTHeight(DoubleTree(\mathcal{C})) = RTRad(\mathcal{C})$ . We define a double-tree cover as a collection of double-trees that cover the whole set of vertices of  $G$ .

We generalize the sparse cover construction in [8] for undirected graphs and one-way distance metric to general directed graphs and any distance metric as follows:

**Theorem 10.** *Let  $DM$  be any distance metric over vertices of some positive-weighted strongly connected directed graph  $G = (V, E)$  with  $|V| = n$ . Given an integer  $k > 1$  and a distance  $d$  such that  $1 \leq d \leq Diam_{DM}(G)$ , it is possible to construct a cover  $\mathcal{T}$  satisfying the following:*

- (1) For every node  $v \in V$  there is a cluster  $T \in \mathcal{T}$  containing all the vertices in  $\hat{N}_{DM}^d(v)$ , where  $\hat{N}_{DM}^d(v) = \{w \in V \mid DM(v, w) \leq d\}$ .
- (2)  $Rad_{DM}(\mathcal{T}) \leq (2k - 1)d$ .
- (3) For any  $v \in V$ ,  $v$  appears in at most  $2kn^{1/k}$  clusters, that is,  $|\{T \mid T \in \mathcal{T} \text{ and } v \in T\}| \leq 2kn^{1/k}$ .

**Proof.** To show the existence of the claimed cover, we use the algorithm in [8] that constructs it. The proof provided here deals with a general distance metric for directed graphs, so even though the notation looks identical to [8], in fact, the underlying graph-theoretic objects we are referring to are different. Thus, for example a new proof is required for property 4 of Lemma 11.

Given a cover  $\mathcal{R}$ , the procedure in Fig. 7 constructs a *partial cover*  $\mathcal{DT}$  (in which some of the clusters  $C \in \mathcal{R}$  are contained in a set  $Y \in \mathcal{DT}$ ) with no overlap among the elements  $Y \in \mathcal{DT}$ . The clusters in  $\mathcal{DT}$  have an increased cluster radius:

**Lemma 11.** Given a positive-weighted strongly connected directed graph  $G = (V, E)$  with  $|V| = n$ , a collection of clusters  $\mathcal{R}$  and an integer  $k$ , the collections  $\mathcal{DT}$  and  $\mathcal{DR}$  output by procedure  $\text{PartialCover}(\mathcal{R}, k)$  satisfy the following properties:

- (1) For every cluster  $C \in \mathcal{DR}$  there is a cluster  $CT \in \mathcal{DT}$  such that  $C \subseteq CT$ .
- (2) For every  $Y, Y' \in \mathcal{DT}$  we have  $Y \cap Y' = \emptyset$ .
- (3)  $|\mathcal{DR}| \geq |\mathcal{R}|^{1-1/k}$ , and
- (4)  $\text{Rad}_{DM}(\mathcal{DT}) \leq (2k - 1)\text{Rad}_{DM}(\mathcal{R})$ .

**Proof.** Property 1 follows directly from lines 11 and 12 in **PartialCover** after noticing that by construction (lines 6, 8):

$$Y = \bigcup_{S \in \mathcal{Y}} S.$$

For Property 2, it is sufficient to notice that when a new cluster  $Y$  is added to  $\mathcal{DT}$  (line 11), the set  $\mathcal{Z}$  of clusters that intersect with  $Y$  are removed from  $\mathcal{U}$  (line 10), and are therefore not eligible to participate in future clusters.

To establish the rest of the properties, denote the initial sets (line 4)  $\mathcal{Z}$  and  $Z$  by  $\mathcal{Z}_0$  and  $Z_0$ , respectively. Denote the set  $\mathcal{Z}$  (respectively,  $Z$  and  $Y$ ) constructed on the  $i$ th internal iteration by  $\mathcal{Z}_i$  (respectively,  $Z_i$  and  $Y_i$ ). Notice that for  $i \geq 1$  it holds that  $Y_i = Z_{i-1}$ .

For Property 3, we notice that from the termination condition of the loop (in line 9), the resulting  $\mathcal{Z}$  is such that  $|\mathcal{Z}| \leq |\mathcal{R}|^{1/k} |\mathcal{Y}|$ . Since  $|\mathcal{R}| = \sum_{\mathcal{Z}} |\mathcal{Z}| \leq \sum_{\mathcal{Y}} |\mathcal{R}|^{1/k} |\mathcal{Y}| = |\mathcal{R}|^{1/k} \sum_{\mathcal{Y}} |\mathcal{Y}| = |\mathcal{R}|^{1/k} |\mathcal{DR}|$ , Property 3 follows.

Finally, to establish Property 4, let  $J$  be the number of times that the internal loop of lines 5–9 is executed in some iteration. First we notice that for  $1 \leq i \leq J$  it holds that  $\text{Rad}_{DM}(Y_i) \leq (2i - 1)\text{Rad}_{DM}(\mathcal{R})$ . The proof is by induction. The base case ( $i = 1$ ) holds immediately since  $Y_1$  is simply one of the clusters in  $\mathcal{R}$  (lines 4, 6) and hence  $\text{Rad}_{DM}(Y_1) \leq (2 - 1)\text{Rad}_{DM}(\mathcal{R})$ . For the induction step, notice that by construction  $\text{Rad}_{DM}(Y_i) = \text{Rad}_{DM}(Z_{i-1})$ . Since  $\mathcal{Z}_{i-1}$  is created from  $Y_{i-1}$  by adding into it all clusters in  $\mathcal{R}$  intersecting it and  $Y_i = Z_{i-1}$  is simply a merge of all the clusters in  $\mathcal{Z}_{i-1}$ , we now see that the largest roundtrip distance between a node in  $Z_{i-1}$  and the center  $c$  of  $Y_{i-1}$  is as follows: let  $w \in Z_{i-1}$  be any vertex that achieves the largest roundtrip distance so that  $\text{Rad}_{DM}(c, Z_{i-1}) = r(c, w)$ , and let  $c'$  be the center of a cluster  $C \in \mathcal{R}$  which intersects with  $Y_{i-1}$  and contains the vertex  $w$ . Let  $w'$  be a vertex that is in  $Y_{i-1} \cap C$ . By the triangle inequality,

$$DM(c, w) \leq \underbrace{DM(c, w')}_{\text{within } Y_{i-1}} + \underbrace{DM(w', c')}_{\text{within } C} + \underbrace{DM(c', w)}_{\text{within } C} \leq \text{Rad}_{DM}(Y_{i-1}) + 2\text{Rad}_{DM}(\mathcal{R}).$$

Since  $\text{Rad}_{DM}(Y_i) = \text{Rad}_{DM}(Z_{i-1}) \leq \text{Rad}_{DM}(c, Z_{i-1})$  we conclude  $\text{Rad}_{DM}(Y_i) \leq \text{Rad}_{DM}(Y_{i-1}) + 2\text{Rad}_{DM}(\mathcal{R})$ . Applying the induction hypothesis, we conclude that

$$\text{Rad}_{DM}(Y_i) \leq (2(i - 1) - 1)\text{Rad}_{DM}(\mathcal{R}) + 2\text{Rad}_{DM}(\mathcal{R}) = (2i - 1)\text{Rad}_{DM}(\mathcal{R}).$$

It is left to upper bound  $J$ , the number of times the internal loop (lines 5–9) can be executed in an iteration of the procedure. By the termination condition, we see that  $|\mathcal{Z}_i| > |\mathcal{R}|^{1/k} |\mathcal{Y}_i|$  for  $1 \leq i < J$ , hence,  $|\mathcal{Z}_i| > |\mathcal{R}|^{1/k} |\mathcal{Z}_{i-1}|$  for  $1 \leq i < J$ . Since  $|\mathcal{Z}_0| = 1$ , it follows that  $|\mathcal{Z}_i| > |\mathcal{R}|^{i/k}$  for  $1 \leq i < J$ . Therefore,  $J \leq k$  since it is not possible that  $|\mathcal{Z}_k| > |\mathcal{R}|$  because  $\mathcal{Z}$  consists of clusters in  $\mathcal{R}$ .

Therefore, for any arbitrary cluster added to  $\mathcal{DT}$  in line 11 it holds that

$$\text{Rad}_{DM}(Y_J) \leq (2J - 1)\text{Rad}_{DM}(\mathcal{R}) \leq (2k - 1)\text{Rad}_{DM}(\mathcal{R}).$$

Hence,  $\text{Rad}_{DM}(\mathcal{DT}) \leq (2k - 1)\text{Rad}_{DM}(\mathcal{R})$  as required.  $\square$

We now describe the cover construction algorithm in Fig. 8:

We proceed now to prove that  $\mathcal{T}$  as computed by the algorithm **Cover** satisfies the properties stated in Theorem 13:

Property 1. Follows directly from Property 1 in Lemma 11 and observing that the final cover  $\mathcal{T}$  is the union of all the partial covers  $\mathcal{DT}$ , that the initial cover consists precisely of  $\mathcal{R} = \{\hat{N}_{DM}^d(v) \mid v \in V\}$ , and finally that the algorithm finishes when all the clusters in  $\mathcal{R}$  are exhausted (and hence covered by some cluster in  $\mathcal{DT}$ ).

Property 2 follows from Property 4 of Lemma 11 and the fact that  $\text{Rad}_{DM}(\mathcal{R}) = d$ .

**Algorithm Cover**( $G = (V, E), k, d$ ) from [8]:  
 1.  $\mathcal{R} \leftarrow \{\hat{N}_{DM}^d(v) \mid v \in V\}; T \leftarrow \emptyset$   
 2. **while** ( $\mathcal{R} \neq \emptyset$ ):  
 3.      $(\mathcal{DR}, \mathcal{DT}) \leftarrow \text{PartialCover}(\mathcal{R}, k)$   
 4.      $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{DR}$   
 5.      $T \leftarrow T \cup \mathcal{DT}$   
 6. **Output** ( $T$ )

Fig. 8. The main cover construction algorithm.

Finally, Property 3 is established as follows. Let  $\mathcal{R}^i$  denote the contents of the set  $\mathcal{R}$  at the beginning of  $i$ th iteration of the main loop (lines 2–5), let  $r_i = |\mathcal{R}^i|$ . Let  $\mathcal{DT}^i$  be the collection  $\mathcal{DT}$  added to  $T$  at the end of iteration  $i$ , and let  $\mathcal{DR}^i$  be the collection  $\mathcal{DR}$  removed from  $\mathcal{R}$  at the end of iteration  $i$ .

From Property 2 of Lemma 11, we know that each vertex appears in at most one cluster in each cover  $\mathcal{DT}^i$ . Therefore, to bound the number of cluster a vertex can appear in, it is sufficient to bound the number of iterations performed by the algorithm. From Property 3 of Lemma 11, after every iteration  $i$ , at least  $|\mathcal{DR}^i| \geq |\mathcal{R}^i|^{1-1/k}$  clusters of  $\mathcal{R}^i$  are removed from the set  $\mathcal{R}^i$ , that is,  $r_{i+1} \leq r_i - r_i^{1-1/k}$ .

From [8], we have that

**Lemma 12.** Consider the recurrence relation  $x_{i+1} = x_i - x_i^\alpha$  for  $0 < \alpha < 1$ . Let  $f(n)$  denote the least index  $i$  such that  $x_i \leq 1$  given  $x_0 = n$ . Then  $f(n) < \frac{n^{1-\alpha}}{(1-\alpha)\ln 2}$ .

We conclude that since  $r_0 = |\mathcal{R}| = n$  and  $\alpha = 1 - 1/k$ ,  $\mathcal{R}$  is exhausted after no more than  $\frac{n^{1/k}}{(1/k)\ln 2} = \frac{kn^{1/k}}{\ln 2} < 2kn^{1/k}$  iterations. This completes the proof of Theorem 10.  $\square$

We can apply the cover construction from Theorem 10 above to positive-weighted directed graphs using our new roundtrip distance metric  $r(\cdot, \cdot)$ . Moreover, we build a double-tree cover by building a double-tree on top of every cluster generated by the cover construction.

Given a positive-weighted strongly-connected directed graph  $G = (V, E)$  with  $|V| = n$ , we define  $\hat{N}^m(v)$  as the set of nodes in  $V$  that are within roundtrip distance  $m$  from  $v \in V$ .

We obtain:

**Theorem 13.** Given an integer  $k > 1$ , a positive-weighted strongly connected directed graph  $G = (V, E)$  with  $|V| = n$  and a roundtrip distance  $r$  such that  $1 \leq r \leq \text{RTDiam}(G)$ , it is possible to construct a double-tree cover  $\mathcal{T}$  satisfying the following:

- (1) For every node  $v \in V$  there is a double-tree  $T \in \mathcal{T}$  spanning all the vertices in  $\hat{N}^r(v)$ .
- (2) For every double-tree  $T \in \mathcal{T}$ :  $\text{RTHeight}(T) \leq (2k - 1)r$ .
- (3) For any  $v \in V$ ,  $v$  appears in at most  $2kn^{1/k}$  double-trees, that is,  $|\{T \mid T \in \mathcal{T} \text{ and } v \in T\}| \leq 2kn^{1/k}$ .

The following is a sketch of our construction. We use a similar hierarchy of covers as in [8], adapted to directed positive-weighted graphs and double-tree covers. For every  $i = 1, \dots, \lceil \log(\text{RTDiam}(G)) \rceil$ , we apply Theorem 13 with  $r = 2^i$  and construct a cover  $\mathcal{T}_i$  such that (1) there exists a double-tree in the cover that includes  $\hat{N}^{2^i}(v)$  for every  $v \in V$ , (2) the roundtrip height of such a double-tree is at most  $(2k - 1)2^i$ , and (3) every vertex appears in no more than  $2kn^{1/k}$  double-trees. For each  $i = 1, \dots, \lceil \log(\text{RTDiam}(G)) \rceil$ , every node  $v$  in the network chooses a double-tree  $C_i$  that contains  $\hat{N}^{2^i}(v)$ . Following [8]’s terminology, we refer to that double-tree as  $v$ ’s home double-tree at level  $i$ . Notice that the existence of such a tree is guaranteed by property (1) above.

The idea is to route within shallow home double-trees first and increasingly search in higher double-trees until one is found that contains both source and destination. To route within a double-tree  $C$ , we will always go through the root of the tree. We use the following result for single-source compact routing in the topology-dependent model with optimal stretch, due to Thorup and Zwick [39], and also to Fraigniaud and Gavoille [18]. While it is stated for undirected graphs, it is straightforward to apply this particular result to the  $\text{OutTree}(\cdot)$  component of a double-tree.

**Lemma 14.** (See [18,39].) *There is a routing scheme for any tree  $T$  with root  $r$  such that given any node  $u$  in  $T$ , the scheme routes along the optimal path from  $r$  to  $u$  in the fixed port model. The storage costs are  $\tilde{O}(1)$  per node in the tree, and the address size is  $O(\log^2 n)$ .*

Given a double-tree  $T$ , let  $TreeTab(T, x)$  and  $TreeR(T, x)$  refer to the storage table and address, respectively of node  $x$  under a tree-routing scheme that satisfies the requirements of Lemma 14 on the tree component  $OutTree(\cdot)$ .

To route within a double-tree  $C$ , every node will keep a pointer  $e_{x,RTCenter(C)}$  following edges in the  $InTree(\cdot)$  component of  $C$ , and also the tables  $TreeTab(C, x)$ . Notice that to route from the center to any node once name-dependent labels are known can be done optimally using a routing scheme such as in Lemma 14. To route from any node to the root can be done optimally using the pointers  $e_{x,RTCenter(C)}$  placed at every node in  $C$ . Notice that the distance traveled in this manner when routing between two arbitrary nodes in  $C$  is at most twice the roundtrip height of the double-tree  $C$ .

#### 4.1. Storage

To simplify the presentation we assume that  $n = q^k$  for an integer  $q$ , and define the alphabet  $\Sigma = \{0, \dots, q - 1\}$ . Let  $\langle u \rangle$  be the length  $k$  string over  $\Sigma$  which is the base  $n^{1/k}$  representation of  $u$ . For each  $0 \leq i \leq k$ , we also define functions  $\sigma^i : \Sigma^k \rightarrow \Sigma^i$ , so that for strings  $x$  and  $y$  over the alphabet  $\Sigma$ ,  $\sigma^i(xy) = x$  if and only if  $|xy| = k$  and  $|x| = i$ .

For every level  $i = 1, \dots, \lceil \log(RTDiam(G)) \rceil$ ,  $u \in V$  stores the following:

- (1) An identifier for  $u$ 's home double-tree at level  $i$ .
- (2) For every double-tree  $C_i$  in the  $i$ th level cover that vertex  $u$  is in,  $u$  stores:
  - (a)  $TreeTab(C_i, u)$  and its own name-dependent label  $TreeR(C_i, u)$ .
  - (b) The first link  $e_{u,RTCenter(C_i)}$  towards the center of  $C_i$ .
  - (c) For every  $\tau \in \Sigma$  (notice there are  $n^{1/k}$  choices) and for every  $j = 0, \dots, k - 1$  ( $k$  choices), the label  $TreeR(C_i, v)$ , where  $v \in C_i$  is the nearest node such that  $\sigma^j(\langle u \rangle) = \sigma^j(\langle v \rangle)$  and the  $(j + 1)$  element of  $v$  is  $\tau$ , if such node  $v$  exists.

Notice that (a) and (b) are used to route within the double-tree  $C_i$  when topology-dependent information is known and (c) implements the distributed dictionary to find topology-dependent labels.

The total storage requirement is

$$\lceil \log(RTDiam(G)) \rceil \times (\text{poly-log}(n) + 2kn^{1/k}) \times (\text{poly-log}(n) + \log(n) + kn^{1/k}),$$

where  $\lceil \log(RTDiam(G)) \rceil$  accounts for all the levels in the hierarchy,  $2kn^{1/k}$  accounts for the maximum number of double-trees a vertex appears in, and  $(\log(n) + \text{poly-log}(n) + kn^{1/k})$  is the combined storage requirement of every node within a single double-tree. The term  $kn^{1/k}$  accounts for the tree routing addresses of prefix-matching closest nodes. Notice also that  $\text{poly-log}(n)$  bits are sufficient to identify a double-tree in a given level since there are at most  $2kn^{1+1/k}$  such double-trees. The total storage at a node is therefore  $\tilde{O}(k^2 n^{2/k} \log(RTDiam(G)))$ , which is  $\tilde{O}(n^{2/k})$  for constant  $k$  and polynomial-sized edge weights.

#### 4.2. Routing algorithm

To route from  $s$  to  $t$  and back, we attempt to find node  $t$  in the home double-tree of  $s$ ,  $C_i$ , for increasing values of  $i = 1, \dots, \lceil \log(RTDiam(G)) \rceil$ .

To route to  $t$  in a double-tree  $C_i$  we go through a series of nodes  $s = v_0, v_1, \dots, v_h = t$  in  $C_i$ . The message always carries the tree routing label of the origin  $s$  and an identifier for  $s$ 's home double-tree  $C_i$ . From any intermediate node, say  $v_j$ , in this series ( $s$  is the first such node), it is routed to a node  $v_{j+1}$  in  $C_i$  (among the nodes  $v_j$  has routing labels for) which matches the largest possible prefix of the name of destination  $t$ , and which has a longer matching prefix than the currently matched prefix at  $v_j$ . If one of these nodes does not exist in  $C_i$ , then the message is returned

```

Algorithm PolynomialStretch( $s, t$ ):
 $i \leftarrow 1$ 
 $Found \leftarrow false$ 
while (not  $Found$  and  $i \leq \lceil \log(RTDiam(G)) \rceil$ ):
    // try for roundtrip through  $t$  in  $C_i$  as follows
     $h \leftarrow 1$  // invariant: current node  $c$  is  $s$ 
     $SourceLabel \leftarrow TreeR(C_i, s)$ 
    repeat:
         $v \leftarrow NextNode(s, c, t, C_i)$ 
         $h \leftarrow$  largest value such that  $\sigma^h((v)) = \sigma^h((t))$ 
        if ( $v \neq s$ ):
             $NextWaypointLabel \leftarrow TreeR(C_i, v)$ 
        else:
             $NextWaypointLabel \leftarrow SourceLabel$ 
        if ( $v = t$ ):
             $Found \leftarrow true$ 
            Route to  $v$  within  $C_i$ 
    until ( $v = s$ )
     $i \leftarrow i + 1$ 

```

Fig. 9. Polynomial stretch algorithm.

to  $s$  (this is when failure is detected and the search continues in the next level).<sup>6</sup> Otherwise, the message reaches the destination after at most  $k$  such trips. A final trip will be needed to go back to the origin. Notice that intermediate nodes  $v_j$  might appear in different double-trees, and we retrieve the information corresponding to the appropriate double-tree (in this case  $C_i$ ). We can do this because an identifier of  $C_i$  is included in the message header.

Figure 9 on page 791 contains a pseudocode summary of the algorithm, in which  $c$  refers to the current node—the value of  $c$  is implicitly changed by a command to “route.” Nodes  $s$  and  $t$  are the source and destination, respectively. For any nodes  $s, c, t \in V$  and for each  $i \leq \lceil \log(RTDiam(G)) \rceil$  and  $h \leq k + 1$  we define  $NextNode(s, c, t, C_i)$  as follows:

- If  $c = t$  then the return value is  $s$ .
- If  $c \neq t$  then we choose the largest possible  $h$  such that there is a node in  $C_i$  that satisfies  $\sigma^h((c')) = \sigma^h((t))$  and  $\sigma^h((c)) \neq \sigma^h((t))$ . Among the nodes with the largest possible  $h$ , we choose the closest node  $c'$  to  $c$  (by roundtrip distance  $Init_v$ ).
- If  $c \neq t$  and such a node (as in previous bullet) does not exist within  $C_i$ , then  $NextNode(s, c, t, C_i)$  is simply the node  $s$ , to enable a return to the starting point.

#### 4.3. Stretch analysis

Let the roundtrip distance between  $s$  and  $t$  be  $r$ . There exists a level  $i \leq \log(2r)$  such that  $s$ 's home double-tree  $C_i$  contains  $t$ . As shown in Fig. 10 on page 792, when routing within the double-tree  $C_i$  there are at most  $k + 1$  roundtrips from those nodes to the center of  $C_i$ . Hence, the total distance  $r(s, t)$  traveled within  $C_i$  is:

$$\begin{aligned}
 r(s, t) &\leq RTHeight(C_i) \times (k + 1) \\
 &\leq (2k - 1)2^i \times (k + 1) \quad (\text{by Theorem 13}) \\
 &\leq (2k - 1)2r(k + 1) \quad (i \leq \log(2r)) \\
 &\leq 4k^2r + 2kr - 2r.
 \end{aligned}$$

The total distance traveled in the whole process is at most twice the distance in the last level visited, hence the total distance is  $8k^2r + 4kr - 4r$ . The stretch is therefore  $8k^2 + 4k - 4$ .

<sup>6</sup> It is important to note that the source node  $s$  will not be visited again in this lookup, since the lookup always tries to match prefixes larger than those matched so far. Therefore, when the message returns to  $s$  it is the case that either it visited the destination and came back (success) or the destination is not in the current home double-tree (failure, continue in higher levels).

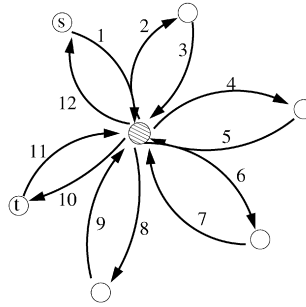


Fig. 10. Example of a route taken by algorithm PolynomialStretch within a cluster. A packet is routed from  $s$  to a first intermediate node  $v_0$ , then to a second intermediate node  $v_1$ , and so on until destination node  $t$  is reached, always routing through the center of the cluster (shaded). After this, the packet is returned to the source  $s$  using one extra trip through the center. Labels of arrows in this figure represent the order in which routes are taken. Notice that there are  $k + 1$  roundtrips between the center and other nodes appearing in this series.

#### 4.4. Remarks on the underlying roundtrip spanner and related work

We are aware of an alternative double-tree sparse cover construction that was used in [35] in a similar way as we use the double-tree sparse cover here. It is indeed possible to use this alternative construction, but it would yield a worse stretch of  $8k^2 + 8k$  while keeping similar storage bounds.

The alternative sparse cover looks more attractive at first since the blow-up in radius that it imposes is of only  $k$  as opposed to the blow-up of  $2k - 1$  in the construction used here. However, the double-trees built by the construction in [35] lack an important property, namely, that given a vertex  $v$  there is a double-tree containing the whole neighborhood of  $v$ . The construction in [35] only guarantees that there is a double tree containing any vertex in the neighborhood of  $v$  and  $v$  itself, but it can be a different tree for different vertices in  $v$ 's neighborhood. This is a problem since every vertex has to choose a single double tree as home double-tree, so that in [35]'s construction it can be the case that the home double-tree of  $v$  does not contain vertices in its close neighborhood. To fix this, [35] have to incur in a blow-up of the radius of 2, so that the total blow-up in distance is  $2k$  which is worse than  $2k - 1$ . We remark that by using the sparse cover presented here, the name-dependent scheme in [35] can be improved to have stretch  $4k - 2 + \epsilon$ .

We also note that their scheme is able to identify the level in which routing will succeed by inspecting the name-dependent labels. This is not possible here since we do not know name-dependent labels at the start, and it is part of the routing scheme to figure these out.

#### 4.5. Distributed implementation notes

This subsection discusses the implementation details of the *PolynomialStretch* algorithm presented in Fig. 9. Distributed pseudocode for the algorithm is presented in Fig. 11 on page 793.

The variable and function names in the pseudocode are similar to that presented in Section 3.6 on Fig. 6, the major differences being that in the *PolynomialStretch* algorithm, the mode is *Enroute* while a packet is moving through the network, because the algorithm is simpler when it does not distinguish between directions using a mode. In this case we use the function *NextNode()* defined in Section 4. Depending on the choice of a name-dependent routing subsystem from either Fraigniaud and Gavoille or Thorup and Zwick [18,39], we define the *GetTreeR()* function which assigns a name-dependent label to a given node id as well as the function *FGorTZNextEdge()* for choosing which edge to use next.

## 5. Lower bound

**Theorem 15.** *There exists an  $n$ -node strongly connected directed network on which every TINN roundtrip routing scheme of stretch  $< 2$  requires  $\Omega(n)$  bits of routing information at some node.*

**Proof.** Let  $N$  be an undirected  $n$ -node network for which every TINN routing algorithm of stretch  $< 3$  requires  $\Omega(n)$  space. Such an  $N$  exists by the result of [20]. Let  $N'$  be the directed network constructed by replacing each undirected

**Algorithm PolynomialStretch():****upon receipt of packet  $P$  at node  $s$ :**

```

ReadPacketHeader()

if (Mode = NewPacket):
    Level  $\leftarrow$  1
    Found  $\leftarrow$  false
    Hop  $\leftarrow$  1
    Mode  $\leftarrow$  Enroute
    SourceID  $\leftarrow$  MyNodeID
    SourceLabel  $\leftarrow$  GetTreeR(1, MyNodeID)
    NextWaypointID  $\leftarrow$  NextNode(SourceID, MyNodeID, DestinationID, Level, Hop)
    NextWaypointLabel  $\leftarrow$  GetTreeR(1, NextWaypointID)

else if (Mode = ReturnPacket):
    Mode  $\leftarrow$  Enroute
    NextWaypointID  $\leftarrow$  SourceID
    NextWaypointLabel  $\leftarrow$  SourceLabel

else if (Mode = Enroute and MyNodeID = DestinationID):
    Found  $\leftarrow$  true
    Deliver the packet to the host (destination) node

else if (Mode = Enroute and MyNodeID = SourceID):
    if (not Found): // Level <  $\lceil \log(\text{RTDiam}(G)) \rceil$ 
        Level  $\leftarrow$  Level * 2
        Hop  $\leftarrow$  1
        SourceLabel  $\leftarrow$  GetTreeR(Level, MyNodeID)
        NextWaypointID  $\leftarrow$  NextNode(SourceID, MyNodeID, DestinationID, Level, Hop)
        NextWaypointLabel  $\leftarrow$  GetTreeR(Level, NextWaypointID)
    else:
        Deliver the packet to the host (original source) node

else if (Mode = Enroute and MyNodeID = NextWaypointID):
    Hop  $\leftarrow$  Hop + 1
    NextWaypointID  $\leftarrow$  NextNode(SourceID, MyNodeID, DestinationID, Level, Hop)
    NextWaypointLabel  $\leftarrow$  GetTreeR(Level, NextWaypointID)

WritePacketHeader()
NextEdge  $\leftarrow$  FGorTZNextEdge(NextWaypointLabel)
Forward the packet on NextEdge

```

Fig. 11. Distributed pseudocode for polynomial stretch algorithm.

edge in  $N$  by two oppositely-directed edges. Let  $R$  be a roundtrip routing scheme for  $N'$  whose local tables are all of size  $o(n)$ , and let  $p_R(u, v)$  denote the (one-way) path a packet will take from  $u$  to  $v$  based on routing scheme  $R$ . Then there exists some  $u$  and  $v$  such that  $p_R(u, v) \geq 3d(u, v)$ ; else since  $R$  is also a routing scheme for  $N$ , this contradicts the lower bound of [20]. Thus  $p_R(u, v) + p_R(v, u) \geq 3d(u, v) + d(v, u) \geq 2d(u, v) + 2d(v, u) \geq 2r(u, v)$ , where the middle inequality follows because by construction of  $N'$ ,  $d(u, v) = d(v, u)$  for all nodes  $u, v$ .  $\square$

**6. Conclusions and open problems**

This paper presents distributed compact roundtrip routing algorithms, based on sublinear space local routing tables. One set of open questions involves how these tables could be most efficiently be set up, and whether this could be done efficiently in a distributed fashion. We note that in a static network, a centralized algorithm could be used to compute

these routing tables in polynomial time: in fact, it is straightforward to show how to do so in time proportional to the time it takes to compute all-pairs shortest paths on a digraph (see [44]). An open problem is how to efficiently maintain these tables in a *dynamic* network, where nodes enter, leave, or there is changing network topology. Notice that the strength of the TINN model is that the node names are decoupled from network topology; thus a distributed algorithm to efficiently update the tables could ultimately lead to a *self-stabilizing* algorithm for routing in the TINN model: packets could wander the network until information about topological updates reach their local neighborhood.

Another open question is as follows. What is the minimum stretch possible in a universal compact (again, meaning sublinear space at every node) roundtrip routing scheme in the TINN model? This paper provides an upper bound of 6, and a lower bound of 2 on the answer to this question; we conjecture that both the upper and lower bounds can be tightened. In the undirected TINN case, there is a tight upper and lower bound of 3 (see [2] and [20], respectively).

Finally, peer to peer networks has been a topic of increasing interest [21,26,31,33,36,37,42,43]. It has been suggested to us [15,32] that some of the techniques developed here, could perhaps be applied to the design of better algorithms for routing and searching in peer-to-peer networks.

## Acknowledgments

We thank the anonymous referees for suggestions that greatly improved the presentation of this paper. Marta Arias was partly supported by NSF grant IIS-0099446. Lenore Cowen was supported in part by the National Science Foundation under Grant No. CCR-0208629. Kofi Laing was supported in part by NSF grant EHR-0227879.

## References

- [1] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Routing with improved communication-space trade-off, in: Proceedings of the Eighteenth International Symposium on Distributed Computing, DISC 2004, October 2004.
- [2] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, Mikkel Thorup, Compact name-independent routing with minimum stretch, in: Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2004, ACM Press, June 2004, pp. 20–24.
- [3] Marta Arias, Lenore Cowen, Kofi Laing, Rajmohan Rajaraman, Orjeta Taka, Compact routing with name independence, *SIAM J. Discrete Math.* 20 (3) (2006) 705–726.
- [4] Marta Arias, Lenore Cowen, Kofi Laing, Rajmohan Rajaraman, Orjeta Taka, Compact routing with name independence, in: Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures, June 2003, pp. 184–192.
- [5] Marta Arias, Lenore J. Cowen, Kofi A. Laing, Compact roundtrip routing with topology-independent node names, in: Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing, ACM Press, July 2003, pp. 43–52.
- [6] Baruch Awerbuch, Amotz Bar-Noy, Nati Linial, David Peleg, Compact distributed data structures for adaptive network routing, in: Proc. 21st ACM Symp. on Theory of Computing, May 1989, pp. 479–489.
- [7] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, David Peleg, Near-linear time construction of sparse neighborhood covers, *SIAM J. Comput.* 28 (1) (1999) 263–277.
- [8] Baruch Awerbuch, David Peleg, Sparse partitions, in: Proc. 31st IEEE Symp. on Found. of Comp. Science, 1990, pp. 503–513.
- [9] Baruch Awerbuch, David Peleg, Routing with polynomial communication-space trade-off, *SIAM J. Discrete Math.* 5 (2) (1992) 151–162.
- [10] Edith Cohen, Fast algorithms for constructing  $t$ -spanners and paths with stretch  $t$ , *SIAM J. Comput.* 28 (1) (1999) 210–236.
- [11] L. Cowen, C. Wagner, Compact roundtrip routing in directed networks, *J. Algorithms* 50 (1) (2004) 79–95.
- [12] L. Cowen, W. Wagner, Compact roundtrip routing in digraphs, in: Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms, 1999, pp. S885–S886.
- [13] L.J. Cowen, C.G. Wagner, Compact roundtrip routing in directed networks, in: Proc. 19th ACM Symp. on Principles of Distrib. Computing, 2000, pp. 51–59.
- [14] Lenore Cowen, Compact routing with minimum stretch, *J. Algorithms* 38 (2001) 170–183.
- [15] M. Crovella, Personal communication, 2002.
- [16] Dorit Dor, Shay Halperin, Uri Zwick, All pairs almost shortest paths, in: 37th Annual Symposium on Foundations of Computer Science, Burlington, Vermont, 14–16 October, IEEE, 1996, pp. 452–461.
- [17] Tamar Eilam, Cyril Gavoille, David Peleg, Compact routing schemes with low stretch factor, in: 17th Annual ACM Symposium on Principles of Distributed Computing, PODC, 1998, pp. 11–20.
- [18] Pierre Fraigniaud, Cyril Gavoille, Routing in trees, in: Fernando Orejas, Paul G. Spirakis, Jan van Leeuwen (Eds.), 28th International Colloquium on Automata, Languages and Programming, ICALP, in: Lecture Notes in Comput. Sci., vol. 2076, Springer, 2001, pp. 757–772.
- [19] Cyril Gavoille, Routing in distributed networks: Overview and open problems, *ACM SIGACT News Distributed Computing Column* 32 (1) (March 2001) 36–52.
- [20] Cyril Gavoille, Marc Gengler, Space-efficiency of routing schemes of stretch factor three, *J. Parallel Distrib. Comput.* 61 (2001) 679–687.
- [21] K. Hildrum, J. Kubiawicz, S. Rao, B. Zhao, Distributed object location in a dynamic network, in: Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures, 2002, pp. 41–52.

- [22] Cyril Gavoille, Ittai Abraham, Dahlia Malkhi, On space-stretch trade-offs: Lower bounds, in: Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architectures, July 2006, pp. 207–216.
- [23] Cyril Gavoille, Ittai Abraham, Dahlia Malkhi, On space-stretch trade-offs: Upper bounds, in: Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architectures, July 2006, pp. 217–224.
- [24] K. Iwama, A. Kawachi, Compact routing with stretch factor of less than three, in: 19th Annual ACM Symposium on Principles of Distributed Computing, PODC, 2000, p. 337.
- [25] Kazuo Iwama, Masaki Okita, Compact routing for average case networks, in: 21st Annual ACM Symposium on Principles of Distributed Computing, PODC, 2002.
- [26] J. Kubiawicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, H. Weatherspoon, R. Gummadi, S. Rhea, W. Weimer, C. Wells, B. Zhao, Oceanstore: An architecture for global-scale persistent storage, in: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, 2000.
- [27] Kofi A. Laing, Name-independent compact routing in trees, Technical Report 2003-02, Tufts University Department of Computer Science, September 2003.
- [28] Kofi A. Laing, Brief announcement: Name-independent compact routing in trees, in: Proc. 23rd ACM Symp. on Principles of Distrib. Computing, July 2004, p. 382.
- [29] T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann Publishers, San Mateo, CA, 1992.
- [30] David Peleg, Distance-dependent distributed directories, Inform. and Comput. 103 (2) (1993) 270–298.
- [31] C.G. Plaxton, R. Rajaraman, A.W. Richa, Accessing nearby copies of replicated objects in a distributed environment, Theory Comput. Syst. 32 (1999) 241–280.
- [32] R. Rajaraman, Personal communication, 2002.
- [33] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Schenker, A scalable content-addressable network, in: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM Press, 2001, pp. 161–172.
- [34] Liam Roditty, Mikkel Thorup, Uri Zwick, Roundtrip spanners and roundtrip routing in directed graphs (full version), submitted for publication.
- [35] Liam Roditty, Mikkel Thorup, Uri Zwick, Roundtrip spanners and roundtrip routing in directed graphs, in: Proc. 13th ACM–SIAM Symp. on Discrete Algorithms, January 2002, pp. 844–851.
- [36] Antony Rowstron, Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in: IFIP/ACM International Conference on Distributed Systems Platforms, Middleware, November 2001, pp. 329–350.
- [37] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, Hari Balakrishnan, Chord: A scalable Peer-To-Peer lookup service for Internet applications, in: Proceedings of the 2001 ACM SIGCOMM Conference, 2001, pp. 149–160.
- [38] Mikkel Thorup, Uri Zwick, Approximate distance oracles, in: Proc. 33rd ACM Symp. on Theory of Computing, May 2001, pp. 183–192.
- [39] Mikkel Thorup, Uri Zwick, Compact routing schemes, in: Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM Press, July 2001, pp. 1–10.
- [40] M.van Steen, F.J. Hauck, A.S. Tanenbaum, A model for worldwide tracking of distributed objects, in: Proceedings of the 1996 Conference on Telecommunications Information Networking Architecture, TINA 96, September 1996, pp. 203–212.
- [41] Christopher Wagner, Drawing with Bezier curves and routing on digraphs, PhD thesis, Dept. of Mathematical Sciences, Johns Hopkins University, 1999.
- [42] B.Y. Zhao, J.D. Kubiawicz, A.D. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing, Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.
- [43] Ben Y. Zhao, Yitao Duan, Ling Huang, Anthony D. Joseph, John D. Kubiawicz, Brocade: landmark routing on overlay networks, in: First International Workshop on Peer-to-Peer Systems, IPTPS/LNCS, March 2002, pp. 34–44.
- [44] Uri Zwick, Exact and approximate distances in graphs—A survey, in: Proc. of 9th European Symposium on Algorithms, 2001, pp. 33–48.