Compact Roundtrip Routing with Topology-Independent Node Names

Marta Arias * Lenore J. Cowen † Kofi A. Laing ‡
Department of Computer Science
Tufts University
Medford, MA 02155
{ marias, cowen, laing }@cs.tufts.edu

ABSTRACT

This paper presents compact roundtrip routing schemes with local tables of size $\tilde{O}(\sqrt{n})$ and stretch 6 for any directed network with arbitrary edge weights; and with local tables of size $\tilde{O}(\epsilon^{-1}n^{2/k})$ and stretch $min((2^{k/2}-1)(k+\epsilon), 16k^2 + 8k - 8)$, for any directed network with polynomially-sized edges, both in the topology-independent node-name model.[1] These are the first topology-independent results that apply to routing in directed networks.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network Communications, Store and Forward Networks*; G.2.2 [**Discrete Mathematics**]: Graph Theory— *Path and Circuit Problems, Network Problems, Trees, Graph labeling.*

Keywords

Compact Routing, Digraphs, Stretch, Distributed Lookup Tables.

1. INTRODUCTION

This paper concerns compact roundtrip routing for 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. [4, 6, 9, 10, 14, 15, 22, 25] and the surveys of [11, 30]) 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, when a packet destined for i arrives, "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. (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.) 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 with its (x, y) coordinates.

In [2], 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 there is changing network topology. There are serious consistency and continuity issues if the identifying label of a node changes as network connectivity 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 to 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. A paper of Awerbuch and Peleg in the following year [3], presented an alternate scheme with a polynomial space/stretch tradeoff. Our recent result joint with Rajaraman and Taka [1], presents compact-routing TINN schemes that achieve a reduction in the maximum length of the routes or *stretch* of these schemes. These schemes, like the ones we present in this paper, are *universal,* meaning they apply to any n-node (weighted) network. However, the previous results all apply only to 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 "compact" routing schemes when the network is directed. For example, it is shown in [8] that distinguishing between pairs of vertices at distances 2 and ∞ even in *unweighted* directed graphs is at least as hard as Boolean matrix multiplication. Roditty

*Supported in part by NSF grant IIS-0099446.
†Supported in part by NSF grant CCR-0208629.
‡Supported in part by NSF grant EHR-0227879.

[1]The $\tilde{O}(R)$ notation denotes $O(R \log^{O(1)} R)$.

PODC'03, July 13–16, 2003, Boston, Massachusetts, USA.

et al. [22] observe that sparse spanners don't exist for all digraphs, and there is further discussion in [6, 27]. In the 2000 PODC conference, Cowen and Wagner [6] 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 [6], sparse roundtrip spanners do exist in this model, and and can be used as a basis for compact roundtrip routing schemes in directed networks. The (name-dependent) roundtrip routing scheme of [6] was subsequently improved by Roditty et al [22]. Here we present the first universal compact roundtrip routing schemes for (weighted) directed networks in the TINN model.

1.1 Model and Definitions

The metric. Let $G = (V, E)$ be a strongly connected 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. If $p(u, v)$ is the length of a path p from u to v, then the *stretch* of p is defined as $p(u, v)/d(u, v)$. The *stretch* of a routing scheme that describes point to point routes between each ordered pair of vertices (u, v) is the maximum stretch over all pairs $u, v \in G$. 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 $r(i, j) = r(j, i)$.

The routing scheme. A *roundtrip routing scheme* is a scheme where a packet 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. 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 α 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 $\alpha \times r(s, t)$.

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 $\{0, \ldots, n-1\}$. 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 $\{0, \ldots, n-1\}$ 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 [1] 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 $\{0, \ldots, n-1\}$ 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.[2] For details see [1]; the generalization of this hashing scheme to roundtrip routing in directed graphs is entirely straightforward. So in what follows, we assume the labels are a permutation of $[n]$ and implicitly apply the reduction at the end to handle the more general case.

Edge names. Each node v is also assumed to have a unique name from a set of size $O(V)$ 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 [10]; in contrast, the *designer-port* model considered by [10] allows the network designer to specify port names dependent on global network topology.

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 forwarded. All TINN schemes therefore require writable packet headers. While packets initially arrive with $O(\log n)$ 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 [18]. 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. [22]. 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 $16k^2+8k-8$ (both schemes require that edge weights be bounded by a polynomial in n). The first gener-

[2] 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.

alized scheme follows easily from the definition of roundtrip routing and a result in [1], the second general scheme involves some substantial new ideas and produces the best space/stretch tradeoff for larger k. Putting the two together gives the result claimed in the abstract.

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 Gavoille and Gengler [12] implies that any compact routing scheme that uses $o(n)$-sized tables at every node in the TINN model, must have stretch ≥ 3. (In fact, the result of [12] 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 [5] and [22], we define the roundtrip distance metric as follows: let $G = (V, E)$ be an 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 $u \prec_v w$ (read: u is closer to v than w is, by the roundtrip metric) if and only if one of the following is true:

1. $r(v, u) < r(v, w)$
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: $v \prec_v u_1 \prec_v u_2 \prec_v \ldots \prec_v u_{n-1}$. We call this sequence $Init_v$. Additionally, we define $u \preceq_v w$ to mean $u \prec_v w$ or $u = w$.

Given a 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 divide the address space $\{0, \ldots, n-1\}$ into $\sqrt{n}$-sized blocks B_i, for $i = 0, \ldots, \sqrt{n}-1$, such that block B_i consists of the node labels $i\sqrt{n}$ to $(i+1)\sqrt{n}-1$. (Assume for simplicity that n is a perfect square). 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 figure 1.

LEMMA 2.1. *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 $\{B_i | 0 \leq i < \sqrt{n}\}$ 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 [1], the only difference being $N(v)$ is now defined in terms of $Init_v$ and roundtrip distance. The proof is identical. □

We also make use of the following result of Roditty et al. [22] in the topology-dependent model:

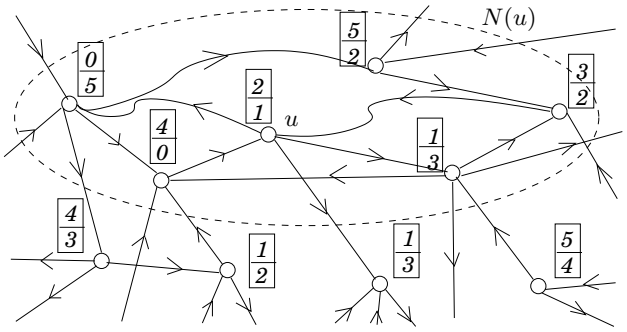

Figure 1: A Block distribution example showing a section of a 36 node directed graph, with a node u and its neighborhood $N(u)$. The example is unweighted, but the lemma works for weighted graphs as well. In this case the possible block labels are $\{0, \ldots, 5\}$. The sets S_v are indicated as vertical 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.

LEMMA 2.2. *[22] There exists a name-dependent compact roundtrip routing algorithm for directed graphs with stretch 3, which uses $\tilde{O}(n^{1/2})$ space. The path taken by a packet routing 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.

2.1 Storage Requirements

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 2.1).
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 $Tab_3(u)$.

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.

```
upon receipt of packet P at node s:

    // initialize local variables from the packet header
    (SrcID, SrcLabel, NextID, NextLabel,
    DestID, DestLabel, DictID, Mode) ← ReadPacketHeader(P)

    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(SrcID, SrcLabel, NextID, NextLabel,
                      DestID, DestLabel, DictID, Mode, P)
    // forward packet
    NextEdge ← GetRTZNextEdge(NextLabel)
    Forward the packet P on NextEdge
    exit(Success)
```

Figure 2: The local routing algorithm at node s.

2.2 Algorithm and Stretch Analysis

The local routing algorithm is presented as pseudocode in Figure 2. 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 2.2, 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 2.2. The return trip to s is accomplished using $R_3(s)$, which is contained in the packet header.

LEMMA 2.3. *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 2.2. Otherwise, the length of the route taken from s to w to t and back to s is given by $p(s,w)+p(w,t)+p(t,s)$, which by Lemma 2.2 is $\leq r(s,w)+d(s,w)+r(w,t)+d(w,t)+r(t,s)+d(t,s)$, and thus certainly $\leq r(s,w)+d(s,w)+r(w,s)+r(s,t)+d(w,t)+r(t,s)+d(t,s)$. But since $w \in N(s)$ but $t \notin N(s)$, it must be that $r(s,w) \leq r(s,t)$ so this is $\leq 4r(s,t)+d(s,w)+d(w,t)+d(t,s)$, which is $\leq 4r(s,t)+d(s,w)+d(w,s)+d(s,t)+d(t,s)$ by the triangle inequality. Using again the fact that $r(s,w) \leq r(s,t)$, we have that $4r(s,t)+d(s,w)+d(w,s)+d(s,t)+d(t,s) \leq 6r(s,t)$, whence the result. □

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 often will result in worse average case performance since it always routes through s when routing from w to t.

3. A GENERALIZED ROUTING SCHEME

3.1 Preliminaries

We use a randomized distribution of blocks of lookup information first introduced by Arias et al.[1]. The following description is nearly syntactically identical to the one in [1] 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 $\Sigma = \{0, \ldots, n^{1/k} - 1\}$. For each $0 \leq i \leq k$, Σ^i is the set of words over Σ 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 $\sigma^i : \Sigma^k \longrightarrow \Sigma^i$, such that $\sigma^i((a_0, \ldots, a_{k-1})) = (a_0, \ldots, a_{i-1})$. That is, σ^i extracts the prefix of length i from a string $\alpha \in \Sigma^k$.

For each $\alpha \in \Sigma^{k-1}$, define the sets $B_\alpha = \{u \in V | \sigma^{k-1}(\langle u \rangle) = \alpha\}$. 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 Σ^k obtained by appending a 0 to α. 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 [1].

LEMMA 3.1. *[1] 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 2.1, 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.[22]. For the definition of the double-trees referred to in the following lemma, see Section 4 of this paper.

LEMMA 3.2. *[22] Let G be a weighted 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 an arbitrary node v in at most $O(\frac{k^2}{\epsilon} n^{1/k} (\log n)^{1-1/k} \log(nW))$ double-trees.*

We will use a routing scheme that achieves stretch $2k + \epsilon$, using local routing tables of size $\tilde{O}(n^{1/k})$, and headers of size $o(log^2 n)$, which is derived from [22] under the assumption that weights are polynomially sized, which is made throughout this section. The stretch in this context is lower than the $4k + \epsilon$ they state, because we can use our lookup tables to store the best double-tree in their spanner for a particular pair of nodes.

3.3 Storage

Let $Tab(x)$ refer to the storage table at node x in the algorithm of Roditty et al.[22], 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 is stored in the algorithm of Roditty et al., and they deduce $R_2(.)$ from their node labels. Their labels are of size $o(\frac{k}{\epsilon} \log^2 n \log(nW))$ bits, are globally valid – they are used for routing to node v from *any* node u in the digraph, with stretch $4k + \epsilon$.

Let $\{S_u | u \in V\}$ be a collection of sets of blocks that satisfies Lemma 3.1. For each node u, let $S'_u = S_u \bigcup \{B_\beta\}$, 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:

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_α, 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 containing a block B_β such that $\sigma^i(B_\beta) = \sigma^i(B_\alpha)$ and the $(i + 1)$st element of $\sigma^{k-1}(B_\beta)$ is τ.
   (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 τ.

LEMMA 3.3. *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 $O(n^{O(1)})$. □

3.4 Routing Algorithm

Given a source node s and destination node t, our roundtrip routing algorithm visits a sequence of nodes $s = v_0, \ldots, v_k = t$ (not necessarily distinct) to reach t, and $v_{k-1}, \ldots, v_0$ to return to s. Just like in [1], the sequence $s = v_0, \ldots, v_k = t$ has the property that each v_i (except v_k) contains a block B_{β_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 3.4 is presented below. The idea of matching increasing prefixes of node names is well known in the parallel algorithms literature for multidimensional array routing (see [17]); it has also been used more recently in the context of peer to peer systems for locating replicated objects [26, 16, 19, 13], and also for compact routing in undirected graphs [1].

ALGORITHM 3.4.
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* $N^{i+1}(v_i) \bigcap \{v | \exists B_\beta \in S_v : \sigma^{i+1}(B_\beta) = \sigma^{i+1}(\langle t \rangle)\}$
  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 $R_2(v_i, v_{i+1})$

LEMMA 3.5. *Algorithm 3.4 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 $R_2(v_i, v_{i+1})$ which suffices to route to the node v_{i+1}. Delivery to node v_{i+1} is assured by the correctness of the algorithm of Roditty et al.[22]. 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. □

3.5 Stretch Analysis

LEMMA 3.6. *For $0 \leq i \leq h-1$, $r(v_i, v_{i+1}) \leq 2^i r(s,t)$.*

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_β such that $\sigma^1(B_\beta) = \sigma^i(t)$.

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:

$$\begin{aligned}
r(v_l, v_{l+1}) &\leq r(v_l, v_{l+1}^*) && (1)\\
&\leq r(v_l, s) + r(s, v_{l+1}^*) && (2)\\
&\leq r(s,t) + r(v_l, s) && (3)\\
&\leq r(s,t) + r(s, v_l) && (4)\\
&\leq r(s,t) + \textstyle\sum_{i=0}^{l-1} r(v_i, v_{i+1}) && (5)\\
&\leq r(s,t)\left[1 + \textstyle\sum_{i=0}^{l-1} 2^i\right] && (6)\\
&\leq 2^l r(s,t)
\end{aligned}$$

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. □

THEOREM 3.7. *Algorithm 3.4 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 3.3 and Lemma 3.5 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.

$$\begin{aligned}
\tilde{r}(s,t) &= \textstyle\sum_{i=0}^{k-1} \tilde{d}(v_i, v_{i+1}) + \sum_{i=0}^{k-1} \tilde{d}(v_{i+1}, v_i) && (1)\\
&\leq \textstyle\sum_{i=0}^{k-1} \tilde{r}(v_i, v_{i+1}) && (2)\\
&\leq \textstyle\sum_{i=0}^{k-1} (2k+\epsilon) r(v_i, v_{i+1}) && (3)\\
&\leq (2k+\epsilon) \textstyle\sum_{i=0}^{k-1} r(v_i, v_{i+1}) && (4)\\
&\leq (2k+\epsilon) \textstyle\sum_{i=0}^{k-1} 2^i r(s,t) && (5)\\
&\leq (2k+\epsilon)(2^k - 1) r(s,t) && (6)
\end{aligned}$$

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.[22]. Step (5) results from applying Lemma 3.6. □

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

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 3.7, the second scheme we present does not follow directly from any of the results in [1]. 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, 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. Our construction is based on a hierarchical sparse double-tree construction similar to the sparse tree construction used in [3] but adapted to directed weighted graphs and roundtrip distance. We need the following definitions:

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

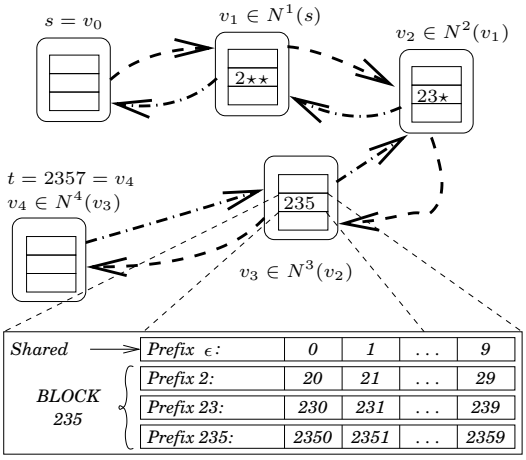

Figure 3: A schematic of how the prefix-matching algorithm of Theorem 3.7 works. The figure only includes the sequence of nodes where the distributed dictionary is read – the other nodes in the path are not shown. Each node contains 3 blocks in this example, and the contents of each block are illustrated 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.

roundtrip distance m from $v \in V$, $RTDiam(G)$ as the maximum roundtrip distance between any pair of nodes in G, $RTRad(v,G)$ as the maximum roundtrip distance between v and any node in G, $RTRad(G)$ as $min\{RTRad(v,G)|v \in V\}$, and $RTCenter(G)$ as any vertex $v \in V$ such that $RTRad(v,G) = RTRad(G)$.

A cluster C is a subset of the nodes in the graph, 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 $RTDiam()$, $RTRad()$, and $RTCenter()$ 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., $RTRad(\mathcal{C}) = max\{RTRad(C)|C \in \mathcal{C}\}$.

Let C be a strongly connected set of vertices, and $v = RTCenter(C)$ its center. We define $OutTree(C)$ as the shortest paths tree rooted at v that spans all the vertices in C. Let $InTree(C)$ be the (reversed) tree that consists of the shortest paths 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. Notice that by construction, we have $RTHeight(DoubleTree(C)) = RTRadius(C)$. We will call a double-tree cover a collection of double-trees that cover the whole set of vertices of G.

We generalize the sparse cover construction in [3] for undirected graphs and one-way distance metric to general directed graphs and any distance metric. Hence, it applies to directed graphs and our new roundtrip distance metric. Moreover, we construct a double-tree sparse cover by building a double-tree on top of every cluster generated by the sparse cover. We obtain:

THEOREM 4.1. *Given an integer $k > 1$, a weighted directed graph $G = (V, E)$ with $|V| = n$ and a roundtrip distance r s.t. $1 \leq r \leq RTDiam(G)$, 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 $T \in \mathcal{T}$: $RTHeight(T) \leq (2k-1)r$.*
3. *For any $v \in V$, v appears in at most $2kn^{1/k}$ double-trees, that is, $|\{T|T \in \mathcal{T} \text{ and } v \in T\}| \leq 2kn^{1/k}$.*

Proof. Omitted from this extended abstract. □

The following is a sketch of our construction. We use a similar hierarchy of covers as in [3], adapted to directed weighted graphs and double-tree covers. For every $i = 1, \ldots, \lceil \log(RTDiam(G)) \rceil$, we apply Theorem 4.1 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 $2kn^{\frac{1}{k}}$ double-trees. For each $i = 1, \ldots, \lceil \log(RTDiam(G)) \rceil$, every node v in the network chooses a double-tree C_i that contains $\hat{N}^{2^i}(v)$. Following [3]'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 [25], and also to Fraigniaud and Gavoille [10]. While it is stated for undirected graphs, it is straightforward to apply this particular result to the $OutTree(\cdot)$ component of a double-tree.

LEMMA 4.2. *[10, 25] 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 4.2 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 4.2. 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 $\Sigma = \{0, \ldots, q-1\}$.

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

For every level $i = 1, \ldots, \lceil \log(RTDiam(G)) \rceil$, $u \in V$ 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 $j = 0, \ldots, k-1$ (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 τ, 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 $\lceil \log(RTDiam(G)) \rceil \times (\text{poly-log}(n) + 2kn^{\frac{1}{k}} \times (\text{poly-log}(n) + \log(n) + kn^{\frac{1}{k}})$, where $\lceil \log(RTDiam(G)) \rceil$ accounts for all the levels in the hierarchy, $2kn^{\frac{1}{k}}$ accounts for the maximum number of double-trees a vertex appears in, and $(\log(n) + \text{poly-log}(n) + kn^{\frac{1}{k}})$ is the combined storage requirement of every node within a single double-tree. The term $kn^{\frac{1}{k}}$ accounts for the tree routing addresses of prefix-matching closest nodes. Notice also that poly-log(n) bits are sufficient to identify a double-tree in a given level since there are at most $2kn^{1+\frac{1}{k}}$ such double-trees. The total storage at a node is therefore $\tilde{O}(k^2 n^{\frac{2}{k}} \log(RTDiam(G)))$, which is $\tilde{O}(n^{\frac{2}{k}})$ 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 $i = 1, \ldots, \lceil \log(RTDiam(G)) \rceil$.

To route to t in a double-tree C_i we go through a series of nodes $s = v_0, v_1, \ldots, v_h = t$ 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 to s (this is when failure is detected and the search continues in the next level)[3]. 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.

In the following pseudocode summary of the algorithm, c refers to the current node, and its value 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 $\sigma^h(\langle c' \rangle) = \sigma^h(\langle t \rangle)$ and $\sigma^h(\langle c \rangle) \neq \sigma^h(\langle t \rangle)$. 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.

Algorithm PolyRoute(s,t)

```
i ← 1
Found ← false
while (not Found and i ≤ ⌈log(RTDiam(G))⌉):
    // try for roundtrip through t in C_i as follows
    h ← 1 // invariant: current node c is s
    SourceLabel ← TreeR(C_i, s)
    repeat:
        v ← NextNode(s, c, t, C_i)
        h ← largest value such that σ^h(⟨v⟩) = σ^h(⟨t⟩)
        if (v ≠ s):
            NextWaypointLabel ← TreeR(C_i, v)
        else:
            NextWaypointLabel ← SourceLabel
        if (v = t):
            Found ← true
        Route to v within C_i
    until (v = s)
    i ← i + 1
```

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. When routing within the double-tree C_i there are at most $k+1$ nodes visited, and the distance traveled between nodes is no more than twice the roundtrip height of C_i. The total distance $r(s,t)$ traveled within C_i is:

$$\begin{array}{rcll} r(s,t) & \leq & 2RTHeight(C_i) \times (k+1) & \\ & \leq & 2(2k-1)2^i \times (k+1) & \text{By Theorem 4.1} \\ & \leq & (4k-2)2r(k+1) & i \leq \log(2r) \\ & \leq & 8k^2 r + 4kr - 4r & \end{array}$$

The total distance traveled in the whole process is at most twice the distance in the last level visited, hence the total distance is $16k^2 r + 8kr - 8r$. The stretch is therefore $16k^2 + 8k - 8$.

[3]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).

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 [22] 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 $16k^2 + 16k$ 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 [22] lack an important property, namely, that given a vertex v there is a double-tree containing the whole neighborhood of v. The construction in [22] 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 [22]'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, [22] 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 [22] 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.

5. LOWER BOUND

THEOREM 5.1. *There exists an n-node 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 [12]. Let N' be the directed network constructed by replacing each undirected 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 s.t. $p_R(u, v) \geq 3d(u, v)$; else since R is also a routing scheme for N, this contradicts the lower bound of [12]. 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. □

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 [30]). 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, the known upper and lower bounds are 5 and 3 (see [1] and [12] respectively).

Finally, peer to peer networks has been a topic of increasing interest [13, 16, 19, 21, 23, 24, 28, 29]. It has been suggested to us [7, 20] 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.

7. REFERENCES

[1] M. Arias, L. Cowen, K. Laing, R. Rajaraman, and O. Taka. Compact routing with name independence. To appear in SPAA, 2003.

[2] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive network routing. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 479–489, May 1989.

[3] B. Awerbuch and D. Peleg. Sparse partitions. In *Proc. 31st IEEE Symp. on Found. of Comp. Science*, pages 503–513, 1990.

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

[5] L. Cowen and W. Wagner. Compact roundtrip routing in digraphs. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages S885–886, 1999.

[6] L. J. Cowen and C. G. Wagner. Compact roundtrip routing in directed networks. In *Proc. 19th ACM Symp. on Principles of Distrib. Computing*, pages 51–59, 2000.

[7] M. Crovella. Personal communication, 2002.

[8] D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. In *37th Annual Symposium on Foundations of Computer Science*, pages 452–461, Burlington, Vermont, 14–16 Oct. 1996. IEEE.

[9] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. In *17th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 11–20, 1998.

[10] P. Fraigniaud and C. Gavoille. Routing in trees. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, 28^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of Lecture Notes in Computer Science, pages 757–772. Springer, 2001.

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

[12] C. Gavoille and M. Gengler. Space-efficiency of routing schemes of stretch factor three. *Journal of Parallel and Distributed Computing*, 61:679–687, 2001.

[13] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao.

Distributed object location in a dynamic network. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 41–52, 2002.

[14] K. Iwama and A. Kawachi. Compact routing with stretch factor of less than three. In *19th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, page 337, 2000.

[15] K. Iwama and M. Okita. Compact routing for average case networks. In *21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[16] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, H. W. R. Gummadi, S. Rhea, W.Weimer, C. Wells, and 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.

[17] T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[18] D. Peleg. Distance-dependent distributed directories. *Information and Computation*, 103(2):270–298, 1993.

[19] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32:241–180, 1999.

[20] R. Rajaraman. Personal communication, 2002.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[22] L. Roditty, M. Thorup, and U. Zwick. Roundtrip spanners and roundtrip routing in directed graphs. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pages 844–851, Jan. 2002.

[23] A. Rowstron and P. 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)*, pages 329–350, Nov. 2001.

[24] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

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

[26] M. van Steen, F. Hauck, and A. Tanenbaum. A model for worldwide tracking of distributed objects. In *Proceedings of the 1996 Conference on Telecommunications Information Networking Architecture (TINA 96)*, pages 203–212, Sept. 1996.

[27] C. Wagner. *Drawing with Bezier Curves and Routing on Digraphs*. PhD thesis, Dept. of Mathematical Sciences, Johns Hopkins University, 1999.

[28] B. Y. Zhao, Y. Duan, L. Huang, A. D. Joseph, and J. D. Kubiatowicz. Brocade: landmark routing on overlay networks. In *First International Workshop on Peer-to-Peer Systems (IPTPS)/LNCS*, pages 34–44, march 2002.

[29] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.

[30] U. Zwick. Exact and approximate distances in graphs - a survey. In *Proc. of 9th European Symposium on Algorithms*, pages 33–48, 2001.