Compact Routing Schemes with Improved Stretch

Shiri Chechik
Microsoft Research Silicon Valley
Mountain View CA, USA
shiri.chechik@gmail.com

ABSTRACT

We consider the problem of compact routing in weighted general undirected graphs, in which the goal is to construct local routing tables that allow information to be sent on short paths in the network. In this paper the first improvement to the work of Thorup and Zwick [SPAA'01] is presented. Specifically, we construct an improved routing scheme obtaining for every k routing tables of size $\tilde{O}\left(n^{1/k}\log D\right)$, and stretch $(4-\alpha)k-\beta$ for some absolute constants $\alpha,\beta>0$, where D is the normalized diameter. This provides a positive answer to a main open question in this area as to the existence of a routing scheme with stretch $c\cdot k$ for some constant $c<4$.

Categories and Subject Descriptors: F.2.2 Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems

Keywords: compact routing, stretch factor, name independent routing

1. INTRODUCTION

Routing is perhaps one of the most fundamental problems in the area of distributed networking. The goal in this problem is to construct a distributed mechanism that allows any node in the network to send packages of data to any other node efficiently. As in all distributed algorithms, a routing scheme runs locally on every node of the network allowing it to forward arriving data while utilizing local information that is stored at the node itself. This local information is commonly referred to as the routing table of the node.

Formally, a routing scheme is comprised of two phases, the preprocessing phase and the routing phase. In the first phase, the preprocessing phase, each node is assigned a routing table and a small size label (poly-logarithmic in the size of the network) that are stored locally at the node. In the second phase, the routing phase, the routing scheme allows any node to send information to any other node in a distributed manner. Specifically, the scheme allows every node, upon receiving a message, to decide whether this message reached its final destination or to which of the node's neighbors this message should be sent next. In order to make such decisions, the node may use its own routing table and the header of the message that contains the label of the final destination and perhaps some additional information.

The stretch of a routing scheme is defined as the worst case ratio between the length of the path obtained by the routing scheme and the length of the shortest path between the source node and the destination node.

There are usually two key concerns in designing routing schemes. The first concern is to minimize the stretch of the routing scheme, and the second concern is to minimize the size of the routing tables. Much of the work on designing routing schemes focuses on the tradeoff between these two concerns.

An extreme case is when the designer is allowed to store linear size memory at the nodes. In this case it is possible to store a complete routing table at all nodes, i.e., for every node s and every destination t store the port of the neighbour of s on the shortest path from s to t. Using these routing tables it is possible to route messages on shortest path, namely, having a stretch of 1. The clear drawback of this solution is that we need routing tables of size $\Omega(n)$. In a large network, having routing tables of size $\Omega(n)$ can be too costly.

In many cases it would be desirable to store much smaller routing tables at the price of larger stretch. We say that a routing scheme is compact if the size of the routing tables is sub-linear in the number of nodes.

Many papers deal with the tradeoff between the size of the routing tables and the stretch (e.g. [16, 5, 6, 15, 7, 8, 13]). The first tradeoff was presented by Peleg and Upfal [16]. Their paper considered unweighted graph and achieved a bound on the total size of the routing tables. A tradeoff for weighted graphs with a guarentee on the maximum table size was later presented by Awerbuch *et al.* [5]. This paper presented a routing scheme that uses table size of $\tilde{O}(n^{1/k})$ with stretch $O(k^2 9^k)$. A better tradeoff was later obtained by Awerbuch and Peleg [6]. Efficient schemes for specific values of k were presented in [7, 8].

The best known tradeoff was achieved by Thorup and Zwick [17]. They showed a routing scheme that uses routing tables of size $\tilde{O}(n^{1/k})$, a stretch of $4k-5$ and labels size of $O(k\log n)$. This routing scheme assumes that the port numbers can be assigned by the routing process. In the case of fixed port model, namely, the port numbers are part of the input of the preprocessing phase, their labels size

PODC'13, July 22–24, 2013, Montréal, Québec, Canada.

increases to $O(k \log^2 n)$. In addition, they showed that if a handshaking is allowed, namely the source node and the target node can communicate before the routing phase starts and agree on an $o(\log^2 n)$ bit header that is attached to the header of all messages, then the stretch can be reduced to $2k-1$. However, in many cases, it would be desirable to avoid the use of handshaking, especially if the source wishes to send only a single message to the destination. In that case the overhead of establishing a handshake could be as high as sending the original message. Thorup and Zwick's scheme [17] of stretch of $2k-1$ established using a handshake is essentially optimal assuming the girth conjecture of Erdős [9]. Erdős [9] conjectured that for every $k > 1$ there are graphs with $\Omega(n^{1+1/k})$ number of edges whose girth is at least $2k+1$. If the conjecture is true, namely, there exists such a graph G, then any routing scheme of stretch less than $2k-1$ requires a total memory of at least $\Omega(n^{1+1/k})$ on some subgraphs of G. Namely, relying on this conjecture it is impossible to achieve a routing scheme that uses $O(n^{1/k})$ routing tables with less than $2k-1$ stretch with or without handshaking. For further lower bounds see [16, 10, 12, 18, 14]. A main open problem in the area of compact routing schemes is on the gap between the stretch $4k-5$ and $2k-1$ in the case of no handshaking. In this paper, we give the first evidence that the asymptotically optimal stretch is less than $4k$ (for the case of routing tables of size $\tilde{O}(n^{1/k})$ and no handshaking). This is the first improvement to the stretch-space tradeoff of routing scheme since the result of Thorup and Zwick [SPAA'01].

A closely related variant is that of name independent routing scheme. In this variant the addresses of the nodes are fixed, namely, they are part of the input network and cannot be changed by the routing scheme. The problem of name independent routing was extensively studied. The first tradeoff was presented by Awerbuch *et al.* [4]. They presented a compact name independent routing scheme with stretch that is exponential in k. This was followed by a series of improvements [6, 5, 3, 2, 1]. In [1], Abraham *et al.* presented a name independent routing scheme that uses $\tilde{O}(n^{1/k} \log D)$ with $O(k)$ stretch, where D is the normalized diameter.

All of our sizes, unless mentioned otherwise, are measured in the number of words, where a word is a storage unit large enough to contain any distance or an ID of a node.

Our contributions: We present the first improvement on the work of Thorup and Zwick [SPAA'01] by constructing a compact routing scheme in weighted general undirected graphs that uses tables of size $\tilde{O}(n^{1/k})$ and has stretch $c \cdot k$ for some absolute constant $c < 4$, thus, obtaining improved results for every $k \geq 4$. Specifically, for $k = 4$ we improve the 11 stretch of Thorup and Zwick to ≈ 10.52. In order to obtain this improved result we prove several structural properties on the Thorup and Zwick construction which might be of independent interest.

Paper Organization: Section 2 contains preliminaries and notations. In Section 3 we present the general framework used in the paper. In order to simplify presentation, we start by focusing on the case where $k = 4$ in Section 4. Section 5 contains the case of general k. For simplicity, we start by describing the scheme for unweighted graphs and later (in Section B) describe the modifications needed in order to handle weighted graphs.

2. PRELIMINARIES AND NOTATION

Let us introduce some notations that will be used throughout the text. For a graph G, denote by $V(G)$ and $E(G)$ respectively the sets of vertices and edges of G. Consider a rooted tree T and a node $v \in V(T)$. Denote by $r(T)$ the root of the tree T. Let $\mathbf{parent}(v,T)$ be the parent of v in the tree T or null in the case where v is the root of T. Let $\mathbf{childs}(v,T)$ be the set of children of v in the tree T. Let $\deg(v,T) = |\mathbf{childs}(v,T)|$, namely, the number of children of v in the tree T. Let $\mathbf{radius}(T)$ be the longest path from $r(T)$ to some node in T.

3. GENERAL FRAMEWORK

An essential ingredient in our routing scheme is a procedure for routing on rooted subtrees of the graph. Given a tree T, the procedure assigns every node v in T a label $L(v,T)$ and a routing table $A(v,T)$. Using the label $L(t,T)$ of some node t and the routing tables $A(v,T)$, it is possible to route to t from any node in T on their shortest (only) path in T. Thorup and Zwick presented a routing scheme on trees that uses $(1+o(1)) \log n$ label size and these labels are the only information stored at the nodes. In the case of fixed port model, namely, the port numbers are not allowed to be changed, their labels size increases to $O(\log^2 n)$. A similar scheme was presented by Fraigniaud and Gavoille [11].

Our scheme is strongly based on Thorup-Zwick construction (with some new ideas). For completeness we now outline the compact routing scheme of Thorup and Zwick. For a given positive integer k, construct the sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1} \supseteq A_k = \emptyset$ as follows. Each A_i for $1 \leq i \leq k-1$ is obtained by sampling the nodes in A_{i-1} independently at random with probability $(n/\ln n)^{-1/k}$. The pivot $p_i(v)$ is defined to be the closest node to v in A_i (break ties arbitrarily).

The bunch $B(v)$ of v is defined as follows. A node $w \in A_i \setminus A_{i+1}$ is added to $B(v)$ if $\mathbf{dist}(v,w) \leq \mathbf{dist}(v,A_{i+1})$. Namely,

$$B(v) = \bigcup_{i=0}^{k-1} \{w \in A_i \setminus A_{i+1} \mid \mathbf{dist}(v,w) < \mathbf{dist}(v,A_{i+1})\}.$$

Note that $\mathbf{dist}(v,A_k) = \infty$ and thus $A_{k-1} \subseteq B(v)$.

For a node $w \in A_i \setminus A_{i+1}$, the cluster $C(w)$ is defined as follows $C(w) = \{v \in V \mid \mathbf{dist}(w,v) \leq \mathbf{dist}(v,A_{i+1})\}$. Or in other words, the cluster of a node w is the set of nodes v such that $w \in B(v)$.

For every node $w \in V$, let $T(w)$ be the shortest path tree rooted at w spanning $C(w)$. For every node $w \in V$, invoke the routing scheme on trees on $T(w)$ and store the label $L(v,T(w))$ at the routing table $A_{TZ}(v)$ of v. The label $L_{TZ}(v)$ of v is the concatenation of $L(v,T(p_i(v)))$ for $1 \leq i \leq k-1$. This completes the construction of the routing tables and the labels.

It was shown in [17] that for every node v, $|B(v)| = O(n^{1/k} \log n)$, namely there are st most $O(n^{1/k} \log n)$ nodes w such that $v \in C(w)$. We get that $|A_{TZ}(v)| = O(n^{1/k} \log^3 n)$. The size of the label $L_{TZ}(v)$ is $O(k \log^2 n)$ (we have to use the fixed port model since the trees may overlap).

The routing process is done as follows. Assume some node s wants to send a message to some node t given the label $L_{TZ}(t)$. The node s finds the first index i such that $p_i(t) \in B(s)$, or in other words that $s \in T(p_i(t))$ and then it routes

the message to t on the tree $T(p_i(t))$ using $L(t, T(p_i(t)))$ $(\subseteq L_{TZ}(t))$.

Thorup and Zwick showed that the above scheme gives a stretch of $4k-3$. The proof of the stretch was based on the following claim. Let i be the first index such that $p_i(t) \in B(s)$ and let $d = \mathbf{dist}(s,t)$. For every $j \leq i$, $\mathbf{dist}(t, p_j(t)) \leq 2jd$ and $\mathbf{dist}(s, p_j(s)) \leq (2j-1)d$.

Note that the algorithm routes the message from s to t on $T(p_i(t))$ and that $\mathbf{dist}(s, p_i(t)) + \mathbf{dist}(t, p_i(t)) \leq \mathbf{dist}(s,t) + \mathbf{dist}(t, p_i(t)) + \mathbf{dist}(t, p_i(t)) \leq d + 4(k-1)d = (4k-3)d$. We thus get a stretch of $4k-3$.

Thorup and Zwick also showed that by using a slightly different sampling procedure it is possible to reduce the stretch to $4k-5$. The new sampling procedure guarantees that $|C(w)| \leq O(n^{1/k})$ for every $w \in A_0 \setminus A_1$. The algorithm stores the set $C(w)$ and the labels $L(v, T(w))$ for every $v \in C(w)$ in the table $A_{TZ}(w)$. In the routing process, the algorithm checks if $s \in A_0 \setminus A_1$ and $t \in C(s)$, if so it routes the message to t in $T(s)$. If $t \notin C(s)$ then by definition $\mathbf{dist}(t, p_1(t)) \leq \mathbf{dist}(s,t)$, (rather than $\mathbf{dist}(t, p_1(t)) \leq 2\mathbf{dist}(s,t)$). This saves up 2 to the total stretch, resulting with a stretch of $4k-5$.

In our scheme we need the following stronger property. For every $w \in A_{\ell-1} \setminus A_\ell$: $|C(w)| \leq O(n^{\ell/k})$ for $\ell \leq r$ for some integer r. We thus employ this sampling procedure for every index i and slightly change the sampling probability used in [17].

We construct the sets A_i as follows, $A_0 = V, A_k = \emptyset$ and for each $1 \leq i \leq k-1$, $A_i = \mathbf{center}(G, A_{i-1}, n^{1-i/k}/\log n)$.

Procedure **center** operates on a given graph G, set of nodes A' and a size s. It operates as follows. Initially set $A \leftarrow \emptyset$ and $W \leftarrow A'$. While $W \neq \emptyset$ do the following. Let $B' = \mathit{sample}(W, s)$, namely, B' is obtained by sampling every node in W independently at random with probability $s/|W|$, or $B' = W$ if $|W| \leq s$. Set $A \leftarrow A \cup B'$. Let $C_A(w) \leftarrow \{v \in V \mid \mathbf{dist}(w,v) < \mathbf{dist}(A,v)\}$, for every $w \in A'$. Set $W \leftarrow \{w \in A' \mid |C(w)| > 4n/s\}$. See Procedure 1 for the formal code.

The following lemma is crucial in our analysis. (The proof is deferred to the appendix.)

LEMMA 3.1. *For every node $w \in A_i \setminus A_{i+1}$: $|C(w)| = O(n^{(i+1)/k})$. In addition, for every node $v \in V$: $|B(v)| = O(n^{1/k} \log n)$.*

4. WARM-UP: THE CASE $K = 4$

In this section we present our routing scheme for the case $k = 4$.

An important ingredient in our routing algorithm is a procedure for name-independent routing on trees inspired from [2]. We present a name-independent routing scheme for a given tree T that is schematically done a follows. For a given tree T, distribute the labels $L(v,T)$ for every node $v \in V(T)$ among a subset of the nodes and design a search mechanism such that given $\mathbf{key}(v)$ can find in a distributed manner $L(v,T)$, where $\mathbf{key}(v)$ is a unique identifier of the node v in $[1..n]$.

Let us start with describing our search mechanism. We will later see how to use this mechanism in our routing scheme. The search mechanism presented here is designed specifically for the case of $k = 4$, in the case where $k > 4$ we need to use a more complicated search tree as will be described later on. We show the following lemma.

LEMMA 4.1. *Consider a tree T of depth d', a set of nodes $\mathbf{core}(T) \subseteq V(T)$ such that $r(T) \in \mathbf{core}(T)$, $|\mathbf{core}(T)| \geq \lceil |V(T)|/n^{1/k} \rceil$ and $|V(T)| \leq n^{2/k}$. One can construct a search scheme with the following properties.*
(1) The scheme stores $O(n^{1/k} \log^2 n)$ information $\mathbf{ST}(v,T)$ at every node $v \in \mathbf{core}(T)$.
(2) Given a key $\mathbf{key}$ the algorithm can find $L(\mathbf{key}, T)$ (or decides that $\mathbf{key} \notin V(T)$) in a distributed manner by traveling on a path from the root $r(T)$ of length at most $\mathbf{radius}(T)$.

Proof: The proof is by construction. Let $\mathcal{K} = \{\mathbf{key}(v) \mid v \in V(T)\}$. First, the algorithm distributes the keys $\mathcal{K}$ such that every node in $\mathbf{core}(T)$ stores $O(n^{1/k})$ keys and their matching labels. The algorithm assigns each node $v \in \mathbf{core}(T)$ an interval $I(v) = [n_1, n_2]$, the node v stores the labels of the keys in $\mathcal{K}$ in the range $[n_1, n_2]$. This is done as follows. Order the nodes at $\mathbf{core}(T)$ by some order. The algorithm stores at the first node in $\mathbf{core}(T)$ the first $n^{1/k}$ keys and their matching labels (the keys with the smallest IDs). The algorithm then assigns to the next node in $\mathbf{core}(T)$ the next $n^{1/k}$ keys and so on.

Recall that $|V(T)| \leq n^{2/k}$, we get that the algorithm assigns keys to at most $n^{1/k}$ nodes in $\mathbf{core}(T)$, let $\mathbf{c\hat{o}re}(T)$ be this set of nodes. Notice also that the algorithm cannot run out of nodes in $\mathbf{core}(T)$ as $|\mathbf{core}(T)| \geq \lceil |V(T)|/n^{1/k} \rceil$.

Consider a node $v \in \mathbf{core}(T)$, Let n_1 be the smallest key assigned to v and n_2 be the largest key assigned to v, the interval $I(v,T)$ of v is $[n_1,n_2]$. Note that all keys in the interval $I(v,T)$ are assigned to v. The node v stores the keys assigned to it and their matching labels. In addition, the root $r(T)$ stores the labels of the set $\mathbf{c\hat{o}re}(T)$ and their matching intervals.

This completes the construction of our search mechanism.

We now turn to describe the search algorithm for a given key $\mathbf{key}$. We assume that the search algorithms starts at the root $r(T)$. Namely, the algorithm first routes the message to the root and only then it invokes the search algorithm. The root $r(T)$ checks which of the nodes in $\mathbf{core}(T)$ stores the label $L(\mathbf{key}, T)$, namely, the node z whose interval contains $\mathbf{key}$. The root $r(T)$ attaches to the header of the message the label $L(z,T)$ and routes the message to the node z using $L(z,T)$. The node z either stores the label $L(\mathbf{key}, T)$ or determines that $\mathbf{key}$ does not exist in T. One can see that the path obtained by this search mechanism is of length at most $\mathbf{radius}(T)$ (until reaching the node in T containing $L(\mathbf{key}, T)$ or deciding that $\mathbf{key} \notin V(T)$). We thus get that property (2) is satisfied. In addition, it is not hard to verify that property (1) is satisfied by construction. ∎

Constructing the labels and routing tables:

We now turn to describe the construction of the labels and routing tables in our routing scheme.

The first step of our algorithm is to assign every node $v \in V$ a unique identifier $\mathbf{key}(v)$ in the range $[1..n]$. Next, construct Thorup-Zwick routing tables and labels. The label $L(v)$ is defined as follows. Add the key $\mathbf{key}(v)$ to the label $L(v)$. Next, add to $L(v)$ the label $L_{TZ}(v)$ assigned to v by the Thorup-Zwick construction. In addition, add to $L(v)$ the distances $\mathbf{dist}(v, P_i(v))$ for every $1 \leq i \leq k-1$.

It is not hard to verify that the asymptotic sizes of the labels $L(v)$ and $L_{TZ}(v)$ are the same.

The routing table $A(v)$ of the node v is constructed as follows. First, add to the routing table $A(v)$ the routing ta-

```
Algorithm center(G(V,E), A', s)

A ← ∅; W ← A'
while W ≠ ∅; do:
{
    A ← A ∪ sample(W, s)
    C_A(w) ← {v ∈ V | dist(w,v) < dist(A,v)}, for every w ∈ A';
    W ← {w ∈ A' | |C(w)| > 4n/s}
}
return A;
```

Figure 1: Choosing a set of centers with small size clusters.

ble $A_{TZ}(v)$ assigned to v by the Thorup-Zwick construction. In addition, we enhance the routing tables with additional information. We now describe the additional information stored at the nodes.

For every node $w \in V$, let $T(w)$ be the shortest path tree rooted at w spanning $C(w)$. For every node $w \in A_1 \setminus A_2$ and every distance $d' = (1+\epsilon)^j$ for $1 \leq j \leq \log D(w)$, where $D(w) = \mathbf{radius}(T(w))$, do the following.

The core $\mathbf{core}(w, d')$ is obtained by sampling every node $v \in T(w)$ independently at random with probability $4 \log n / n^{1/k}$. Let $T(w, d')$ be the tree $T(w)$ trimmed at distance d', i.e., the tree that is obtained by deleting all nodes from $T(w)$ that are at distance greater than d' from w. Let $C(w, d')$ be set of nodes in $T(w, d')$, i.e., all nodes v in $C(w)$ such that $\mathbf{dist}(v, w) \leq d'$.

For every node $w \in V$, and every distance $d' = (1+\epsilon)^j$ construct the search mechanism of Lemma 4.1 on $T(w, d')$ and $\mathbf{core}(w, d') \cup \{w\}$. Add to the routing tables $A(v)$ the information $\mathbf{ST}(v, T(w, d'))$ of the search mechanism for every $v \in \mathbf{core}(w, d') \cup \{w\}$.

This completes the construction of our labels and routing tables.

Let us now analyze the size of the routing tables.

By Chernoff bounds we show the following.

LEMMA 4.2. *With high probability the following two events occur. 1. For every node $w \in A_1 \setminus A_2$: $|\mathbf{core}(w, d')| \geq \lceil |C(w)|/n^{1/k} \rceil$. 2. For every node $v \in V$, there are at most $O(\log^2 n)$ nodes w and distances $d' = (1+\epsilon)^j$ for $1 \leq j \leq \log D(w)$ such that $v \in \mathbf{core}(w, d')$.*

Proof: The expected size μ_1 of $\mathbf{core}(w, d')$ is $\mu_1 = (4 \log n |V(T)|)/n^{1/k}$.

Recall that by Chernoff's bound, we have for a binomial random variable X such that $E[X] = \mu$, $Pr[X < (1+\delta)\mu] \leq exp(\mu\delta^2/2)$. We thus get that $Pr[|\mathbf{core}(w, d')| \leq (\log n |V(T)|)/n^{1/k}] \leq exp(\mu) < 1/n^2$. The first part follows.

To see the second part, recall that a node v belongs to $O(n^{1/k} \log n)$ clusters. For each cluster, the algorithm considers $O(\log n)$ distances d'. The probability that a node v belongs to $\mathbf{core}(w, d')$ is $4 \log n / n^{1/k}$. Thus the expected number μ_2 of nodes w and distances d' such that $v \in \mathbf{core}(w, d')$ is $O(\log^2 n)$. By applying Chernoff bound we get that with high probability there are at most $O(\log^2 n)$ nodes w and distances d' such that $v \in \mathbf{core}(w, d')$. ∎

We thus conclude the following.

LEMMA 4.3. *For every node v the expected size of the routing table $A(v)$ is $O(n^{1/k} \log^4 n)$.*

Proof: Recall that there are two main parts in the routing table $A(v)$. The first part is the routing table $A_{TZ}(v)$ of Thorup-Zwick scheme and the second part is the information $\mathbf{ST}(v, T(w, d'))$ of the search mechanism for every w and d' such that $v \in \mathbf{core}(w, d') \cup \{w\}$. It was shown in [17] that the size of $A_{TZ}(v)$ is $O(n^{1/k} \log^3 n)$. We left with bounding the size of the second part. By Lemma 4.2 there are $O(\log^2 n)$ sets $\mathbf{core}(w, d')$ such that $v \in \mathbf{core}(w, d')$. By Lemma 4.1 for each such set the size of $\mathbf{ST}(v, T(w, d'))$ is $O(n^{1/k} \log^2 n)$. The lemma follows. ∎

The routing phase: We now describe the routing phase for the case $k = 4$.

For a node v let $\Delta_j(v) = \mathbf{dist}(v, p_{j+1}(v)) - \mathbf{dist}(v, p_j(v))$. Let k' be the minimal index such that $p_{k'-1}(t) \in B(s)$, or 1 in case either $t \in B(s)$ or $s \in B(t)$.
Let $M = \max\{\Delta_j(s)/2, \Delta_j(t)/2 \mid 1 \leq j \leq k'-2\} \cup$
$\{\Delta_0(s), \Delta_0(t)\} \cup$
$\{\mathbf{dist}(s, p_{j+1}(s)) - \mathbf{dist}(t, p_j(t)) \mid 1 \leq j \leq k'-2\}$. Note that the information needed to calculate k' and M can be extracted during the routing process from the label $L(t)$ and the routing table $A(S)$.

The routing phase is done as follows.

First check if $s \in A_0 \setminus A_1$ and $s \in B(t)$, if so route the message to t on $T(s)$ using $L(t, T(s))$. Note that, this information can be extracted from s's routing table. If $s \in B(t)$ then by definition $t \in C(s)$ and recall that $s \in A_0 \setminus A_1$ stores the set $C(s)$ and the labels $L(x, T(s))$ for every $x \in C(s)$ (as part of the Thorup-Zwick routing $A_{TZ}(s)$).

Otherwise, check if either $p_1(t) \in B(s)$ or $p_2(t) \in B(s)$. If so route the message to t on $T(p_i(t))$ where $i \in \{1, 2\}$ is the minimal index such that $p_i(t) \in B(s)$

In all other cases, check if $\Delta_1(t) \leq c \cdot M$ (for some parameter $1 < c < 2$ to be fixed later on). If so, invoke the standard Thorup-Zwick routing algorithm. Otherwise do the following. Let $d' = (1+\epsilon)^i$ for some index i be the minimal distance such that $d' \geq (1+c) \cdot M$.

Check using the search mechanism if $p_1(s) \in B(t)$. This is done by routing the message from s to $p_1(s)$ in $T(p_1(s))$. Then search for the key $\mathbf{key}(t)$ in the tree $T(p_1(s), d')$, if the key exists in $T(p_1(s), d')$ then route the message to t and quit. Otherwise if the $\mathbf{key}(t)$ was not found, return the message to s and invoke the standard Thorup-Zwick routing algorithm. This completes the description of the routing process.

We now turn to analyze the stretch of our routing scheme.

Let $d = \mathbf{dist}(s, t)$. By Thorup-Zwick analysis, either both $\Delta_0(s) \leq d$ and $\Delta_0(t) \leq d$, or we can route the message on

the exact shortest path from s to t. To see this, note that if $\Delta_0(s) > d$, then $t \in B(s)$ and the message can be routed from s to t on $T(t)$ using the label $L(t, T(t))$, where the label $L(t, T(t))$ can be extracted from t's label. Similarly, if $\Delta_0(t) > d$, then $s \in B(t)$ or in other words, $t \in C(s)$ and the message can be routed from s to t in $T(s)$ using $L(t, T(s))$. The label $L(t, T(s))$ can be extracted from s's routing table. So assume this is not the case, namely, $k' > 1$. We now show that M is a lower bound on the distance from s to t.

LEMMA 4.4. *If $k' > 1$ then $\mathbf{dist}(s,t) \geq M$.*

Proof: By the definition of k' and the assumption that $k' > 0$, we have $t \notin B(s)$ and $s \notin B(t)$. By the definition of $B(s)$ and $B(t)$, we have
$\mathbf{dist}(s,t) \geq \mathbf{dist}(s, p_1(s)) = \Delta_0(s)$ and
$\mathbf{dist}(s,t) \geq \mathbf{dist}(t, p_1(t)) = \Delta_0(t)$.

We now turn to show that
$\mathbf{dist}(s,t) \geq \max\{\Delta_j(s)/2, \Delta_j(t)/2 \mid 1 \leq j \leq k'-2\}$. Note that for every j such that $p_j(t) \notin B(s)$, we have
$\mathbf{dist}(s, p_{j+1}(s)) \leq d + \mathbf{dist}(t, p_j(t))$. Hence
$\mathbf{dist}(t, p_{j+1}(t)) \leq d + \mathbf{dist}(s, p_{j+1}(s)) \leq 2d + \mathbf{dist}(t, p_j(t))$. Hence, $d = \mathbf{dist}(s,t) \geq (\mathbf{dist}(t, p_{j+1}(t)) - \mathbf{dist}(t, p_j(t)))/2$ $= \Delta_j(t)/2$. In addition, we have
$\mathbf{dist}(t, p_j(t)) \leq d + \mathbf{dist}(s, p_j(s))$. Hence, $\mathbf{dist}(s, p_{j+1}(s)) \leq d + \mathbf{dist}(t, p_j(t)) \leq$
$d + d + \mathbf{dist}(s, p_j(s))$. We get that $d = \mathbf{dist}(s,t) \geq (\mathbf{dist}(s, p_{j+1}(s)) - \mathbf{dist}(s, p_j(s)))/2 = \Delta_j(s)/2$.

We are left to show that $\mathbf{dist}(s,t) \geq \max\{p_{j+1}(s) - p_j(t) \mid 1 \leq j \leq k'\}$. Consider $1 \leq j \leq k'$, notice that by the definition of k', $p_j(t) \notin B(s)$. We get that $\mathbf{dist}(s, p_{j+1}(s)) \leq \mathbf{dist}(s,t) + \mathbf{dist}(t, p_j(t))$. Hence, $\mathbf{dist}(s,t) \geq p_{j+1}(s) - p_j(t)$, as required. ∎

Lemma 4.4 gives us a good starting point. We already have a lower bound on $\mathbf{dist}(s,t)$. Notice that in the worst case, where Thorup-Zwick analysis gives a stretch of $4k-5$ is when $\Delta_j(s)$ and $\Delta_j(t)$ are roughly $2M$ for any $j > 0$, and M is slightly smaller than $\mathbf{dist}(s,t)$.

We now turn to bound the stretch of our routing process.

LEMMA 4.5. *The stretch of our routing process is at most* $\max\{7 + 2c, 9, 15/c + 2 + 2\epsilon(1/c+1)\}$.

Proof: First, notice that in one of the following three cases the stretch of the routing process is at most 7 (instead of 11) and we are done. 1. $s \in A_0 \setminus A_1$ and $s \in B(t)$. 2. $p_1(t) \in B(s)$. 3. $p_2(t) \in B(s)$. In the above mentioned cases, by following Thorup-Zwick analysis, one can show that the stretch of the routing process is at most 7.

So assume this is not the case. Recall that by Lemma 4.4, we have
$d = \mathbf{dist}(s,t) \geq \max\{\Delta_j(s)/2, \Delta_j(t)/2 \mid 1 \leq j \leq k'-2\} \cup \{\Delta_0(s), \Delta_0(t)\} \cup \{\mathbf{dist}(s, p_{j+1}(s)) - \mathbf{dist}(t, p_j(t)) \mid 1 \leq j \leq k'-2\}$.

Consider first the case where $\Delta_1(t) \leq c \cdot M$. Recall that in this case the algorithm routes the message to t on $T(p_3(t))$. Thus the length of path obtained by the routing scheme in this case is $\mathbf{dist}(s, p_3(t)) + \mathbf{dist}(t, p_3(t))$. Notice that in this case we have $\mathbf{dist}(t, p_2(t)) \leq \Delta_0(t) + \Delta_1(t) \leq M + c \cdot M = (1+c)M \leq (1+c)\mathbf{dist}(s,t)$. We thus get, $\mathbf{dist}(t, p_3(t)) \leq \Delta_0(t) + \Delta_1(t) + \Delta_2(t) \leq (1+c)M + 2M \leq (3+c)\mathbf{dist}(s,t)$. In addition, $\mathbf{dist}(s, p_3(t)) \leq \mathbf{dist}(s,t) + \mathbf{dist}(s, p_3(t)) \leq (4+c)\mathbf{dist}(s,t)$. We get that $\mathbf{dist}(s, p_3(t)) + \mathbf{dist}(t, p_3(t)) \leq (7+2c)\mathbf{dist}(s,t)$. Hence, the stretch in this case is $7 + 2c$.

Finally, consider the case where $\Delta_1(t) > c \cdot M$. Let $d' = (1+\epsilon)^i$ be the minimal distance such that $d' \geq (1+c) \cdot M$ for some index i.

The algorithm searches for $\mathbf{key}(t)$ in the tree $T(p_1(s), d')$. Recall that $\mathbf{dist}(s, p_1(s)) \leq M$. Hence the algorithm traverses a path of at most M until reaching $p_1(s)$, it then finds $L(t, T(p_1(s), d'))$ or decides that $t \notin T(p_1(s), d')$ by traveling on a path of length at most d'. We now need to consider two subcases. The first subcase is when $t \in T(p_1(s), d')$. In that case the algorithm routes the message to t in $T(p_1(s), d')$. Note that the overall path in this case is at most $M + d' + d' + \mathbf{dist}(p_1(s), t) \leq 3\mathbf{dist}(s,t) + 2(1+\epsilon)(1+c) \cdot M \leq 9\mathbf{dist}(s,t)$.

The last case is when $t \notin T(p_1(s), d')$. Note that this could happen only if $\mathbf{dist}(s,t) \geq cM$.

In this case the algorithm reaches the root $p_1(t)$, tries to find $\mathbf{key}(t)$ in $T(p_1(s), d')$ fails, returns the message to s and then invoke the standard Thorup-Zwick algorithm. The total path traveled by the algorithm in this case is $M + 2d' + M + 11M$. Since $\mathbf{dist}(s,t) \geq cM$, we get that the stretch is $(2d' + 13M)/\mathbf{dist}(s,t) \leq (2d' + 13M)/(cM) \leq (2(1+\epsilon)(1+c)M + 13M)/(cM) == 15/c + 2 + 2\epsilon(1/c+1)$. ∎

By setting $c = (\sqrt{145} - 5)/4$ and taking ϵ to be small enough, we get a stretch of roughly 10.52 instead of the stretch 11 obtained by Thorup-Zwick's routing scheme.

5. THE GENERAL CASE $K > 4$

For simplicity, we present the construction for unweighted graphs, we later (in Section B) explain the modifications needed to handle weighted graphs.

Let us start with a general overview of our routing scheme for general k. As in the case of $k = 4$, we start by constructing Thorup-Zwick routing tables and labels. Each node stores the routing table and the label assigned to it by the the Thorup-Zwick scheme. As in the case of $k = 4$, we construct a name-independent routing mechanism on the trees $T(w)$, by storing additional information at the nodes of $T(w)$, while keeping the tables size $\tilde{O}(n^{1/k})$.

Roughly speaking, the routing phase is done as follows. The source node s checks if the target node t satisfies $\mathbf{dist}(p_r(t), t) \leq c \cdot r \cdot M$ (for some parameters r and $1 < c < 2$ to be fixed later on). If so, the node s invokes the standard Thorup-Zwick routing algorithm. Otherwise an attempt is made to route the message to t on the tree $T(p_j(s))$ for some $j \leq r$. This is done by first searching for the label $L(t, T(p_j(s)))$ using the search mechanism constructed for $T(p_j(s))$. If $t \in T(p_j(s))$ then the label $L(t, T(p_j(s)))$ is found using the search algorithm and the message is then routed to t using $L(t, T(p_j(s)))$, otherwise the message is bounced back to s and s invokes the standard Thorup-Zwick routing algorithm.

Recall that in the Thorup-Zwick analysis we have $\mathbf{dist}(p_r(t), t) \leq (2 \cdot r - 1) \cdot M$, thus if $\mathbf{dist}(p_r(t), t) \leq c \cdot r \cdot M$ for small enough c then following Thorup-Zwick analysis we get a better stretch. Otherwise, there must be an index j such that $\Delta_j(t) > c \cdot M$ and the algorithm tries to route the message to t on $T(p_j(s))$ using the search mechanism. If $t \in T(p_j(s))$ then the algorithm finds $L(t, T(p_j(s)))$ and the message is routed to t on $T(p_j(s))$. Otherwise, the algorithm invokes the standard Thorup-Zwick routing algorithm. The detour for searching the label $L(t, T(p_j(s)))$ and returning back to s appears to be a waste in the case

where $t \notin T(p_j(s))$, however, we show that the only case that $t \notin T(p_j(s))$ is when $\mathbf{dist}(s,t) \geq c \cdot M$. In this particular case we can actually show that the Thrup-Zwick algorithm gives a much better stretch. Hence, even combined with the path traveled by the search algorithm, overall we still get a smaller stretch than in the general case of the Thrup-Zwick analysis.

The search mechanism we use in this section is slightly different than the one we presented for the case of $k = 4$. In the general case, we are given a tree $T(w)$ such that $|T(w)| \leq n^{r/k}$ and $\mathbf{radius}(T(w)) \leq r\rho$ for some distance ρ and the goal is to design a search mechanism that finds the label $L(t, T(w))$ by traveling on a short path, where by a short path we mean $O(\mathbf{radius}(T))$ length. The main difficulty with the previous search mechanism is that in the general case, the node s cannot store a complete map that indicates which node in $T(w)$ contains the label of t (as otherwise we will have to violate the constraint that every node stores $\tilde{O}(n^{1/k})$ information). In fact, it is possible to show that if $|T(w)| > n^{r/k}$, then in the worst case the algorithm must visit at least r different nodes in order to find $L(t, T(w))$. Naively, we could partition the set of nodes $V(T(w))$ into $n^{1/k}$ sets and pick $n^{1/k}$ nodes $\mathbf{helpers}(w)$ in $T(w)$ and assign each such node with one of these sets. We could partition $V(T(w))$ into sets in such a way that each such set S corresponds to a continuous interval I. The node w would store a map between these intervals and the corresponding nodes $\mathbf{helpers}(w)$. The size of each such set is $|V(T(w))|/n^{1/k}$. We could continue this process partitioning these sets into smaller sets until we have sets of size $O(n^{1/k})$ that a single node can store. Using this search mechanism, it is possible to find the label $L(t, T(w))$ by vising r nodes in $T(w)$. However, note that naively these r nodes could be far away from one other (distance $2\mathbf{radius}(T(w))$), so naively this process could yield a path of length $O(r \cdot \mathbf{radius}(T(w)))$. We thus pick the nodes of $\mathbf{core}(T(w))$ in a more careful way, in order to reduce the maximum length of the path traveled by the search process to $O(\mathbf{radius}(T(w)))$ rather than $O(r \cdot \mathbf{radius}(T(w)))$. We pick the nodes $\mathbf{core}(T(w))$ in such a way that for every node $v \in T(v)$, $|\mathbf{core}(T) \cap \mathbf{childs}(v,T)| \geq \log n \lceil |\mathbf{childs}(v,T)|/n^{1/k} \rceil$. At every node $v \in \mathbf{core}(T(w))$ our search mechanism stores $O(\min\{\mathbf{deg}(\mathbf{parent}(v,T)), n^{1/k}\})$ information. We later show that we can pick $\mathbf{core}(T(w))$ in a way that maintain the requirement that every node stores at most $\tilde{O}(n^{1/k})$ information in total.

More precisely, we show the following search scheme (the proof is deferred to the appendix).

LEMMA 5.1. *Consider a tree T of depth d', a set of nodes $\mathbf{core}(T) \subseteq V(T)$ such that $r(T) \in \mathbf{core}(T)$, and that for every $v \in T(v)$, $|\mathbf{core}(T) \cap \mathbf{childs}(v,T)| \geq \lceil |\mathbf{childs}(v,T)|/n^{1/k} \rceil$. One can construct a search mechanism with the following properties.*
1. The search mechanism stores $O(s(v,T)\log^2 n)$ data size at every node $v \in \mathbf{core}(T)$ and at most $O(\log^2 n)$ data at the rest of the nodes of T, where $s(v,T) = \min\{\mathbf{deg}(\mathbf{parent}(v,T)), n^{1/k}\}$. Let $\mathbf{ST}(v,T)$ be the data stored at a node $v \in V(T)$.
2. Given a key $\mathbf{key}$, the algorithm can find $L(\mathbf{key}, T)$ (or decides that $\mathbf{key} \notin V(T)$) from $r(T)$ in a distributed manner, by traveling on a path of length at most $3\mathbf{radius}(T) + 2(\mathbf{ind}(T) - 1)$, where $\mathbf{ind}(T)$ is the minimal index such that $|V(T)| \leq n^{\mathbf{ind}(T)/k}$.

Constructing the labels and routing tables:

We now describe the construction of the labels and routing tables of our routing scheme. First, construct Thorup-Zwick routing tables and labels.

The label $L(v)$ is defined as follows (similarly as in the case where $k = 4$). Add the key $\mathbf{key}(v)$ to the label $L(v)$. Next, add to $L(v)$ the label $L_{TZ}(v)$ assigned to v by the Thorup-Zwick construction. In addition, add to $L(v)$ the distances $\mathbf{dist}(v, P_i(v))$ for every $1 \leq i \leq k-1$.

The routing table $A(v)$ of the node v is constructed as follows. First, add to the routing table $A(v)$, the routing table $A_{TZ}(v)$ assigned to v by the Thorup-Zwick construction. In addition, we enhance the routing tables with the following additional information.

For every node $w \in V$ and distance $d' = (1+\epsilon)^j$ for $1 \leq j \leq \log n$ do the following. The core $\mathbf{core}(w, d')$ is obtained by sampling every node $v \in T(w)$ independently at random with probability $\min(1, 4\log n/\mathbf{deg}(\mathbf{parent}(v,w)))$ when $\mathbf{deg}(\mathbf{parent}(v,w)) < n^{1/k}$ and with probability $4\log n/n^{1/k}$ otherwise.

For every node $w \in V$, and every distance $d' = (1+\epsilon)^j$ construct the search mechanism of Lemma 5.1 on $T(w, d')$ and $\mathbf{core}(w, d') \cup \{w\}$. Add to the routing tables $A(v)$ the information $\mathbf{ST}(v, T(w, d'))$ of the search mechanism for every $v \in T(w, d')$.

This completes the construction of our labels and routing tables.

We now turn to analyze the size of the routing tables.

By Chernoff bounds we show the following two lemmas.

LEMMA 5.2. *With high probability for every node $w \in V$, distance d' and node $v \in T(w, d')$, $|\mathbf{childs}(v,w) \cap \mathbf{core}(w, d')| \geq \lceil |\mathbf{childs}(v,w)|/n^{1/k} \rceil$.*

LEMMA 5.3. *With high probability for every node v, $|A(v)| = \tilde{O}(n^{1/k})$.*

Proof: Recall that $|B(v)| \leq O(n^{1/k}\log n)$, namely, there are at most $O(n^{1/k}\log n)$ nodes w such that $v \in C(w)$. Consider such a node w and distance $d' = (1+\epsilon)^j$ for some $1 \leq j \leq \log n$. Let $p(v,w,d')$ be the probability that $v \in \mathbf{core}(w, d')$, where $p(v,w,d')$ is $\min(1, 4\log n/\mathbf{deg}(\mathbf{parent}(v,w)))$ if $\mathbf{deg}(\mathbf{parent}(v,w)) < 1/n^{1/k}$ and $4\log n/n^{1/k}$ otherwise. Recall that by Lemma 5.1 the size of $\mathbf{ST}(v, T(w, d'))$ is $O(s(v,T)\log^2 n)$ if $v \in \mathbf{core}(T)$ and $O(\log^2 n)$ otherwise, where $s(v,T) = \min\{\mathbf{deg}(\mathbf{parent}(v,T)), n^{1/k}\}$.

We get that the expected size of $\mathbf{ST}(v, T(w, d'))$ is $O(\log^2 n s(v,T) \cdot p(v,w,d') + \log^2 n) = O(\log^2 n)$.

There are $O(n^{1/k}\log^2 n)$ nodes w such that $v \in C(w)$, for each such w the algorithm considers $\log n$ different distances d'. Hence, the expected size of $A(v)$ is $O(n^{1/k}\log^4 n)$. By applying Chernoff bound we can show that with high probability, the size of $A(v)$ is roughly its expectation, i.e., $O(n^{1/k}\log^4 n)$. ∎

The routing phase: The routing phase is done as follows.

Let k' be the minimal index such that $p_{k'-1}(t) \in B(s)$, or 1 in case either $t \in B(s)$ or $s \in B(t)$.

Let $M = \max\{\Delta_j(s)/2, \Delta_j(t)/2 \mid 1 \leq j \leq k'-2\} \cup \{\Delta_0(s), \Delta_0(t)\} \cup \{\mathbf{dist}(s, p_{j+1}(s)) - \mathbf{dist}(t, p_j(t)) \mid 1 \leq j \leq k'-2\}$. Recall that by Lemma 4.4, we have $\mathbf{dist}(s,t) \geq M$.

We now present the routing process. The source node s checks if there exists an index $j \leq r$ such that $\mathbf{dist}(p_{j+1}(t),t) - \mathbf{dist}(p_{j-1}(t),t) \geq 2cM$ (for some integer $r < k$ and number $1 < c < 2$ to be fixed later on). If no such index exists, s invokes the standard Thorup-Zwick routing algorithm, otherwise do the following.

Let $d' = (1+\epsilon)^i$ be the minimal distance such that $d' \geq 2M + (j-2)cM + (2c-1)M$ for some index i .

The node s tries to find $L(t, T(p_j(s)))$ using the search mechanism constructed on $T(p_j(s), d')$. If the algorithm finds $L(t, T(p_j(s)))$ then the message is routed to t using $L(t, T(p_j(s)))$. Otherwise, the message is bounced back to s and the standard Thorup-Zwick routing algorithm is invoked.

This conclude the routing process.

The following lemma bound the stretch of our routing scheme.

LEMMA 5.4. *The maximum stretch obtained by our routing process is at most* $\max\{4k-1-4r-2c+2rc, (1+\epsilon)(6+6rc)/(2c-1) + (4k-4r+2rc-2c+2)/(2c-1) + 2(r-1)/M((2c-1)), (1+\epsilon)6(1+rc) + 2(r-1)/M\}$.

We need to consider few cases.

The simplest case is when there is no index $j \leq r$ such that $\mathbf{dist}(p_r(t),t) - \mathbf{dist}(p_{r-2}(t),t) \geq 2cM$. In this case the algorithm invokes the standard Thorup-Zwick algorithm. Note that in this case we have $\mathbf{dist}(t, p_r(t)) \leq M + (r-1)cM$, whereas in Thorup-Zwick analysis they have $\mathbf{dist}(t, p_r(t)) \leq M + 2(r-1)M$. This saves $2(r-1)(2-c)$ in the final stretch. So the total stretch is $4k-5-2(r-1)(2-c) = 4k-1-4r-2c+2rc$.

Consider now the case where there exists an index $j \leq r$ such that $\mathbf{dist}(p_r(t),t) - \mathbf{dist}(p_{r-2}(t),t) \geq 2cM$. Let $d' = (1+\epsilon)^i$ for some index i be the minimal distance such that $d' \geq 2M + (j-2)cM + (2c-1)M$.

There are two subcases. The first subcase is when $t \in T(p_j(s), d')$ and the second when $t \notin T(p_j(s), d')$. Consider the first subcase. In that case the algorithm routes the message to t in $T(p_j(s), d')$. The node s tries to find $L(t, T(p_j(s)))$ using the search mechanism constructed on $T(p_j(s), d')$. The algorithm traveled to $p_j(s)$ and from $p_j(s)$ it tried to find $L(t, T(p_j(s)))$ using the search mechanism. By Lemma C.1 the path traveled by the search mechanism from $p_j(s)$ until finding $L(t, T(p_j(s)))$ or deciding that $t \notin T(p_j(s))$ is $3d' + 2(j-1)$. Note that the overall path in this case is at most $6d' + 2(j-1) \leq (1+\epsilon)6(2M + (j-2)cM + (2c-1)M) + 2(j-1)$. The stretch in this case is at most $(1+\epsilon)6(1+jc) + 2(j-1)/M$.

Consider now the subcase where $t \notin T(p_j(s), d')$. Note that this could happen only if $\mathbf{dist}(s,t) \geq (2c-1)M$. The total path traveled by the algorithm in this case is $6d' + 2(j-1) + (M + (j-1)cM + (k-j)2M) \leq (1+\epsilon)6(2M + (j-2)cM + (2c-1)M) + 2(j-1) + 2(M + (j-1)cM + (k-j)2M) \leq (1+\epsilon)M(6+6jc) + M(4k-4j+2jc-2c+2) + 2(j-1)$.

We know however that in this case $\mathbf{dist}(s,t) \geq (2c-1)M$. Hence the stretch in this case is at most $(1+\epsilon)(6+6jc)/(2c-1) + (4k-4j+2jc-2c+2)/(2c-1) + 2(j-1)/M((2c-1))$. ∎

By minimizing over r and c, one can show that the stretch obtained by Lemma 5.4 is less than $3.68 \cdot k$.

Graphs of bounded degree: We note that it is possible to further decrease the stretch in the case of bounded degree graphs. In the case of bounded degree graphs, one can construct a more efficient search mechanism while maintaining the required constraints on the tables size. By using these better search mechanism we can further decrease the stretch to roughly $3.58k$ (details deferred to the full version).

6. CONCLUSIONS

In this paper we provide the first improvement to the work of Thorup and Zwick[SPAA'01], presenting a compact routing scheme for weighted general undirected graphs which uses tables of size $\tilde{O}\left(n^{1/k}\right)$ and has stretch $c \cdot k$ for some absolute constant $c < 4$, for every $k \geq 4$. We note that it is possible to obtain an improved guarantee for the stretch by a more careful analysis of our routing scheme. However, it seems unlikely that our scheme might allow the stretch to go as low as $2k$ since the algorithm must "detour" in order to find the tŠs label in the tree $T(p_i(s))$ for some i. The main question that still remains unresolved is to prove or disprove the existence of a compact routing scheme that utilizes tables of size $\tilde{O}\left(n^{1/k}\right)$ and has stretch of $2k$ without the use of a handshake.

Acknowledgement: I'm extremely grateful to Ittai Abraham for very helpful discussions.

7. REFERENCES

[1] I. Abraham, C. Gavoille, and D. Malkhi. Routing with improved communication-space trade-off. In *Proc. 18th Annual Conference on Distributed Computing (DISC)*, 305–319, 2004.

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

[3] M. Arias, L. Cowen, K. Laing, R. Rajaraman, and O. Taka, Compact routing with name independence. In *Proc. 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 184Ű-192, 2003.

[4] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive routing. In *Proc. 21st ACM Symp. on Theory of Computing (STOC)*, 479Ű-489, 1989.

[5] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Improved routing strategies with succinct tables. In *J. Algorithms*, 11(3):307–341, 1990.

[6] B. Awerbuch and D. Peleg. Sparse partitions. In *Proc. 31st IEEE Symp. on Foundations of Computer Science (FOCS)*, 503–513, 1990.

[7] L.J. Cowen. Compact routing with minimum stretch. *J. Alg.*, 38:170–183, 2001.

[8] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. In *J. Algorithms*, 46:97–114, 2003.

[9] P. Erdős. Extremal problems in graph theory. In *Theory of graphs and its applications*, pages 29Ű-36, 1964.

[10] P. Fraigniaud and C. Gavoille. Memory requirement for universal routing schemes. In *Proc. 14th ACM Symp. on Principles of Distributed Computing (PODC)*, 223–230, 1995.

[11] P. Fraigniaud and C. Gavoille. Routing in Trees. In *28th Int'l Coll. on Automata, Languages and Programming (ICALP)*, 757–772, 2001.

[12] C. Gavoille and M. Gengler. Space-efficiency for routing schemes of stretch factor three. In *J. Parallel Distrib. Comput.*, 61:679–687, 2001.

[13] C. Gavoille and D. Peleg. Compact and localized distributed data structures. In *Distributed Computing*, 16:111–120, 2003.
[14] C. Gavoille and C. Sommer. Sparse spanners vs. compact routing. In *Proc. 23th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 225–234, 2011.
[15] D. Peleg. Distributed computing: a locality-sensitive approach. In *SIAM*, 2000.
[16] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. In *J. ACM*, 36(3):510–530, 1989.
[17] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 1–10, 2001.
[18] M. Thorup and U. Zwick. Approximate distance oracles. In *J. ACM*, 52, 1–24, 2005.

APPENDIX

A. SOME PROOFS

Proof of Lemma 3.1:

The proof of Lemma 3.1 is strongly based on Lemma 3.2 and Theorem 3.1 from [17]. We state the lemmas here for completeness.

Let $B_A(v) = \{w \in V \mid \mathbf{dist}(w,v) < \mathbf{dist}(A,v)\}$ and $C_A(w) = \{v \in V \mid \mathbf{dist}(w,v) < \mathbf{dist}(A,v)\}$.

LEMMA A.1. *Let $W \subseteq V$, $1 \leq s \leq n$ and let $A' \leftarrow sample(W,s)$, namely A' is obtained by sampling every node in W independently at random with probability $s/|W|$, or $A' = W$ if $|W| \leq s$. Then, for every $v \in V$, we have $E[B_{A'}(v) \cap W] \leq |W|/s$.*

THEOREM A.2. *The expected size of the set A returned by algorithm* **center** *is at most $2s \log n$. In addition, for every $w \in A'$, $C_A(w) \leq 4n/s$.*

By Theorem A.2, we get that $|A_i| = O(n^{1-i/k})$ for $1 \leq i \leq k-1$.

Consider a node $w \in A_i \setminus A_{i+1}$. By Theorem A.2, $C(w) = C_{A_{i+1}}(w) \leq 4n/(n^{1-i+1/k}/\log n) = 4n/(n^{1-(i+1)/k}/\log n) = O(\log n \cdot n^{(i+1)/k})$.

We are left to show that $|B(v)| = O(n^{1/k} \log n)$ for every $v \in V$.

Let $\hat{A}$ be the set A after the first iteration of procedure **center** when invoked on the input $(G, A_{i-1}, n^{1-i/k}/\log n)$.

Note that $\hat{A} \subseteq A_i$. In addition, notice that for every two sets S_1 and S_2 such that $S_1 \subseteq S_2$: $B_{S_2}(v) \subseteq B_{S_1}(v)$. In particular, $B_{A_i}(v) \subseteq B_{\hat{A}}(v)$. We thus have by Lemma A.1, $E[B(v) \cap A_{i-1}] = E[B_{A_i}(v) \cap A_{i-1}] \leq E[B_{\hat{A}}(v) \cap A_{i-1}] \leq |A_{i-1}|/(n^{1-i/k}/\log n) = O(n^{1/k} \log n)$.

B. WEIGHTED GRAPHS

Let us explain the modifications needed to handle weighted graphs. The only place in the proof where we use the fact that the graph is unweighted is in the construction of the search mechanism. Recall that in our search mechanism, the set of intervals of the children of some node $v \in V(T)$ is stored in one of the children $\mathbf{helper}(v)$ of v (in the case where the degree is smaller than $n^{1/k}$). Notice that the weight of the edge $(v, \mathbf{helper}(v))$ may be large and thus may increase our stretch by a lot. The high level idea is to partition the children of every node v into $\log D$ sets such that the edges leading from v to the nodes in the same set is roughly of the same weight (up to $1+\epsilon$ factor), where D is the diameter of the graph. Each such set is handled separately, where the keys assign to each set are consecutive, namely belong to one continuous interval. The node v stores the intervals of these $\log D$ sets and thus in the search process, the node v knows in each interval out of the $\log D$ intervals to search for the key. This increases the length of the path obtained by the search algorithm by at most a factor of $1+\epsilon$ and the size of the routing tables by at most a factor of $\log D$.

C. THE SEARCH MECHANISM

In this section we prove Lemma 5.1.

Our search mechanism is operated on given tree T, and set of core nodes $\mathbf{core}(T) \subseteq V(T)$. Every node v in $V(T)$ has a unique key $\mathbf{key}(v)$ in $[1..n]$. The set $\mathbf{core}(T)$ satisfies the following property. For every $v \in T(v)$, $|\mathbf{core}(T) \cap \mathbf{childs}(v,T)| \geq \log n \lceil |\mathbf{childs}(v,T)|/n^{1/k} \rceil$. In addition, the root $r(T)$ of the tree belongs to $\mathbf{core}(T)$.

The goal is to design a search mechanism such that later in the routing phase it would be possible to find the label $L(v,T)$ given $\mathbf{key}(v)$ from any node in T by traveling on a "short" path. Our search scheme stores at every node $v \in \mathbf{core}(T)$ at most $O(s(v,T) \log n)$ data where $s(v,T) = \min\{\mathbf{deg}(\mathbf{parent}(v,T)), n^{1/k}\}$ and at most $O(\log n)$ data at the rest of the nodes.

Let $\mathbf{ST}(v,T)$ for some node $v \in V(T)$ be the data stored at the node v by our search mechanism. We then show that using the data stored at the nodes of the tree, it is possible to find the label of a given key (or decide that this key is not in $T(v)$) from the root $r(T)$ of T in a distributed manner on path of length at most $3\mathbf{radius}(T) + 2(\mathbf{ind}(T) - 1)$, where $\mathbf{ind}(T)$ is the minimal index such that $|V(T)| \leq n^{\mathbf{ind}(T)/k}$.

Let us now describe our search algorithm. Let $\mathcal{K} = \{\mathbf{key}(v) \mid v \in V(T)\}$. First the algorithm distributes the keys $\mathcal{K}$ such that every node stores the matching labels of $O(1)$ keys as follows. For a node $v \in T$, let $T[v]$ be the subtree of v in T.

The algorithm assigns each node v an interval $I(v) = [n_1, n_2]$, the subtree $T[v]$ contains the labels of the keys in $\mathcal{K}$ in the range $[n_1, n_2]$. The root of the tree $r(T)$ corresponds to the interval $[1..n]$.

Order the children z of $r(T)$ by $|T[z]|$ in non-increasing order. Let $z_1, ..., z_\ell$ be the children of $r(T)$ by that order. The algorithm stores at $r(T)$ the first key and its matching label (the key with the smallest ID). The algorithm then assigns to z_1 the next $|T[z_1]|$ keys and to z_2 the next $|T[z_2]|$ keys and so on. Let n_i^1 be the smallest key assigned to z_i and n_i^2 be the largest key assigned to z_i, the interval of z_i is $[n_i^1, n_i^2]$. The algorithm continues with this process recursively until all keys are assigned.

For a node $v \in V(T)$, let $I(v)$ be its corresponding interval. Consider a consecutive set of children $S = \{z_{j_1} ... z_{j_2}\}$. Let $I(z_{j_1}) = [n_1^l, n_1^h]$ and $I(z_{j_2}) = [n_2^l, n_2^h]$. Let $I[S] = [n_1^l, n_2^h]$. Note that all keys in $[n_1^l, n_2^h]$ are stored at one of nodes in S.

We now enhance the nodes with additional information that will enable them to find the right label of a given key later in the routing process with the desired stretch.

Ideally we would like to store in each node the intervals of its children. If we could that, later in the routing process ev-

ery node in the tree knows exactly to which child to forward the message in order to find the desired key. This way the key could be found from the root $r(T)$ by traveling over a path of at most $\mathbf{radius}(T)$ length. However this could result in storing too much information for some nodes. Therefore, in order to store at the nodes the desired amount of information, we use the children of the nodes to store some of this information.

Consider a node $v \in V(T)$. Let $w_1, ..., w_\ell$ be the set of children of v. Let $\mathbf{core}(v) = \mathbf{core}(T) \cap \mathbf{childs}(v, T)$. Let $\hat{I}(v)$ be the set of intervals of v's children together with the port leading from v to the child containing the relevant interval. If $|\mathbf{childs}(v, T)| \leq n^{1/k}$ then pick one node $\mathbf{helper}(v)$ in $\mathbf{core}(v)$ and store $\hat{I}(v)$ at $\mathbf{helper}(v)$. The node v stores the port leading to $\mathbf{helper}(v)$.

If $|\mathbf{childs}(v, T)| > n^{1/k}$ then we cannot store $\hat{I}(v)$ at a single node without violating the requirement that every node stores $\tilde{O}(n^{1/k})$ data.

Hence in this case we use techniques from [1]. It was shown in [1] that it is possible to store $O(n^{1/k})$ data at $\lceil |\mathbf{childs}(v, T)|/n^{1/k} \rceil$ children of v (in our case in $\mathbf{core}(T)$) such that it is possible to find the right child z of v such that $\mathbf{key} \in I(v)$ while traveling to at most j children of v and in addition, $|T[z]| \leq |T[v]|/n^{(j-1)/k}$. The general idea is to partition the children w_j of v into $n^{1/k}$ sets and pick $n^{1/k}$ children of v in $\mathbf{core}(T)$ and assign each such child with one of the sets. Each child continue partitioning its sets and pick other helper children of v that would be responsible for these sets, this process continue until the sets are of size at most $n^{1/k}$ and then the node just store the relevant intervals and matching children of v. This process give preference to nodes with larger subtrees. This increases the total length traveled by the search mechanism by at most $(\mathbf{ind}(T) - 1)$, since each time the algorithm traveled to j children (instead of 1 in the case where $\deg(v, T) \leq n^{1/k}$) of v in order to find the child z such that $\mathbf{key} \in I(z)$, the size of subtree of z decreases by at least $n^{(j-1)/k}$. Thus, our searching process might traveled to at most $\mathbf{ind}(T) - 1$ extra children and thus the total path traveled by the algorithm increases by $2(\mathbf{ind}(T) - 1)$ (the factor of 2 comes from the fact that the algorithm needs to travel to the child and back). For more details about this process we deferred the reader to [1] or to the full version of this paper.

This completes the construction of the search tree. Let us now turn to the searching phase. In the searching phase, the root $r(T)$ wants to find the label of some key or to discover that this key does not exist in $\mathcal{K}$. The algorithm checks if the label of the key is stored in $r(T)$. If not, the algorithm seek the child w of $r(T)$ such that $key \in I(w)$. The node $r(T)$ checks if it contains a reference to interval that contain key, if so travel to the relevant child.

Otherwise (in the case where $\deg(r(T), T) > n^{1/k}$), the algorithm finds the child z of $r(T)$ such that $\mathbf{key} \in I(z)$ using the method presented in [1]. This process continues recursively until finding the node u that contain $L(\mathbf{key}, T)$ or decides that $\mathbf{key} \notin V(T)$.

LEMMA C.1. *The path traversed by the algorithm until finding $L(t, T)$ or deciding that $t \notin V(T)$ is $3\mathbf{radius}(T) + 2(\mathbf{ind}(T) - 1)$.*

Proof: Consider first the case where the degree of all nodes in T is $O(n^1/k)$. In that case notice that in each step the algorithm travel to the to one child before traveling to the child z such that $\mathbf{key} \in I(z)$. Therefore the path traversed by the algorithm is at most $3\mathbf{radius}(T_1)$ (for every edge $e = (u, v)$ the algorithm traveled to a child z of u, then traveled back to u and continues to v, hence for each edge the algorithm traveled a path of 3). In the case where we have larger degrees recall that we use the method of [1] and as mentioned above the total increase of the length traveled by the search algorithm is $2(\mathbf{ind}(T) - 1)$. ∎