Locality and Efficiency in Distributed Computing

Thesis for the degree of

DOCTOR of PHILOSOPHY

by

Ittai Abraham

SUBMITTED TO THE SENATE OF
THE HEBREW UNIVERSITY OF JERUSALEM

April 2009

This work was carried out under the supervision of
Prof. Danny Dolev and Prof. Dahlia Malkhi

Acknowledgements

I have been blessed by having two amazing advisors, Danny Dolev and Dahlia Malkhi. I am forever grateful for the tremendous amount of time, energy and support that both have spent during my six years as a phd student. I cannot even begin to explain how much I have learned from them as a researcher and as a person. In many crossroads of my research it was Danny's wise advice or Dahlia's insightful views that allowed me to overcome difficulties and continue.

Cyril Gavoille introduced me to the world of compact routing. Cyril has been a great source of knowledge and a wonderful co-author. I would also like to thank Baruch Awerbuch, Yossi Azar, Yair Bartal, Elan Pavlov, Noam Nisan, Mikkel Thorup, Oren Dobzinski, Ankur Badola, Danny Bickson, Sharad Maloo, Saar Ron, David Peleg and Nati Linial who either co-authored a compact routing paper with me or who have significantly impacted my understanding of the topic.

This thesis is centered around results that were developed mostly during the first three years of my phd. In the remaining years I continued to work with Dahlia on Compact Routing Schemes, with Danny Dolev and Joe Halpern on Game Theory and Distributed Computing, and with Yair Bartal and Ofer Neiman on Metric Space Embedding. I would like to thank all my co-authors. I have learnt tremendously from each one of them.

Finally I would like to thank my father, Uri for teaching me that math is simple when viewed in the right way and my wife, Mor for reminding me that there are more important things in life.

Abstract

A prominent challenge is improving the *efficiency* of distributed computation: reducing the cost, load, and resource requirements while still maintaining the same quality of service. Due to the shear amount of information and the size of the network, executing global system-wide operations becomes very expensive. When possible, we wish to minimize the work done by the system and avoid the heavy cost incurred by global operations.

One of the main tools to achieve efficient distributed computation is to exploit the issue of *locality*. Specifically, some tasks require the participation of a small region of the network. In such a case one would want the computation to span only the minimal required region.

Routing is a prominent example of a task that can dramatically benefit from exploiting locality. If the source and destination nodes are close by, routing should involve only the small region that contains the source and destination.

One of the fundamental trade-offs in compact routing schemes is between the *space* used to store the routing table on each node and the *stretch* factor of the routing scheme – the maximum ratio over all pairs between the cost of the route induced by the scheme and the cost of a minimum cost path between the same pair.

A *name-independent routing scheme* is a distributed algorithm that allows any source node to route messages to any destination node, given the destination's network identifier.

This thesis suggests new ways in which distributed routing schemes can exploit locality in an efficient manner by proposing new and improved *space-stretch tradeoffs* for compact name-independent routing schemes.

Contents

Chapter 1

Introduction

In the past 30 years, there has been tremendous growth in the use of computer systems and computer networks. As the demands from computers become more complex and wide reaching, there is an inherent need to distribute the system over many processors. Today, most large scale systems are to some extent distributed. The Internet as a whole can be viewed as a distributed system.

The traditional Client-Server paradigm is well suited to today's Internet. As the computational power and bandwidth of computers increases, I envision tomorrow's Internet as a meeting place in which computational entities can interact. Unlike the passive browsers of today, tomorrow's computational entities will have an active network presence. These entities will act both as client and server, and will perform complex proactive tasks. In the last ten years there has been enormous interest by the research community in symmetric distributed systems that allow participants to actively share information among the participating peers. This interest was partly due to the phenomenal popularity of Peer-to-Peer file sharing applications like Napster and BitTorrent.

I have a strong belief that in the future, distributed and non-centralized systems will have an ever increasing role in all aspects of mankind. As these distributed systems evolve, the complexity and the amount of information stored in them will grow exponentially. Managing the tremendous amount of information stored on these systems raises many fundamental challenges.

A prominent challenge is improving the *efficiency* of distributed computation: reducing the cost, load, and resource requirements while still maintaining the same quality of service. Due to the shear amount of information and the size of the network, executing global system-wide operations becomes very expensive. When possible, we wish to minimize the work done by the system and avoid the heavy cost incurred by global operations.

One of the main tools to achieve efficient distributed computation is to exploit the issue of *locality*. Specifically, some tasks require the participation of a small region of the network. In such a case one would want the computation to span only the minimal required region.

Routing is a prominent example of a task that can dramatically benefit from exploiting

locality. If the source and destination nodes are close by, routing should involve only the small region that contains the source and destination.

The *locality* of a routing scheme can be captured by its *stretch* - the multiplicative overhead relative to the optimal route, and the *efficiently* of a routing scheme can be captured by its *memory* - the number of bits stored for the routing scheme on each node. This thesis suggests new ways in which distributed routing schemes can exploit locality in an efficient manner by proposing new and improved *space-stretch tradeoffs*.

Consider an n-node weighted undirected graph $G = (V, E, \omega)$. Each node in V is given an arbitrary unique name with $O(\log n)$ bits. In addition, for each node $u \in V$, each out-going edge is given an arbitrary unique port name in $\{1, \ldots, deg(u)\}$.

A *routing scheme* is a distributed algorithm that, given a destination node's name, allows any node to route messages that will eventually arrive at the destination node. Specifically, a routing scheme can be viewed as a function on each node that maps from a given header and incoming port number to an outgoing port number and new message header. For example, the trivial solution to routing on minimum cost paths is for each node to store for each of the possible $(n-1)$ destinations, a port number leading to the next node on a minimum cost path to the destination. This solution requires each node to store $\Omega(n \log n)$ bits of routing information and thus does not scale well as the number of nodes in the system increases.

In order to reduce memory overhead and incur routing costs that are proportional to the actual distances between interacting parties, there are two parameters that routing schemes aim to minimize:

- *Stretch*: the maximum ratio over all source-destination pairs between the cost of the path taken by the routing scheme and the cost of a minimum cost path for the same source-destination pair.
- *Memory*: the maximum over all nodes of the number of bits stored for the routing scheme.

Routing schemes which require nodes to store a linear number of bits are not scalable. The challenge is to construct in polynomial time a *compact routing scheme* that minimizes the stretch bound for any weighted graph while requiring only $o(n)$ bits of routing information per node. We refer the reader to Peleg's book [Pel00] and to the surveys of Gavoille and Peleg [Gav01, GP03] for comprehensive background on compact routing schemes.

The problem of devising compact routing schemes has two basic variants: *labeled routing* and *name-independent routing*. Awerbuch et al. [ABNLP89] were the first to distinguish between solutions that allow/disallow the designer to choose destination labels for nodes as part of the solution. The variant that allows the designer to name nodes with arbitrary destination labels is called *labeled routing*. A packet carries the chosen label of the destination in its header, and we can use the label to code topological information useful for routing. The power to choose destination labels of polylogarithmic size tends to make the routing much easier.

The variant that does not allow labeling of nodes in this way is called *name-independent routing*. In this variant node destination names are given as part of the input, e.g., these could be standard IP addresses. Generally this makes routing harder: Intuitively, the routing algorithm must first discover information about the location of the target, and only then route to it.

This thesis is composed of five papers that all deal with compact routing space-stretch tradeoffs for name-independent routing schemes.

1.1 Compact Name-Independent Routing with Minimum Stretch

This is a joint work with Cyril Gavoille, Dahlia Malkhi, Noam Nisan and Mikkel Thorup [AGM+04b] that appeared in SPAA 04 [AGM+04b] and TALG 08 [AGM+08]. Given a weighted undirected network with arbitrary node names, we present a compact routing scheme, using a $\widetilde{O}(\sqrt{n}\,)$ space routing table at each node, and routing along paths of stretch 3, that is, at most thrice as long as the shortest paths. This is optimal in a very strong sense. It is known that no compact routing using $o(n)$ space per node can route with stretch below 3. Also, it is known that any stretch below 5 requires $\Omega(\sqrt{n}\,)$ space per node.

1.2 Name-Independent Routing with Improved Communication-Space Trade-Off

This is joint work with Cyril Gavoille and Dahlia Malkhi [AGM04a] that appeared in DISC 04. Given a weighted undirected network with arbitrary node names, we present a family of routing schemes characterized by an integral parameter $\kappa \geq 1$. The scheme uses $\widetilde{O}(n^{1/\kappa} \log D)$ space routing table at each node, and routes along paths of linear stretch $O(\kappa)$, where D is the normalized diameter of the network. When D is polynomial in n, the scheme has asymptotically optimal stretch factor. With the same memory bound, the best previous results obtained stretch $O(\kappa^2)$.

1.3 Name-Independent Routing for Growth Bounded Network

This is joint work with Dahlia Malkhi [AM05] that appeared in SPAA 05. A weighted undirected network is Δ growth-bounded if the number of nodes at distance $2r$ around any given node is at most Δ times the number of nodes at distance r around the node. Given a weighted undirected network with arbitrary node names and $\epsilon > 0$, we present a

routing scheme that routes along paths of stretch $1+\epsilon$ and uses with high probability only $O(\frac{1}{\epsilon}^{O(\log \Delta)} \log^5 n)$ bit routing tables per node.

1.4 Scale-Free Name-Independent Routing

This is joint work with Cyril Gavoille and Dahlia Malkhi [AGM06] that appeared in SPAA 06. All previous routing schemes required storage that is dependent on the diameter of the network (its aspect ratio). We present a new *scale-free* routing scheme, whose storage and header sizes are independent of the aspect ratio of the network. Our scheme is based on a decomposition into sparse and dense neighborhoods. Given an undirected network with arbitrary weights and n arbitrary node names, for any integer $k \geq 1$ we present the first scale-free routing scheme with asymptotically optimal space-stretch trade-off that does *not* require edge weights to be polynomially bounded. The scheme uses $\widetilde{O}(n^{1/k})$ space routing table at each node, and routes along paths of asymptotically optimal linear stretch $O(k)$.

1.5 LLS : Name-Independent Routing for Mobile Ad Hoc Networks

This is joint work with Danny Dolev and Dahlia Malkhi [ADM04] that appeared in DIALM-POMC 04. Coping with mobility and dynamism is one of the biggest challenges in ad hoc networks. An essential requirement for such networks is a service that can establish communication sessions between mobile nodes whose location is unknown. A *location service* for ad hoc networks is a distributed algorithm that allows any source node s to know the location of any destination node t, simply by knowing t's network identifier.

A location service has a *locality aware lookup* algorithm if the cost of locating destination t from source s is proportional to the cost of the minimal cost path between s and t. A location service has a *locality aware publish* algorithm if the cost of updating the location service due to a node moving from x to y is proportional to the distance between x and y.

We present LLS, the first location service for the Unit Disk Graph model whose lookup and publish algorithms have worst case locality guarantees and average case locality awareness efficiency for any source destination pair.

Chapter 2

Compact Name-Independent Routing with Minimum Stretch

2.1 Introduction

Consider an n-node weighted undirected graph $G = (V, E, \omega)$. Each node in V is given an arbitrary unique name with $O(\log n)$ bits. In addition, for each node $u \in V$, each out going edge is given an arbitrary unique port name in $\{1, \ldots, \deg(u)\}$.

A *routing scheme* is a distributed algorithm that, given a destination node's name, allows any node to route messages that will eventually arrive at the destination node. Specifically, a routing scheme can be viewed as a function on each node that maps from a given header and incoming port number to an outgoing port number and new message header. For example, the trivial solution to routing on minimum cost paths is for each node to store for each of the possible $(n-1)$ destinations, a port number leading the next node on a minimum cost path to the destination. This solution requires each node to store $\Omega(n \log n)$ bits of routing information and thus does not scale well as the number of nodes in the system increases.

In order to reduce memory overhead and incur routing costs that are proportional to the actual distances between interacting parties, there are two parameters that routing schemes aim to minimize:

- *Stretch*: the maximum ratio over all source-destination pairs between the cost of the path taken by the routing scheme and the cost of a minimum cost path for the same source-destination pair.
- *Memory*: the maximum over all nodes of the number of bits stored for the routing scheme.

Routing schemes which require nodes to store a linear number of bits are not scalable. The challenge is to construct in polynomial time a *compact routing scheme* that minimizes the stretch bound for any weighted graph while requiring only $o(n)$ bits of routing information

per node. We refer the reader to Peleg's book [Pel00] and to the surveys of Gavoille and Peleg [Gav01, GP03] for comprehensive background on compact routing schemes.

Gavoille and Gengler [GG01] show that compact routing schemes must have stretch of at least 3. Specifically they prove that there exist n node networks in which any scheme with stretch less than 3 requires a total of $\Omega(n^2)$ bits of routing information. Thorup and Zwick [TZ05] show that any scheme with stretch less than 5 must have networks which require $\Omega(n^{3/2})$ bits of routing information. These lower bounds imply that compact routing schemes must have at least stretch 3 and that stretch 3 routing schemes require at least $\Omega(\sqrt{n})$ bits per node.

The problem of devising compact routing schemes has two basic variants: *labeled routing* and *name-independent routing*. Awerbuch et al. [ABNLP89] were the first to distinguish between solutions that allow/disallow the designer to choose destination labels for nodes as part of the solution. The variant that allows the designer to name nodes with arbitrary destination labels is called *labeled routing*. A packet carries the chosen label of the destination in its header, and we can use the label to code topological information useful for routing. The power to choose destination labels of polylogarithmic size tends to make the routing much easier.

The variant that does not allow labeling of nodes in this way is called *name-independent routing*. In this variant node destination names are given as part of the input, e.g., these could be standard IP addresses. Generally this makes routing harder: Intuitively, the routing algorithm must first discover information about the location of the target, and only then route to it.

Indeed, optimal stretch compact routing schemes for labeled routing are already known. Eilam et al. [EGP03] present a stretch 5 labeled scheme with $\widetilde{O}(n^{1/2})$ memory[1], whereas Cowen [Cow99, Cow01] presents a stretch 3 labeled scheme with $\widetilde{O}(n^{2/3})$ memory. Later, Thorup and Zwick [TZ01b] achieve the optimal stretch of 3 using only $\widetilde{O}(n^{1/2})$ bits. They also give a generalization of their scheme and using techniques from their distance oracles [TZ01a, TZ05], obtaining labeled schemes with stretch $4k-5$ (and $2k-1$ with handshaking) using $\widetilde{O}(n^{1/k})$-bit routing tables. Additionally, there exist various labeled routing schemes suitable only for certain restricted forms of graphs. For example, routing in a tree is explored, e.g., in [FG01, TZ01b], achieving optimal routing. This routing requires $O(\log^2 n/\log\log n)$ bits for local tables and for headers, and this is tight [FG02].

As for name-independent routing, the situation is quite different. Initial results in [ABNLP90] provide stretch 3 non-compact name-independent routing with $\widetilde{O}(n^{3/2})$ total memory. However this scheme is unbalanced, $\Omega(\sqrt{n})$ nodes must store $\Omega(n)$ bits of routing information. Awerbuch and Peleg [AP90] were the first to show that constant-stretch is possible to achieve with $o(n)$ memory per node, albeit with a large constant. Arias et al. [ACL+03] significantly reduce the stretch to 5 with $\widetilde{O}(\sqrt{n})$ memory per node. This chapter closes the gap between these results and the known lower bound of stretch 3.

[1]We use the notation $\widetilde{O}(f(n)) = f(n)(logn)^{O(1)}$ to hide poly-logarithmic factors.

2.1.1 Our results

We present the first optimal compact name-independent routing scheme for arbitrary undirected graphs. The scheme has stretch 3, and requires $O(\log^2 / \log\log n)$-bit headers and $\widetilde{O}(\sqrt{n})$ bits of routing information per node. When routing along our stretch 3 paths, each routing decision is performed in constant time. Given the graph and the node names, we can construct the routing information in $\widetilde{O}(n|E|)$ time.

Besides improving the stretch of Arias et al. [ACL+03] from 5 to 3, our results answer affirmatively the challenge of optimal name-independent routing that was open since the initial statement of the problem in 1989 [ABNLP90]. Surprisingly, our results show that with $\widetilde{O}(\sqrt{n})$ bits of routing information per node, allowing the designer to label the nodes does **not** improve the stretch factor compared to the task when node labels are predetermined by an adversary.

We note that our solution does not contain any strikingly new technique. Rather our new scheme is a non-trivial combination of simple standard techniques.

2.2 Preliminaries

Consider a set V of n nodes wishing to participate in a distributed routing scheme. We assume the nodes are labeled with an arbitrary unique identifier that can be represented by $O(\log n)$ bits.

We assume a graph $G = (V, E, \omega)$ with positive edge cost ω. For $u, v \in V$, let $d(u, v)$ denote the cost of a minimum cost path from u to v in G, where the cost of a path is the sum of the weights along its edges.

Each node has, for each outgoing edge, a unique port name from the set of integers $\{1, \ldots, n\}$. We assume the fixed-port model [FG01]. In this model the name of each outgoing edge is fixed by the adversary. Thus the name of the outgoing edge may have no connection to the label of the node on the other side of the edge.

We assume that initially the sender only knows the name of the destination node. This destination is written in the header of the message. We require *writable packet headers*, namely, we allow the routing algorithm to write a reasonable amount of information into the headers of messages as they are routed. In our case, we use $O(\log^2 n / \log\log n)$-bit headers.

We note that our use of headers aims at useful tradeoffs between current techniques used in the real world: source-directed routing, where the source puts the whole path to the destination in the header, and routing with a fixed header, where each router knows how to forward packets to any destination. For source-directed routing, the header may be very large, and for routing with a fixed header, the routing tables may become huge. In either case, we have problems with scaling. Our point here is that writing a small amount of information in the header can dramatically reduce the amount of information needed at the routers.

It is no coincidence that our scheme and indeed all previous name-independent schemes use writable packet headers. A scheme that does not rewrite packet headers must be loop free and thus must have stretch 1 on any tree. Clearly in a tree that is a star the center would have to code a permutation using $\Omega(n \log n)$ bits on the average.

Lemma 2.2.1. *There do not exist loop-free name-independent routing schemes with $o(n)$ bits for each node on every graph.*

As for lower bounds for compact routing, note that for the related problem of labeled routing, the work of [GG01] shows that any stretch < 3 scheme must use a total of $\Omega(n^2)$ bits. Thus it cannot be the case that all nodes use $o(n)$ bits. Clearly this bound holds also for the name-independent model.

Actually, a slightly stronger memory bound of $\Omega(n^2 \log n)$ bits for stretch < 3 can be proven for the name-independent model. This is derived by examining the complete bipartite graph $K_{n/2,n/2}$ with uniform weights (likewise, the metric space it induces). For stretch < 3, each node must route optimally to its distance one neighbors. By counting all the permutations on names it is clear that each node must use $\Omega(n \log n)$ bits.

Lemma 2.2.2. *Any name-independent routing scheme with $o(n \log n)$ bits per node must have stretch at least 3.*

2.3 The Stretch 3 Scheme

In Sections 2.3.1, 2.3.2, 2.3.3, 2.3.4, 2.3.5, 2.3.6, we will first present some simple ingredients for our optimal stretch 3 scheme. Then, in Section 2.3.8, we will combine them in our optimal solution. Finally, in Section 2.3.9, we prove that the scheme has the optimal stretch of 3.

2.3.1 Vicinity balls

For every integer $\kappa \geq 1$, and for a node $u \in V$, let the *vicinity* of u, denoted by $B_\kappa(u)$, be the set consisting of u and the κ closest nodes to u, breaking ties by lexicographical order of node names. Vicinities satisfy the following Monotonicity Property:

Property 2.3.1. [ABNLP90] *If $v \in B_\kappa(u)$ and w is on a minimum cost path from u to v, then $v \in B_\kappa(w)$.*

Proof. Seeking a contradiction, suppose $v \notin B_\kappa(w)$. For any $z \in B_\kappa(w)$ we have $d(w,z) < d(w,v)$ (or $d(w,z) = d(w,v)$ and $z < v$). So $d(u,z) < d(u,w) + d(w,v)$ (or $d(u,z) = d(u,w) + d(w,v)$ and $z < v$). Since w is on a minimum cost path from u to v, $d(u,z) < d(u,v)$ (or $d(u,z) = d(u,v)$ and $z < v$), hence $B_\kappa(w) \subseteq B_\kappa(u)$. But since $v \in B_\kappa(u) \setminus B_\kappa(w)$ we have $|B_\kappa(w)| < |B_\kappa(u)|$ a contradiction. □

Hereafter, the size of the vicinities is set to $\kappa = \lceil 4\sqrt{n}\log n \rceil$ and denote simply by $B(u) = B_\kappa(u)$. Let $b(u)$ denote the radius of $B(u)$, $b(u) = \max_{w \in B(u)} d(u,w)$.

As in previous compact routing schemes (see, e.g., [ABNLP89, Cow01]), each node u will know its vicinity $B(u)$. We assume that u has a standard dictionary over the names in $B(u)$ so that in constant time it can check membership and look up associated information.

2.3.2 Coloring

Our construction uses a partition of nodes into sets $C_1, \ldots, C_{\sqrt{n}}$, called *color-sets*, with the following two properties:

Property 2.3.2.

1. *Every color-set has at most* $2\sqrt{n}$ *nodes.*
2. *Every node has in its vicinity at least one node from every color-set.*

From here on, if node $u \in C_i$ we say that it has "color i", and denote $c(u) = i$. By standard Chernoff bounds and a union bound Property 2.3.2 clearly holds with high probability if every node independently chooses a random color. Constructing a polynomial-time coloring satisfying Property 2.3.2 is discussed in Section 2.4.

2.3.3 Hashing names to colors

We shall assume a mapping h from node names to colors which is balanced in the sense that at most $O(\sqrt{n})$ names map to the same color. Each node u should be able to compute $h(w)$ for any destination w. If the names were a permutation of $\{1, \ldots, n\}$, we could just extract $\frac{1}{2}\log n$ bits from the name, but we want to deal with arbitrary names such as IP addresses. Arias et al. [ACL+03] use a $(\log n)$-universal hash function for a similar purpose. In Section 2.5 we will present a function h that can be computed in constant time.

2.3.4 Stretch 3 for complete graphs

To illustrate the use of these first three ingredients, we here observe a very simple stretch 3 scheme with $\widetilde{O}(\sqrt{n}\,)$ bits per node given a complete graph whose edge weights satisfy the triangle inequality.

Every node u stores the following:

1. The names of all the nodes in the vicinity $B(u)$ and what port number to use to reach them.
2. The names of all the nodes v such that $c(u) = h(v)$ and what port number to use to reach them.

Routing from u to v is done in the following manner:

1. If $v \in B(u)$ or $c(u) = h(v)$ then u routes directly to v with stretch 1.
2. Otherwise, u forwards the packet to $w \in B(u)$ such that $c(w) = h(v)$. Then from w the packet goes directly to v. The stretch is at most 3 since $d(u,w) + d(w,v) \leq d(u,v) + 2d(u,v)$.

Note that in a general graph the main difficulty is in implementing the path from w to v.

2.3.5 Routing on trees

We make use of the following labeled routing scheme for trees:

Lemma 2.3.3. [FG01, TZ01b] *For every weighted tree T with n nodes, in the fixed-port model, there exists a labeled routing scheme that, given any destination label, routes optimally on T from any source to the destination. The storage per node in T, the label size, and the header size are $O(\log^2 n / \log\log n)$ bits. Given the stored information of a node and the label of the destination, routing decisions take constant time.*

For a tree T containing a node v, let $\mu(T,v)$ denote the routing information stored at node v and $\lambda(T,v)$ denote the destination label of v in T as defined by the labeled routing scheme of Lemma 3.3.4.

We shall apply the scheme Lemma 3.3.4 to several trees in the graph. Each node will participate in $\widetilde{O}(\sqrt{n})$ trees, so its total tree routing information will be of size $\widetilde{O}(\sqrt{n})$.

2.3.6 Landmarks

Designate one color to be special and call it the landmark color. Let L denote the set of nodes with the chosen color. By Property 2.3.2 we have $|L| \leq 2\sqrt{n}$ and for every $v \in V$, $B(v) \cap L \neq \emptyset$. For a node $v \in V$, let ℓ_v denote the closest landmark node in $B(v)$ (breaking ties by lexicographical order of node names).

2.3.7 Partial shortest path trees

For any node u let $T(u)$ denote a single-source minimum-cost-path tree rooted at u. In a partial shortest path tree, every node v maintains $\mu(T(u),v)$ if and only if $u \in B(v)$. Notice that the set of nodes that maintain $\mu(T(u),\cdot)$ is a subtree of $T(u)$ that contains u.

Lemma 2.3.4. *If $x \in B(y)$ then given the label $\lambda(T(x),y)$, node x can route to node y along a minimum cost path.*

Proof. By Property 2.3.1 for any node w on the minimum cost path of $T(x)$ between x and y we have $x \in B(w)$. Thus every node w on this path maintains $\mu(T(x),w)$. □

2.3.8 The stretch 3 scheme

Every node u stores the following:

1. For every $w \in B(u)$, the name w and the port name $u \to y$, where $(u \to y)$ is the port number to use to get to node y which is the next hop on a minimum cost path from x to w.
2. For every landmark node $\ell \in L$, routing information $\mu(T(\ell), u)$ and label $\lambda(T(\ell), \ell)$ of the tree $T(\ell)$.
3. For every node $x \in B(u)$, routing information $\mu(T(x), u)$ of the tree $T(x)$.
4. For every node v such that $c(u) = h(v)$, store one of the following two options that produces the minimum cost path out of the two:
   (a) Store the labels $\langle\, \lambda(T(\ell_v), \ell_v),\ \lambda(T(\ell_v), v)\,\rangle$. The routing path in this case would be from u to $\ell_v \in B(v)$ using $\lambda(T(\ell_v), \ell_v)$ on the tree $T(\ell_v)$, and from ℓ_v to v using $\lambda(T(\ell_v), v)$ on the same tree $T(\ell_v)$.
   (b) Let $P(u, w, v)$ be a path from u to v composed of a minimum cost path from u to w, and of a minimum cost path from w to v with the following properties: $u \in B(w)$, and there exists an edge (x, y) along the minimum cost path from w to v such that $x \in B(w)$ and $y \in B(v)$. If such paths exists, choose the lowest cost path $P(u, w, v)$ among all these paths and store the labels $\langle\, \lambda(T(u), w),\ x,\ (x \to y),\ \lambda(T(y), v)\,\rangle$.
   The routing path in this case would be from u to w on $T(u)$ using $\lambda(T(u), w)$. This part is possible by Lemma 2.3.4 on $u \in B(w)$. Then from w to y since $x \in B(w)$ and the port number $(x \to y)$ is stored. Finally from y to v on $T(y)$ using $\lambda(T(y), v)$. This part is possible by Lemma 2.3.4 on $y \in B(v)$.

Routing from u to v is done in the following manner:

1. If $v \in B(u)$ or $v \in L$ (v is a landmark node) or $c(u) = h(v)$ then u routes to v using its own information.
2. Otherwise, u forwards the packet to $w \in B(u)$ such that $c(w) = h(v)$. Then from w the packet goes to v using w's routing information.

2.3.9 Analysis

Theorem 2.3.5. *Let $s, t \in V$ be any two nodes. The route of the above scheme from s to t has stretch at most 3.*

Proof. There are three cases to consider (see Figure 2.1):

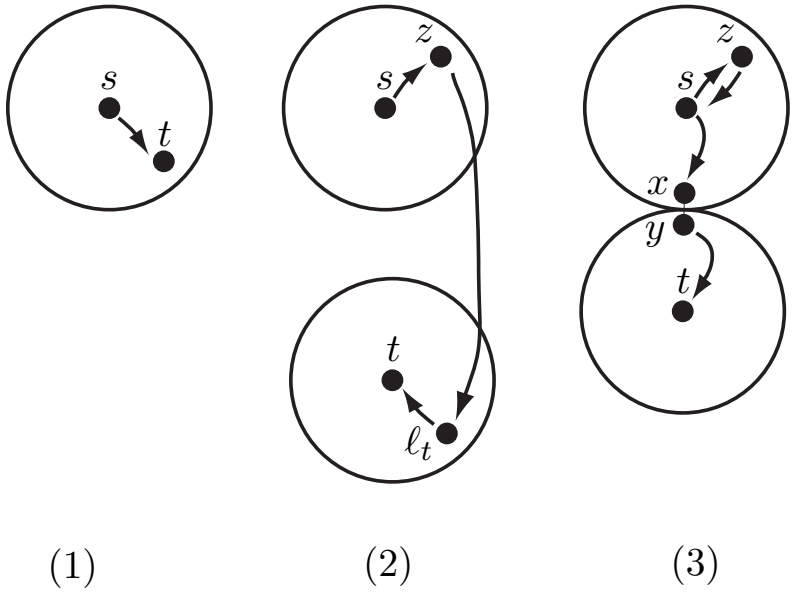

Figure 2.1: The three cases in the proof of Theorem 2.3.5: (1) target is inside the source vicinity, (2) source and target vicinities are far apart, (3) source and target vicinities are close.

1. If $t \in B(s)$ or $t \in L$ then s routes on a minimum cost path directly to t.

Otherwise, denote $d = d(s,t)$, let z be a node such that $z \in B(s)$ and $c(z) = h(t)$. For the case $c(s) = h(t)$, we set $z = s$. Let $p(z,t)$ be the cost of the path chosen by z as the lowest cost path from z to t among options 4a and 4b of Section 2.3.8.

2. On every minimum cost path from s to t there is a node y such that $y \notin B(s)$ and $y \notin B(t)$. In this case $b(s) + b(t) \leq d(s,t)$ (recall that $b(u) = \max_{w \in B(u)} d(u,w)$).

   By examining option 4a the cost $d(s,z) + p(z,t)$ of the path taken by our routing scheme is bounded by the cost of the path $s \rightsquigarrow z \rightsquigarrow \ell_t \rightsquigarrow t$, where $u \rightsquigarrow v$ denotes a minimum cost path from u to v. Thus $d(s,z) + p(z,t) \leq d(s,z) + d(z,\ell_t) + d(\ell_t,t) \leq b(s) + [b(s) + d + b(t)] + b(t) \leq 3d$.

3. There exists a minimum cost path, in which every node is in $B(s) \cup B(t)$. Let (x,y) be an edge of this path such that $x \in B(s)$ and $y \in B(t)$.

   By examining the best choice in option 4b the cost $d(s,z) + p(z,t)$ of the path taken by our routing scheme is bounded by the cost of the path $s \rightsquigarrow z \rightsquigarrow s \rightsquigarrow x \rightarrow y \rightsquigarrow t$. Thus $d(s,z) + p(z,t) \leq d(s,z) + d(z,s) + d(s,t) \leq b(s) + b(s) + d \leq 3d$.

□

2.4 On Polynomial Time Coloring

In this section, we discuss how to "derandomize" the coloring discussed in Section 2.3.2. This is done via the method of conditional probabilities using pessimistic estimators [Rag88]. Let $C = \{1, \ldots, \sqrt{n}\}$. We begin with a simple analysis of the randomized algorithm.

- For every $v \in V$ and $c \in C$ let $b_{v,c}$ be a $\{0,1\}$ random variable that equals 0 if and only if there exists $u \in B(v)$ with $c(u) = c$. Note that for any $b_{v,c}$, $E[b_{v,c}] = \Pr[b_{v,c} > 0] \leq (1 - 1/\sqrt{n})^{4\sqrt{n}\log n} \leq n^{-4}$.
- For every $c \in C$ let e_c be a $\{0,1\}$ random variable that equals 0 if and only if $|\{u \mid c(u) = c\}| \leq 2\sqrt{n}$.
- For every $c \in C$ and $v \in V$ let $f_{c,v}$ be a $\{0,1\}$ random variable that equals 1 if and only if $c(v) = c$.

Using Chernoff bounds and its analysis (see [MR95], chapter 4, theorem 4.1 and equation (4.3)) we have

$$\Pr[e_c > 0] \leq \Pr\left[\sum_{v \in V} f_{c,v} > 2\sqrt{n}\right] < \alpha E\left[e^{t\sum_{v\in V} f_{c,v}}\right] \tag{2.1}$$

for $t = \ln 2$ and $\alpha = e^{-2t\sqrt{n}}$. If follows that $\alpha E[e^{t\sum_{v\in V} f_{c,v}}]$ is a pessimistic estimator for e_c. Let $A = \sum_{v\in V, c\in C} b_{v,c} + \alpha \sum_{c\in C} e^{t\sum_{v\in V} f_{c,v}}$. We now show that the four requirements for the method of conditional probabilities with pessimistic estimators are fulfilled. When no node is colored yet, using union and Chernoff bounds, $E[A] < 1$. For any partial coloring, the expression $E[A]$ can be computed in polynomial time. Since $E[A]$ is an expectation of a random variable then any partial coloring can be extended by one more node with a color that does not increase $E[A]$. Finally, since Eqn. 2.1 is true for any partial coloring then for the full coloring $\sum b_{v,c} + \sum e_c < E[A] < 1$ as required.

Computing this derandomization can be done in $\widetilde{O}(n^2)$ time. There are n stages in the process. At each stage we are given a partial coloring that induces some value A and an uncolored node u. For each color $c \in C$ we need to compute $A[u,c]$ (the value of A induced by the partial coloring extended by coloring u with c) and choose the color with the minimum $A[u,c]$.

For a given node u, the cost of checking if $b_{v,c}$ needs to change requires scanning $|\{v \mid u \in B(v)\}|$ balls for each color. So the total is $\sqrt{n}\sum_{u\in V} |\{v \mid u \in B(v)\}| = \sqrt{n}\sum_{v\in V} |B(v)| = \widetilde{O}(n^2)$.

Computing the changes in $f_{c,v}$ from one color c' to another c'' can be done in $\widetilde{O}(\sqrt{n})$ time since for each color $c \in C$ only two terms in $e^{t\sum_{v\in V} f(c,v)}$ change. So the total cost of computing the changes in $f_{c,v}$ is $\widetilde{O}(n^2)$ as required.

2.5 On Hashing in Constant Time

In this section, we will implement a constant time hashing function h from n arbitrary names to $\sqrt{n}$ colors, as assumed in Section 2.3.3. The constant time assumes that we can perform standard arithmetic operations on names in constant time, hence that each name is stored in a constant number of words. For example, these names could be IP addresses. The representation of our hash function will take $O(\sqrt{n})$ space. We can store such a representation with each node without violating our space bounds.

Previous work of Arias et al. [ACL+03] used a $(\log n)$-universal hash function, but with current implementations via degree-$(\log n)$ polynomials, the evaluation of this function takes more than constant time. With the constant time hash function we propose, each routing decision is made in constant time.

We will now give a randomized construction of the hash function which works with high probability. For simplicity we assume $\sqrt{n}$ is a power of two so that $(\log n)/2$ is an integer. First we use a standard universal hash function h_0 mapping names into $\{1, \ldots, n^{2.5}\}$ in constant time. With high probability this mapping is collision free (see, e.g., [MR95, §8.4.1]).

Set $q = (\log n)/2$. We are now dealing with n distinct reduced names of $5q$ bits, and we want to get down to colors of q bits. We will use an idea of Tarjan and Yao [TY79]. For $i = 1, \ldots, 4$, let T_i be a random table mapping q bits into iq bits. Note that each table has $2^q = \sqrt{n}$ entries. We then hash a $(5q)$-bit reduced name x as follows.

Let $x_4 = x$, for $i = 4, \ldots, 1$, let y_i be the q least significant bits of x_i, let z_i be the remaining iq bits of x_i. Set $x_{i-1} = T_i[y_i] \oplus z_i$ (where $\oplus$ is the bit-wise xor operator). At the end, x_0 has only $q = (\log n)/2$ bits which we return as the color.

The above computation of colors from reduced names takes constant time. We will now bound the number of reduced names mapping to each color.

Lemma 2.5.1. *W.h.p., there are $O(\sqrt{n})$ reduced names mapping to each of the $\sqrt{n}$ colors.*

Proof. We start with n unique $5q$-bit reduced names. In each of 4 iterations, the names get further reduced by q bits. In this process, some names collide. We will use Chernoff and union bounds four times iteratively to show that no more than $3\sqrt{n}$ original names are reduced to each of the $\sqrt{n}$ colors.

For each $i = 4, 3, 2, 1$ and each iq-bit name x let $a_{i,x}$ be the number of original names reduced to x at the i-th iteration. Let $\alpha_5 = 1, \alpha_4 = e, \alpha_3 = 2e^2, \alpha_2 = \log n, \alpha_1 = 3\sqrt{n}$. Let $B(i) = \{0,1\}^{iq}$ be the set of all iq-bit names. Let $\mathcal{M}_i$ be the event that at most α_i original names are mapped to each iq-bit name, $\mathcal{M}_i = \{\max_{x \in B(i)} a_{i,x} \leq \alpha_i\}$. Given non-negative values $\{a_{i+1,x}\}_{x \in B(i+1)}$ such that $\sum_{x \in B(i+1)} a_{i+1,x} = n$ then for any $v \in B(i)$ we have

$$E[a_{i,v}] = \sum_{y \in B(1), z \in B(i)} \Pr[v = T_i(y) \oplus z] a_{i+1,yz} = 2^{-iq} \sum_{x \in B(i+1)} a_{i+1,x} = n2^{-iq}.$$

In addition, given $\mathcal{M}_{i+1}$ then $a_{i,v}$ is a sum of independent variables, where each such random variable is dominated by $\alpha_{i+1}Y_{v,x}$ where $Y_{v,x}$ is an independent $\{0,1\}$ Bernoulli random variable with $p_x = a_{i+1,x}2^{-iq}/\alpha_{i+1}$. Observe that $E[\sum_{x\in B(i)} Y_{v,x} \mid \mathcal{M}_{i+1}] = n^{1-i/2}/\alpha_{i+1}$

Using standard Chernoff bounds (see, e.g., [MR95, eq. (4.10)]), it follows that for each $v \in B(i)$,

$$\Pr[a_{i,v} > \alpha_i \mid \mathcal{M}_{i+1}] \leq \Pr\left[\sum_{x\in B(i)} Y_{v,x} > \frac{\alpha_i}{\alpha_{i+1}} \frac{n^{1-i/2}/\alpha_{i+1}}{n^{1-i/2}/\alpha_{i+1}} \mid \mathcal{M}_{i+1}\right] \leq \left(\frac{e}{\alpha_i n^{i/2-1}}\right)^{\alpha_i/\alpha_{i+1}}$$

Using a union bound it can be checked that for each $i = 4, 3, 2, 1$, we have

$$\Pr[\mathcal{M}_i \mid \mathcal{M}_{i+1}] \geq 1 - \sum_{v\in B(v)} \Pr[a_{i,v} > \alpha_i \mid \mathcal{M}_{i+1}] \geq 1 - n^{i/2}\left(\frac{e}{\alpha_i n^{i/2-1}}\right)^{\alpha_i/\alpha_{i+1}} \geq 1 - n^{2-e}.$$

So we conclude that $\mathcal{M}_1$ occurs w.h.p. □

Thus it follows that we expect a maximum of $O(\sqrt{n})$ reduced names to map to the same color. Moreover the mapping took constant time, and its representation took $O(\sqrt{n})$ space. This completes our randomized construction of the desired hash function. The construction uses the same ingredients as are used for the deterministic dictionaries of Hagerup et al. [HMP01]. Using the techniques from [HMP01], we can derandomize our construction to run in $O(n \log n)$ time.

2.6 Combining the Ingredients

Theorem 2.6.1. *Given a network with n nodes and m edges, in which nodes have arbitrary unique names and arbitrary port name permutations, there exists an algorithm that runs in $\widetilde{O}(mn)$ time and produces a routing scheme which requires $O(\sqrt{n}\log^3 n/\log\log n)$-bit routing tables per node and $O(\log^2 n/\log\log n)$ headers that performs routing decisions in constant time and routes along paths of stretch at most 3.*

Proof. We combine the results of the previous sections:

Running time: Since the graph is undirected and weights are positive, performing APSP takes $O(mn)$ time. Computing $B(u)$ for all $u \in V$ takes $\widetilde{O}(n^{3/2})$ time using results of [RTZ05]. Finding the coloring via derandomization takes $\widetilde{O}(n^2)$ time as shown in Section 2.4. Then building the tree routing scheme requires $\widetilde{O}(n)$ time for each of the n trees. Computing the routing scheme information requires $\widetilde{O}(n^2)$ time. For each node, computing items $1, 2, 3$ of Section 2.3.8 takes $\widetilde{O}(n)$ time. In addition, for item 4, each node v is queried by each node $u \in B(v)$ at most $\sqrt{n}$ times. Hence the number of queries is at most $\sqrt{n}\sum_{v\in V}|B(v)| = \widetilde{O}(n^2)$ and the time is $\widetilde{O}(n^2)$.

Storage: From Section 2.3.8, each node u stores: (1) $O(\sqrt{n}\log n)O(\log n)$ bits for the names of all the nodes in the vicinity $B(u)$ and what link to use to reach them, (2) $O(\sqrt{n})O(\log^2 n/\log\log n)$ bits for storing $\mu(T(\ell), u)$ of the tree $T(\ell)$ for every landmark node $\ell \in L$, (3) $O(\sqrt{n}\log n)O(\log^2 n/\log\log n)$ bits for storing $\mu(T(x), u)$ of the tree $T(x)$ for every node $x \in B(u)$, (4) $O(\sqrt{n})O(\log^2 n/\log\log n)$ for storing either $\langle\, \lambda(T(\ell_v), \ell_v),\ \lambda(T(\ell_v), v)\,\rangle$ or $\langle\, \lambda(T(u), w),\ x,\ (x \to y),\ \lambda(T(y), v)\,\rangle$ for every node v such that $c(u) = h(v)$.

Headers: Observe that messages headers always contain a constant number of port numbers, node names, and tree labels. Hence by Lemma 3.3.4 headers require $O(\log^2 n/\log\log n)$ bits. *Decision time*: Follows from Section 2.5. *Stretch:* Follows from Theorem 2.3.5. □

2.7 Conclusion

For any integer $k \geq 2$, consider schemes with $\tilde{O}(n^{1/k})$-bit routing tables. For labeled routing the best known schemes obtain stretch $4k-5$, while for name-independent routing the best known schemes obtain stretch $C \cdot k$ where C is a large constant. In this chapter we give optimal bounds on the stretch for $k = 2$. A natural open question is to obtain tighter bounds on stretch for $k > 2$. We explore this question in the chapter.

Chapter 3

Name-Independent Routing with Improved Communication-Space Trade-Off

3.1 Introduction

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

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

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

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

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

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

Our contributions. We first present in Section 3.3 a family of routing schemes parameterized by an integer $\kappa > 0$, that has the complexity measures below. The $\widetilde{O}()$ notation denotes complexity similar to $O()$ up to poly-logarithmic factors. Concrete constants are provided in the body of the chapter.

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

When D is polynomial in n, the scheme has asymptotically optimal stretch factor, as proven by [PU89]. With the same memory bound, the best previous results obtained stretch $O(\kappa^2)$ [AP90, ACL+03].

Previous results. There is a subtle distinction between a *designer port* model and a *fixed port* model. In the *fixed port* model (also known as the adversarial port model) the names of outgoing links, or ports, from each node may be arbitrarily chosen by an adversary from the set $\{1, \ldots, n\}$. In the *designer port* model they may be determined by the designer of the routing scheme. In particular, Gavoille and Gengler [GG01] indicate at least stretch-3 when each node has memory $o(n)$. For stretch-k routing scheme Peleg and Upfal [PU89] prove that a total of $\Omega(n^{1+1/(2k+4)})$ routing information bits is required. Thorup and Zwick refine this bound and show in [TZ01b] that the stretch is at least $2k+1$ when each node has memory $o(n^{1/k})$, proved for $k = 1, 2, 3, 5$ and conjectured for other values of k. For comprehensive surveys on compact routing and compact network data structures, see [Gav01, GP03].

Initial results in [ABNLP90] provide name-independent routing with $\widetilde{O}(n^{3/2})$ total memory. Awerbuch and Peleg [AP90] presented a scheme that for any k, requires $\widetilde{O}(k^2 n^{1/k} \log D)$ bits per node and routes on paths of stretch $O(k^2)$. Arias et al. [ACL+03] present a slight improvement that uses the same memory bounds but improves the constant in the $O(k^2)$ stretch by a factor of 4.

All known name-independent schemes that are "combinatorial" and do not rely on the normalized diameter, D, in their storage bound have exponential stretch factor. Awerbuch et al. [ABNLP89] achieve with $\widetilde{O}(n^{1/k})$ memory stretch $O(9^k)$, and [ACL+03] improved to stretch $O(2^k)$ with the same memory bound. For $\widetilde{O}(\sqrt{n}\,)$ memory Arias et al. provide stretch 5. Recently, Abraham et al. [AGM+04b], achieve optimal stretch 3 with $\widetilde{O}(\sqrt{n}\,)$.

A weaker variant of the routing problem, *labeled routing*, was initiated in [ABNLP89]. In this problem model, the algorithm's designer can choose the network addresses of nodes (and of course, use node names to store information about their location in the graph). This paradigm does not provide for a realistic network design, however, the tools devised for its solution have proven useful as building blocks of full routing schemes (in fact, we make use here of certain building blocks devised in the context of labeled routing schemes).

Indeed, optimal compact schemes for labeled routing are known. The first non trivial stretch-3 scheme was given by Cowen [Cow01] with $\widetilde{O}(n^{2/3})$ memory. Later, Thorup and Zwick [TZ01a, TZ01b] improved the memory bound to only $\widetilde{O}(\sqrt{n}\,)$ bits. They also gave an elegant generalization of their scheme, achieving stretch $4k-5$ (and even $2k-1$ with handshaking) using only $\widetilde{O}(n^{1/k})$ bits. Additionally, there exist various labeled routing schemes suitable only for certain restricted forms of graphs. For example, routing in a tree is explored, e.g., in [FG01, TZ01b], achieving optimal routing. It requires $\widetilde{O}(1)$ bits for local tables and $\widetilde{O}(1)$ bits for headers.

3.2 Preliminaries

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

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

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

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

3.3 Linear Communication-Space Trade-Off

Let $G = (V, E, \omega)$ be a graph, where $|V| = n$. In this section, we provide a family of name-independent routing schemes for G parameterized by κ, in which each node keeps $\widetilde{O}(n^{1/\kappa} \log D)$ storage, where D is the normalized diameter of the graph, and each route has stretch $O(\kappa)$. When D is polynomial in n, the scheme has asymptotically optimal stretch factor, as proven by [PU89].

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

3.3.1 Tree cover based on Sparse Partitions

Lemma 3.3.1. [AP90, AP92, Pel00] *For every weighted graph $G = (V, E, \omega)$, $|V| = n$ and integers $\kappa, \rho \geq 1$, there exists a polynomial algorithm that constructs a collection of rooted trees $\mathcal{TC}_{\kappa,\rho}$ such that:*

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

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

Note that property (4) is a novel property that does to appear in the tree covers of [AP90, AP92, Pel00]. However, it is crucial for our construction and its proof is a simple consequence of the manner in which the cover algorithm works: in each iteration, any cluster S added to a cover Y has $\mathrm{rad}(S) \leq \rho$. The end result is a set of covers $\mathcal{R}$ that has properties (1),(2), and (3). For every cover $Y \in \mathcal{R}$ define $r(Y)$ as the initial node that started that cover, and $G[Y]$ as the subgraph containing Y and all the edges connecting nodes in Y whose cost is at most 2ρ. $G[Y]$ spans Y because Y is formed by a connected union of clusters whose radius is at most ρ. The set $\mathcal{TC}_{\kappa,\rho}$ is defined by taking every $Y \in \mathcal{R}$ and setting $T_Y \in \mathcal{TC}_{\kappa,\rho}$ to be a minimum cost path tree spanning $G[Y]$ whose root is $r(Y)$.

W.l.o.g. assume that the minimum cost edge is 1. We define an index set $I = \{1, \ldots, \lceil \log D \rceil\}$. For all $i \in I$, we build a tree cover $\mathcal{TC}_{\kappa,2^i}$ according to Lemma 3.3.1 above. For all $v \in V$ and $i \in I$, let $\mathrm{Tree}_v[i]$ be a tree $T \in \mathcal{TC}_{\kappa,2^i}$ such that $B(v, 2^i) \subseteq T$.

3.3.2 Bounded cost name-independent tree-routing

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

Theorem 3.3.2. *For every tree* $T = (U, E, \omega)$, $|U| = m$, $U \subset V$, $|V| = n$, *and integer* κ *there exists a name-independent routing scheme on* T *with error-reporting that routes on paths of length bounded by* $4\mathrm{rad}(T) + 2\kappa\mathrm{maxE}(T)$, *each node requires* $O(\kappa n^{1/\kappa} \log^2 n)$ *memory, and headers are of length* $O(\log^2 n)$. *(And routing for a non-existent name in* T *also incurs a path of length* $4\mathrm{rad}(T) + 2\kappa\mathrm{maxE}(T)$ *until a negative result is reported back to the source.)*

The proof of Theorem 3.3.2 is deferred until Section 3.3.4.

For a tree T containing a node v, we let $\phi(T, v)$ denote the routing information of node v as required from Theorem 3.3.2.

3.3.3 The name-independent routing scheme

We now combine Theorem 3.3.2 with Lemma 3.3.1 in a manner similar to the hierarchical routing scheme of Awerbuch and Peleg [AP92].

Storage. For all $v \in V$, $i \in I$, and $T \in \mathcal{TC}_{\kappa,2^i}$ such that $v \in T$ node v stores $\phi(T, v)$. According to Lemma 3.3.1 and Theorem 3.3.2 above, the total storage of each node is $O(\kappa^2 n^{2/\kappa} \log D \log^2 n)$.

Routing. The sender s looks for destination t in the tree $\text{Tree}_s[i]$ successively for $i = 1, 2, \ldots, \lceil \log D \rceil$ using the construction in Theorem 3.3.2.

Stretch analysis. From Lemma 3.3.1 for $T \in \mathcal{TC}_{\kappa,\rho}$ we have that the cost $4\text{rad}(T) + 2\kappa\text{maxE}(T)$ is bounded by $4(2\kappa-1)\rho + 2\kappa 2\rho \leq 12\kappa\rho$. Hence, for any source s, integer $i \in I$, the cost of searching for any target t in $\text{Tree}_s[i]$ is at most $12\kappa 2^i$.

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

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

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

Theorem 3.3.3. *For every weighted graph $G = (V, E, \omega)$ whose normalized diameter is D and integer $\kappa \geq 1$, there is a polynomial time constructible name-independent routing scheme with stretch $O(\kappa)$ and memory $O(\kappa^2 n^{1/\kappa} \log D \log^2 n)$.*

In the remainder of this section, we provide the construction that proves Theorem 3.3.2 above.

3.3.4 Bounded-cost name-independent tree-routing

Consider a set V of n nodes in which every node $u \in V$ has a unique name $n(u) \in \{1, \ldots, n\}$. Let $T = (U, r, E, \omega)$ be a rooted tree with $r \in U \subseteq V$ and $|U| = m$.

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

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

First, we give v its name in the labeled tree-routing of Thorup & Zwick [TZ01b] and Fraigniaud & Gavoille [FG01]:

Lemma 3.3.4. **[FG01, TZ01b]** *For every weighted tree T with n nodes there exists a labeled routing scheme that, given any destination label, routes optimally on T from any source to the destination. The storage per node in T, the label size, and the header size are $O(\log^2 n / \log\log n)$ bits. Given the information of a node and the label of the destination, routing decisions take constant time.*

For a tree T containing a node v, we let $\mu(T, v)$ denote the routing information of node v and $\lambda(T, v)$ denote the destination label of v in T as required from Lemma 3.3.4. Thus, the first name we assign with v is $\ell(v) = \lambda(T, v)$.

Secondly, $d(v)$ denotes the depth-first-search (DFS) preorder enumeration of the rooted tree, note that $\{d(u) | u \in U\} = \{1, \ldots, m\}$. Finally every node has a name $s(v)$ which will

be defined as a function of its own subtree size relative to its siblings' subtree sizes. In some sense this reflects its rank among its siblings. The formal value of $s(v)$ will be defined later.

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

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

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

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

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

1. Let $y \in T$ be such that $o(x) = n(y)$. Node x stores the tuple $(y,\ n(y),\ \ell(y))$.
2. Node x stores $A(x) = \{o(y) \mid y \in \text{child}(x)\}$ together with a map from any $o(y) \in A(x)$ to the corresponding port name $\text{port}(x, y)$ to reach the child y.
3. x stores $\mu(T, x)$, its tree-routing label as required from Lemma 3.3.4.

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

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

This procedure is similar to the interval routing of [SK85, vLT86]. If the label $\ell(u)$ is found, routing proceeds using the labeled tree-routing scheme of Lemma 3.3.4. In the simple scheme presented above, the cost of reaching root is at most $\text{rad}(T)$, cost of reaching the node storing the required label is bounded by $\text{rad}(T)$ and reaching the target (or reporting an error to the source) requires at most another $2\text{rad}(T)$. In the fixed port model the storage per node is $\widetilde{O}(|\text{child}(v)| + 1) = \widetilde{O}(n)$.

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

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

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

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

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

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

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

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

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

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

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

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

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

More precisely, the routing algorithm is as follows.

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

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

Directory algorithm.

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

3.3.5 Analysis

Lemma 3.3.5. *Given a parameter κ, the name-independent error-reporting tree-routing scheme requires $O(\kappa n^{1/\kappa} \log^2 n)$ bits of storage per node in the tree.*

Proof. Each node v stores $\widetilde{O}(1)$ information for each child $u \in \text{child}(v)$ such that $s(u) \in S(v)$. By definition of $S(v)$, it contains at most $\kappa n^{1/\kappa}$ members, hence the storage is $\widetilde{O}(\kappa n^{1/\kappa})$. In addition node v maintains information to assist its parent node $p(v)$. This includes $\widetilde{O}(1)$ storage per each member in $S'(v)$, which, by definition, also requires $\widetilde{O}(\kappa n^{1/\kappa})$ bits. Finally, each node v stores the routing information $\mu(T, v)$ according to Lemma 3.3.4, requiring $\widetilde{O}(1)$ storage. Thus, the total storage of this scheme is $\widetilde{O}(\kappa n^{1/\kappa})$ items per node, each of $O(\log^2 n)$ bits at most. □ □

Lemma 3.3.6. *Given a parameter κ, the name-independent error-reporting tree-routing scheme routs on paths whose cost is at most*

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

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

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

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

Then u_1 routes up to v and down to u_2, which has a length-i child name that matches $s(y)$ in at least the first two digits. This process continues until the correct child y is reached after at most $i-1$ intermediate steps from v to a child and back.

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

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

Chapter 4

Name-Independent Routing for Growth Bounded Networks

4.1 Introduction

Given a network of processes in which each process has a unique name, a routing scheme is a distributed algorithm in which, given a destination's name, any node can route messages that will eventually reach the destination.

Modeling the network as an undirected weighted graph G there is a well known trade-off between two conflicting parameters of a routing scheme RS. The first is the space complexity, the maximum over all nodes of the number of bits of information required by RS, we denote this as space(RS, G). The second is the stretch factor denoted by stretch(RS, G) which is the maximum ratio over all pairs between the cost of the shortest path between the pair denoted $d(s, t)$ and the cost of the path induced by the routing scheme denoted $d_{RS}(s, t)$ for the same source destination pair.

The most studied problem in this context is the *universal* trade-off between space and stretch. Specifically, let $\mathcal{G}(n)$ denote the set of all connected weighted graphs on n nodes, then for any routing scheme RS, let space$(RS, n) = \max_{G \in \mathcal{G}(n)}$ space(RS, G) and stretch$(RS, n) = \max_{G \in \mathcal{G}(n)}$ stretch(RS, G). The universal trade-off problem for a parameter $k \geq 1$ is to find a polynomial scheme RS that as a function of n, minimizes stretch(RS, n) given the restriction that space$(RS, n) = O(n^{1/k})$.

The lower bounds [PU89, GG01, TZ01b] for *universal* compact routing schemes come from graphs with many edges and high girth. These bounds show that there exist high girth n-node graphs in which any scheme that wants to achieve stretch less than $2k+1$ must require some node to store $\Omega(n^{1/k})$ bits of routing information. These high girth graphs seem very far from a typical real life connected system. To the contrary, most Internet networks tend to have multiple relatively short paths from any source destination pair. Looking at asymptotic behavior on the size of the network, worst case analysis over all the input space is one of the most studied questions in theoretical computer science. However it may be that for a given

network or for a large family of networks there are polynomial time constructible schemes that have much better trade-offs than that of a universal scheme on the same network.

There are two variants on the assumptions of node names: *labeled* and *name-independent*. In the labeled model [Cow99, TZ01b, TZ01a, EGP03] the designer of the routing scheme is allowed to give each node a poly-logarithmic label. This name is then used in order to route to the destination. In the name-independent model, the names of nodes are independent of the routing scheme. For brevity's sake, assume names are unique indexes from $\{1, \ldots, n\}$. Name-independent schemes [ABNLP89, ABNLP90, AP90, AP92, ACL+03, AGM+04b, AGM04a] are inherently harder than labeled schemes. Informally, before routing on a low stretch path one needs some low stretch directory service to learn where the destination is located.

The name-independent model is suitable when node names are required to have some specific value that is not related to the routing scheme. For example if nodes participate in a forming a Distributed Hash Table (DHT), their names should be arbitrary points in a unit segment; and in a mobile setting nodes may need persistent names in order to be consistently identified independently of their current location. Generally, name-independent schemes allow the network designer to label nodes with names that do not necessarily need to change every time the topology changes.

4.1.1 Problem definition

In this chapter we study the space-stretch trade-off for name-independent routing schemes on growth-bounded graphs.

Let $G = (V, E, \omega)$ be an undirected weighted graph. For any $u, v \in V$ let $d(u, v)$ denote the cost of a minimum cost path between u and v, where the cost of a path is the sum of the weight of its edges. For any $v \in V$, $r \in \mathbb{R}^+$ define $N(v, r) = \{u \mid d(u, v) \leq r\}$.

Definition 4.1.1. *For a real number $\Delta > 0$, an undirected weighted graph G is* Δ growth-bounded *if for all $v \in V$ and $r \in \mathbb{R}^+$, if $|N(v, r)| > 1$ then $|N(v, 2r)| \leq \Delta |N(v, r)|$.*

This definition captures the growth dimensionality of a network. Define that G has *growth dimension* s iff G is 2^s growth-bounded.

In this chapter we study the problem of achieving the minimum stretch over all growth-bounded networks. Let $\mathcal{G}(n, \Delta)$ be the set of all n-node undirected weighted graphs G that are Δ growth-bounded. The goal is to find a routing scheme RS that minimizes $\text{stretch}(RS, n, \Delta) = \max_{G \in \mathcal{G}(n,\Delta)} \mathit{stretch}(RS, G)$ given a bound on $\text{space}(RS, n, \Delta) = \max_{G \in \mathcal{G}(n,\Delta)} \mathit{space}(RS, G)$.

4.1.2 Our results

In this chapter we present a polynomial time constructible name-independent routing scheme with stretch $1 + \epsilon$ that requires with high probability only a poly-logarithmic number of bits

of routing information per node, for any n-node Δ growth-bounded network. This result is in sharp contrast to the universal space stretch lower bounds [PU89, GG01, TZ01b].

Theorem 4.1.2 (Main). *For any $\epsilon > 0$, n, and Δ there exists a polynomial time constructible name-independent routing scheme with* $\text{stretch}(RS, n, \Delta) \leq 1 + \epsilon$ *and* $\text{space}(RS, n, \Delta) = O(\frac{1}{\epsilon}^{O(\log \Delta)} \log^5 n)$ *with high probability.*

4.1.3 Related work

A less restrictive model is one in which a metric space is given and the designer is required to construct both a low degree overlay network and an accompanying routing scheme. Compared to routing schemes on graphs, the advantage is that the overlay can connect between any nodes that the designer desires. Plaxton, Rajaraman and Richa [PRR97] give an object location scheme[1] for metric spaces that are growth-bounded and shrink-bounded (exists constants δ, Δ such that $\delta \leq \frac{N(u,2r)}{N(u,r)} \leq \Delta$). For such metrics they give a randomized solution in which the expected stretch is constant and the overlay degree is logarithmic and hence the memory requirement is poly-logarithmic per node. This scheme was later adapted to dynamic settings by Hildrum et al. [HKRZ02, HKMR04]. Using a distributed node emulation technique, Abraham et al. [AMD04] show how to reduce the stretch to $1 + \epsilon$ while achieving expected logarithmic degree and requiring only a growth-bound on the metric space. A method that does object location for more realistic networks was given by Hildrum et al. [HKK04]. Indeed our construction has roots in the PRR object location overlay [PRR97, AMD04], while extending the treatment from metric spaces to graphs.

On graphs, the universal space-stretch trade-off has been extensively studied under various models and extensions. We refer the reader to Peleg's book [Pel00] and to the surveys of Gavoille and Peleg [Gav01, GP03] for background.

Trees are another family of graphs that is well studied. Labeled routing on a trees is explored in [FG01, TZ01b], achieving stretch 1 with $O(\log^2 n / \log \log n)$ bits for local tables and for headers, and this is tight [FG02]. Laing [Lai03] presents a routing scheme on trees with arbitrary names that obtains stretch $2^k - 1$ with $\widetilde{O}(n^{1/k})$ bit routing tables. With the same bit complexity the author gives a *single-source* routing scheme with stretch $2k - 1$.

Iwama and Okita study compact routing schemes on *flat* and *almost-flat* networks [IO03]. We note that flat networks are growth-bounded so our result is applicable to their model.

Recently there have been several efforts to devise labeled routing schemes and distance labels for graphs whose induced metric space has constant *doubling dimension*. A metric space is said to have doubling dimension δ if any ball with radius $2r$ can be covered by at most 2^δ balls of radius r. Informally the doubling dimension indicates how far the metric is from having a uniform sub-metric. It is well known (see [GKL03]) that any metric with constant growth-bound has constant doubling dimension and that the opposite need not

[1] Object location schemes are stronger than name-independent routing schemes. They alow targets to be replicated and grantee stretch relative to the closest copy from the source.

be true. Hence a constant growth-bound is a strictly more restrictive requirement than a constant doubling dimension. However for doubling metrics only labeled routing schemes are known.

Given an n-node network with diameter D whose induced metric space has a constant doubling dimension δ, for any constant $\epsilon > 0$ let $K = \frac{1}{\epsilon}^{O(\delta)}$. The following results were obtained for stretch $1+\epsilon$ distance labels: Gupta et al. [GKL03] $O(K \log n \log D)$, Talwar [Tal04a] $O(K \log^2 D)$, Chan et al. [CGMZ05] $O(K \log n \log D)$, Slivkins [Sli05a] $O(K \log^2 n(\log n + \log\log D))$, Har-Peled, Mendel [HPM05] $O(K \log n(\log n + \log\log D))$, Slivkins [Sli05b] $O(K \log n \log\log D)$. Papers [Tal04a, CGMZ05, Sli05a, Sli05b] also give labeled routing schemes based on their distance oracles.

The line of works above focuses on labeled schemes and obtains stretch $1+\epsilon$ for any fixed $\epsilon > 0$. Recently, Abraham et al. [AGM05] prove that any name-independent routing scheme on networks with doubling dimension δ must have stretch at least $3-\epsilon$ if less than $\Omega(\delta n)$ bits are used. This lower bound implies that the stretch $1+\epsilon$ schemes mentioned above cannot be extended with the same stretch factor to name-independent schemes on networks with constant doubling dimension. This leaves open the question of achieving stretch $1+\epsilon$ name-independent routing on other constrained families of graphs, which our works addresses.

4.2 Overview

In this section, we give an informal overview of the scheme.

Nodes are assigned virtual B-ary identifiers uniformly at random. The base $B = \lceil \Delta^2 \rceil$ is determined by the growth-bound. For each level $\ell \in \{1, ..., \log_B n\}$, each node is assigned $O(\log n)$ identifiers of length ℓ. Each node defines a self-centric partition of the set of nodes, with gradually increasing vicinities around itself, each one containing B times as many nodes as the former. We denote the vicinity of node v containing B^i nodes by $A(v,i)$, and its diameter by $a(v,i)$.

Given a B-ary identifier x of length ℓ, $A(v,\ell)$ is expected to contain $\Theta(\log n)$ nodes whose level ℓ identifiers equal x.

First, we construct a stretch $1+\epsilon$ labeled routing scheme using *zero assisted routing*. Call a node whose identifier is 0^ℓ a level-ℓ zero node. For each node, its label contains the names of the $\Theta(\log n)$ zero nodes closest to it, one from each level. Each node stores labeled tree-routing routing information on trees rooted at zero nodes in its vicinity. Specifically, it stores routing tables for all level-i zero nodes within $A(v, i+\alpha+2)$, where $\alpha = O(\log 1/\epsilon)$. The expected amount of storage is $\widetilde{O}(2^\alpha)$.

Consider two nodes u and v whose distance is $d \approx a(v,i) * 2^\alpha$. Because $A(v,i)$ contains with high probability a level-i zero node, then v has a level-i zero node z_i "close-by", namely within distance ϵd. The main property we obtain from the growth-bound, is the following. Blowing up the radius of $A(v,i)$ by a factor of 2^α results in a vicinity that contains u on

the one hand, and on the other hand, contains at most a factor Δ^α nodes over $A(v,i)$. Consequently, we prove in Lemma 4.3.2 below that $A(u, i+\alpha+2)$ contains $A(v,i)$, and hence, contains z_i. Therefore, all nodes from z_i towards u, including u, store the tree routing information of z_i's tree. Hence, given u's label, v can route to u over z_i's tree, paying ϵd extra distance.

Second, when v routes to u, we need to store u's label so that v can find it within $\approx \epsilon d$ distance. This is done by storing u's label at all nodes with matching level-i identifiers to the length i prefix of u within $A(u, i+\alpha+2)$. Once again, the storage inflicted by this on any node is bound by $\widetilde{O}(2^\alpha)$.

The strategy for finding u's label is to perform iterative routing by fixing one-bit at a time; this is called *prefix routing*. Within $A(v,i)$, v can find a node x_i that "fixes" i bits in u's name. As above, due to the growth-bound, x_i is within $A(u, i+\alpha+2)$, hence it stores u's label, and we are done.

4.3 Preliminaries

Virtual Identifiers and Vicinities. Our construction makes heavy use of virtual node identifiers, which are drawn at random from certain alphabets. We now introduce the relevant definitions concerning alphabets, identifiers and vicinities.

Given a Δ growth-bounded network we set B, the size of the alphabet, to $B = \lceil \Delta^2 \rceil$. Denote the alphabet $\Sigma = \{0, 1, \ldots, B-1\}$. Given a letter $b \in \Sigma$ denote b^i as the word $b, \ldots, b \in \Sigma^i$ and given a word $w = w_1, \ldots, w_i \in \Sigma^i$ and letter $b \in \Sigma$ denote $w||b$ as the word $w_1, \ldots, w_i, b \in \Sigma^{i+1}$.

In our scheme, identifers will be chosen in various lengths. Denote by M the maximal length, such that $M = \lceil \log_B n \rceil$. Denote the lengths set by $L = \{1, 2, \ldots, M\}$. We frequently refer to a length ℓ as *level* ℓ.

Definition 4.3.1 (ℓth vicinity around v)**.** *For all $v \in V, \ell \in L$ denote $A(v,\ell)$ as the B^ℓ closest nodes to v with ties broken by the node identifiers. Let $a(v,\ell)$ be the radius of the ball $A(v,\ell) = N(v, a(v,\ell))$.*

The important properties of vicinities, derived from the growth-bound assumption, are stated in the following lemma. Parts (i)-(iii) of this lemma borrow from [AMD04], though the definitions of vicinities there are slightly different.

Lemma 4.3.2. *Let x and y be any two nodes, for any i such that $y \in A(x,i)$:*
(i) $A(x,i) \subseteq A(y, i+1)$.
(ii) $A(y,i) \subseteq A(x, i+1)$.
(iii) $a(x, i+1) \geq 4a(x,i)$.
(iv) $a(y,i) \leq 2a(x,i)$,

Proof. Let $r = a(x,i)$ denote the radius of $A(x,i)$. Since $y \in A(x,i)$ then (see Figure 4.1)

$$N(x,r) \subseteq N(y,2r) \subseteq N(x,3r) \ .$$

From the growth-bounded assumption we can bound the number of nodes in $N(x,3r)$ using $|N(x,r)|$ as follows: $|N(x,3r)| \leq \Delta^2|N(x,r)| = \Delta^2|A(x,i)| = \Delta^2 B^i \leq B^{i+1}$.

For (i), $N(x,3r) \subseteq A(x,i+1)$, and so by node count, $A(y,i+1)$, the ball around y with B^{i+1} nodes, must contain $N(y,2r)$ and so must contain $A(x,i)$. For (ii), $A(y,i) \subseteq N(y,2r) \subseteq N(x,3r) \subseteq A(x,i+1)$.

For (iii), $|N(x,4r)| \leq \Delta^2|N(x,r)| \leq B^{i+1}$, so $A(x,i+1) \supseteq N(x,4r)$.

Finally, for (iv), $N(y,2r) \supseteq A(x,i)$, so $A(y,i) \not\supset N(y,2r)$. Hence, $a(y,i) \leq 2r$.

□

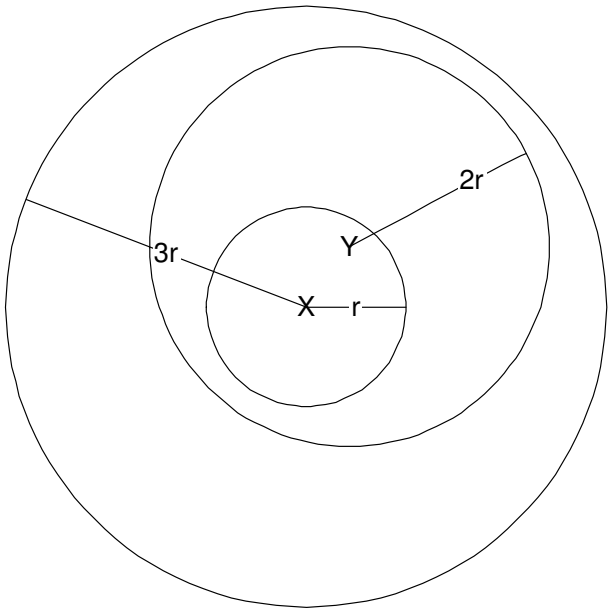

Figure 4.1: The circles $N(x,r)$, $N(y,2r)$, and $N(x,3r)$

We will select virtual identifiers with certain redundancy. To this end, set $\rho > 2$ as a confidence parameter, and let R denote a replication factor $R = \lceil \rho \ln n \rceil$. Let K denote the replication set $K = \{0, 1, \ldots, R\}$

Finally, let $\alpha = \alpha(\epsilon)$ be a constant parameter of the construction which will be determined below (See Eqn. 4.1 in Section 4.5).

4.4 The Scheme

4.4.1 Identifiers and the zero-sets.

Every node chooses $R \cdot M = O(\log^2 n)$ identifiers in the following manner. For every level ℓ in L a node chooses R length ℓ random words.

Definition 4.4.1 (The identifiers)**.** *$\forall v \in V, \ell \in L, k \in K$ let $I(v, \ell, k)$ denote a random variable chosen independently and uniformly out of Σ^ℓ.*

For an identifier w let $C(w)$ denote the nodes that have chosen w.

Definition 4.4.2 (The prefix set)**.** *$\forall \ell \in L, w \in \Sigma^\ell$ denote $C(w) = \{u \mid (\exists k \in K) I(u, \ell, k) = w\}$.*

Specifically, nodes that have an all zero identifer will be part of the zero set. Every node records the closest zero nodes as its zero link.

Definition 4.4.3 (The zeros)**.** *For all $\ell \in L$ define $Z(\ell) = C(\{0\}^\ell)$.*

Definition 4.4.4 (The zero link)**.** *For all $v \in V, \ell \in L$ define $z(v, \ell)$ as the node closest to v in the set $Z(\ell)$.*

There are two key properties relating the random virtual identifier selection and vicinities. One bounds the density of an identifier within a vicinity from below; the other from above. Both are stated in the following lemma.

Lemma 4.4.5. *Let $w \in \Sigma^\ell$ be any specific identifier. Then for all $v \in V$ and $j \geq \ell$ we have $|C(w) \cap A(v, j)| \in (\frac{1}{2} R\, B^{(j-\ell)},\, 2R\, B^{(j-\ell)})$ w.h.p.*

Proof. The expected value of $|C(w) \cap A(v, j)|$ is $R\, B^j\, B^{-\ell} = R\, B^{j-\ell}$. Using standard Chernoff bounds we get

$$Pr[\, |C(w) \cap A(v, j)| \in (\frac{1}{2} R\, B^{j-\ell},\, 2R\, B^{j-\ell})\,] \geq$$
$$1 - 2e^{\frac{1}{8} R B^{j-\ell}} \geq 1 - \frac{2}{n^{(\rho/8)}}$$

and the lemma follows by choosing a large enough ρ. □

4.4.2 Zero-Assisted Routing

Routing on the graph is done via the assistance of the zero nodes. This is done by utilizing labeled-tree routing on partial trees. More specifically, we repeatedly make use of the single source labeled routing scheme stated in the following lemma.

Lemma 4.4.6. [FG01, TZ01b] *For every weighted tree T with n nodes there exists a labeled routing scheme that, given any destination label, routes optimally on T from any source to the destination. The storage per node in T, the label size, and the header size are $O(\log^2 n/\log\log n)$ bits. Given the information of a node and the label of the destination, routing decisions take constant time.*

For a tree T containing a node v, we let $\mu(T,v)$ denote the routing information of node v and $\lambda(T,v)$ denote the destination label of v in T as required from Lemma 3.3.4.

Denote all the zeroes as $\mathcal{Z} = \bigcup_{\ell\in L} Z(\ell)$. For any $z \in \mathcal{Z}$ let $T(z)$ denote a minimum cost path tree rooted at z.

The key element we use in forming graph routing is the following. Let us have a node v and another node u such that u is in the $(i+\alpha+2)$'th vicinity of v, i.e., $u \in A(v, i+\alpha+2)$. We want to make use of a zero-node z_i in order to route from u to v. In order to provide this, every node x on the path from z_i to v needs to maintain $\mu(T(z_i), x)$; and every node x on the path from u to z_i must also maintain $\mu(T(z_i), x)$, and thus, given $\lambda(T(z_i), v)$, we can route from u to v. The following lemma states that these provisions are satisfied if every node maintains tree routing information on zeroes in its $A(*, i+\alpha+4)$ vicinity:

Storage 4.4.7. *For every $i \in M$, let each node $v \in V$ maintain $\mu(T(z_i), v)$ for every $z_i \in Z(i) \cap A(v, i+\alpha+4)$.*

We have obtained the following.

Lemma 4.4.8. *Let $v \in V$ be a node and $u \in A(v, i+\alpha+2)$. Then for every zero node $z \in A(u, i+\alpha+2)$, where $z \in Z(\ell)$ for any $i \le \ell \le M$, given the label $\lambda(T(z), v)$, node u can route to v with route length at most $d(v,u) + 2a(u,\ell)$ w.h.p.*

Proof. For every node w on a shortest path from u to z, we have $z \in A(w, i+\alpha+2) \subseteq A(w, \ell+\alpha+2)$ because $z \in A(u, i+\alpha+2)$. Therefore, w maintains $\mu(T(z), w)$.

Now, by Lemma 4.3.2(ii), we have that $z \in A(u, i+\alpha+2) \subseteq A(v, i+\alpha+3)$. Therefore, every node w on any shortest path from v to z also has $w \in A(v, i+\alpha+3)$. Applying Lemma 4.3.2(i), we obtain $z \in A(v, i+\alpha+3) \subseteq A(w, i+\alpha+4)$. Therefore, w maintains $\mu(T(z), w)$.

Together, we have that all nodes w on the path from v to u over $T(z)$ maintain $\mu(T(z), w)$. We obtain that given the label $\lambda(T(z), v)$, node u can route to v over $T(z)$.

By Lemma 4.4.5, $z \in A(u, \ell)$ w.h.p. Hence, the length of the routing path is at most $d(u,z) + d(z,v) \le d(u,z) + d(u,z) + d(v,u) \le d(v,u) + 2a(u,\ell)$, as required. □

4.4.3 Prefix routing

In order to perform prefix routing every node with identifier w stores the closest node that contains an identifier that extends w by one bit.

Definition 4.4.9 (The neighbor link)**.** *For all $v \in V, \ell \in L, k \in K, b \in \Sigma$ define $n(v, \ell, k, b)$ as the node closest to v in the set $C(I(v, \ell, k)||b)$.*

By Lemma 4.4.5 above, we have that neighbor links that fix the ℓth bit are in $A(v, \ell)$.

Lemma 4.4.10. *For all $v \in V, \ell \in L, k \in K, b \in \Sigma$, w.h.p. $n(v, \ell, k, b) \in A(v, \ell)$.*

Proof. This follows immediately from Lemma 4.4.5. □

Every node stores an appropriate tree-label for every neighbor:

Storage 4.4.11. *For all $\ell \in L, k \in K, b \in \Sigma$, let $u = n(v, \ell, k, b)$ be the appropriate neighbor. Node v stores $z(v, \ell), \lambda(T(z(v, \ell)), u)$ (for prefix routing to the neighbor link).*

Together with the zero-assisted routing construction above, we get that a node v can route to its level-i neighbor via a route of distance proportional to $a(v, i)$:

Lemma 4.4.12. *Let $v \in V$ be a node, $u = n(v, i, k, b)$ a level-i neighbor. Then v can route to u with route length at most $3a(v, i)$ w.h.p.*

Proof. By Lemma 4.4.10, w.h.p. $u \in A(v, i)$. Using Lemma 4.3.2(i), we have $v \in A(v, i) \subseteq A(u, i + 1)$. Applying Lemma 4.4.8, we obtain that v can route to u with route length at most $d(v, u) + 2a(v, i) \leq 3a(v, i)$. □

4.4.4 The Directory

The final component of our construction is a directory of node labels, that guarantees routing with $(1 + \epsilon)$ bounded stretch.

In order to disperse directory entries such that they can be found, we use a hash function $h : V \to \Sigma^M$ that is $e^2RB^{\alpha+3}$-wise independent. Carter and Wegman [CW79] show how to build such a function and represent it with $O(e^2RB^{\alpha+3} \log n) = O(\log^2 n)$ bits. Now for all $v \in V$ denote $h(v) = h(v)_1, \ldots, h(v)_M$. For all $\ell \in L$ denote the subsequence $h(v, \ell) = h(v)_1, \ldots, h(v)_\ell$. The following defines the set of nodes that implement the directory for a node v; these are vicinity nodes that have an identifier that coincides with $h(v, i)$.

Definition 4.4.13 (The directory set)**.** *Let v be a node. For every $i \in L$, the level-i directory set $D(v, i)$ is defined as $D(v, i) = A(v, i + \alpha + 2) \cap C(h(v, i))$.*

Storage 4.4.14. *For any node v, and every level $i \in L$, we store a reference of the form $v \longrightarrow \langle z(v, i), \lambda(T(z(v, i)), v)\rangle$ at all the nodes in the directory set $D(v, i)$.*

Lemma 4.4.15. *With high probability, the directory storage requires at most $e^2MR^2B^{\alpha+3} \log^2 n$ bits of storage per node.*

Proof. Fix a node $u \in V$. We want to count the nodes v for which u stores a reference, i.e., for which there exists $i \in L$, $u \in D(v,i)$.

For every $\ell \in L, k \in K$ define $X(u,\ell,k) = \{v \mid I(u,\ell,k) = h(v,\ell) \text{ and } v \in A(u,\ell{+}\alpha{+}3)\}$. Define $X(u) = \bigcup_{\ell\in L, k\in K} X(u,\ell,k)$. The relationship of X to directory storage is as follows. If $u \in D(v,i)$ for some $i \in L$, then there exists $k \in K$ for which $I(u,i,k) = h(v,i)$, and furthermore, $u \in A(v,i{+}\alpha{+}2)$. By Lemma 4.3.2(ii), $v \in A(u,i{+}\alpha{+}3)$. Hence, if $u \in D(v,i)$ then $v \in X(u)$ (though note that the converse need not be true). Our strategy is to bound the size of $X(u)$, and thereby bound u's storage requirements from the above.

The probability that $|X(u,\ell,k)| \geq e^2RB^{\alpha+3}$ is less than the probability that there exists a set of $e^2RB^{\alpha+3}$ identifiers in $A(u,\ell+\alpha+3)$ such that all these identifiers equal $I(u,\ell,k)$.

$$\begin{aligned}
Pr\left[|X(u,\ell,k)| \geq e^2RB^{\alpha+3}\right] &\leq \binom{RB^{\ell+\alpha+3}}{e^2RB^{\alpha+3}}(B^{-\ell})^{e^2RB^{\alpha+3}} \\
&\leq (RB^{\ell+\alpha+3})^{e^2RB^{\alpha+3}}\frac{1}{(e^2RB^{\alpha+3})!}(B^{-\ell})^{e^2RB^{\alpha+3}} \\
&\leq (RB^{\alpha+3})^{e^2RB^{\alpha+3}}\left(\frac{e}{e^2RB^{\alpha+3}}\right)^{e^2RB^{\alpha+3}} \\
&\leq (1/e)^{RB^{\alpha+3}} \\
&\leq n^{-\rho}
\end{aligned}$$

Note that this argument only requires a $e^2RB^{\alpha+3} = O(\log^2 n)$-wise independent hash function. Hence by union bound $Pr\left[|X(u)| \leq e^2MR^2B^{\alpha+3}\right] \geq 1 - RMn^{-\rho} \geq 1 - n^{1-\rho}$. Finally, each element in the directory storage requires $O(\log^2 n)$ space, totalling $O(e^2MR^2B^{\alpha+3}\log^2 n)$ storage bits. □

When routing toward v, we use $h(v)$ as a target for bit-fixing. Let the sequence of nodes visited by fixing the bits of $h(v)$ be $s = x_0, x_1, x_2, \ldots$ When the distance from x_i to v is at most $a(v, i+\alpha+2)$, a directory reference is guaranteed to be found. And since $x_i \in A(v, i+\alpha+2)$, by Lemma 4.4.8 we get that x_i can route to v given $\lambda(T(z(v,i)), v)$. This is stated in the following lemma.

Lemma 4.4.16. *Let $v \in V$ be a node. Then for any node u, such that $u \in A(v, i+\alpha+2)$ and $\exists k : I(u,i,k) = h(v,i)$, u can route to v with route length at most $d(v,u) + 2a(u,i)$.*

Proof. By construction, we have that u is a level-i directory node for v, i.e., $u \in D(v,i)$. Therefore, u maintains a directory reference on v. The fact that u can route to v with the specified route length then follows directly from Lemma 4.4.8, since $u \in A(v, i+\alpha+2)$. □

4.4.5 The Routing Algorithm

Assume the source is $s \in V$ and the target is $t \in V$. Set $i = 0$ and $x_0 = s$. Routing has two stages.

Phase 1: (Prefix routing) If x_i does not contain a pointer $t \rightarrow u$ then let k, b be such that $I(x_i, i, k)||b = h(t, i+1)$. Let the neighbor information corresponding to $n(x_i, i, k, b)$ at x_i be z_i, λ_i, route on $T(z_i)$ to λ_i. Set $x_{i+1} = n(x_i, i, k, b)$, $i = i+1$ and repeat Phase 1.

Phase 2: (Directory routing) Once x_i contains a pointer $t \rightarrow \langle z, \lambda \rangle$ then on tree $T(z)$ use label λ to route to t.

4.4.6 Correctness

Lemma 4.4.17. *From any starting node $s \in V$, given any node $t \in V$, the routing algorithm finds t within a finite number of steps.*

Proof. During the first phase, every bit-fixing step succeeds according to Lemma 4.4.12. Let $\ell \in L$ be a level such that $B^{(\ell+\alpha+2)} \geq n$. At the latest, when the first phase has made ℓ steps, it must find a reference to t. This holds since it reaches a node x_ℓ that satisfies $x_\ell \in A(t, \ell+\alpha+2) \cap C(h(t,\ell)) \Longrightarrow x_\ell \in D(t, \ell)$. Once a reference to the target t is found, Phase 2 succeeds by Lemma 4.4.16. □

4.5 Stretch Analysis

Throughout the analysis below, we denote the source node by s, the target node by t. The series of neighbor-steps during Phase 1 are denoted $s = x_0, x_1, x_2, ..., x_i$. Phase 2 starts at x_i and ends at t.

Lemma 4.5.1. *For all $s \in V, 1 \leq i \in L$, during Phase 1 of routing, $x_i \subseteq A(s, i+1)$.*

Proof. By induction on i. For $i = 1$ we have $s = x_0$, and by Lemma 4.4.10, the neighbor satisfies $x_1 = n(s, 1, k, b) \in A(s, 1)$. Also, clearly $A(s, 1) \subseteq A(s, 2)$.

Assume by induction that $x_{i-1} \in A(s, i)$. By Lemma 4.3.2(ii), $A(s, i+1) \supseteq A(x_{i-1}, i)$. By Lemma 4.4.10, $x_i \in A(x_{i-1}, i)$, and hence, $x_i \in A(s, i+1)$. □

Lemma 4.5.2. *The total distance of the path from $s = x_0$ to x_i is at most $2a(s, i+1)$.*

Proof. By Lemma 4.5.1 for every $1 \leq j \leq i$, $x_j \in A(s, j+1)$. Applying Lemma 4.3.2(iv), $a(x_j, j+1) \leq 2a(s, j+1)$. By Lemma 4.4.12, the neighbor-routing from x_j to x_{j+1} has length at most $3a(x_j, j+1)$. Putting the above together, the route from x_j to x_{j+1} is bounded by $6a(s, j+1)$.

By Lemma 4.3.2(iii), $a(s, j+1) \leq 4^{-(i-j)} a(s, i+1)$. Hence, the total distance of the path from x_0 through x_i is at most

$$\sum_{j=0}^{i-1} 6a(s, j+1) \leq 6a(s, i+1) \sum_{j=0}^{i-1} 4^{-(i-j)} \leq \frac{6}{4-1} a(s, i+1) \ .$$

□

Lemma 4.5.3. *Let j be the first index such that $s \in A(t, j+\alpha+2)$ then $i \leq j$.*

Proof. From Lemma 4.5.1, $x_j \in A(s, j+1)$. Applying Lemma 4.3.2(ii) on $s \in A(t, j+\alpha+1)$ gives $A(s, j+\alpha+1) \subseteq A(t, j+\alpha+2)$. Combining the above $x_j \in A(s, j+\alpha+1) \subseteq A(t, j+\alpha+2)$. Also, by the routing algorithm, $x_j \in C(h(t, j))$. Therefore, $x_j \in D(t, j)$, and x_j must contain a reference to t. □

Theorem 4.5.4. *The stretch of the path from s to t is $1+\epsilon$.*

Proof. First, if $s \in A(t, \alpha+1)$, then by definition, $j \in D(t, 0)$ and hence stores a directory reference to t. In this case, Phase 1 is degenerate, and we move directly to Phase 2. Since $z(t, 0) = t$, the reference at s on t is of the form $t \longrightarrow t, \lambda(T(t), t)$, and routing is along the shortest path from s to t.

Otherwise, as in Lemma 4.5.3 above, let j be the first index such that $s \in A(t, j+\alpha+2)$. The first phase of the route is the path from $s = x_0$ to x_j. We now make use of the assumption that $s \notin A(t, j+\alpha+1)$, so $d(s, t) \geq a(t, j+\alpha+1)$. By node count, since $A(t, j+\alpha+1) \not\subset A(s, j+\alpha+1)$, we obtain $a(s, j+\alpha+1) \leq d(s, t) + a(t, j+\alpha+1) \leq 2d(s, t)$. This, we note by Lemma 4.3.2(iii) implies $a(s, j+1) \leq 2 \cdot 4^{-\alpha} d(s, t)$.

With Lemma 4.5.3, we obtain that the route length from s to x_j is bounded by

$$2a(s, j+1) \leq 4 \cdot 4^{-\alpha} d(s, t) \ .$$

The second phase is the traversal from x_j to t. With Lemma 4.4.16, its length is bounded by $d(x_j, t) + 2a(x_j, j)$. Here, we use from Lemma 4.5.1 the fact that $x_j \in A(s, j+1)$. With the triangle inequality, we have $d(x_j, t) \leq d(x_j, s) + d(s, t) \leq a(s, j+1) + d(s, t)$. For $a(x_j, j)$, we use Lemma 4.3.2(iv) to obtain $a(x_j, j) \leq 2a(s, j+1)$. Putting all of the above together, the length of the route from x_j to t is bounded by

$$\begin{aligned} & d(x_j, t) + 2a(x_j, j) \\ & \quad \leq a(s, j+1) + d(s, t) + 4a(s, j+1) \\ & \quad \leq d(s, t) + 10 \cdot 4^{-\alpha} d(s, t) \end{aligned}$$

The resulting total stretch is $1 + 14 \cdot 4^{-\alpha}$, and the theorem is proven by choosing

$$\alpha = \log_4 \frac{14}{\epsilon} = O(\log(\frac{1}{\epsilon})) \ . \tag{4.1}$$

□

4.6 Space Analysis

For all $v \in V$ node v stores the following:

1. For all $i \in L, z \in Z(i) \cap A(v, i+\alpha+4)$ store $\mu(T(z), v)$ (for zero-assisted routing).
2. For all $i \in L, k \in K, b \in \Sigma$, and for $u = n(v, i, k, b)$, store $z(v, i), \lambda(T(z(v, i)), u)$ (for prefix routing to the neighbor link).
3. For all $i \in I$ store $v \longrightarrow \langle z(u, i), \lambda(T(z(u, i)), v)\rangle$ at all the nodes u in the directory set $D(v, i)$ (for finding v).

Theorem 4.6.1. *With high probability, the network storage requires at most* $O(\frac{1}{\epsilon}^{O(\log \Delta)} \log^5 n)$ *bits of storage per node.*

Proof. The storage consists of the following.

For the first storage item above, v stores routing information of size $O(\log^2 n)$ of at most $2RB^{\alpha+4}$ zero nodes for each $i \in L$, and the total is $O(MRB^{\alpha+4} \log^2 n)$.

Using $B^\alpha = B^{\log_4 \frac{14}{\epsilon}} = \frac{14}{\epsilon}^{\log_4 B} \leq \Delta^4 \frac{1}{\epsilon}^{\log \Delta}$ we get that the total is $O(\frac{1}{\epsilon}^{\log \Delta} \Delta^{12} \log^4 n)$.

For the second item, v stores label information of size $O(\log^2 n)$ of B neighbors for each $i \in L$, and $k \in K$, totalling $O(BMR \log^2 n) = O(\Delta^2 \log^4 n)$.

The storage of the third item is bounded by Lemma 4.4.15 to be $O(e^2 MR^2 B^{\alpha+3} \log^2 n) = O(\frac{1}{\epsilon}^{\log \Delta} \Delta^{12} \log^5 n)$.

Summing it all up, we have $O(\frac{1}{\epsilon}^{O(\log \Delta)} \log^5 n)$ bits of storage per node. □

Chapter 5

Scale-Free Name-Independent Routing

5.1 Introduction

One of the most basic functionalities of any distributed network is the ability to route messages between pairs of nodes. Given that each node has an arbitrary network identifier, a routing scheme allows any source node to route messages to any destination node, given the destination's network identifier. It is natural to consider a weighted network in which the cost of routing a message is proportional to the cost of the path taken from source to destination. In such a model it is desirable to minimize routing costs by routing on short paths. In this sense the efficiency of a routing scheme is measured by its *stretch factor*, the maximum ratio over all source destination pairs, between the cost of routing from the source to the destination and the cost of a minimum cost path. The trivial solution to routing on shortest paths with stretch factor 1 is for each node to store a routing table with $(n-1)$ entries that contains the next hop of an all pairs shortest path algorithm. This solution is very expensive as it requires each node to store $\Omega(n \log n)$ bits. Thus, network designers are faced with two conflicting goals: reduce both the stretch factor and the size of the routing tables.

For a weak variant of this problem, called *labeled routing*, both lower bounds and asymptotically optimal upper bounds are known (see [TZ01b]). In this version of the problem, the designer of a solution may pick node names that contain (bounded size) information about their location in the network. This variant is useful in many aspects of network theory, but less so in practice: Knowledge of the labels needs to be disseminated to all potential senders, as these labels are not the addresses by which nodes of an *existing* network, e.g., an IP network, are known. Furthermore, if the network may admit new joining nodes, all the labels may need to be re-computed and distributed to any potential sender. Finally, various recent applications pose constraints on nodes addresses that cannot be satisfied by existing labeled-routing schemes. E.g., Distributed Hash Tables (DHTs) require nodes names in the

range $[1..n]$, or ones that form a binary prefix.

In this chapter we assume a network with arbitrary node names and arbitrary edge weights. This model is called the *name-independent* model because the designer of the routing scheme has no control over node names. This routing problem may appear daunting: In order to route to a node, we must first somehow gain knowledge about its location in the network, but we must do so without exceeding the distance to the target.

A fundamental difficulty in all previous schemes is their heavy dependence on the scale of the network. Let the *aspect ratio* $\Delta = \max d(u,v)/\min_{u\neq v} d(u,v)$ be the ratio between the largest distance and the smallest distance, then many schemes require memory that tends to infinity as Δ increases. This suggests that there might exist a lower bound associated with the aspect ratio. However, the best known lower bound for name-independent routing [TZ01b] does not contain such dependence. Hence one would hope to remove the dependence on Δ altogether. We will say that a routing scheme is *scale-free* if its memory requirement is independent of the aspect ratio. Obtaining scale-free schemes is a challenging goal: Until now, the only scale-free schemes for general graphs have exponential stretch [ABNLP89, ABNLP90, ACL+03]. Specifically, when each node stores $\widetilde{O}(n^{1/k})$ bit routing information the best stretch bound achieved is $O(2^k)$. Obtaining such scale free solutions was raised as an open question in [ACL+03, LR05]. In this chapter we fully answer this problem and provide an exponential improvement from $O(2^k)$ to asymptotically optimal $O(k)$ stretch.

5.1.1 Our contribution

We construct for any $k \geq 1$ a routing scheme with linear stretch factor of $O(k)$ and with $\widetilde{O}(n^{1/k})$-bit routing tables per node and $\widetilde{O}(1)$-bit headers[1]. .1.0

Theorem 5.1.1. *For each weighted n-node graph, and integer $k \geq 1$, there is a polynomial time constructible name-independent routing scheme with stretch factor $O(k)$ that uses $O(k^2 n^{1/k} \log^3 n)$-bit routing tables per node.*

5.1.2 Techniques

Broadly speaking, there are two main techniques used to construct routing schemes. The first technique is *random sampling*, in which landmark nodes are selected randomly. This technique has been successful in labeled routing, providing asymptotically optimal space-stretch trade-offs [TZ01b]. For name-independent schemes, random sampling based schemes were used for optimal trade-offs for stretch 3 schemes with $\widetilde{O}(\sqrt{n})$ space [AGM+04b]. However, all general schemes with $\widetilde{O}(n^{1/k})$ space, based on random sampling, obtained exponential stretch $O(2^k)$ (see [ABNLP89, ABNLP90, ACL+03]).

The second technique is *sparse covers*, in which the graph is covered by clusters of bounded diameter such that each node belongs to a small number of clusters. This technique

[1]The notation $\widetilde{O}(\cdot)$ indicates complexity similar to $O(\cdot)$ up to poly-logarithmic factors.

was used for several name-independent schemes [PU89, AP90, AP92, ACL+03, AGM04a]. In all the schemes above that use sparse covers, a cover with clusters of diameter $< 2^i$ is constructed for each $i \in \{1, \ldots, \lceil \log \Delta \rceil\}$, hence these schemes are inherently *not* scale-free.

Our scheme is based on a new decomposition into dense and sparse neighborhoods. It uses a subtle combination of random sampling and sparse cover routing techniques depending on the density or sparsity of each neighborhood. This decomposition allows us to remove any dependence on the aspect ratio. Informally, a sparse neighborhood is one where the number of nodes does not increase to much if the radius is increased by a constant factor. For sparse neighborhoods, we use random sampling techniques that turns out to be efficient in this case. Specifically, nodes maintain a tree-routing scheme for all the nearby landmarks. For dense neighborhoods, we use sparse cover based routing techniques. Since dense neighborhoods imply that the number of nodes is multiplied when the diameter is increased by a constant, the number of dense neighborhood scales a node belongs to is $O(\log n)$. This fact allows to use sparse cover techniques in a scale-free manner. While our decomposition was developed independently of [KLMN04], it bears some similarities to the "measured decent" approach of Krauthgamer et al. [KLMN04]. However, using the "measured decent" approach for routing fails since it chooses scales for each density change. Hence searching on a ball with $\Omega(\log n)$ density changes may incur $O(\log n) \gg k$ stretch which is unacceptable. Our decomposition circumvents this by decomposing both by density change and by diameter change.

Our sparse/dense decomposition technique has interest in its own, as a general approach to remove the aspect ratio parameter in many other constructions. For example for labeled and name-independent routing schemes for networks with low doubling dimension [AGGM05].

5.1.3 Related work

The space-stretch trade-off has been extensively studied under various models and extensions. We refer the reader to Peleg's book [Pel00] and to the surveys of Gavoille and Peleg [Gav01, GP03] for comprehensive background.

Peleg and Upfal [PU89] were the first to study this trade-off in a parameterized manner. For unit cost networks they achieve $O(k)$ stretch with a total of $O(k^3 n^{1+1/k} \log n)$ bits for all routing tables. For weighted networks, Awerbuch et al. [ABNLP89, ABNLP90] present a scale-free routing scheme with exponential stretch of $O(k^2 9^k)$ that requires $\widetilde{O}(n^{1/k})$ bits per node. Arias et al. [ACL+03] improve the stretch to $O(k^2 2^k)$ with the same memory bound.

More recently, constant stretch routing schemes have been designed for networks whose induced metric space has a low doubling dimension [AGGM05, KRX06], and for unweighted graphs excluding a fixed minor [AGM05] (including trees and planar graphs). These schemes require a polylogarithmic space, hidding a multiplicative constant depending on the doubling dimension, or on the minor excluded.

Awerbuch and Peleg use *sparse covers* [AP90] in order to build a hierarchal routing scheme [AP92]. Their scheme is based on tree covers with geometrically increasing radii.

Therefore there is an inherent geometric factor in their memory requirement. They achieve stretch $O(k^2)$ with $\widetilde{O}(n^{1/k}\log\Delta)$-bit routing tables, and $O(\log\Delta)$ headers, where Δ is the maximum weighted distance between any two nodes divided by the minimum weighted distance between any two unique nodes. Abraham, Gavoille, and Malkhi [AGM04a] improve the stretch factor to $O(k)$ with the same memory requirement. This solution is adequate if the network weights are polynomial in the number of nodes. However for arbitrary networks the diameter may be arbitrarily large, for instance $\Delta = \Omega(2^n)$, and solutions based on the aspect ratio of the network may become unusable.

A weaker variant of compact routing is based on the *labeled routing* model. Instead of assuming nodes have arbitrary names, in this model, the network designer is allowed to name the nodes in a topology dependent manner. This paradigm does not provide for a realistic network design, however, the tools devised for its solution have proven useful as building blocks of full routing schemes.

Eilam et al. [EGP03] present a stretch 5 labeled scheme with $\widetilde{O}(n^{1/2})$ memory, whereas Cowen [Cow01] presents a stretch 3 labeled scheme with $\widetilde{O}(n^{2/3})$ memory. Later, Thorup and Zwick [TZ01b] improve to stretch 3 using $\widetilde{O}(n^{1/2})$ bits. These three schemes uses $O(\log n)$-bit node names. Thorup and Zwick also give in [TZ01b] a generalization of their scheme and using techniques from their distance oracles [TZ05], achieve labeled schemes with stretch $4k-5$ (and even $2k-1$ with handshaking) using $\widetilde{O}(n^{1/k})$-bit routing tables and $o(k\log^2 n)$-bit node names. Labeled routing on a trees is explored in [FG01, TZ01b], achieving stretch 1 with $O(\log^2 n/\log\log n)$ bits for local tables and for headers, and this is tight [FG02].

Thorup [Tho04] showed that planar graphs support stretch $1+\varepsilon$ labeled routing schemes with polylogarithmic space. This has been generalized by Abraham and Gavoille [AG06] to graphs excluding a fixed minor with same stretch and space bounds. For low doubling dimension networks, strech $1+\varepsilon$ labeled schemes exist [Tal04b, CGMZ05, Sli05a], but all of them have a dependency in the aspec ratio Δ in the memory bounds.

A variation of our sparse-dense decomposition was recently used to provide some scale-free labeled and name-independent routing schemes for networks with low doubling dimension [AGGM05], this solved a question raised by Slivkins [Sli05a]. While the decomposition in that paper is superficially similar, the techniques used in this chapter are significantly different. Specifically, the sparse level case uses a landmark property of Lemma 5.2.6 together with a new error-reporting tree routing scheme of Lemma 5.3.6. The dense level case is based on applying spares covers of [AP90] with the routing extensions of [AGM04a].

5.2 Sparse and Dense Neighborhood Decomposition

5.2.1 Preliminaries

Given is a weighted graph $G = (V, E, \omega)$ of size $n = |V|$ with a non-negative weight function $\omega : E \to \mathbb{R}^+$. Let the cost of a path be the sum of the weights of its edges. For any two nodes

$u, v \in V$ let $d(u, v)$ denote the cost of a minimum cost path between u and v. Let Δ denote the *aspect ratio* (normalized diameter) of G, $\Delta = \max_{u \neq v} d(u, v) / \min_{u \neq v} d(u, v)$. In order to avoid dragging a normalization constant, from here on assume that $\min_{u \neq v} d(u, v) = 1$. Define the radius r ball around node u, $B(u, r)$, as the set of nodes whose distance is at most r from u, $B(u, r) = \{v \mid d(u, v) \leq r\}$. For any node u, let $T(u)$ denote a minimum cost path spanning tree rooted at u. Given a lexicographic order on the nodes, for any node $u \in V$, set $Z \subseteq V$, and integer $m > 0$ define $N(u, m, Z)$ as the m closest nodes from Z to node u, i.e., as the set $N(u, m, Z) = N$ such that $N \subseteq Z$, $|N| = m$ and for all $x \in N$ and $y \in Z \setminus N$ either $d(u, x) < d(u, y)$ or $d(u, x) = d(u, y)$ and x is lexicographically smaller than y. Let I denote the set $I = \{0, 1, \ldots, \lceil \log \Delta \rceil\}$. Given a parameter k, let K denote the level set $K = \{0, 1, 2, \ldots, k\}$. Each node has an arbitrary unique network identifier consisting of polylog(n) bits. Using standard hashing techniques it is possible to generalize the model and assume nodes have arbitrarily long unique labels.

Our solution is based on using a new decomposition into a series of balls around each node that have a combined combinatorial and geometric restriction. Each ball has at least $n^{1/k}$ more nodes than the previous *and* its radius is at least twice the radius of the previous.

Definition 5.2.1. *For all $u \in V$ and $i \in K$ define the* range $a(u, i)$ *as follows. Let $a(u, 0) = 0$. Then recursively let $a(u, i + 1)$ be the smallest positive integer $j > 0$ such that*

$$|B(u, 2^j)| \geq n^{1/k} |B(u, 2^{a(u,i)})|$$

(or let $a(u, i + 1) = \log \Delta$ if there does not exist such an integer).

For all $u \in V$ and $i \in K$ denote the *neighborhood* ball $A(u, i)$ as the ball who's radius is $2^{a(u,i)}$ around u. Formally, $A(u, i) = \{u\}$ for $i = 0$ and $A(u, i) = B(u, 2^{a(u,i)})$ for $i > 0$.

Intuitively, if the gap between $a(u, i)$ and $a(u, i + 1)$ is small, then the neighborhood $A(u, i + 1)$ is "dense" relative to the neighborhood $A(u, i)$, otherwise $A(u, i + 1)$ is "sparse" relative to $A(u, i)$. A central definition capturing this intuitive notion is the following.

Definition 5.2.2 (Dense level)**.** *For $u \in V$ and $i \in K$, define that i is a* dense level *for node u if*

$$a(u, i) \ < \ a(u, i + 1) \ \leq \ a(u, i) + 3$$

Define that i is a *sparse level* for node u if it is not a dense level. In words, in a dense level, we find at least $n^{1/k}$ times as many nodes as the current level by looking at a ball whose radius is at most 2^3 times the current level.

5.2.2 Dense Levels

For every $u \in V$ define the *range set of node u*, denoted $L(u)$, as $L(u) = \{a(u, i) \mid i \in K\}$ and define the *extended range set* $R(u)$ as,

$$R(u) = \{i \in I \mid \exists a \in L(u), -1 \leq a - i \leq 4\} \ .$$

Define $F(u,i) = B(u, 2^{a(u,i)-1})$. The main property of dense levels is captured in the following lemma.

Lemma 5.2.3 (Dense neighborhoods)**.**
If i is a dense level for u and $v \in F(u,i)$, then $a(u,i) \in R(v)$.

Proof. Recall that $F(u,i) = B(u, 2^{a(u,i)-1})$. Let $v \in F(u,i)$, then $B(v, 2^{a(u,i)-1}) \subseteq A(u,i)$ (see Figure 5.1), and hence

$$|B(v, 2^{a(u,i)-1})| \;\leq\; |A(u,i)| \;.$$

Since $a(u,i+1) \leq a(u,i)+3$, then $B(v, 2^{a(u,i)+4}) \supseteq A(u,i+1)$, and hence

$$|B(v, 2^{a(u,i)+4})| \;\geq\; |A(u,i+1)| \;\geq\; |A(u,i)| \cdot n^{1/k} \;.$$

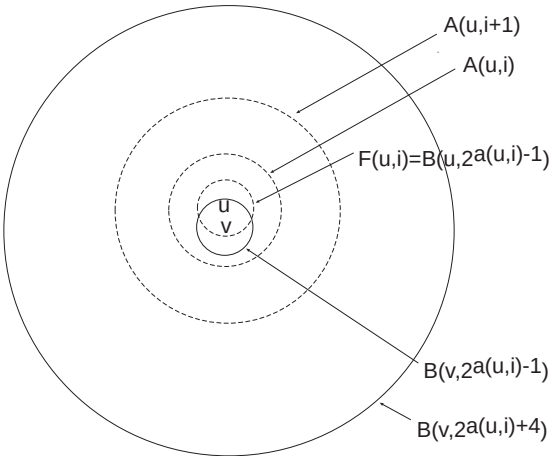

Figure 5.1: Example of a level i dense neighborhood for node u and a node $v \in F(u,i)$.

Together, these imply $|B(v, 2^{a(u,i)+4})| \geq n^{1/k} \cdot |B(v, 2^{a(u,i)-1})|$. Therefore, there exists some index $a(v,j)$ such that $a(u,i) - 1 \leq a(v,j) \leq a(u,i) + 4$. □

5.2.3 Sparse Levels

We use a low discrepancy cover of 'landmark' nodes. We use $k+1$ sets $V = C_0 \supseteq C_1 \supseteq \cdots \supseteq C_k = \emptyset$ of landmarks defined as follows. Let $C_0 = V$. For $i = 1$ to $k-1$ iteratively set C_i to contain each element of C_{i-1} independently, with probability $(n/\ln n)^{-1/k}$. The randomized procedure can be de-randomized using the method of conditional probabilities and pessimistic estimators. Let $\mathcal{B} = \{B(u, 2^i) \mid u \in V, i \in I\}$, note that $|\mathcal{B}| \leq |V|^2$. We will use two simple properties.

Claim 5.2.4. *With high probability, for any $B \in \mathcal{B}$, if $4(\ln n)^{(k-j)/k} n^{j/k} \leq |B|$ for $j \in K$ then $B \cap C_j \neq \emptyset$.*

Proof. Union bound and $\Pr[B \cap C_j = \emptyset] \leq (1-(n/\ln n)^{-j/k})^{4(\ln n)^{(k-j)/k}n^{j/k}} \leq e^{4\ln n}$. □

Claim 5.2.5. *With high probability, for any $B \in \mathcal{B}$, if $|B| < 4(\ln n)^{(k-(j+1)/k)}n^{(j+2)/k}$ for $j \in K$ then $|B \cap C_j| \leq 16n^{2/k}\ln n$.*

Proof. Union bound, Chernoff bound and $E[|B \cap C_j|] \leq 4(\ln n)^{(k-(j+1)/k)}n^{(j+2)/k}(n/\ln n)^{-j/k} \leq 4n^{2/k}(\ln n)^{(k-1)/k}$. □

If $x \in C_j$, and $x \notin C_{j+1}$, define that node x has *rank* j. For every $u \in V$ and $i \in K$ define the *nearby landmarks* $S(u,i)$ to be the $n^{2/k}\log n$ closest nodes in C_i.

$$S(u,i) = N(u, 16n^{2/k}\log n, C_i)$$

and define $S(u) = \bigcup_{i\in K} S(u,i)$. Define $m(u,i)$ as the highest rank of any node in $A(u,i)$. Formally,

$$m(u,i) = \max\{\ell \in K \mid A(u,i) \cap C_\ell \neq \emptyset\} \ .$$

Define the *center* $c(u,i)$ as the closest node to u from $C_{m(u,i)}$. Let $E(u,i) = B(u, 2^{a(u,i+1)}/6)$. The main property of sparse levels is captured in the following lemma.

Lemma 5.2.6 (Sparse neighborhoods)**.**
Let i be a sparse level for u, i.e., $a(u,i+1) > a(u,i)+3$. If $v \in E(u,i)$, then $c(u,i) \in S(v)$.

Proof. Recall that $E(u,i) = B(u, 2^{a(u,i+1)}/6)$ and $m(u,i)$ is the highest rank of any node in $A(u,i)$. Formally,

$$m(u,i) \ = \ \max\{\ell \in K \mid A(u,i) \cap C_\ell \neq \emptyset\} \ .$$

Let $j \in K$ be the index such that $4(\ln n)^{(k-j/k)}n^{j/k} \leq |A(u,i)| < 4(\ln n)^{(k-(j+1)/k)}n^{(j+1)/k}$ then from Claim 5.2.4 it follows that $m(u,i) \geq j$. For any $v \in E(u,i)$, we have (see Figure 5.2)

$$c(u,i) \in A(u,i) \subseteq E(u,i) \subseteq B(v, 2^{a(u,i+1)}/3) \subseteq B(u, 2^{a(u,i+1)}/2) \ .$$

Since level i is sparse for u, by definition there are strictly fewer than $n^{1/k}|A(u,i)|$ nodes in $B(u, 2^{a(u,i+1)}/2)$. Therefore

$$|B(v, 2^{a(u,i+1)}/3)| \ < \ n^{1/k}|A(u,i)| \ \leq \ 4(\ln n)^{(k-(j+1)/k)}n^{(j+2)/k} \ .$$

From Claim 5.2.5 and since $m(u,i) \geq j$ it follows that there are less than $16n^{2/k}\log n$ nodes of rank $m(u,i)$ in $B(v, 2^{a(v,i+1)}/3)$. Since $c(u,i) \in B(v, 2^{a(v,i+1)}/3)$ then $c(u,i) \in S(v)$ as required. □

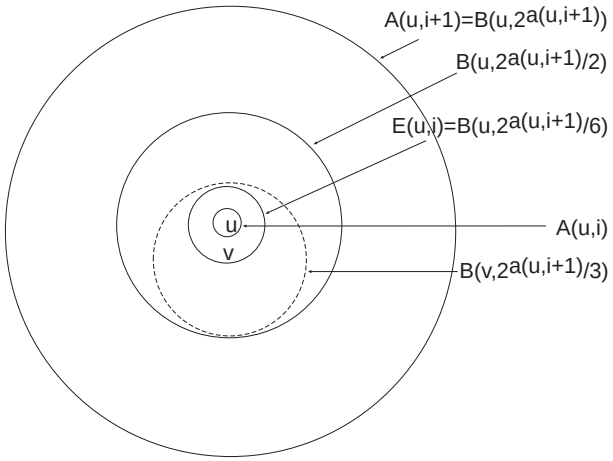

Figure 5.2: Example of a level i sparse neighborhood for node u and a node $v \in E(u,i)$.

5.3 A Scale-Free Routing Scheme

The goal of this section is to present the construction for the following upper bound:

.3.0

Theorem 5.3.6. *For each weighted n-node graph, and integer $k \geq 1$, there is a polynomial time constructible name-independent routing scheme with stretch factor $O(k)$ that uses $O(k^2 n^{1/k} \log^3 n)$-bit routing tables per node.*

The high level view of the routing scheme is a simple iterative protocol. For phases $i = 1$ to k, search for v as follows: If $A(u,i)$ is sparse, use the sparse neighborhood routing strategy. If $A(u,i)$ is dense, use the dense neighborhood routing strategy. We begin by describing the two routing strategies.

5.3.1 Sparse neighborhood routing strategy

For every center $c(u,i)$ define $T(c(u,i))$ as a minimum cost path tree rooted at $c(u,i)$ that spans all nodes v such that $c(u,i) \in S(v)$. Routing on these trees is done using the following name-independent tree-routing scheme, which is an enhancement of Laing's algorithm [Lai04].

.3.5

Lemma 5.3.6. *For any $k \geq 1$, and for any weighted tree $T = (V, E, \omega)$ and for any designated root $r \in V$, there exists a name-independent error-reporting tree-routing scheme with the following properties:*

1. *Each node stores $O(kn^{1/k} \log^2 n)$ bits of routing information.*

2. *For any $j \in K$, the root can perform a* j-bounded search *for destination v.*

   A j-bounded search for v has the following properties:

   (a) *If $v \in N(r, n^{j/k}, V)$ then it reaches v with stretch $2j-1$;*

   (b) *Otherwise it returns a negative response to the root incurring a cost of at most $(2j-2)\max\left\{d(r,v) \mid v \in N(r, n^{(j-1)/k}, V)\right\}$.*

Proof. Given a tree $T = (V, E)$ with $m \leq n$ nodes and a root $t \in T$. We give each node $v \in V$ three names. Let $a_0, \ldots, a_{m-1}$ denote the nodes of T sorted by increasing distance from the root. Formally, for any two indexes, if $i < j$ then $d_T(r, a_i) < d_T(r, a_j)$ or $d_T(r, a_i) = d_T(r, a_j)$ and a_i is lexicographically smaller than a_j. The first name we give nodes, called their primary name, makes use of words of the alphabet $\Sigma = \left\{0, 1, 2, \ldots n^{1/k} - 1\right\}$. Specifically, name $a_0 = r$ as the empty word (ε). Then name $a_1, \ldots, a_{n^{1/k}}$ respectively with one digit in Σ in increasing order $(0), (1), \ldots, (n^{1/k} - 1)$ respectively. Then name each of the next $n^{2/k}$ nodes $a_{1+n^{1/k}}, \ldots, a_{1+n^{1/k}+n^{2/k}}$ by a name in Σ^2 in increasing lexicographic order, $(0,0), (0,1), \ldots, (0, n^{1/k}-1), (1,0), (1,1), \ldots, (1, n^{1/k}-1), \ldots, (n^{1/k}-1, 0), \ldots, (n^{1/k}-1, n^{1/k}-1)$ respectively. Let $j_i = \sum_{j=0}^{i} n^{j/k}$. Continue this naming process, naming nodes $a_{j_{i-1}+1}, \ldots, a_{j_i}$ respectively by a name in Σ^i until all nodes in T are exhausted, up to at most a k-digit node name in Σ^k. For $0 \leq j \leq k$ let V_j denote the set of nodes whose name contains at most j digits. Next, we give v its name based on the labeled tree routing of Thorup and Zwick [TZ01b] and Fraigniaud and Gavoille [FG01]:

Lemma 5.3.7. **[FG01, TZ01b]** *For every integer $k > 1$ and every weighted tree T with m nodes there exists a labeled routing scheme that, given any destination label, routes optimally on T from any source to the destination. The storage per node is $O(m^{1/k} \log m)$ bits, the label size, and the header size are $O(k \log m)$ bits.*

For a tree T containing a node v, let $\mu(T, v)$ denote the routing information of node v and $\lambda(T, v)$ denote the destination label of v in T as required from Lemma 3.3.4. We require from each node v to store $\mu(T, v)$. The second name we assign v is $\lambda(T, v)$. The third node-name makes use of a hash function $h : T \to \Sigma^k$. We require that, for all $0 \leq j \leq k$, $\max_{u \in \Sigma^{j-1}} |\{v \in V_j \mid u \text{ is a prefix of } h(v)\}| \leq |\Sigma| \log n = n^{1/k} \log n$. This requirement can be fulfilled with high probability using a $\Theta(\log n)$-wise independent hash function that requires $\Theta(\log^2 n)$ bits of storage [CW79, MR95]. A node $u \in V$ with name $(x_1, \ldots, x_j)$ stores:

1. Information for labeled tree routing $\mu(T, u)$. This requires $O(n^{1/k} \log n)$ bits.
2. The labels $\lambda(T, v)$ of all the nodes v whose name is $(x_1, \ldots, x_j, y)$ for all $y \in \sigma$. This requires $O(kn^{1/k} \log n)$ bits.
3. The map $v \to \lambda(T, v)$ of the $n^{1/k} \log n$ closest nodes v (from the root) whose first j indexes of their hash $h(v)$ equal $x_1, \ldots, x_j$. Formally, define $Z = \{z \mid h(z)[1 \ldots j] = x_1, \ldots, x_j\}$ and store $v \to \lambda(T, v)$ for all $v \in N(r, n^{1/k} \log n, Z)$. This requires $O(kn^{1/k} \log^2 n)$ bits.

To search from the root r for a node t whose hash is $y_1, \ldots, y_k$ on a j-bounded search, do the following:

1. $current := r$; $round := 1$;
2. if $round = j$ and $current$ does not know of t then return to root r with negative response.
3. Otherwise if $current$ knows of t's label then route to t.
4. Otherwise route to the node whose name is $(y_1, \ldots, y_{round})$, set $round := round + 1$, and update $current$ to be the current node.
5. goto 2;

Suppose the destination is a node t whose hash is $y_1, \ldots, y_k$ and whose name has length i ($t \in V_i$ and $t \notin V_{i+1}$). The destination will be found after at most i iterations of the algorithm. This is true since if $i = 1$, then the root knows about the destination. Otherwise, at the $i-1$th iteration we reach the node x whose name is $(y_1, \ldots, y_{i-1})$ and due to the properties of the hash function we know that $|\{v \in V_i \mid (y_1, \ldots, y_{i-1}) \text{ is a prefix of } h(v)\}| \leq |\Sigma| \log n$ and hence x stores the label of t.

Since all the nodes that are visited have smaller names than i, it is easy to see that the stretch is bounded by $2i - 1 \leq 2k - 1$.

If a j-bounded search is performed and $j < i$ then t may not be found. At the $(j-1)$th iteration, a node whose name is $(y_1, \ldots, y_{j-1})$ will report the error to the root. Since all nodes visited have names with at most $j - 1$ digits then the total cost is bounded by $(2j - 2) \max_{v \in V_{j-1}}\{d(r, v)\}$.

The storage per node v is $O(\log^2 n) + O(|\Sigma| \log n) + O(k|\Sigma| \log n) + O(k|\Sigma| \log^2 n) = O(kn^{1/k} \log^2 n)$, for the hash function h, routing information $\mu(T, v)$, the primary name entries, and the hash name entries respectively. □

For all $u \in V$ and $i \in K$ recall that $E(u, i) = B(u, 2^{a(u,i+1)}/6)$. Given a sparse level i the algorithm routes to the root $c(u, i)$. We prove that $E(u, i) \subseteq T(c(u, i))$ and hence searching from $c(u, i)$ for $v \in E(u, i)$ on the tree $T(c(u, i))$ will succeed with stretch of at most $2k - 1$. In order to bound the cost incurred if $v \notin E(u, i)$ every node $u \in V$ stores for every $i \in K$ the index $b(u, i)$. Where $b(u, i)$ is the minimal integer j such that a j-bounded search on $T(c(u, i))$ of any node in $E(u, i)$ succeeds. Since $b(u, i)$ is defined as the minimal index to find in $T(c(u, i))$ all nodes of $E(u, i)$ then we prove that a negative result from this search has cost proportional at most to the diameter of $E(u, i)$. More precisely, we now define the information stored by every node, and the routing algorithm for sparse neighborhoods.

5.3.2 Storage for sparse neighborhood strategy

For any tree T and node $u \in T$, let $\tau(T, u)$ denote the information stored on u that is induced by the name-independent tree-routing scheme of Lemma 5.3.6. Every node $u \in V$ stores $\tau(T(v), u)$ for all $v \in S(u)$. In addition, for every $i \in K$, u records $c(u, i)$ and $b(u, i)$.

5.3.3 Routing algorithm for sparse neighborhood strategy

1. route to the root $c(u,i)$.
2. Perform a $b(u,i)$-bounded search on $T(c(u,i))$ for destination v.
3. If destination is not found, continue to iteration $i+1$.

5.3.4 Dense neighborhood routing strategy

If i is a dense level, the routing strategy uses tree covers that have a bounded radius. A source of difficulty in arbitrarily weighted graphs is that the logarithm of the diameter may be arbitrarily large ($\gg n$, for example $\Delta = 2^n$). Hence, tree covers of geometrically increasing radii, e.g., as used in [AP90, AGM04a], require each node to participate in too many partition levels any may require $\Omega(n)$ bits per node. Our solution for general graphs is to let each node maintain routing information only for a limited number of radii that surround its range set. However, special care must be taken to make sure that the destination and all the nodes along the way store information for the same radius that the source is using. Recall that the range set is defined as $L(u) = \{a(u,i) \mid i \in K\}$, and the *extended range set* $R(u)$ is defined as, $R(u) = \{i \in I \mid \exists a \in L(u), -1 \leq a - i \leq 4\}$. For every $i \in I$ define $G_i = (V_i, E_i)$ as the subgraph induced by the nodes $V_i = \{u \mid i \in R(u)\}$. We prove that in a dense level i for u, if v is in $B(u, 2^{a(u,i)-1})$ then $i \in R(v)$ and moreover both source and destination are connected in G_i. Hence a tree cover in G_i may be used for routing. The tree cover is built using the construction of [AP90] with the improvements of [AGM04a].

Lemma 5.3.8. **[AP90, AGM04a]** *For every weighted graph $G = (V, E, \omega)$, $|V| = n$ and integers $k, \rho \geq 1$, there exists a polynomial algorithm that constructs a collection of rooted trees $\mathcal{TC}_{\kappa,\rho}(G)$ such that:*

1. *(Cover) For all $v \in V$, there exists $T \in \mathcal{TC}_{\kappa,\rho}(G)$ such that $B(v,\rho) \subseteq T$.*
2. *(Sparse) For all $v \in V$, $|\{T \in \mathcal{TC}_{\kappa,\rho}(G) \mid v \in T\}| \leq 2kn^{1/k}$.*
3. *(Small radius) For all $T \in \mathcal{TC}_{\kappa,\rho}(G)$, $\mathrm{rad}(T) \leq (2k-1)\rho$, where $\mathrm{rad}(T) = \max_u \{d_T(r,u)\}$.*
4. *(Small edges) For all $T \in \mathcal{TC}_{\kappa,\rho}(G)$, $\mathrm{maxE}(T) \leq 2\rho$, where $\mathrm{maxE}(T) = \max_{e \in E(T)} \{\omega(e)\}$.*

For every $i \in I$ we build a tree cover $\mathcal{TC}_{k,2^i}(G_i)$ only on the graph G_i (note that G_i may have several connected components, so a tree cover is built for each connected component separately). Since $|R(u)| = O(k)$ then every node u participates only in $O(k)$ such tree covers. For all $u \in V$ and $i \in K$, denoting $j = a(u,i)$, define $W(u,i) \in \mathcal{TC}_{k,2^j}(G_j)$ to be the tree such that $B(u,2^j) \subseteq W(u,i)$. On a dense level i, routing towards a destination begins by routing to the root of the tree $W(u,i)$ and then using a name-independent tree-routing scheme. For all $i \in I$ and $T \in \mathcal{TC}_{\kappa,\rho}(G_i)$ we use the name-independent error-reporting tree-routing scheme of [AGM04a] with an improved analysis.

Lemma 5.3.9. *For every tree $T = (U, E, \omega)$, $|U| = m$, $U \subset V$, $|V| = n$, and integer k there exists a name-independent tree-routing scheme on T with error-reporting that routes on paths of length bounded by $4\mathrm{rad}(T) + 2k\mathrm{maxE}(T)$, each node requires $O(kn^{1/k} \log n)$ memory bits, and headers are of length $O(\log^2 n)$. Moreover, routing for a non-existent name in T also incurs a (closed) path of length $4\mathrm{rad}(T) + 2k\mathrm{maxE}(T)$ until a negative result is reported back to the source.*

sketch. We use the same construction of [AGM04a], the only change is to use Lemma 3.3.4 that requires $O(k \log n)$ bit labels instead of $O(\log^2 n / \log \log n)$ bit labels used in [AGM04a]. Since each node stores only one such label, the space requirement is only $O(kn^{1/k} \log n)$ bits (all other labels used require only $O(\log n)$ bits). □

We prove that if i is a dense level for u then $\forall v \in F(u, i)$ we have $i \in R(v)$ (Lemma 5.2.3) and hence the tree routing scheme on $W(u, i)$ will reach any node in $F(u, i)$ or report a negative response at a cost proportional to k times the radius of $F(u, i)$ (recall $F(u, i) = B(u, 2^{a(u,i)-1})$). We now define the information stored by every node, and the routing algorithm for dense neighborhoods.

5.3.5 Storage for dense neighborhood strategy

For any tree T and node $u \in T$, let $\phi(T, u)$ denote the information stored on u that is induced by the name-independent error-reporting tree-routing scheme of Lemma 5.3.9. For every $u \in V$ and $i \in R(u)$ node u stores $\phi(T, u)$ for all $T \in \mathcal{TC}_{k,2^i}(G_i)$ such that $u \in T$. In addition, every node $u \in V$ records $w(u, i)$ the root of the tree $W(u, i)$.

5.3.6 Routing algorithm for dense neighborhood strategy

1. Route to the root $w(u, i)$.
2. Route on the tree $W(u, i)$ to either find destination v or get a negative response.
3. If destination is not found, continue with iteration $i + 1$.

5.3.7 Analysis

Throughout the description below, the source node is denoted u and the destination node is denoted v. Routing from u to v is done by iteratively expanding the search through the neighborhoods of u, $A(u, 1), A(u, 2), \ldots, A(u, k)$ until the destination is found. The routing strategy in each neighborhood $A(u, i)$ depends on whether level i is sparse or dense for node u. If $A(u, i)$ is sparse, we use the sparse neighborhood strategy. If $A(u, i)$ is dense we use the dense neighborhood routing strategy.

The following technical lemmata are used to prove Theorem 5.1.1.

Lemma 5.3.10. *Let i be a dense level for u, and let $v \in F(u,i)$. Then a dense neighborhood routing strategy from u will reach v.*

Proof. Recall that $R(u) = \{i \mid \exists \ell \in L(u), -1 \leq \ell - i \leq 4\}$. Denote $a(u,i) = j$, by the properties of dense neighborhood stated in Lemma 5.2.3, for every $x \in F(u,i)$ node x has $a(u,i) \in R(x)$ and hence $x \in G_j$. Recall that the tree $W(u,i) \in \mathcal{TC}_{k,2^j}(G_j)$ is defined to be the tree such that $B_{G_j}(v, 2^j) \subseteq W(u,i)$. Therefore, $F(u,a) \subseteq W(u,i)$ and it is possible to reach v by routing on $W(u,i)$. □

Lemma 5.3.11. *Let $i \in K$ be a dense level for $u \in V$. Then the dense neighborhood routing strategy requires $O(k^3 n^{2/k} \log n)$ memory bits and incurs a cost of $O(k \cdot 2^{a(u,i)})$ either to reach v if $v \in B(u, 2^{a(u,i)-1})$ or otherwise to return a negative response to the source u.*

Proof. Storage: For the dense level routing strategy every node maintains $O(k)$ ranges. For each range, every node participates in $O(kn^{1/k})$ trees, each tree requires $O(kn^{1/k} \log n)$ bits. For a total of $O(k^3 n^{2/k} \log n)$ bits.

Routing cost: By combining Lemma 5.3.8 and Lemma 5.3.9 the cost of searching for v on $G_{a(u,i)}$ is at most $O(k \cdot 2^{a(u,i)})$. From Lemma 5.3.10, the destination will actually be found on $G_{a(u,i)}$ if $v \in F(u,i) = B(u, 2^{a(u,i)-1})$. □

Lemma 5.3.12. *Let i be a sparse level for u, and let $v \in E(u,i)$. Then the sparse neighborhood routing strategy from u will reach v.*

Proof. Denote $c = c(u,i)$, and recall that c is the closest landmark of highest rank in $A(u,i)$. By the properties of sparse neighborhoods stated in Lemma 5.2.6, every node $v \in E(u,i)$ has $c \in S(v)$. Hence, all nodes in $E(u,i)$ store $\tau(T(c))$, including all nodes on the shortest path of $T(c)$ from u to c. Since $b(u,i)$ is defined so that a $b(u,i)$-bounded search on $T(c)$ will find all nodes in $E(u,i)$ then $v \in E(u,i)$ will be found. □

Lemma 5.3.13. *Let $i \in K$ be a sparse level for $u \in V$. Then the sparse neighborhood routing strategy requires $O(k^2 n^{3/k} \log^3 n)$ memory bits and incurs a cost of $O(k \cdot (d(u,v) + 2^{a(u,i)}))$ to reach v if $v \in B(u, 2^{a(u,i+1)}/6)$ or otherwise returns a negative response to the source at a cost $O(k \cdot 2^{a(u,i+1)})$.*

Proof. Storage: For the sparse level routing strategy every node maintains k closest landmark sets $S(u,1), \ldots, S(u,k)$. For each set, a node participates in $16n^{2/k} \log n$ trees, each tree requires $O(kn^{1/k} \log^2 n)$ bits. For a total of $O(k^2 n^{3/k} \log^3 n)$ bits.

Routing cost: If $v \in E(u,i) = B(u, 2^{a(u,i+1)}/6)$, then by Lemma 5.3.12 the routing strategy will route from u to $c(u,i)$ and then to v. By definition $c(u,i) \in A(u,i)$ so the path to $c(u,i)$ incurs a cost of at most $d(u,c(u,i)) \leq 2^{a(u,i)}$. Then by *Lemma* 5.3.6 the $b(u,i)$-bounded search will find v at cost at most $(2k-1)d(c(u,i),v) \leq (2k-1)(d(u,v) + d(u,c(u,i))) \leq (2k-1)(d(u,v) + 2^{a(u,i)})$. So the total cost is bounded by $2^{a(u,i)} + (2k-1)(d(u,v) + 2^{a(u,i)}) = O(k(d(u,v) + a(u,i)))$ as required. Otherwise if $v \notin E(u,i) = B(u, 2^{a(u,i+1)}/6)$ then the round-trip cost of reaching $c(u,i)$ and returning

is at most $2^{a(u,i)+1}$. Then by Lemma 5.3.6 the cost of a $b(u,i)$-bounded search is bounded by $(2k-2)2^{a(u,i+1)}/6$. This is true since $b(u,i)$ is defined so that all the $n^{b(u,i)/k}$ closest nodes in $T(c(u,i))$ all belong to $E(u,i)$. Hence the total cost for a negative answer is $2^{a(u,i)+1} + (2k-2)2^{a(u,i+1)} = O(k \cdot 2^{a(u,i+1)})$. □

We now prove the main theorem.

of Theorem 5.1.1. For storage, by combining Lemma 5.3.13 and Lemma 5.3.11 it follows that every node stores $O(k^2 n^{3/k} \log^3 n)$-bit routing tables. For stretch analysis, let $i \in K$ be the first iteration index in which v is found. There are two cases to consider.

1. If level $i-1$ is sparse for u, then given that v is not found in iteration $i-1$, by Lemma 5.3.13, v is not inside $E(u, i-1)$, and hence, $d(u,v) \geq 2^{a(u,i)}/6$.
2. Otherwise, level $i-1$ is dense for u, and again, v is not found in iteration $i-1$. In this case, by Lemma 5.3.11, v is not inside $F(u, i-1) = B(u, 2^{a(u,i-1)}/2)$. By density, $a(u,i) \leq a(u,i-1)+3$. Putting these two facts together, we have $d(u,v) \geq 2^{a(u,i-1)}/2 \geq 2^{a(u,i)-3}/2$.

In either case, $2^{a(u,i)} = O(d(u,v))$. From Lemma 5.3.13 and Lemma 5.3.11, reaching v on level i will cost $O(k \cdot 2^{a(u,i)})$ if i is dense or $O(k(d(u,v) + 2^{a(u,i)}))$ if i is sparse. Hence in both cases, level i searches are bounded by $O(k \cdot d(u,v))$. For the cost of the negative responses, note that the highest level that fails is $i-1$. From Lemma 5.3.13 and Lemma 5.3.11 the cost of a negative response for level j is either $O(k \cdot 2^{a(u,j)})$ for a dense level, or $O(k \cdot 2^{a(u,j+1)})$ for a sparse level. Hence, the total cost of negative response is at most $\sum_{j=0}^{i-1} O(k \cdot 2^{a(u,j+1)}) = O(k \cdot 2^{a(u,i)}) = O(k \cdot d(u,v))$. □

5.4 Conclusion

Our routing scheme can be adopted to work on strongly connected directed graphs, this extension will appear in the full paper. Our upper bounds have asymptotically optimal stretch with poly-logarithmic storage overhead. There are two natural open questions. First, what is the exact lowest stretch obtainable with a $\widetilde{O}(n^{1/k})$ memory? Even for the labeled case, the bound is not known to be tight. Second, what is memory requirement for schemes with $\Theta(\log n)$ stretch? Our upper bounds requires $O(\log^5 n)$ bits in such cases. We believe at least one logarithm can be removed by improving the hash function used by Lemma 3.3.4.

Chapter 6

LLS : Name-Independent Routing for Mobile Ad Hoc Networks

6.1 Introduction

In the widely used geometric ad hoc Unit Disk Graph model each node knows its location on the Euclidean plane and it can communicate with all other nodes whose distance is at most one unit. Consider the following natural question for mobile networks; A source node s wants to initiate a communication session with a destination node t. The main problem is that node s has no a priori knowledge of t's current location. Thus, the first step in establishing a connection is to locate t's whereabouts. Once the location of the destination is discovered the source can route messages and establish communication using well known geometric routing algorithms. A *location service for ad hoc networks* is a fundamental building block that allows any source s to know the location of any destination t.

More generally, a geometric location service is useful in any Euclidean metric space: one needs to find the coordinates of targets in order to route toward them. Using geometric coordinates in general networks is made relevant not only by the ubiquity of GPS devices, but also by several recent techniques that embed internet nodes in a coordinate space. One of the pioneering mechanisms to predict network latency is based on the work of Ng and Zhang [NZ02]. They embed the Internet latencies into a virtual geometric space (e.g., 3-D Euclidean) and characterize the position of any node with coordinates. The computed distances are used to predict the actual network distances. Following [NZ02] other schemes have been developed to improve the embedding of internet hosts into virtual geometric spaces, e.g., [GSG02], [WSPC03], [CDK+03], and [ST03]. Although the principles of LLS are applicable for general Euclidean spaces, the remainder of this chapter focuses on describing LLS for ad hoc mobile networks.

In order to formally assess the utility of a geometric location service, we define formal measures of efficiency. Let the cost of a path be the sum of the costs of its edges. An important measure of a location service is its *lookup cost*: Given a source node s and destination

node t, the cost of locating t from s is the cost of the path induced by the location service until the location of t known. A *locality aware lookup algorithm* is an algorithm whose lookup cost is proportional to the cost of routing between s and t when the location of the destination is known.

For mobile networks, whenever a node changes its location it must update the location service. Therefore, another important measure of a location service is its *publish cost*: Given that a node moves from location x to a new location y, the publish cost is the total cost of the paths induced by the publish algorithm. A *locality aware publish algorithm* is one whose cost is proportional to the distance between the old location x and the new location y.

6.1.1 Our Results

We present a Locality aware Location Service named LLS. Our location service is the first location service which has both worst case guarantees and average case efficiency. For worst case networks our lookup incurs a lookup cost of $O(d^2)$ for a source and a destination whose minimal cost path has length d.

For networks in which the expected ad hoc routing costs Δ times the distance from source to destination, LLS achieves in addition an average case linear cost over the distance, $O(d)$. Thus, in average case networks of nodes randomly positioned in the plane, where the routing cost is proportional to the distance from source to destination, lookup in LLS costs only a constant factor more than the distance between the source and the destination.

Our scheme is also the first to provide guaranteed average case efficient publishing. That is, our service ensures that the expected cost of updating the data structures due to a node's movement is bounded as a function of the distance of the movement. Specifically, when a node moves distance d, the average cost of publishing its new location is $O(d \log d)$.

The inherent locality of our scheme makes it fault tolerant both to node failure and to network partitions. When the network partitions, nodes within a connected component can locate each other because the location service is also collocated with them within the component. As for node failures, when some node containing location information fails, there is sufficient redundancy at incrementally increasing distances in the network to transparently make up for it.

6.1.2 Related work

Geometric ad hoc routing. The first step in our approach, is a routing algorithm for known geometric locations. The first routing algorithm to guarantee delivery is [KSU99] (Kranakis et al.). Their face routing algorithm has no bound on the ratio between the cost of route and the cost of the minimal cost path. Both Bose et al. [BMSU01] (CGF) and Karp and Kung [KK00] (GPSR) propose an algorithm that combines greedy routing with face routing. These algorithms guarantee delivery and any source destination pair have expected cost $O(d)$, for average case networks, where d is the distance between the source and the destination. The

first algorithm that gives worst case guarantees is by Kuhn et al. [KWZ02]. They present a scheme in which, if the minimal cost path has cost d, then delivery with cost $O(d^2)$ is guaranteed, which is asymptotically optimal. In a follow up paper [KWZ03], they combine their bounded face routing with greedy routing to achieve a scheme that is both worst case asymptotically optimal and average case efficient. In our construction, we make use of an underlying geometric routing protocol, and assume both linear average case behavior and quadratic worst case.

Location algorithms. Location algorithms are measured by their lookup and publish costs. The basic location services studied by Camp et al. [CBW02] either have an unbounded publish cost that may require to flood the whole network with updated location information (DLS, SLS) or have an unbounded lookup cost that may require to flood the whole network to find the destination (RLS). A standard technique for name services in wireless cellular networks (e.g., ISA-4 [ISA], GSM MAP [MP92]) employs a *home location register* (HLR) for each mobile host. A publish algorithm stores the whereabouts of a node at its home location. The lookup first routes to the home location, and from there to the current destination of the node. However, even this simple approach is challenging in a mobile ad hoc network, since there is no fixed infrastructure and the location servers themselves are dynamic.

One of the first approaches to address that in ad hoc networks, by Hubaux et al. [HLBTG01], is to define the home of a node as a geometric area, and have all nodes in that area store location information. A similar solution for finding the home location is suggested in the context of sensor networks in the Geographic Hash Table (GHT) of [RKL+02]. In their approach, a home location is defined as a virtual coordinate. They enhance the underlying routing to reach the closest node to the virtual point. A similar concept is employed in the GeoQuorums of [DGL+], where geometric coordinates determine the location of home servers. In GeoQuorums, these *focal point* coordinates define geographic areas that must be inhibited by at least one server at any time. The drawback of *all* of the home-based approach is that the cost of the lookup and of the publish may be arbitrarily high compared to the optimal path between source and destination.

In order to provide for better scalability and alleviate the problem of reaching specific location servers, several works suggest to replicate home location servers using quorum systems for availability and load balancing. Among these, the works of [PS97, KMP99, HLV03, HL99] have no locality awareness. Other quorum based location services addressed locality in a partial way.

One of the early locality-aware location services that employs a hierarchy of partitions is provided for the general problem of object location in graphs by Awerbuch and Peleg [AP95]. Their solution does not make use of the geometric structure of ad hoc mobile networks, nor address their high dynamism. Consequently, their solution is not easily adaptable to dynamic mobile settings. In addition, their locality factors are somewhat large (polylogarithmic).

The approach taken by several works, e.g., in [Sto99, NN02, AS02, TV04], for quorum construction makes use of the planar structure of ad hoc networks. It defines a write quorum for updating location information of a node as a *column* of some choice trajectory, and

potential some choice thickness. Similarly, a read quorum for querying location information is a row (of a choice trajectory and thickness). Trajectories are determined such that in average density networks, read and write quorums are likely to intersect. This method has good average case locality for lookup, but its publish cost is always the full diameter of the network, thus not proportional to the size of the movement. In addition, there are extreme cases in which read and write quorums might not intersect.

The *position based multi-zone* routing method of Amouris et al. [APL99] stores location information about each node in geometrically increasing discs, each disc referencing the smaller disc that contains the node. When a node moves a distance 2^i, it broadcasts an update about the change to an area of radius 2^{i+1}. Thus, both lookup and publish have locality awareness. The drawback of the scheme is that within a 2^i zone, location update is flooded to all nodes. This implies that each node in the network needs to maintain information (albeit not accurate) about every other node.

One of the pioneering works on efficient and scalable location services is by Li et al. in [LJD+00]. Similarly to the multi-zone method of [APL99], GLS utilizes a hierarchy of exponentially decreasing sets of regions (GLS uses squares rather than discs) that cover the plane. Every node belongs to only $\log M$ squares (were M is the diameter of the network). Using ingenious techniques drawn from the consistent hashing approach [KLL+97], every node has a designated hashed location server within each square, thus distributing the load of location services across the network. The path taken by a GLS lookup operation is bounded inside the minimal square that contains both the source and the destination.

Yet GLS does not achieve either of our goals, and supports neither worst case locality aware lookup, nor locality aware publish. There are several reasons for that, none of which is trivial to fix. First, their scheme makes little effort to proactively handle updates and out of date information. This problem arises when, for instance, a node crosses a grid boundary line. This causes a change in the role the node plays as a location server for others, performing this change of roles may cause the publish cost to be arbitrarily high compared to the distance taken. The authors state that indeed a remaining open question is improving the handling of node mobility. Second, there may be a source s and destination t that are arbitrarily close to each other, but the smallest square that contains both of them is arbitrarily large. As a consequence, even in average case networks, the cost of lookup does not have worst case bounds. Third, within each square GLS routes to a location server in order to find the target. In extreme network conditions when the network is sparse, routing to the location server, even if close by, could require worst case cost, while routing directly to the destination could be efficient. Thus, the worst case lookup cost could be arbitrarily high and have no worst case locality guarantees. This degraded performance in sparse networks was also observed by Guba and Camp [GC02].

A totally different approach focusing on worst case analysis is discussed in the Conclusion section of [KWZ02]. The authors describe an algorithm that we name the Iterative Bounded Flooding (IBF) algorithm. This algorithm runs in phases beginning with phase 1 and incrementing the phase by one until the destination is found. At phase i, the algorithm

floods the network to all nodes whose minimal cost path from the source is at most 2^i. IBF is asymptotically worst case optimal. Specifically, if the minimal cost of the path from source to destination is d then IBF guarantees to reach the destination in cost $O(d^2)$. The main drawback of this approach is that its average cost is also $\Omega(d^2)$.

Recently, in the context of sensor networks and a slightly different model, Demirbas et al. [DANL04], achieve $O(d \log d)$ move time and $O(d)$ find time.

6.1.3 Technical approach

For each destination node t, we define a virtual hierarchical cover of the $M \times M$ plane consisting of exponentially decreasing squares, whose origin depends on the node's id. Our solution is built incrementally in three steps, *Spiral*, *Spiral-Flood*, and *LLS*. In our basic Spiral algorithm, we publish t's location on a spiral that spans increasingly large squares in the hierarchy, and likewise, search for t in increasing spirals on the same virtual hierarchy. The lookup and publishing paths are guaranteed to cross at the first hierarchy level in which the squares containing the source and the destination intersect. This cover bears resemblance to the hierarchical grid of GLS [LJD+00]. However, we address mobility with techniques borrowed from GHT [RKL+02] by using virtual coordinates within the squares for storing information on t rather than search for certain pre-designated nodes. This allows us to search in four grid squares around each point, and circumvent grid-boundary problems. Unlike GLS, our hierarchical cover forms a subsuming partition of the plane that is similar in spirit to the work on sparse partitions of Awerbuch and Peleg [AP90].

For most networks, this scheme suffices to have a lookup that is within an expected constant factor over the most optimal route. In order to further address worst case scenario, in our Spiral-Flood scheme we carefully interweave depth-bounded flooding stages with spiral lookup stages. This only increases the total cost by a constant factor, yet provides worst case quadratic location guarantee.

Finally, the full LLS scheme addresses publishing in the following way. First, we modify Spiral (or Spiral-Flood) so that within each lattice in the hierarchy, we publish a node's location not only in the square containing it, but also in the eight squares surrounding it. When a node moves within the nine square boundary, nothing happens. The stale information may eventually lead to a square that does not contain the node, but then one of the surrounding squares does. Location information in a lattice is updated only when the node leaves the nine-square boundary, at which time the distance traversed by the node (possibly over multiple steps) already exceeds the diameter of one square. As a consequence, we prove that the amortized cost of information updating due to a cumulative movement of distance d is $O(d \log d)$.

In summary, our LLS scheme provides the first scheme which has both locality aware lookup and locality aware publish. Lookup is linear in average case networks, and is quadratic in the worst case, and publish is sub-quadratic.

6.2 Model and Notations

Consider a set of n nodes, V, that exists in the Euclidean plane $\mathbb{R}^2$. Each node v has a unique name denoted $v.id$. Denote by $|uv|$ the L_2 distance between the location of the nodes u and v. The underlying network is formed by a Unit Disk Graph $UDG = \langle V, E \rangle$ in which $u, v \in V$ have an edge $(u, v) \in E$ iff $|uv| \leq 1$.

We assume a nondecreasing cost function c, mapping edge lengths to real numbers. Formally, $c : [0, 1] \mapsto \mathbb{R}^+$, and for all $0 \leq x \leq y \leq 1$ we have $c(x) \leq c(y)$. This cost function abstraction generalizes the three common cost measures: hop count $c(x) = 1$, Euclidean distance $c(x) = x$, and energy $c(x) = x^\alpha$ for $\alpha \geq 2$. Given a path $p = u_1, u_2, \ldots, u_k$ such that $(u_i, u_{i+1}) \in E$ define the cost of the path to be $c(p) = \sum_{i=1}^{k-1} c(|u_i u_{i+1}|)$. Given two nodes u, v let $d(u, v)$ denote the cost of the minimal cost path in UDG from u to v.

In order to be able to obtain worst case bounds on the cost of geometric routing we assume $\Omega(1)$ density. In this model there exists a minimum length d_0 such that for any network $\min_{u,v} |uv| \geq d_0$. Following the results in [KWZZ03] it is also possible to remove the $\Omega(1)$ density assumption if the cost function, c, is linearly bounded. Formally c is *linearly bounded* if there exists a constant m such that $\forall x \in [0, 1] : \quad c(x) \geq mx$. For linearly bounded cost functions it is possible to construct a connected dominating set backbone. Routing can be done on this backbone, which, by its nature, has bounded density. We note however that maintaining this backbone in a dynamic network incurs additional costs.

We assume all nodes are located inside a bounded square of size $M \times M$. Without loss of generality we normalize the coordinates so that all nodes, V, are inside the square whose diagonal corners are $(0, 0), (M, M)$. We further assume that each node knows its own location in the plane and its neighbors' locations.

6.2.1 Virtual Coordinates

A recurring issue in our scheme is the use of virtual coordinates that map to real nodes. The points are called virtual, since there is no guarantee that there is a node exactly at these points.

The natural way to map between nodes and virtual points is to partition the plane using a Voronoi diagram (see [OBSC00] for a survey on Voronoi diagrams and their applications). This partition maps each point p in the square $M \times M$ to the node closest to p. For a node x let $A(x)$ be the Voronoi polygon of x (the area of all the points to which x is the closest node). For any point $p \in \mathbb{R}^2$ let $\ell(p)$ be the closest node to p, thus, $p \in A(x) \quad \Leftrightarrow \quad \ell(p) = x$.

Traditional geometric routing algorithms that combine greedy with perimeter routing [KK00, KWZ03] guarantee that given the location of a node, the algorithm can route to it. These algorithms can be augmented as in GHT [RKL+02] so that given a virtual point p, the routing algorithm can reach the node $\ell(p)$ that is responsible for that virtual point. Their technique also adapts to node mobility. That is, when a node moves, the lines between its Voronoi polygon and its neighbors' Voronoi polygons changes. The nodes communicate

and swap information about all the virtual points that cross boundaries due to these line changes.

In summary, there is a mapping between virtual points in the plane and the nodes responsible for them; and we can employ routing to virtual points that reaches the corresponding nodes. Hence, by a slight abuse of terminology, given a point p, we refer from here on to the node $\ell(p)$ simply as the node at p, or even the node p. And to the contrary, given a node s we refer to the location of s as the point s.

6.3 Problem Definition

We consider algorithms for the geometric location problem. In this problem there are two basic operations: Publish and Lookup. The *Publish*$(t.id; x, y)$ operation is called every time a node t changes its location from x to a new location y. It is also used in the first time a node joins the system, by executing *Publish*$(t.id; \perp, y)$. The *Lookup*$(t.id, s)$ operation is called by a source node s in order to find location information on a destination node t whose location is unknown.

We now define the complexity measure associated with locality aware location services. The first time a node joins, we would like to minimize the cost of its publish operation. For subsequent publish operations we would like the cost of publishing to be proportional to the distance taken. This is formally captured in the following definition.

Definition 6.3.1. *For a function f, a publish algorithm is* f-locality aware *if for any node t with location x that moves to a new location y the expected cost of* Publish$(t.id; x, y)$ *is at most $f(|xy|)$.*

If the destination node does not exist we would like to minimize the cost of lookup until a failure result is returned to the source. Otherwise, we would like to have a locality aware lookup whose cost is proportional to the minimal cost path as defined below.

Definition 6.3.2. *For a function g, a lookup algorithm is* g-locality aware *if for any source s, and destination t, such that the minimal cost path costs $d = d(s, t)$, the cost of* Lookup$(t.id, s)$ *from source s to destination t is at most $g(d)$.*

Note that Definition 6.3.2 is a worst case bound that compares the lookup cost to the minimal cost path. Although it may not be possible for all networks, we would also like to get an average case bound on the cost of the lookup as a function of the *distance* between source and destination. The following definition captures this notion.

Definition 6.3.3. *For a function g, a lookup algorithm is* g-average case efficient *if for any source s, and destination t, the expected cost of* Lookup$(t.id, s)$ *from source s to destination t is at most $g(|st|)$.*

Certainly, building a lookup service with minimal cost can be done simply by storing complete location information at each node about all other nodes. The cost of an update, such as a node joining or moving, would be very high in this case. The goal is therefore to build a location service that simultaneously has the following properties: (1) a locality aware publish algorithm; (2) a locality aware lookup algorithm; (3) an average case efficient lookup on a large class of networks.

It is known from the lower bounds of [KWZ02] that even if t's location is known, routing on UDGs may cost $\Omega(d^2)$, where d is the cost of the minimal cost path. Simulations in [KWZ03] show that the cost of routing from source s to destination t is expected to be $O(d(s,t))$, which is also $O(|st|)$ for average case networks.

Our goal is to match these bounds for location services. Worst case lookup should cost $O(d^2)$, but for average case networks we aim to preform lookup with average cost that is only a constant factor more than the distance between the source and the destination, $O(d) = O(|st|)$. In case a node moves from location x to destination y we strive that the expected cost of the publish algorithm be a function of $|xy|$.

6.4 LLS Architecture

We present our location service in a modular way. We decompose the scheme into three algorithms, each algorithm builds upon its predecessor's techniques and enhances it:

1. Spiral algorithm. This location service obtains a locality aware lookup algorithm for average case networks.
2. Spiral-Flood algorithm. This location service enhances the Spiral algorithm by obtaining locality awareness both for average case lookup, and for lookup in worst case networks.
3. LLS. This algorithm enhances the Spiral-Flood service with a locality aware publish algorithm.

6.4.1 Mapping to Hierarchical Lattices

In our construction, for each node identifier, we build a hierarchy of lattices associated with the node's identifier. The node's location is published on each lattice to the points closest to its location, and is looked up on lattice points closest to the location of the searching node.

Given a node t, we use a double index hash function $H(t.id) = \langle h_1(t.id), h_2(t.id)\rangle$ that maps node identifiers to coordinates inside $M \times M$. We denote $H(t.id)$ the *primary virtual home* of node t. Formally, $H : V \mapsto [0, M] \times [0, M]$.

For a node t, and a parameter $k \in \{0, 1, \ldots \log M\}$ we define $L_k(t.id)$ to be the lattice consisting of all lattice points $(h_1(t.id) + 2^k i, h_2(t.id) + 2^k j) \in M \times M$, for all $i, j \in \mathbb{Z}$. Alternatively, $L_k(t.id)$ can be thought of as the set of corner points of the tiling with squares of size $2^k \times 2^k$ having $H(t.id)$ as its origin.

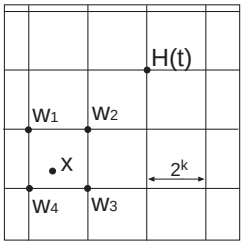

Figure 6.1: Example of the lattice $L_k(t.id)$ and of $W_k(t.id, x) = \{w_1, w_2, w_3, w_4\}$

For any point x and tiling of size 2^k originating at the virtual home of t, $H(t.id)$, we define the *level-k location points of t from x*, $W_k(t.id, x)$, to be the set of the 4 corner points of the tile that covers x. Formally, for $k \geq 1$ we define $W_k(t.id, x) = \{w_1, w_2, w_3, w_4\}$ for every point x and lattice $L_k(t.id)$, to be the set of the 4 lattice points that are closest to x in the L_∞ norm (with ties broken lexicographically), see Figure 6.1 for an example. When clear from the context, we will abuse notation and use $W_k(t.id, x)$ to denote the nodes that are responsible for the virtual points.

x

6.5 The Spiral Algorithm

We begin by presenting the basic spiral algorithm. It achieves a locality aware lookup algorithm that performs as well as the underlying geometric routing algorithm. Intuitively, a node publishes its location information in a set of virtual points that form a virtual spiral that exponentially increases in distance. When a node initiates a lookup operation, it too performs a spiral like search path. The lookup finds the destination's location when the two spirals intersect. These two spirals intersect because the points in both spirals are computed relative to the same hierarchical grid whose origin is the primary virtual home of the destination.

The first time a node t joins the network at location y, it invokes $Publish(t.id; \perp, y)$ to register its location. Whenever it moves from x to y in updates its location using $Publish(t.id; x, y)$. When a node s wants to establish communication with node t, it issues $Lookup(t.id, s)$.

Publish: A node t that moves from location x to y performs $Publish(t.id; x, y)$. This operation deletes the old location information (if there was any) and registers the new location y of the node by inserting location information in $O(\log M)$ different virtual points.

Registering location information is done by storing *location pointers* in the auxiliary memory of the nodes in $W_i(t.id, y)$. Each location pointer contains two fields: the identifier of the destination and its location, in the form $\langle t, y \rangle$. Figure 6.2 depicts an example of the set of points to which information is published.

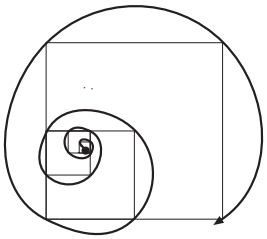

Figure 6.2: The corners of all the tiles that cover the node are the set of virtual points to which location pointers are published. The spiral depicts a path that the *Publish* algorithm executes in order to reach the virtual points.

Deleting the information of the old location x is done by visiting the respective nodes responsible for the old virtual points and deleting the information in them.

Lookup: A node s that executes $Lookup(t.id, s)$ reaches the destination node t by querying location information at the points associated with destination t, with increasingly larger distances from s. This process is very similar to the Publish operation. Once a location pointer is found, the information in the pointer is used to reach the destination.

6.5.1 Analysis

The lookup operation of the Spiral Algorithm is only as efficient as the underlying geometric routing that is used for routing to the nodes that are responsible for the virtual points. In this subsection we show that given a network with an efficient routing layer, the lookup operation costs only a constant factor more than the distance from the source to the destination.

In order to analyze the cost of the lookup operation in the Spiral Algorithm we first define the *locality awareness* of geometric ad hoc routing schemes in which the location of the destination is known.

Definition 6.5.1. *A routing protocol is* Δ-locality aware *if given a source s and point $y \in R^2$, the expected cost of routing from s to the node at y (more precisely, recall that this should actually reach $\ell(y)$, which is the node closest to y) is at most $\Delta|sy|$. Formally, let $r(s, y)$ be the cost of routing from s to y given that s knows the approximate location y then, $E_{s \in V, y \in R^2}\left[\frac{r(s,y)}{|sy|}\right] \leq \Delta$.*

In Section 6.10 we present simulation results showing that Greedy Face Routing is 2-locality aware for average case networks where nodes are uniformly distributed.

The following lemma shows that there is a location pointer at a distance that is proportional to the distance between source and target.

Lemma 6.5.2. *Let $\hat{k}$ be the minimal index such that $|st| = d \leq 2^{\hat{k}}$ then at least one of the nodes in $W_{\hat{k}}(t.id, s)$ contains a location pointer to node t.*

Proof. Denote t's location by y. Since destination node t performed $Publish(t.id; y)$ then the nodes in $W_{\hat{k}}(t.id, y)$ contain location pointers towards t. Let Q_1 be the square of size $2^{\hat{k}} \times 2^{\hat{k}}$ that covers t whose corners are $W_{\hat{k}}(t.id, y)$. Let $Q_2, \ldots, Q_9$ be the set of 8 adjacent squares of same size that have a joint edge or joint corner with Q_1.

Recall that we chose $\hat{k}$ such that $|st| = d \leq 2^{\hat{k}}$ therefore s must be inside one of the squares $Q_1, \ldots, Q_9$. Thus when source s performs the $\hat{k}$th phase of its lookup algorithm it will find a location pointer towards t.

Note that a similar argument holds for the degenerate case in which t lies exactly on a lattice line or intersection. □

Hence for networks that have efficient routing, the cost of lookup is a linear function of the distance between source and destination.

Theorem 6.5.3. *For networks in which routing is Δ-locality aware, for any source s and destination t the expected cost of locating t is $O(|st|)$*

Proof. Due to Lemma 6.5.2, the lookup algorithm will find a location pointer at phase $\hat{k}$, where $\hat{k}$ is the minimal index such that $|st| = d \leq 2^{\hat{k}}$. The expected cost of routing to all the points $W_1(t.id, s), \ldots, W_{\hat{k}}(t.id, s)$ can be bounded by the following expression: $\sum_{i \leq \hat{k}} 2 \cdot 4 \cdot 2^i \cdot \Delta = O(2^{\hat{k}}) = O(|st|)$ where the factor 2 is due to going to each of the virtual points and returning to s, 4 is due to the fact that there are 4 virtual points, 2^i is due to the maximal distance of the virtual points from s, and Δ is the overhead due to the underlying routing layer. Once a location pointer of level $\hat{k}$ is found, the expected cost of following the location pointers until t is found can be similarly bounded by $O(|st|)$. □

6.6 The Spiral-Flood Algorithm

The main drawback of the basic spiral algorithm is that it relies solely on the underlying routing algorithm for performance. In particular, there may exist extreme situations in which there is a low cost path from source to destination, but the cost of the minimal cost path from the source to the first virtual point is arbitrarily high.

The Spiral-Flood algorithm overcomes such cases. The algorithm combines the iterated bounded flooding (IBF) ideas from [KWZ02] for worst case bounding, with the average case efficiency of the Spiral algorithm.

In order to describe Spiral-Flood, we need to enumerate in a sequence the nodes visited in the Spiral algorithm. For a destination t and a source s, the Spiral lookup operation $Lookup(t.id, s)$, coupled with the underlying geometric routing mechanism determine the sequence of nodes that are visited. We denote this sequence by $Spiral(t.id, s)$. Define $Spiral_j(t.id, s)$ to be the prefix path of $Spiral(t.id, s)$ containing the first j nodes.

The Spiral-Flood lookup consists of phases, $i = 1 \ldots \log M$. When the accumulated cost of the Spiral lookup reaches 4^i, we switch to flooding of depth 2^i. This ensures that the

Lookup$(t.id, s)$
Initialize $phase := 0$
Until a location pointer for t is found:

1. *Spiral stage:* Route on the path $Spiral(t.id, s)$ for j hops, as long as the cumulative cost $c(Spiral_j)$ is less than 4^{phase}.
2. Return back to the source node s.
3. *Flooding stage:* Flood a search message to all nodes r whose minimal cost path $d(s, r)$ is at most 2^{phase}.
4. Converge the search results back to source node s.
5. Set $phase := phase + 1$.

Figure 6.3: The Spiral-Flood Lookup Algorithm. Node s wants to establish a communication session with node t.

cost of the combined algorithm is not more than a constant factor over the basic Spiral algorithm, and at the same time the cost does not exceed the worse case cost of the flooding algorithm. The pseudo code for the Spiral-Flood algorithm appears in Figure 6.3. The Publish algorithm is the same as the in basic Spiral algorithm.

6.6.1 Analysis

In this subsection we prove that the Spiral Flood algorithm worst case quadratic cost.

Lemma 6.6.1. *Spiral-Flood locates any destination t from any source s with cost $O(d(s,t)^2)$.*

Proof. Denote $d = d(s,t)$, let j be the minimal index such that $d \leq 2^j$. Clearly before the end of phase j of the Spiral-Flood algorithm the destination will be found. This is true since the jth flooding phase would reach all nodes whose cost is a most 2^j.

Given the $\Omega(1)$ assumption of a minimal distance between nodes (or any other bounded density assumption due to a CDS backbone), the sphere of nodes whose cost is at most 2^j has $O(2^{2j})$ nodes and hence also $O(2^{2j})$ edges. Each such edge is traversed at most 4 times. Therefore, the cost of a flooding during phase j is $O(2^{2j})$.

By construction, the cost of the bounded phase-j walk on $Spiral(t.id, s)$ is $O(4^i)$. Therefore, the total cost until the end of phase j is $\sum_{0 \leq i \leq j} O(4^i) + (2^{2i}) = O(4^j) = O(d^2)$. □

Compared with the basic Spiral Algorithm, the Spiral Flood Algorithm incurs a constant factor overhead.

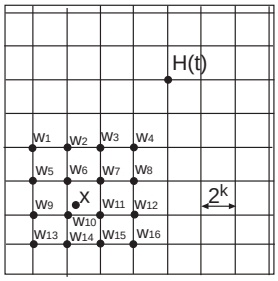

Figure 6.4: Example of the lattice $L_k(t.id)$ and of $Z_k(t.id, x) = \{w_1, \ldots, w_{16}\}$

Lemma 6.6.2. *The cost of reaching the points $W_1(t.id, s)$ through $W_i(t.id, s)$ in the Spiral Flood Algorithm is at most a constant factor more than the cost of the basic Spiral Algorithm to reach the same set of points.*

The proof follows by noting that in each stage of the algorithm, the cost of the flooding stage is at most a constat factor more than the cost of the spiral search stage for that phase.

6.7 The LLS Algorithm

In this section we describe our full scheme, LLS, in which proactive updating of location information due to node movements is proportional to the distance of the move. Formally we present a publish algorithm that is average case efficient. Achieving this aim is somewhat more involved than simply writing to all the virtual points in $Spiral(t.id, y)$ and deleting the points in $Spiral(t.id, x)$.

First, instead of publishing to only 4 virtual points at each level, our publish algorithm publishes to 16 virtual points. For any point x and tiling of size 2^k originating at the virtual home of t, $H(t.id)$, we define $Z_k(t.id, x)$ to be the set of corner points of the 9 tiles of size $2^k \times 2^k$ that form a square $3 \cdot 2^k \times 3 \cdot 2^k$ whose center tile covers x. Formally we define $Z_k(t.id, x) = \{w_1, \ldots, w_{16}\}$ for every point x and lattice $L_k(t.id)$, to be the set of the 16 lattice points that are closest to x in the L_∞ norm (with ties broken lexicographically). See Figure 6.4 for an example.

Secondly, in order to reduce costs, a node does not update its level k location pointers as long as it does not move a total of a certain distance proportional to 2^{k-1}. This lazy update process is carefully done in a manner that still allows the lookup algorithm to be efficient and locality aware. Accordingly, we modify the location information stored at the publish nodes $Z_k(t.id, y)$ as follows. At the location nodes of $Z_i(t.id, y)$, we store information on a set of virtual points that defines the next hop towards the destination, in the form $\langle t, W_{i-1}(t.id, y)\rangle$. At $Z_0(t.id, y)$, we store the location y itself.

Publish: A node t that moves from location x to y performs $Publish(t.id; x, y)$.

1. If $W_0(t.id, x) = W_0(t.id, y)$ then no update is required.

Publish$(t.id; x, y)$

1. If all nodes $W_0(t.id, y)$ contain a location pointer for t then RETURN. Otherwise initialize $i := 1$
2. STEP A: Read all nodes in $W_i(t.id, y)$
3. If all nodes $W_i(t.id, y)$ contain a location pointer for t goto STEP B.
4. Write in all 16 nodes of $Z_i(t.id, y)$ a location pointer $\langle t, W_{i-1}(t.id, y)\rangle$.
5. Set $i := i + 1$ and goto STEP A.
6. STEP B: Update the content of the 16 existing level i location pointers to point to $\langle t, W_{i-1}(t.id, y)\rangle$.
7. For $j = i$ to 0 do

   Read all level j nodes. Using the location pointers in the level j nodes, route to all the level $j-1$ nodes and delete their location pointers.

Figure 6.5: The LLS Publish Algorithm. Node t updates location pointers once it arrived to a new location y.

2. Otherwise, do a spiral search, find the minimal index i such that all four points of $W_i(t.id, y)$, have location information regarding t.
   (a) If $i = 0$ then no update is required. Otherwise, when $i > 0$:
      i. Using the location information stored in $W_i(t.id, x)$ (which contains $Z_{i-1}(t.id, x)$), recursively erase all the location information in the 16 points of each level $i - 1, \ldots, 0$.
      ii. For $j = 1, \ldots, i - 1$, store in $Z_j(t.id, y)$ the location pointer $\langle t, W_{j-1}(t.id, y)\rangle$.
      iii. Set the location pointers of level i to point to the new location $\langle t, W_{i-1}(t.id, y)\rangle$.

Note that the minimality of the index i implies that if $i > 0$ then the node should have moved out of the 9 tiles of level $i - 1$. Thus, the node has moved a distance of at least 2^{i-1} since the last time the related nodes were updated. See Figure 6.5 for the LLS Publish algorithm.

Lookup: Node s executing *Lookup*$(t.id, s)$ starts the lookup algorithm as before (Spiral or Spiral Flood). However, the search ends when a recursive location pointer is found of the form $\langle t, W(t.id, y)\rangle$, where y is the location of t.

Once such a node is found, repetitively use *location pointers* in order to route from a point in $W_j(t.id, y)$ to a point in $W_{j-1}(t.id, y)$ and eventually for $j = 1$, to destination t.

At the lowest level, if the destination is not found at the square of the lowest level (of size 1×1), then a search is conducted in the 8 unit squares surrounding that square.

6.7.1 Analysis

To address the complexity of the LLS publish algorithm we carefully study the movement of the node. Assume that the node performed h publish operations, the *location history* of the node is the sequence $x_0, x_1, \ldots, x_h$ of locations. In order to bound the total cost of its h publish operations we will keep track of the relative location of the intermediate nodes. We prove the bound on the cost in an amortized sense. The reason is that the node does not need to update its location pointers as long as it does not cross its 16-point boundary, which happens only if it moves a total of a certain distance. Once the node publishes its location, the cost of publishing is offset by the total distance it has covered since its last update.

Our analysis focuses on average case networks, and in particular, assumes that the network has Δ-locality aware routing, Let $d(X_j)$ denote the total distance moved by the node in its first $j \leq h$ hops, thus $d(X_j) = \sum_{i=1}^{j-1} |x_i x_{i+1}|$. For any level k, let $c_k(x_j, x_{j+1})$ denote the cost associated with writing and deleting location pointers on level k virtual points as a result of *Publish*$(t.id; x_j, x_{j+1})$. Denote the cost of updating the level k virtual points due to the first j location changes as $c_k(X_j) = \sum_{i=1}^{j-1} c_k(x_i, x_{i+1})$. Our goal is to bound the total cost $c(X_j) = \sum_{i=1}^{\log M} c_i(X_j)$ as a function of total distance $d(X_j)$.

Lemma 6.7.1. *If* Publish$(t.id; x_j, x_{j+1})$ *changes level k location pointers then the expected cost of this change is bounded by $E\left[c_k(x_j, x_{j+1})\right] \leq \Delta \cdot 2^{k+6}$.*

Proof. When the *Publish*$(t.id; x_j, x_{j+1})$ changes level k location pointers, it deletes old level k pointers and writes to its new ones. The expected cost of each of these two operations can be bounded by $\Delta \cdot 2 \cdot 16 \cdot 2^k$.

Where the factor $2 \cdot 16 \cdot 2^k$ is due to the total distance of a spiral path visiting all 16 points and returning and Δ is the overhead due to the underlying routing layer. □

When level k location pointers are not changed despite the move, there is a potential gain towards future updates. This is formally stated in the following Theorem.

Theorem 6.7.2. *The total expected cost of publishing of location history $x_1, \ldots, x_h$, is bounded by $E[c(X_h)] \leq O(d \log d)$ where $d = d(X_h)$.*

Proof. While moving, the node occasionally updates its location pointers. Consider a specific level k. A node moving from x_{j-1} to x_j updates its level k pointers when the location of its level $k-1$ pointers at x_{j-1} differ from its new location's (x_j) level $k-1$ pointers. Formally, let $i_1 < i_2 < \cdots < i_m$ be the sequence of all indices such the level k location pointers were updated when the node reached x_{i_ℓ} for $1 \leq \ell \leq m$. Due to our publish algorithm, the distance the node moves between any two updates of level k pointers is at least 2^{k-1}. That is, for any such index i_ℓ we have $\sum_{i_{\ell-1} < j \leq i_\ell} |x_{j-1} x_j| = d(X_{i_\ell}) - d(X_{i_{\ell-1}}) \geq 2^{k-1}$. Thus the

number of updates m is bounded by $d(X_h)/2^{k-1}$. From Lemma 6.7.1, the expected cost of the update of level k pointers is $\Delta \cdot 2^{k+6}$. Therefore, by summing on the whole sequence, $E[c_k(X_h)] \leq \Delta \cdot 2^{k+6} m \leq \Delta \cdot 2^7 \cdot d(X_h)$. The expected total cost of publishing the updates during the location history $x_1, \ldots, x_h$, is the sum of the costs over all levels that were updated during the total move. Thus, the cost is $E[c(X_h)] \leq \Delta 2^7 \cdot d(X_h) \log d(X_h)$. If $d(X_h) \geq M$ then the expected cost may be bounded by $\Delta 2^7 \cdot d(X_h) \log M$. □

6.8 Fault Tolerance

Our location service is inherently fault tolerant both to network partitions and to node failures.

In case of a network partition, nodes within the vicinity of a detached node remain able to locate it and communicate with it. More precisely, let a set of nodes S detach from $V \setminus S$. For a node $t \in S$, let k be the maximal index such that $W_k(t.id, t) \subseteq S$. Then all of the nodes within distance 2^k can locate t and communicate with it. In fact, even in extreme cases where a partition forms weird shapes, for any $\ell > k$ such that we have $W_\ell(t.id, t) \cap S \neq \emptyset$, far nodes in S that have lookup spirals intersecting these location points are also able to locate t.

As for node failures, our scheme provides immediate fallback due to the multiplicity of location points that store information about any node. In particular, when a node s searching for $t.id$ encounters a faulty location point, it simply skips it and moves on to the next higher level. This guarantees location at the first non-faulty level that is greater than the distance from the source and the target. That this works follows from the following fact. If the minimal index at which intersection occurs between $W_k(t.id, t)$ and $W_k(t.id, s)$ is k, then higher level location points intersect as well.

As a final frontier for fault tolerance, the flooding stage will always locate the target (if it is connected) at a quadratic cost.

Additional fault tolerance is provided by employing the perimeter replication techniques of [RKL+02], by storing all of the information belonging to virtual point x also on all the nodes surrounding its perimeter.

6.9 Improving Locality Awareness

Assume a network in which routing is Δ-locality aware. A natural question to ask is whether on top of the locality-aware routing mechanism, our Spiral paradigm can provide location services with costs arbitrarily close to minimal. In addition to its theoretical value, this question has real-life motivation: For nodes that incur frequent lookups, we would like to drive the lookup cost down as much as possible.

In this section, we focus on optimizing the cost of the lookup operation. We obtain a

lookup scheme whose cost is $(1+\varepsilon)\Delta$ on top of the actual distance to target. The decreased lookup cost is obtained at the cost of increasing the publish cost by a constant factor. We could do the reverse in a similar manner, if the frequency of updates is expected to exceed that of lookups.

In order to reduce the cost of lookup, denote by $B(u,r)$ the ball around node u with radius r. Let us define $W_{k,\ell}(t.id,x)$ to be $L_k(t.id) \cap B(x,2^\ell)$; that is, the set of lattice points in $L_k(t.id)$ within distance 2^ℓ from x. The trick will be to slightly increase the range of publishing into $W_{k,k+\rho}$, for a parameter ρ of the construction determined below; and to query only one node in $W_{k,k}$. More precisely, we now define publish and lookup as follows:

Publish: For any $i \in \{0,1,2,\ldots,\log M\}$, location pointers on $t.id$ located at y are stored in the auxiliary memory of the nodes in $W_{i,i+\rho}(t.id,y)$, where the parameter ρ is determined below. A node t that moves from location x to y performs $Publish(t.id;x,y)$ by deleting old information and storing new information in the nodes described above.

Lookup: A node s executes $Lookup(t.id,s)$ by performing the following loop. Until t is found, for $i=0,1,2,\ldots,\log M$, read from the closest node in $W_{i,i}(t.id,s)$ until a node is found that has a location pointer $\langle t, W(t.id,y)\rangle$, where y is the location of t. Once such a node is found, repetitively use the *location pointers* in order to route from a point in $W_j(t.id,y)$ to a point in $W_{j-1}(t.id,y)$ and eventually for $j=1$, to destination t.

Locality awareness stems from the following lemma:

Lemma 6.9.1. *Denote $|st|=d$. Let $\hat{k}$ be the minimal index such that $2^{\hat{k}}+d \le 2^{\hat{k}+\rho}$. Then the $\hat{k}$'th lookup step from s finds a location pointer to node t.*

Proof. The lemma follows from the fact that $B(s,2^{\hat{k}}) \subseteq B(t,2^{\hat{k}+\rho})$. Hence, every node in $W_{\hat{k},\hat{k}}(t.id,s)$ is contained in $W_{\hat{k},\hat{k}+\rho}(t.id,t)$. Therefore, every node in $W_{\hat{k},\hat{k}}(t.id,s)$ holds a location pointer to node t. □

As a consequence of this lemma, the cost of lookup from s to t is bounded by $\sum_{i\le\hat{k}} 2^i\Delta \le 2\cdot 2^{\hat{k}}\Delta$. Bearing in mind that $\hat{k}$ was the first index for which $2^{\hat{k}}+d \le 2^{\hat{k}+\rho}$, we now simply need to set $\rho=\rho(\varepsilon)$ to be $\rho \ge \log(1+\frac{8\Delta}{\varepsilon}) = O(\log(\varepsilon^{-1}))$, in order to obtain that the total lookup length is at most $(1+\varepsilon)\Delta d$.

We now get to the increased cost of publishing. We only sketch the difference here due to space limitations. In our original Spiral method, publishing a node's new location was done in the 4 corner points of the tile covering the node. We now change that to include all the lattice-k points within distance $2^{k+\rho}$ from the node. Their count is at most $2^{2\rho}$, and the total cost of reaching them is proportional to $2^{k+\rho}/2^k$. Therefore, the increased cost is a constant factor over our original scheme, where the constant depends on $\rho=\rho(\varepsilon)$.

We also briefly comment on how to complete our scheme for locality-aware updates. This requires maintaining an invariant that all lattice-k points within a ball of radius $2^{k+\rho}$ surrounding a node have up-to-date information about the interior ball containing it. In order to prevent frequent updates, we initially publish within a ball with double this radius,

i.e., a ball of radius $2^{k+1+\rho}$, and update only when the invariant is broken. This again incurs only a constant-factor increase over the original LLS publish costs.

In summary, at the cost of a constant factor increase in update costs, we obtain a location service whose cost is almost as low as the cost of the underlying routing infrastructure.

6.10 Simulations

In this section we present simulation results for the LLS scheme. Following the observations of [KWZ03] about the importance of the network density on routing performance, we tested our scheme in randomly generated graphs with varying densities. The density of a graph is defined relative to a unit disk as follows, a random graph on a $T \times T$ square with density δ has $\delta T^2/\pi$ nodes.

We tested the underlying Greedy Face Routing algorithm to virtual points, and the LLS Lookup algorithm. Our measurements were performed on Unit Disk Graphs that were generated by randomly placing nodes on a square with 15 units side length.

Routing. For the underlying greedy face routing algorithm, for each density δ, we generated 1200 random graphs for $\delta \leq 8$ and 200 random graphs for $\delta > 8$, for each graph, we randomly chose 50 sources nodes, for each source s we chose a random point p on the square and found the closest node $\ell(p)$ to the virtual point p. We measured the following parameters for each density:

1. Connectivity: The percentage of node pairs $s, \ell(p)$ that were connected on the Unit Disk Graph.
2. Cost/Distance: The average ratio between the cost of routing from a source s to a destination $\ell(p)$ and the Euclidean distance between s and p.

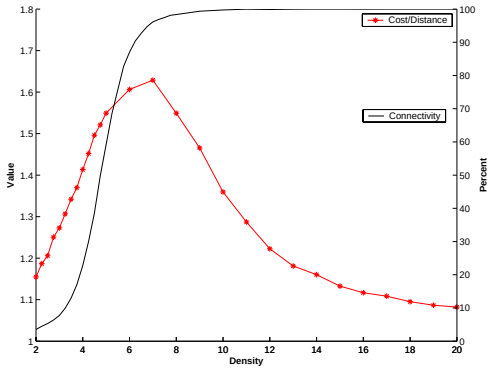

Figure 6.6: Connectivity percentage and Cost/Distance values for greedy face routing from a random source to the node closest to a random point.

Our results in Figure 6.6 show that the average cost of routing on a random graph to the node closest to a virtual point is only a small constant factor times the Euclidean distance

between the node and the virtual point.

Lookup. For each density value δ, we generated 500 random graphs for $\delta \leq 10$ and 100 random graphs for $\delta > 10$, for each graph, we randomly chose 10 sources nodes, for each source we randomly chose a destination node. We measured the following 4 parameters for each density:

1. Cost/Distance: The average ratio between lookup cost and Euclidean distance between source s and target t.
2. Cost/Shortest path: The average ratio between lookup cost and the minimum cost path from s to t on the UDG.
3. Connectivity: The percentage of s, t pairs that were connected.
4. Flood: the percentage of cases in which the lookup algorithm had to use flooding in order to find the destination.

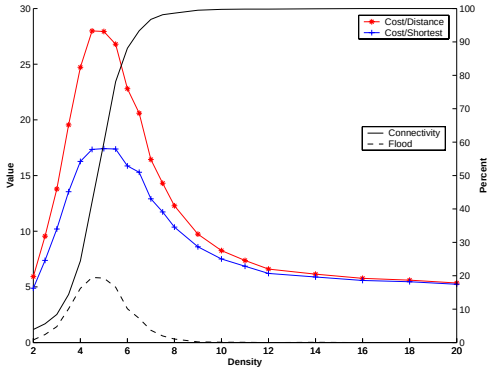

Figure 6.7: Measurements for Lookup algorithm: Cost/Distance and Cost/Shotest path values, Connectivity and Flood percentages.

Our results in Figure 6.7 show that lookup is locality aware, the average cost of locating a mobile node is only a constant factor from the optimum. For low densities, the low stretch factor may be attributed to the fact that the connected components are relatively small. Both Cost/Distance, Cost/Shortest, and Flood ratio peak around density 5. In this critical density, the graph is mostly connected but finding short paths to route on is still hard.

Publish. We measured the cost updating the location pointers due to a random hop. The distance moved in each hop was a varied by a parameter ℓ. For each hop length ℓ, we generated 100 graphs, for each graph we chose 20 sources and for each source we performed a random hop, with a randomly chosen angle and a randomly chosen length in $[\ell, \ell + 1/2)$. For every value of ℓ, we measured the average cost of updating location pointers due to the random hop over all sources and graphs.

Our results in Figure 6.8 show that the cumulative cost of updating location pointers due to a random walk is proportional to the length of the average hop. This result agrees with our theoretical bounds of $O(d \log d)$.

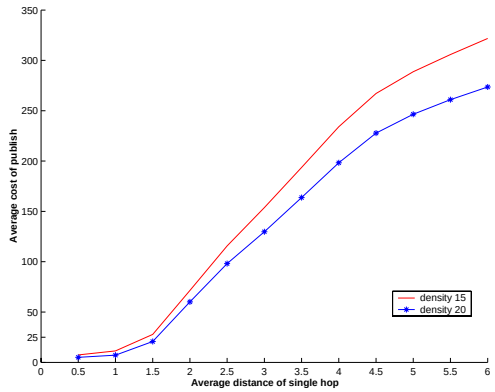

Figure 6.8: Average cost of publish relative to the average length of the hop for graphs with densities 15 and 20.

6.11 Conclusions

This chapter presents LLS, the first location service for mobile ad hoc networks to guarantee both worst case bounds and average case efficiency. Our scheme has inherent fault tolerance both for node failures and for network partitions. The generalization of our construction results in a geometric location service that is locality aware in any Euclidean metric space.

For any network the worst case cost of our lookup is quadratic $O(d^2)$. For average case networks LLS achieves linear $O(d)$ expected lookup cost. Mobility of nodes is handled in an efficient proactive manner. LLS ensures that the expected cost of publish is $O(d \log d)$. We provide simulation results for the LLS scheme that conform our theoretical bounds.

Our simulations show that routing in random ad hoc networks is 2-locality aware for various densities. It would be interesting to give a formal proof backing this result.

A potential area of improvement is reducing the asymptotic cost of the publish algorithm. The worst case cost of the publish algorithm can be easily bounded using similar techniques as in our lookup algorithm. An open question is whether an expected $O(d)$ is achievable for *both* publish and lookup.

Acknowledgments

The authors would like to thank Marina Shudler for simulations. Fabian Khun and the anonymous referees provided helpful comments.

Bibliography

[ABNLP89] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Compact distributed data structures for adaptive routing. In 21^{st} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 479–489. ACM Press, May 1989.

[ABNLP90] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *Journal of Algorithms*, 11(3):307–341, 1990.

[ACL+03] Marta Arias, Lenore J. Cowen, Kofi Ambrose Laing, Rajmohan Rajaraman, and Orjeta Taka. Compact routing with name independence. In 15^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 184–192. ACM Press, June 2003.

[ADM04] Ittai Abraham, Danny Dolev, and Dahlia Malkhi. Lls: a locality aware location service for mobile ad hoc networks. In *DIALM-POMC '04: Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 75–84, New York, NY, USA, 2004. ACM.

[AG06] Ittai Abraham and Cyril Gavoille. Object location using path separators. In 25^{rd} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM Press, July 2006.

[AGGM05] Ittai Abraham, Cyril Gavoille, A.V. Goldberg, and Dahlia Malkhi. Routing in networks with low doubling dimension. Technical Report MSR-TR-2005-175, Miscrosoft Research, December 2005. To appear in The 26th International Conference on Distributed Computing Systems (ICDCS 06).

[AGM04a] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. Routing with improved communication-space trade-off. In 18^{th} *International Symposium on Distributed Computing (DISC)*, volume 3274 of Lecture Notes in Computer Science, pages 305–319. Springer, October 2004.

[AGM+04b] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. In 16^{th} *Annual ACM Symposium on Parallel Algorithms and Architecture (SPAA)*, pages 20–24. ACM Press, July 2004.

[AGM05] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. Compact routing for graphs excluding a fixed minor. In 19^{th} *International Symposium on Distributed Computing (DISC)*, volume 3724 of Lecture Notes in Computer Science, pages 442–456. Springer, September 2005.

[AGM06] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On space-stretch trade-offs: upper bounds. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 217–224, New York, NY, USA, 2006. ACM.

[AGM+08] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. *ACM Trans. Algorithms*, 4(3):1–12, 2008.

[AM05] Ittai Abraham and Dahlia Malkhi. Name independent routing for growth bounded networks. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 49–55, New York, NY, USA, 2005. ACM.

[AMD04] Ittai Abraham, Dahlia Malkhi, and Oren Dobzinski. LAND: stretch $(1 + \varepsilon)$ locality-aware networks for DHTs. In 15^{th} *Symposium on Discrete Algorithms (SODA)*, pages 550–559. ACM-SIAM, 2004.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In 31^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 503–513. IEEE Computer Society Press, October 1990.

[AP92] Baruch Awerbuch and David Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Discrete Mathematics*, 5(2):151–162, May 1992.

[AP95] B. Awerbuch and D. Peleg. Online tracking of mobile users. *Journal of the ACM*, 42(5):1021–1058, 1995.

[APL99] K.N. Amouris, S. Papavassiliou, and M. Li. A position-based multi-zone routing protocol for wide area mobile ad-hoc networks. In *Proceedings of the IEEE Vehicular Technology Conference (VTC)*, pages 1365–1369, 1999.

[AS02] I. Aydin and C. C. Shen. Facilitating match-making service in ad hoc and sensor networks using pseudo quorum. In *Proceedings of 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, 2002.

[BMSU01] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[CBW02] T. Camp, J. Boleng, and L. Wilcox. Location information services in mobile ad hoc networks. In *Proceedings of the IEEE International Conference on Communications (ICC '02)*, 2002.

[CDK+03] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical, distributed network coordinates. In *HotNets Workshop*, 2003.

[CGMZ05] T.-H. Hubert Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. In 16^{th} *Symposium on Discrete Algorithms (SODA)*, pages 762–771. ACM-SIAM, January 2005.

[Cow99] Lenore J. Cowen. Compact routing with minimum stretch. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 255–260, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[Cow01] Lenore J. Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38:170–183, 2001.

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[DANL04] M. Demirbas, A. Arora, T. Nolte, and N. Lynch. Brief announcement: Stalk: a self-stabilizing hierarchical tracking service for sensor networks. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 378–378. ACM Press, 2004.

[DGL+] S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC 2003).*

[EGP03] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *Journal of Algorithms*, 46:97–114, 2003.

[FG01] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In 28^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of Lecture Notes in Computer Science, pages 757–772. Springer, July 2001.

[FG02] Pierre Fraigniaud and Cyril Gavoille. A space lower bound for routing in trees. In 19^{th} *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2285 of Lecture Notes in Computer Science, pages 65–75. Springer, March 2002.

[Gav01] Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News - Distributed Computing Column*, 32(1):36–52, March 2001.

[GC02] N. Guba and T. Camp. Recent work on gls: a location service for an ad hoc network. In *Proceedings of the Grace Hopper Celebration (GHC '02)*, 2002.

[GG01] Cyril Gavoille and Marc Gengler. Space-efficiency of routing schemes of stretch factor three. *Journal of Parallel and Distributed Computing*, 61:679–687, 2001.

[GKL03] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 534, Washington, DC, USA, 2003. IEEE Computer Society.

[GP03] Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Journal of Distributed Computing*, 16:111–120, May 2003. PODC 20-Year Special Issue.

[GSG02] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the SIGCOMM Internet Measurement Workshop (IMW 2002)*, 2002.

[HKK04] Kirsten Hildrum, Robert Krauthgamer, and John Kubiatowicz. Object location in realistic networks. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 25–35. ACM Press, 2004.

[HKMR04] Kirsten Hildrum, John Kubiatowicz, Sean Ma, and Satish Rao. A note on the nearest neighbor in growth-restricted metrics. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 560–561. Society for Industrial and Applied Mathematics, 2004.

[HKRZ02] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distributed object location in a dynamic network. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 41–52. ACM Press, 2002.

[HL99] Z. J. Haas and B. Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ ACM Transactions on Networking*, 7(2):228–240, 1999.

[HLBTG01] J. P. Hubaux, J. Y. Le Boudec, and M. Vetterli Th. Gross. Towards self-organizing mobile ad-hoc networks: the terminodes project. *IEEE Comm Mag*, 39(1):118 –124, January 2001.

[HLV03] J. L. Welch H. Lee and N. H. Vaidya. Location tracking with quorums in mobile ad hoc networks. *Ad Hoc Networks*, 1(4):371–381, 2003.

[HMP01] Torben Hagerup, Peter Bro Miltersen, and Rasmus Pagh. Deterministic dictionaries. *J. Algorithms*, 41(1):69–85, 2001.

[HPM05] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *SCG '05: Proceedings of the twenty first annual symposium on Computational geometry*, New York, NY, USA, 2005. ACM Press.

[IO03] Kazuo Iwama and Masaki Okita. Compact routing for flat networks. In 17^{th} *International Symposium on Distributed Computing (DISC)*, pages 196–210. Springer, 2003.

[ISA] Cellular radio telecommunication intersystem operations. Technical Report Technical Report TIA/EIA-41-D, Telecommunications Industry Association, Washington, DC.

[KK00] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM Press, 2000.

[KLL+97] D. Karger, E. Lehman, F. T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 654–663, 1997.

[KLMN04] Robert Krauthgamer, James R. Lee, Manor Mendel, and Assaf Naor. Measured descent: A new embedding method for finite metrics. In 45^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 434–443. IEEE Computer Society Press, October 2004.

[KMP99] G. Karumanchi, S. Muralidharan, and R. Prakash. Information dissemination in partitionable mobile ad hoc networks. In *Symposium on Reliable Distributed Systems*, pages 4–13, 1999.

[KRX06] Goran Konjevod, Andréa Werneck Richa, and Donglin Xia. On optimal stretch name-independent compact routing in doubling metrics. In 25^{rd} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM Press, July 2006.

[KSU99] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11 th Canadian Conference on Computational Geometry*, pages 51–54, Vancouver, August 1999.

[KWZ02] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *Proc. of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications (Dial-M)*, pages 24–33. ACM Press, 2002.

[KWZ03] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In *Proc.* 4^{th} *ACM Int. Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc)*, 2003.

[KWZZ03] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In *Proc.* 22^{nd} *ACM Int. Symposium on the Principles of Distributed Computing (PODC)*, 2003.

[Lai03] K. A. Laing. Name-independent compact routing in trees. Technical Report 2003-02, Tufts University Department of Computer Science, November 2003.

[Lai04] Kofi Ambrose Laing. Brief announcement: name-independent compact routing in trees. In 23^{rd} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 382–382. ACM Press, July 2004.

[LJD+00] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130, August 2000.

[LR05] Kofi Ambrose Laing and Rajmohan Rajaraman. Brief announcement: A space lower bound for name-independent compact routing in trees. In 17^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, page 216. ACM Press, July 2005.

[MP92] M. Mouly and M. Pautet. *The GSM System for Mobile Communications*. 1992.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[NN02] B. Nath and D. Niculescu. Routing on a curve. In *HotNets-I, Princeton, NJ*, 2002.

[NZ02] E. Ng and H. Zhang. Predicting internet network distance with coordiantes-based approaches. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM '02)*, 2002.

[OBSC00] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000. 671 pages.

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[PRR97] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In 9^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 311–320. ACM Press, 1997.

[PS97] R. Prakash and M. Singhal. Dynamic hashing + quorum = efficient location management for mobile computing systems. In *Proceedings of ACM Symposium on Principles of Distributed Computing*, 1997.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, July 1989.

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *J. Comput. Syst. Sci.*, 37(2):130–143, 1988.

[RKL+02] S. Ratnasamy, B. Karp, Y. Li, F. Yu, R. Govindan, S. Shenker, and D. Estrin. GHT: A geographic hash table for data-centric storage. In *The First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, October 2002.

[RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 261–272. Springer, 2005.

[SK85] N. Santoro and R. Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28(1):5–8, February 1985.

[Sli05a] Aleksandrs Slivkins. Distance estimation and object location via rings of neighbors. In 24^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 41–50. ACM Press, July 2005.

[Sli05b] Aleksandrs Slivkins. Distance estimation and object location via rings of neighbors. In 24^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2005.

[ST03] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM '03)*, San Francisco, CA, 2003.

[Sto99] I. Stojmenovic. A scalable quorum based location update scheme for routing in ad hoc wireless networks. Technical Report TR-99-09, SITE, University of Ottawa, 1999.

[Tal04a] Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 281–290. ACM Press, 2004.

[Tal04b] Kunal Talwar. Bypassing the embedding: Algorithms for low dimensional metrics. In 36^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 281–290. ACM Press, June 2004.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, November 2004.

[TV04] J. Tchakarov and N. Vaidya. Efficient content location in wireless ad hoc networks. In *IEEE International Conference on Mobile Data Management (MDM)*, 2004.

[TY79] Robert Endre Tarjan and Andrew Chi-Chih Yao. Storing a sparse table. *Commun. ACM*, 22(11):606–611, 1979.

[TZ01a] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In 33^{rd} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 183–192. ACM Press, July 2001.

[TZ01b] Mikkel Thorup and Uri Zwick. Compact routing schemes. In 13^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10. ACM Press, July 2001.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, January 2005.

[vLT86] J. van Leeuwen and R. B. Tan. Computer networks with compact routing tables. In G. Rozemberg and A. Salomaa, editors, *The Book of L*, pages 259–273. Springer-Verlag, 1986.

[WSPC03] S. Wilbur, T. H. Saleem, B. M. Pias, and J. Crowcroft. Lighthouses for scalable distributed location. In *The 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.