Department of Computer Science, University of Copenhagen

Introduction to Compact Name-Independent Routing with Minimum Stretch

Henrik Bendt & Jonas Brunsgaard

May 27, 2015

Contents

1 Introduction to routing schemes

Routing is one of the most fundamental problems in the area of distributed networking. The goal in this problem is to find a distributed algorithm that allows any source vertex v in a network $G = (V, E, \omega)$ to route messages to any destination vertex u, where $u, v \in V$.

Each node $v \in V$ is given a unique name with $O(log\ n)$ bits. Also for each vertex, each outgoing edge is given a unique port number in $\{1, \ldots, deg(v)\}$.

Formally, a routing scheme is comprised of two phases. The first phase is called the preprocessing phase. In this phase G is preprocessed and the derived routing information is stored locally with the relevant vertices.

In the second phase - the routing phase - the routing scheme allows any vertex to route messages to any vertex in a distributed manner only based on the label of the destination vertex.

A naive approach to routing, is to preprocess a graph by running a SSSP algorithm from each vertex $v \in V$ and at each vertex v store a hashmap mapping any vertex in V to a port number.

In the routing phase a vertex will only need to look at the destination label of the message. If the destination label match the one of the vertex then we are done, otherwise we lookup what forwarding port for the destination vertex and forwarding the message.

This approach will ensure a routing of stretch-1 (a min cost path) and the lookup performed at each vertex can be done in constant time. But this approach will use a lot of space as each vertex will store a hashmap with n entries where each key is a destination label using $O(log(n))$ bit.

Thus the space consumed at each node is $\Omega(n\ log\ n)$. This space complexity renders our naive approach useless as we can not scale it.

In order to reduce memory and ensure for routing costs that are proportional to distances, there are two parameters that routing schemes try to minimize.

Stretch The max ratio over all source-destination pairs between the cost of the path taken by the routing scheme and the cost of a min cost path.

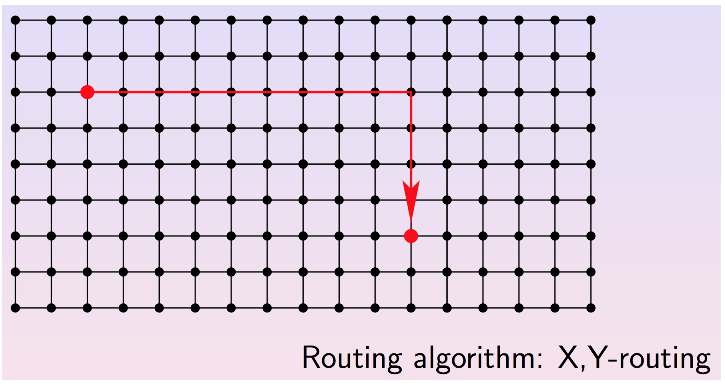
Memory The max number of bits over all nodes stored for the routing scheme. (balanced is preferred)

This article presents a routing scheme that uses $\tilde{O}(\sqrt{(}n))$ routing table space per vertex, and routes along paths of stretch-3.

Labeled Routing or Name-Independent Routing

In the literature we distinguish between *labeled routing* or *name-independent routing*. To illustrate the differences lets look at a grid and imagine that the intersections is vertices and the lines are edges.

In labeled routing we are allowed to label the vertices ourself. This allows us to give every vertex a label representing the (x, y)-coordinate of that vertex. This way we encode topological information into the label names and we can easily navigate in the graph by forwarding messages to desired port. In this scenario the available ports for each vertex would be a subset of $\{+y, -y, +x, -x\}$. Figure **??** illustrates this scenario.



In contract, when doing name independent routing, an adversary will label the vertices. If the labeling is done randomly it will hold no routing information and we can't use the label names to navigate. Thus, we need some kind of technique to figure out what port to use for message forwarding.

The routing scheme presented in this report is name independent.

2 The Stretch 3 Scheme

We shall firstly introduce a stretch 3 scheme for complete graphs and then generalize this to generic graphs.

2.0.1 Vicinity Balls

We introduce vicinity balls as the notion of areas containing the k closest nodes. For every integer $k \geq 1$, and for a node $u \in V$, let the *vicinity* of u, denoted by $B_k(u)$, be the set consisting of u and the k closest nodes to u.

`Property 3.1 (Awerbuch Et Al. 1990).` *If $v \in B_k(u)$ and w is on a min cost path from u to v, then $v \in B_k(w)$.*

`Proof.` Suppose $v \notin B_k(w)$. For any $z \in B_k(w)$, we have distnace $d(w,z) < d(w,v)$ (or $d(w,z) = d(w,v)$ and $z < v$, i.e. z is chosen before v). We have that $d(u,z) < d(u,w) + d(w,v)$. Since w is on a min cost path from u to v, $d(u,z) < d(u,v)$ and hence $B_k(w) \subset B_k(u)$. But since $B_k(u) \backslash B_k(w)$ we have that $|B_k(w)| < |B_k(u)|$, a contradiction.

Set $k = [4\sqrt{n} log\ n]$ and denote $B(u) = B_k(u)$. Denote $b(u)$ the radius of $B(u)$, $b(u) = max_{w \in B(u)} d(u,w)$.

2.0.2 Coloring

We introduce coloring. Partition nodes into the color sets $C_1, \ldots, C_{\sqrt{n}}$, with the following properties:

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.

If node $u \in C_i$, it has "color i" and we denote this $c(u) = i$. If every node independently chooses a random color, the properties holds with high probability by Chernoff bounds and union bound. It can also be derandomized, as shown in the following section.

Polynomial Time Coloring

The coloring can be derandomized via the method of conditional probabilities using pessimistic estimators, to show that the coloring can be constructed in polynomial time with the properties satisfied.

Let $C = \{1, \ldots, \sqrt{n}\}$ and let the randomized algorithm be as follows:

- For every $v \in V$ and $c \in C$ let $b_{v,c}$ be a binary random variable, equal 0 iff there exists a $u \in B(v)$ with $c(u) = c$. Note that for any $b_{v,c}$, $E[b_{v,c}] = Pr[b_{v,c} > 0] \leq (1 - \frac{1}{\sqrt{n}})^k = (1 - \frac{1}{\sqrt{n}})^{4\sqrt{n}\ log\ n} \leq n^{-4}$ (by directly applying the terms). Thus this represents the second property.
- For every $c \in C$ let e_c be a binary random variable equal 0 iff $|\{u|c(u) = c\}| \leq 2\sqrt{n}$. Thus this represents a bound on the number of nodes sharing the same color, i.e. the first property.
- For every $c \in C$ and $v \in V$ let $f_{c,v}$ be a binary random variable equal 1 iff $c(v) = c$. Thus this is an indicator variable on whether a node v has color c.

The derandomization follows from [1][37:8-37:9] and can be done in $\tilde{O}(n^2)$ time.

2.0.3 Hashing Names To Colors

Assuming a mapping h from node names to colors is balanced such that at most $O(\sqrt{n})$ names map to the same color. Each node $u \in V$ should be able to compute $h(w)$ for any destination w in constant time. Note that this hashing is for node names and not the node itself. Thus a node now has a node color and a name color.

2.1 Stretch 3 for Complete Graph

2.1.1 Storing

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 nodes v such that $c(u) = h(v)$ and what port number to use to reach them.

2.1.2 Routing

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 (i.e. min cost)
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 $d(u, w) + d(w, v) \leq d(u, v) + 2d(u, v)$.

3 Stretch 3 Scheme (for all graphs)

We now generalizes the stretch 3 scheme for complete graphs to generic graphs. To do this, we introduce routing on trees, landmarks and partial shortest path trees.

3.0.3 Routing on Trees

From Fraigniaud and Gavoille 2001, Thorup og Zwick 2001B, we have:

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 $O(log2n/loglogn)$ bits. Given the information of a node and the label of the destination, routing decisions take constant time, by Lemma 3.3 [1][37:6].

3.0.4 Landmarks

Designate one color to be the landmark color. Let L denote the set of nodes with this color. Because of the way coloring works we have that $|L| \leq 2\sqrt{n}$ and that 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 $B(v)$.

3.0.5 Partial shortest path trees

For any node u, let $T(u)$ denote a singlesource 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 $\mu(T(u), \dot{})$ is a subtree of $T(u)$ that contains u.

`Lemma 3.4.` *If $x \in B(y)$, then given the label $\lambda(T(x), y)$, node x can route to node y along a min cost path.*

`Proof.` By Property 3.1 for any node w on the min cost path of $T(x)$ between x and y we have $x \in B(w)$. Thus every node w on this path maintains $\mu(T(x), w)$

3.0.6 Storing

Every node 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 min cost path from u 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 $< \lambda(T(\ell_v), \ell_v), \lambda(T(\ell_v), v) >$. 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 followingproperties: $u \in B(w)$, 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 $< \lambda(T(u), w), x, (x \to y), \lambda(T(y), v) >$.
   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 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 3.4 on $y \in B(v)$.

3.0.7 Routing

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.

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.

3.1 Theorem 3.5

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

`Proof`. There are three cases to consider.

3.1.1 Analysis

1. *The taget is inside the source vicinity*: If $t \in B(s)$ or $t \in L$, then s routes on a minimum cost path directly to t.

2. *The source and target vicinities are far apart*: Let z be a node such that $z \in B(s)$ and $c(z) = h(t)$. For the case $c(s) = h(t)$, 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 (from options 4(a) and 4(b)). On every min 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, $b(s) + b(t) \leq d(s,t)$.

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

3. *The source and atarget vicinities are close*: There exists a min cost path in which every node is in $B(s) \cup B(t)$. Let (x,y) be an edge on this path such that $x \in B(s)$ and $y \in B(t)$. From 4(b), the cost $d(s,z) + p(z,t)$ of the path taken is bounded by the cost of path $s \rightsquigarrow z \rightsquigarrow x \rightarrow y \rightsquigarrow t$.
   Thus $d(s,z) + p(z,t) \leq d(s,z) + d(z,s) + d(s,t) \leq b(s) + b(s) + d(s,t) \leq 3d(s,t)$.

3.2 Results

The routing information stored at each node is bounded by

- $\tilde{O}(\sqrt{n})$ for 1, as $|B(u)| = \tilde{O}(\sqrt{n})$.
- $\tilde{O}(\sqrt{n})$ for 2, as each node will participate in $\tilde{O}(\sqrt{n})$ trees and thus its total tree routing information is of size $\tilde{O}(\sqrt{n})$ by Lemma 3.3.
- $\tilde{O}(\sqrt{n})$ for 3, by the same logic as for the above.
- $\tilde{O}(\sqrt{n})$ for 4, as for a node u the number of nodes v such that $c(u) = h(v)$ is bound by $\tilde{O}(\sqrt{n})$ (by the property 3.2 and the assumption of the hashing being balanced), and the lables stored are of $\tilde{O}(1)$ size.

giving the a total bound of size of $\tilde{O}(\sqrt{n})$.

The described stretch 3 scheme has the following bounds:

Stretch 3

Header size $O(log^2/log\ log\ n)$ bits

Routing information stored per node $\tilde{O}(\sqrt{n})$ bits

Routing $O(1)$ time

Construction time $\tilde{O}(n|E|)$

where the routing information stored per node

Improves the stretch of Arias et al. [2003] from 5 to 3, the known lower bound and answers the open problem from 1989 [Awerbuch et al. 1990].

Bibliography

[1] Mikkel Thorup et. al, *Compact name-independent routing with minimum stretch*, ACM Transactions on Algorithms **4** (2008), no. 3, 37:1–37:11.