1 Introduction

During this lecture the topic of Location Free Routing will be considered. We will be looking at the definition of the problem and three different papers that attempt to solve it.

What is Location Free Routing? The overall routing problem is the same, sending a message from source to destination. Nevertheless in location free routing the way coordinates are assigned are different. These coordinates that defines location are not based on geography or “Real Coordinates”, they are Virtual Coordinates.

The three paper that will be considered present similar “Related Work” which will help us to get a better idea of what the problem is.

1.1 Previous Work

All of these papers considered the algorithms available for routing and the ones that use Geographic Coordinates. Some of these algorithms are summarized in this section.

- **Distance Vector, Link State** This algorithm computes the shortest path between all possible pairs and because of the messages overhead it scales poorly.

- **Hierarchical Addressing (Internet)** This is the protocol that is used for routing in the Internet. But this will be hard to implement in nodes because the configuration and topology of the network is not known.

- **Geographic Routing (uses GPS)** In this algorithm the nodes know their geographical coordinates. It requires GPS or some other way to find out the coordinates, and uses greedy routing. The problem with these algorithms is that GPS can be expensive and it does not always work, for example indoors or under foliage.

Now that we have seen some of the related work for all the three papers let’s discuss the way each of them tries to solve the routing problem using virtual coordinates instead of geographic ones.
2 Geographic Routing without Location Information, Rao et.al.

2.1 Preliminaries

This paper starts talking about the usefulness of virtual coordinates. These virtual coordinates are defined as the hop count between nodes in a network. In other work the distance between node $u$ and $v$ is the hop count from $u$ to $v$. Then the considered that for virtual coordinates to be useful the following issues must be addressed:

1. Routing in virtual coordinates
2. How routing in virtual coordinates can be used in traditional ad-hoc routing and data centric networks

2.2 Routing Algorithm

They dont want to improve the geographic routing algorithms that already exists. They are interested in providing a set of virtual coordinates over which they can operate instead. The forwarding algorithm then works as follows: Each node has a routing table with their own coordinates, their neighbors coordinates and the neighbor neighbors coordinates (2-hop neighbors coordinate table). To route a packet the follow the following three rules:

- **Greedy**: Packet is forwarded to a node closest to the destination if (and only if) that node is closer than current node.
- **Stop**: If the node is closer than any other node to the destination is its routing table and higher layers determine that the packet is indeed bounded by this node, then the packet is considered to have arrived its destination.
- **Dead-End**: If the node cant make greedy progress and hasn’t reached a stopping point then it will perform an expanding ring search until finding a node closer to the destination.

2.3 Distributed Hash Table

For some application is it needed to maintain a Distributed Hash Table(DHT). For example Ad-Hoc and sensornets needs this kind of structure. The DHT works as follows. Each object in the network has a key and the DHT supports the put and get operations. When a node receives a put or get command with a key then it will hash the key to the virtual coordinates of the node and route the message to that virtual coordinate and then execute the put or get at that coordinate accordingly.
2.4 Coordinate Construction

To test their algorithm in this paper they test three different scenarios, decreasing the amount of location information about the perimeter nodes. These three scenarios are the following:

1. Perimeter know their location
2. Perimeters know they are perimeter but don't know their locations
3. Nodes doesn't know if they are perimeter nor their location

Now they give a description of the Simulation Scenario which consist of 3200 Nodes in a 200 X 200 units area with 64 perimeter nodes (15/side + 1/corner). The range of each node is 8 units and in average each node has 16 neighbors. Some assumption they made are that they ignore packet loss and signal attenuation. Then they proceed to present the simulation results for each the cases listed above.

2.5 Perimeter know their location

For this approach the paper describes a relaxation procedure for non-perimeter nodes to determine their virtual coordinates. The procedure starts with the perimeter knowing their positions and the non-perimeter node starting with a default location, which can be the same or different. In the examples in the paper they use the same default location for all nodes at the beginning of this procedure. Then the nodes will estimate their x and y coordinates based on their neighbors coordinates and will move closer to the position they should correspond. Nodes estimate their x and y coordinates based on the following formulas:

\[
    x_i = \frac{\sum_{k \in \text{neighbor set}(i)} x_k}{\text{size of}(\text{neighbor set}(i))} \\
    y_i = \frac{\sum_{k \in \text{neighbor set}(i)} y_k}{\text{size of}(\text{neighbor set}(i))}
\]

These relaxation procedure implies that non-perimeter nodes that have a perimeter as a neighbor will move towards it because the x and y position will be based on the average of the neighbors coordinates.

The results for this approach was tested using simulation and some of the results found are that the succes rate of packets arriving to its destination is very high, with 99.3% successful arrives. The average path length for this algorithm is 17.1. These results are very close to the case in which the nodes have true geographic coordinates in which case the success rate is 98.3% (slightly lower) and the average path length is 16.8 (slightly shorter).

The same test is run but this time not all perimeter nodes know their location. They tested it with only 8 nodes out of 64 knowing their true geographic location and the result are a success Rate of 98.9% and an average path length of 16.8. One disadvantage of this algorithms is that it can take long time to converge.
2.6 Perimeter Nodes dont know their Location

In this section the scenario in which the nodes know that they are in the perimeter, but do not know their locations is considered. A pre-phase is needed in order to compute the perimeter nodes locations and then previous algorithm discussed can be used.

A perimeter node will estimate it location by broadcasting a HELLO message to the entire network. When perimeter nodes receive the message they can compute the distance between each other. Once every perimeter node compute this distances, assume they are stored in a vector of distances. Each perimeter broadcast to the entire network its vector of distances for the other perimeters. This way every perimeter receive all the other perimeter vector of distance and make use of a triangulation algorithm to compute the coordinates to the other perimeter nodes.

Something interesting of this approach is that non-perimeter nodes can take advantage of the flooding that the perimeters nodes are doing and calculate their approximate positions. This will allow this algorithm to converge faster than the previous one presented. In the simulation results it only took one iteration for the nodes to achieve a very high success rate of 99.2% and an average path length of 17.2.

2.7 Perimeter nodes dont know they are

This section describes the scenario in which no nodes know if they are in the perimeter or not and also they do not know their location. For the algorithm to work, another preparatory stage is necessary and proceeds as follow. A subset of nodes elect themselves as perimeter nodes. These nodes broadcast HELLO messages to all the network. The other nodes use this information and the perimeter node criterion to see if they are in the perimeter or not. The perimeter node criterion consists of checking if the node is the farthest away one from all its 2-hop neighbors. If this is the case then the node elects itself as a perimeter nodes. The paper point out that not all nodes will make an accurate desicion, but this should not represent a problem if a subset of nodes did an accurate decision because othetr steps in the algorithms should solve this.

They presented more simulation results for this approach too and they are similar to the previous cases, the succes rate is very high and close to routing using geographic coordinates, the average path length is similar also. Nevertheless in the simulation results can be notice it that all three algorithms perform better in the presence of obstacles. This is because virtual coordinates represent the network connectivity better by using hops and not real distances. Now we are ready to take a look to another protocol to Location Free Routing with some similarities to this one, called Beacon Vector Routing.

3 Beacon Vector Routing: Scalable Point-to-Point in Wireless Sensornets, Fonseca et. al.

This paper is about Beacon Vector Routing (BVR) which is an algorithm for point-to-point routing in wireless sensornets. In this algorithm the nodes coordinates are assigned based on
hop counts to a set of beacons and are not based on the geographic location of nodes. The routing algorithm consist of a greedy approach which forward a message to a node closer to the destination. This paper present simulation results as well as a prototype implementation.

3.1 Motivation for BVR

First generation sensornets applications mainly focused on data collection. This limit the routing options using tree based to many-to-one or one-to-many. One-to-one routing is needed for the new applications for sensornets. Actually no practical point-to-point implementation exists.

For a one-to-one routing protocol be successful some of the design requirements are that the algorithms should be scalable, robust, energy efficient and minimal algorithm complexity. Complexity issues are mainly based on the experience of the developers of tinyOS while implementing basic algorithms like flooding a network. It took year to make this and the authors feel that the history can be repeated if complexity is not addressed properly. Other requirements simplicity and minimal assumptions about radio quality, precense of GPS and so forth.

3.2 Previous Work

They present some previous work that includes discussions on shortest path, hierarchical addressing, geographic coordinates, and virtual coordinates. The problem with the shortest path approach is that scales poorly in large network. Hierarchical addressing they think the algorithm is complex and can take year to implement. The geographic coordinates approach need some way to get its geographic coordinates like GPS, but this can be expensive and GPS does not work well under certain environments. The use of virtual coordinates like the previous paper, they claim scalability issues and can be hard to implement in practice. Let consider then the BVR algorithm.

3.3 The BVR Algorithm

The BVR algorithm defines a set of coordinates and a distance function. The coordinates are defined as a vector of distances (hops count) to a set of beacon nodes. The algorithms uses a reverse path tree construction method to compute these distances. This is represented mathematically as follows: Let \( q_i \) denote the hop count from node \( q \) to beacon \( i \), and let \( r \) be the total number of beacon nodes. A node’s \( q \) position is then \( P(q) = < q_1, q_2, \ldots, q_r > \). These nodes are assumed to have a unique identifier which in case that the vector of coordinates computed is the same, will differentiate one node from the other. For this algorithm to work the nodes must know their neighbors positions.

3.4 The Distance Function

The distance function defined by this algorithm takes two arguments, a node \( p \) and a particular destination \( d \). This metric measures how good node \( p \) would be as the next hop to reach node \( d \).
The metric favors neighbors whose coordinates are more similar to the destination, and always moves toward beacons if the beacons are closer to the destination. The metric is the following:

\[
\delta^+_k(p, d) = \sum_{i \in C_k(d)} \max(p_i - d_i, 0)
\]

\[
\delta^-_k(p, d) = \sum_{i \in C_k(d)} \max(d_i - p_i, 0)
\]

Here \(C_k(d)\) is the set of \(k\) closest beacons to \(d\). \(\delta^+_k\) is the sum of the differences for the beacons that are closer to the destination \(d\) than to the current routing node \(p\), while \(\delta^-_k\) measures the sum of the distances to the farthest beacons. The algorithm chooses the next hop that minimizes \(\delta^+_k\) and if there is a tie then is broken by minimizing \(\delta^-_k\). Actually what they do is to implement \(\delta_k = A\delta^+_k + \delta^-_k\) for some large constant \(A\).

To route a message to a destination \(d\), a packet has three header fields: (1) the destination’s unique identifier, (2) its position among the \(k\) closest beacons and (3) the minimum \(\delta\) the node has seen so far. Forwarding a message proceeds as follows: The packet carries the minimum \(\delta\) it has seen so far, according to the metric defined above. Greedily tries to find a neighbor which minimizes the current \(\delta\). If it finds such a node the forward the packet to it. If not then the node will forward the packet to the beacon closest to the destination \(d\), by sending it to the node’s parent in the beacon spanning tree. Once received by the parent then a greedy attempt to forward will be tried. If it fails all the way up to the beacon node then a controlled flood, ensuring that the packet will be delivered in the worst flooding the network.

Because in sensor networks nodes are prone to fail, a mechanism to maintain the beacon nodes must be provided. The algorithm for Beacon Maintenance works as follows. The beacon nodes will be constantly updating their sequence number. When a node detects that a beacon is not updating its sequence number after certain amount of time, it will remove it from its set of beacon nodes. When the number of beacons is below a threshold \(r\), then non-beacon nodes will nominate themselves as beacon nodes. If the number of beacon nodes goes over \(r\) then some beacon nodes will cease to serve as beacons, based on its identifier.

### 3.5 Simulation Results

The presented their work with a series of high level simulations. The following assumptions are made: fixed circular radio range for the nodes. The simulator ignores capacity, congestion and packet losses. The simulation is carried in the following scenario: 3200 nodes in a 200 X 200 square units area. They assume a radio range of 8 units and the average number of neighbors is 16. They considered the following metrics in their experiments: greedy success rate, flood scope, node load, control overhead, per-packet header overhead and routing state.

When testing for Routing Performance vs. Overhead they noticed that the number of beacon nodes can be kept to a minimum and the necessary number of beacon nodes to achieve a high success rate is in the order of 10. When testing the Impact of Node Density with 10 beacon nodes
nodes the case for high density performed really close to the routing with true geographic positions, but when the density is low this algorithm performs a lot better than with true position with success rates in the 90% while with true position can only achieve about 60%. When testing for Scaling the Network Size we can see that if a 2-hop information is maintained, then the number of beacon nodes can be kept low (about 10 will be fine) while using 1-hop info we will need to perform more floodings. The test for Performance Under Obstacles showed that this algorithm performs better than the case with true positions. This is because the virtual coordinates represents better the network connectivity.

3.6 BVR Implementation (key issues)

When implementing the algorithms in real nodes, they indentified four key issues that must be addressed: Link estimation, link/neighbor selection, distance estimation and route selection. They come up with the following conclusions for each of these issues. For Link estimation they noticed that the nodes always selects a high quality neighbor, for Routing performance the found out success rate about 97% when the network load is low. In terms of Dynamics BVR sustains high performance under high node failure rate and in Coordinate Stability they noticed this one varies little in magnitude and over time. In other words changes to the overall topology are minimum. The next section give a high level description of the last paper for this topic, the GLIDER approach.

4 GLIDER: Gradient Landmark-Based Distributed Routing for Sensor Networks, Fang et. al.

4.1 GLIDER

For this algorithm the authors assume that the nodes in the network are fixed and they dont necessarily know their geographic location. Some pre-processing phase is necessary to discover the global topology information and connectivity information of the nodes. This algorithm what it does is to divide the nodes in tiles. Each tile will have a node called a perimeter which will know their location. Within each tile, nodes will try to forward a message using greedy routing algorithms. The topology inside a tile should be trivial so these algorithms should work easily. To identify a node within this network two pieces of information are crucial: the node id and the tile it belongs to. The paper goes on to define the Node address = (tile_id, node_id). For routing using the GLIDER algorithm the landmarks will be forwarding the message from tile to tile until finding the tile that contains the destination node. Within that tile the nodes then will forward using greedy measures as mentioned above.

4.2 Background

Some motivation for this work is presented in the background section. New sensornets applications that require point-to-point routing is one of it. Also having algorithms that depend
on geographical coordinates can be expensive, inaccurate, and can not be used always like in the case of GPS that does not work indoors. The virtual coordinates algorithms available for location free routing they complain that can be time consuming and may not reflect well the network geometry.

4.3 Topology-Enabled Routing

In this section a high level description of the overall approach is given followed by the specifics for this algorithm in the next section. When using virtual coordinates should be noticed that these depends on node connectivity and not on actual positions. To be able to route a message the following two phases are necessary

Phase 1: Global preprocessing: Discover global topology, connected components, holes, and divide the nodes into tiles with trivial topology

Phase 2: Local routing Use greedy forwarding within a tile with trivial topology. All the forwarding decisions are local to the tile.

Both phases use a subset of landmarks nodes which know their location. The algorithms the makes use of Voronoi and Delauney techniques to extract a complex whose vertices are the landmarks. The local coordinates are computed based on the distance to nearby landmarks. The next section explains how the GLIDER algorithm does this.

4.4 Overview of GLIDER

In the GLIDER algorithm we will assume that the network can be described as a graph G=(V, E), and we also have the graph distance information based on hop count between nodes. The algorithm will assign a name to each node in V. It will then construct an Atlas M(G) which describes the general topology of the sensor field, nodes divided in routable tiles, and tiles relationship.

Constructing Atlas M To construct the Atlas M the algorithm constructs a Voronoi Complex and then takes a Delaunay triangulation. The resulting Voronoi cells will become the tiles and the Combinatorial Delaunay triangulation will become the map M.

4.5 Routing a packet

For routing a packet in this network we should remember that the nodes name are give as (tile_id, node_id). The algorithm will the route over the tiles until finding the appropriate one that matches the tile_id using the map M. Then within that tile use greedy forwarding methods until finding the node whose id is node_id

4.6 Local Landmarks Coordinates

Define the local coordinates to route within a tile using greedy approach. These coordinates will use hop count instead of Euclidean distance. They provide two examples the continuous
case that uses Euclidean distance and the Discrete Case that uses hop counts. The discrete case is the one that they later use in their simulation.

4.7 Naming and Routing

It should be noticed that in this algorithm the $node_i \neq node_{ame}$. $node_{ame}$ consist of the ID of its home landmark and the list of neighbors distances to its reference landmark. This name may not be unique and they mention that this should not happen very often and should be easy to resolve.

4.8 Simulation and Results

They present some simulation result to test their algorithms. The assumed that there is no packet loss, no delay and the nodes have a fixed radius range. They placed 2000 nodes on what they called a perturbed grid, which has holes, etc. They concluded that their algorithm has a high delivery rate in general and that it performs similar to the case with geographic coordinates but as in all the location free routing algorithm seeing so far it performs better in the presence of obstacles.