6.885: Distributed Algorithms for Mobile ad hoc Networks
Fall, 2008

Class 10
Today

• Point-to-point routing without location information

• Reading:
  – Rao, Papadimitriou, Shenker, Stoica. Geographical routing without location information
  – Fang, Gao, Guibas de Silva, Zhang. GLIDER: Gradient Landmark-based Distributed Routing for Sensor Networks
  – Fonseca, Ratsanamy, Zhao, Ee, Coller, Shenker, Stoica. Beacon Vector Routing: Scalable point-to-point routing in wireless sensors
Point-to-point routing

• Scenario:
  – Network of nodes
  – Want to send data from node A to node B (point-to-point)
  – Sensor networks, MANETs, etc.

• Challenge: Find an efficient route from A to B
  – Nodes may not fully know network topology
  – Topology may change as devices are added/fail
The naïve way

- Epidemic routing / flooding
  - We’ve seen this a lot before
  - Nodes forward packets to all of their neighbors (usually broadcast)
  - Aim for total network coverage
  - Will get a message from A to B, but at massive overhead cost!

- Can we do better?
Some better ways

- Distance Vector routing
  - Compute shortest paths between all pairs.
  - Every node stores next hop to every destination
  - $O(n^2)$ storage – doesn’t scale very well

- On-demand routing
  - Routes established on demand by flooding route requests
  - $O(n^2)$ overhead to establish route – high latency

- Hierarchical addressing
  - Used in internet
  - Hard in mobile networks where structure is changing

- Landmark routing
  - Nodes choose own address, based on concatenation of nearest landmark nodes
  - Difficult to maintain under mobility
An even better way: Geographic

• Geographic routing
  – Nodes know their coordinates, the coordinates of their neighbors, and the coordinates of the destination node.
  – Simple, greedy algorithm:
    • Always forward data to the neighbor closest to the destination
  – O(1) storage requirement: just maintain coordinates of your neighbors
  – Extremely efficient; quite reliable: ~98% success rates
  – Has some problems:
    • Local minima (no one nearer to destination than current node, but current node is not destination)
    • Need to know coordinates, not just address, of destination node
    • There are ways to get around these limitations; we’ll see them in future lecture
Problems with Geographic : GPS

• Geographic routing has one big flaw: the need for GPS
• GPS has some problems
  – Cost
  – Energy usage
  – Needs line-of-sight; doesn’t work indoors or under heavy foliage
  – Proximity not always indicative of connection quality (may have blockage between two physically-near nodes)
• But it would be great if we could use this simple greedy algorithm without running into the issues with GPS...
Virtual Coordinates

• Today’s papers look at the viability of greedy routing with “virtual coordinates”
• Virtual coordinates don’t necessarily have to be representative of physical coordinates, just need to work well under greedy routing
• Essentially heuristic-based routing
• We’ll see a number of ways of assigning and maintaining virtual coordinates
• Authors find that success rates can be just as high as those recorded with true geographic routing, if not higher
Geographic Routing without Location Information

- Rao, Papadimitriou, Shenker, Stoica
- Goal: assign nodes virtual coords and evaluate greedy routing performance
- Simple routing algorithm. Three cases at each step:
  - **STOP**: If I am the destination, stop
  - **GREEDY**: If I am not the destination, and a neighbor of mine is nearer to the destination than I am, forward to the neighbor
  - **DEAD-END**: If I am not the destination, and no neighbor is closer, perform expanding ring search until I find a closer node, or TTL expires
- Distinguish between perimeter and non-perimeter nodes
- Consider three cases. Perimeter nodes know:
  - That they are perimeter nodes, and their true 2D coordinates
  - That they are perimeter nodes, and nothing more
  - Nothing
Evaluation framework

• Simulation
• 3200 nodes, distributed randomly across a 200x200 area
• 64 ‘perimeter’ nodes
• Nodes have range of 8 units
• Average neighbor count of 16

• Assume:
  – Perfect radios within 8 units
  – No collisions
  – Clearly not realistic, but argue they are trying to evaluate how good the coordinates they assign are for routing, compared to true geographic coordinates.
Evaluation framework, cont.
1. Perimeter nodes know their location

- Simple iterative relaxation algorithm
- Perimeter nodes know their coords, all other nodes start with some default coords
- Until convergence: set my (x,y) to be the average of my neighbors (x,y)’s

What’s the concept at play?
- Model network as a spring
- Each ‘link’ (neighbor relation) represented by force
- Force is the difference between neighbor’s x (or y) coordinates
- If you hold the neighbors of a node fixed, its equilibrium position (Σforces = 0) is where x-coordinate is avg. of neighbors’
- So set x to average of x, iterate until total equilibrium achieved
- Non-perimeter nodes will move towards perimeter and bring their neighbors with them
Results

• Results:
  – True coordinates: 98.9% success rate; 16.8 hop path length
  – Virtual coordinates: 99.3% success rate; 17.1 hop path length

• Higher success rate with virtual coords!
  – Their coordinates reflect underlying topology; fewer local minima

• Show that using 8 perimeter nodes (instead of 64/3200) gives 98.1% success rate.

• But, coordinate assignment took 1000 iterations… not good
Coordinates after several iterations

Figure 2: The virtual coordinates of the non-perimeter nodes after (a) 10, (b) 100, and (c) 1000 iterations, respectively. The initial virtual coordinates of non-perimeter nodes are set to (100, 100).
True coordinates
2. Perimeter nodes don’t know location

- Preprocessing step so perimeter (P) nodes can find coords:
- Each P node floods HELLO throughout network
- After this, every P node knows all other P nodes and its distance, in hops, to each
- Then, all P nodes share this information. So all P nodes know all other P nodes’ hop vector to every other P node
- Now, they all perform a triangulation algorithm
- Minimize sum of squares of all pairwise distance errors.
- Error = difference between measured distance in hops and distances calculated from coordinates
- These coordinates are only unique up to rotation, reflection and translation, so compute center of gravity of the P nodes and use two P nodes to resolve rotation and reflection
Results

• They make an optimization: non-perimeter nodes will hear all the preprocessing messages so can come up with good proposals for their own initial coordinates
• Only need one iteration to get good results
• Find that 1 iteration of the relaxation algorithm gives 99.2% success rate, 17.2 hop path length.
  – Recall: 99.3% success rate; 17.1 hop path length when perimeter nodes knew coordinates
3. Perimeter nodes know nothing

- Add another preprocessing stage:
- Choose a bootstrap node to broadcast HELLO
- Each node elects itself to be a perimeter node if it is the furthest from the bootstrap perimeter node amongst all its 2-hop neighbors.
- Not a massive problem if nodes misclassify themselves as perimeter – show that success rate of 99.6% with 17.3 hop count can be achieved after 10 iterations even with misclassified node errors
Figure 4: The virtual coordinates of a network with 3200 nodes and with 64 perimeter nodes. The perimeter nodes use triangulation to compute their coordinates, while non-perimeter nodes use the relaxation algorithm to compute their coordinates.

Figure 5: The virtual coordinates of a network with 3200 nodes where no node knows its coordinates or whether it is on the perimeter.
Figure 9: The success rate of greedy routing with virtual and true coordinates for increasing network sizes at two different densities.
Summary

• Overall, interesting paper with some novel ideas
• They claim that the process supports node mobility, but with all the preprocessing involved, this seems dubious…
• Shows that virtual coordinates are viable, and that they can perform than true coordinates
• As scalable as geographic, though heavier pre-processing required as network grows
Beacon Vector Routing

- Fonseca, Ratsanamy, Zhao, Ee, Coller, Shenker, Stoica
- Similar to last paper
- Goal: create a simple way to assign virtual coordinates and route based on these coordinates successfully
- From the world of sensor networks; interested in practical implementation
- Simplicity, simplicity, simplicity!
  - Note that simple algorithms are hard to implement in real sensor networks. (Trivial flooding, tree construction took years to get right.)
  - Want robustness guarantees
  - Want to be able to go back and improve it (hard if it’s complex)
BVR - Strategy

- Randomly choose a small set of ‘beacons’ – like perimeter nodes, but have no special properties (e.g. not necessarily on perimeter)
- Each beacon constructs a spanning tree of the network, so every node knows its shortest distance to each beacon
- Each node then creates a ‘beacon vector’: a vector containing its hop count to each beacon
- No need for virtual 2D coordinates; route using a heuristic based on beacon vectors
BVR - Heuristic

• Assume we know a node $d$'s ‘beacon’ vector.
• We need to route to $d$. How do we know which neighbor to forward to so as to maximize $\Pr(d \text{ gets message})$?
• Define a distance metric $\delta(p, d)$ that measures how near a neighbor $p$ is to the destination $d$:
  – Choose $C_k$, set of $k$ beacons closest to $d$
  – Consider subset $l$ of beacons $i$ s.t. $d_i < p_i$ – the beacons that the destination is closer to than $p$ is
  – Intuition: use beacons near $d$ for routing; far beacons not helpful
  – For each $i$ in $l$, calculate $p_i - d_i$
  – $\Sigma p_i - d_i$ gives a heuristic for how far $p$ is from $d$
  – Break ties by considering beacons in $C_k - l$, the beacons $i$ for which $p_i < d_i$ – the beacons that $p$ is closer to than the destination
  – Move away from the neighbor
  – But can be wrong metric: trying to get closer to $d$ by moving away from a beacon that is far from $d$ – could be moving the wrong way
# BVR - Example

<table>
<thead>
<tr>
<th></th>
<th>$d$'s vector</th>
<th>my vector</th>
<th>$p$'s vector</th>
<th>$q$'s vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1$</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>$B_2$</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$B_3$</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$B_4$</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>$B_5$</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

$k = 3$

$C_k = \{B_2, B_3, B_4\}$

$I = \{B_2, B_3, B_4\}$

$I = \{B_2, B_4\}$

$I = \{B_2, B_4\}$

$\delta(me, d) = (3-1) + (4-3) + (4-2) = 5$

$\delta(p, d) = (2-1) + (3-2) = 2$

$\delta(q, d) = (2-1) + (5-2) = 4$

So choose $p$. 
BVR – Routing Algorithm

• Plan: be greedy!

• Packet carries best distance vector encountered so far
  – Usually this will be the current node’s distance vector

• At each hop, look for a better distance vector and forward there

• When this fails, use ‘fall-back’ mode:
  – Forward the packet towards the beacon closest to \( d \), but this time do so by sending it to your parent on that beacon’s spanning tree.
  – Parent tries to continue greedily
  – If it gets to the root of the tree, initiate a controlled flood. (Fair, as the node should be near the root)
Evaluation framework

- 3200 nodes, spread across a 200x200 square
- Radio range of 8 units
- Average neighbor count 16

Assumptions:
- Perfect radios within 8 units
- No packet loss
Plan

• Come up with a few metrics:
  – Greedy success rate: how often is purely greedy routing successful?
  – Flood scope: when flooding required, how many hops do floods cover?
  – Path stretch: what is the ratio of the length of BVR routes vs. routes found with true geographic coordinates?

• Vary total number of beacons, \( r \). Vary \( k \), the number of beacons to use for routing. See how algorithm performs.
Performance vs. Overhead

• They find that $k = 10$ routing beacons beats geographic
• Little improvement with $k > 10$, so stick with $k = 10$
• Find that $r = 20-30$ for $k = 10$ gives excellent performance
• This is 1% of total number of nodes, so acceptable overhead.
• Floods reduce as function of $r$ from 7 at $r = 10$ to 3 at $r = 70$
• Path stretch is 1.05 – not bad at all
Pure greedy success vs. # beacons

![Graph showing the success rate without flooding versus the number of beacons for different numbers of routing beacons: 5, 10, and 20. The graph includes a line for true positions.](image)
Results, cont.

- Low density: BVR outperforms geographic routing, though neither excellent

- On-demand 2-hop neighbors in low density:
  - If you can’t find a route, ask about your 2-hop neighborhood and see if someone is nearer the destination. Increases success rate for both BVR and true positions

- Fix target success rate at 95%. Set $k = 10$. What value of $r$ can guarantee this success as # nodes grows?
  - $r = 10$ is enough using BVR plus on-demand 2-hop neighborhoods.
  - With just 1-hop neighborhoods, the number $r$ grows steadily, but still small, so not horrible
Effect of density

Figure 2: Success rate of routes without flooding, for 3200 node topologies with different densities, for $k = 10$ routing beacons.

Figure 3: Success rate of routes without flooding, for the same topologies as in figure 2, comparing the on demand acquisition of 2-hop neighborhood information.
Implementation

• Simplifying details:
  – Always use \( k = r \)
  – Don’t implement some optimizations (2-hop neighborhoods)

• Issues that arise:
  – Link estimation
    • They use passive link estimation: tag packets with seq numbers, estimate how many you’ve received
  – Route selection (they use product of link quality and distance to choose next link)
  – More in the paper

• Not really the same algorithm…
Resilience to node failures
BVR - Conclusions

• BVR has high routing success
• Relatively simple
• High performance even under node failure (can route with partial beacon set)
• Not a lot of overhead
GLIDER

• Gradient Landmark-Based Distributed Routing for Sensor Networks
• Fang, Gao, Guibas, de Silva, Zhang

• Idea: Route based on two sets of information: global and local topologies.

• Lots of technical terms, but really quite a simple idea
GLIDER

- Goal: partition a network of nodes into a set of overlapping clusters called “tiles”
- A node’s address consists of its tile’s address and some local routing information
- All nodes maintain information about tile topology, nothing about inner-tile ‘local’ topology
- When routing to a destination, $d$, consider an abstract route over the tiles only
- Leads to a small amount of state that every node must remember. They call this an ‘atlas’
- Abstraction: assume large-scale topology is slow changing. Aim to route over this topology and let the nodes in each tile implement the actual greedy routing.
GLIDER: Overview

Two phases:

1. Global preprocessing
   - Divide graph into tiles of uninteresting clusters of nodes, centered by a landmark node
     - Uninteresting = no distinct topological features
     - Topological features (such as holes) hurt routing
   - Tiles should reflect underlying network structure

2. Local routing
   - Greedy forwarding based on local coordinates within each tile
   - This works well in a tile, as tiles have no topological features

Claim this extends to 3D, but really only cover the 2D case
Landmark example
GLIDER: Overview, cont.

- $M$, an ‘atlas’ that contains adjacency information for tiles
  - Construct a Voronoi complex, then take its combinatorial Delaunay triangulation
  - Explanation:
    - Landmark Voronoi Complex: tiles are Voronoi cells of the landmarks. Nodes belong to the cell of the nearest landmark
    - Nodes can be in multiple tiles, as there can be ties when deciding on closest landmark
    - Combinatorial Delaunay triangulation is just the adjacency graph among the tiles

- Every node maintains $M$

- A node has a name $v$
  - Two parts: the global tile name (id of closest landmark, break ties) and the local landmark coordinates. (Distance from the node to its own and nearby landmarks, similar to BVR)
Routing in GLIDER

• Given a destination $d$, known by name, come up with a high-level sequence of tiles.
• Once in the tile, use gradient descent towards next landmark, or towards $d$ if in last tile.

• Example:
Sanity-check

• For $v \in L$, ($L$ is the set of landmarks) define Voronoi cell $T(v)$ to be the set of nodes whose nearest landmark is $v$. A node may be in more than one Voronoi cell.

• Lemma 1: For any node $u \in T(v)$, every shortest path from $u$ to $v$ in the network $G = (V, E)$ is completely contained in $T(v)$.

• Thus, the spanning graph on each V-cell is connected.

• For CDT, they restrict it as follows: $D(L)$ is a simple connectivity graph for the landmarks.

• Two landmarks are connected iff their V-cells share a common node, or there are two “witness” nodes in the two V-cells that are directly connected in $G$.

• Theorem 2: If $G$ is connected, then $D(L)$ is also connected.

• Proofs: See paper.
Landmark selection

- Atlas, $M$, should be small so we have little overhead
- But, need to ensure that each V-cell (tile) has a simple, featureless topology. This is more important!
- So put landmarks near topological features; then their cells won’t contain them accidentally
- Suggest either:
  - Choosing landmarks manually
  - Select landmarks by discovering hole boundaries (an easy topological feature to find)
Gradient property of local coords – continuous version

- Take the set \( \{u_1, u_2, ..., u_k\} \) of landmarks
- Assume nodes know their position \( p \) in Euclidean space
- Define \( B(p) \) to be the \( k \)-vector of squared distances to the \( k \) landmarks: \( B(p)_i = |p - u_i|^2 \)
- Define \( B'(p) \) to be the average squared distance over all \( k \) landmarks
- Define \( C(p) \) to be the centered landmark-distance of a node \( p \), where \( C(p)_i = B(p)_i - B'(p) \)
- Distance \( d(p, q) \) between two points = \( |C(p) - C(q)|^2 \)
- Lemma: If \( k \geq 3 \), then (for 2D), gradient descent always converges to the target
  - Proof: Show that \( p \) to \( C(p) \) is an affine linear transformation, so is one-to-one
Proof of convergence

• Evaluate \([B(p)]_i = |p|^2 - 2p \cdot u_i + |u_i|^2\)
• \([C(p)]_i = B(p)_i - B'(p)\)
• \([C(p)]_i = -2p \cdot (u_i - \hat{u}) + w_i\), where \(\hat{u} = \frac{1}{k} \sum_j u_j\)
  and \(w_i = |u_i|^2 - \frac{1}{k} \sum_j |u_j|^2\)
• So, the function \(p \rightarrow C(p)\) is an affine linear transformation.
• If there are three non-collinear landmarks, the map is one-to-one.
  – Find one point in map determined uniquely by its coordinates, because then (for an affine map) the same must be true of all points in the plane
  – The circumcenter of any 3 non-collinear landmarks is such a point (in 2D)
  – Thus, the map is one-to-one, and affine linear
  – Thus, the gradient is zero nowhere except at the destination itself
Local coordinate generation

- Discrete version: this is the one that is needed, as we only have discrete numbers of hops
- Same basic idea, except use hops to landmarks instead of actual distances
- Can have local minima, though claim that sufficient node density reduces local minima
Naming protocol

- Once landmarks are selected, nodes must be named
- This is done in a distributed way
- Landmarks initiate flooding to determine graph distances from each node to nearby landmarks, and to compute Voronoi cells
- One centralized node receives all CDT information and distributes it
- Each node computes distances between itself and its reference landmarks
Results

• Simulation
• 2000 nodes
• Find that the algorithms work well if there are >= 5 neighbors; not very well otherwise. (20% success with 3 neighbors, 95% success with 4 neighbors.)
• Performs better than geographic routing in face of obstacles, as have the other algorithms
• Overall, seems good, and claims to work in 3D
GLIDER vs GPSR