Routing in Ad Hoc Mobile Networks

Readings:
Routing in mobile ad hoc networks:
*** Johnson, Maltz. DSR paper, Section 3
Hu, Johnson. Caching strategies in on-demand routing protocols for wireless networks.
*** Gafni, Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology.
*** Park, Corson. TORA. A highly adaptive distributed routing algorithm for... Section 2
*** Perkins, Royer. AODV. Ad hoc on-demand distance vector routing. Section 2

Next:
Location-aided routing:
*** Ko, Vaidya. LAR. Location-aided routing in mobile ad hoc networks. Section 3
*** Ko, Vaidya. Geocast. Geocasting in mobile ad hoc networks: location-based multicast algorithms. Section 3

Mutual exclusion, leader election, etc.
*** Malpani, Vaidya, Welch. Distributed token circulation on mobile ad hoc networks. Focus on the LR (Local Recency) algorithm.
*** Malpani, Welch, Vaidya. Leader election algorithms for mobile ad hoc networks.
*** Walter, Welch, Vaidya. A mutual exclusion algorithm for ad hoc mobile networks.
Walter, Cao, Mohanty. A k-mutual exclusion algorithm for wireless ad hoc networks.

1 Gafni, Bertsekas

1.1 Overview

A famous paper, forms the foundation of many loop-free routing algorithms, and other algorithms (see next time), for ad hoc mobile networks.

Setting: a fixed central station plus any number of mobile nodes.
Nodes may join, leave, fail, move,...
Assume that, at any time, certain pairs of nodes can communicate. Communication on each link is bidirectional. Nodes can determine which neighbors they can communicate with.

Problem: Set up routes from all the nodes to the central station.
Do this by orienting all edges in the communication graph, in such a way that no loops are ever produced, and such that, if the network stops changing, then eventually the edge orientations stabilize so that every node has a path consisting of outward-directed edges, from itself to the central station.

The trick to making the graph loop-free is that each node maintains a "height" a, which is an element of a totally-ordered alphabet A, which it can increase from time to time.

Heights are always different at different nodes.
Each edge is defined to be oriented from the node with larger height to the node with smaller height.
That guarantees loop-freedom.

They define a class of algorithms.
Each specific algorithm is defined by a set of local rules for when a node should increase its height, and by how much.

Features of their approach:
Loop-free
Local decisions and changes only. Keeps communication low. Avoids flooding.
Provides alternative routes, based on multiple outgoing edges (redundancy).
Can respond to changes easily.

1.2 Graph problem

Problem is to set up a "destination-oriented" directed acyclic graph (DAG), which means that every node has a directed path in the graph to the distinguished destination node (central station).

Claim: A DAG is not destination oriented if and only if there exists some node other than the destination that has no outgoing link.

Proof: If there is such a node, then clearly it has no path to the destination so the graph is not destination oriented.
Conversely, suppose that the DAG is not destination oriented, and let i be some node that has no path to the destination.
If i has no outgoing edges then we are done, so assume it has an outgoing edge. Follow the edge to another node. If that has no outgoing edge then we're done, etc. So, we may assume we can keep following outgoing edges forever. But this must eventually produce a cycle, contradiction.
(Assuming only finitely many nodes.)

So, an equivalent problem statement is to produce a digraph such that every node has some outgoing link.

1.3 Full reversal algorithm

The simplest such algorithm: If a node sees that it has no outgoing edges, then it reverses the directions of all its incoming links, making them all outgoing.

Implemented using the "heights" strategy:
Heights are pairs (sequence-number, node-id).
The node-id is for uniqueness, as usual.
The fixed destination is assigned a minimum value, like (0,0).

Some atomicity assumption is needed, e.g., a node can determine the heights of its neighbors and change its own, without interference.

Can weaken this?
Actually, in the paper, they assume that all the nodes do all of this in parallel: all read the old values, then all change their values.
But I think this can be done by any subset.

The rule is simply that a (non-destination) node i that sees that all its neighbors have pairs larger than its own increases its own pair:
It sets its pair to (α, i), where α is one more than the max sequence-number of any of its neighbors.

1.4 Partial reversal algorithm

Observes that a node i with no outgoing edges doesn't really need to reverse all its edges—it's enough to reverse at least one edge.

The partial reversal algorithm maintains triples (α, β, i), where i is the process identifier, as before.

Again, each node examines its neighbors, and can tell by comparing heights if all the neighbors have larger heights (so it has no outgoing edges).
In this case, it raises its height, but not necessarily to be greater than all of its neighbors: it sets α to be one more than the *minimum* of its neighbors' α components.
That ensures that the new height is greater than that of at least one of its neighbors.

And it resets its β component, to be less than the second component of all of its neighbors who share the new α component.
This seems to be designed to make the new height greater than the fewest possible neighbors. ???

1.5 General algorithm

They generalize both the preceding algorithms by presenting a generic algorithm, which they then prove correct (safety and termination).

Arbitrary totally ordered height set A.
They let all the nodes with no outgoing links take steps at once (though I don't think this is necessary).
Each node must apply a local rule, based on examining neighbors only, to raise its own height. The choice may be nondeterministic.
Constraints:
1. Only nodes that have no outgoing edges can change their heights.
2. Every node with no outgoing edges must increase its height.
(Seems like overkill.)
3. The amount of increase isn't specified, but it must lead to unbounded increase if applied infinitely many times.

With only these assumptions, GB prove that:
Proposition 1: Starting from any assignment of heights, eventually, the edge directions stabilize.

Proposition 2: If a path is ever established from a node to the destination, then no node along the path will ever perform a reversal. (Thus, paths are stable.)

Prop 2 is easy to see: In order for a path edge to get reversed, the node at the target end would have to have all incoming edges. But each node on the path except the destination always has an outgoing edge. The destination never reverses anyway.

For Prop. 1, suppose that it doesn't terminate. Let N be the set of nodes that increase their heights infinitely many times, F the set of nodes that increase their heights finitely many times.
The destination never increases its height, so it's in F.
N is nonempty, since the algorithm doesn't terminate.

Eventually, all the nodes in N have heights bigger than all the nodes in F.
But some node i in N is adjacent to some node j in F (assuming connectivity).
Then eventually, the edge from i to j stays oriented outward from i.
Thereafter, i can never increase its height again (because it always have an outgoing edge).

Property 1 implies the basic property we need: that eventually the system stabilizes to a state in which the DAG is destination-oriented.
Proof: Property 1 says that the system eventually stabilizes to some state. In the final state, if anyone other than the destination has only incoming edges, then it would perform another step, which contradicts stability. So, it follows that in the final state, no node except for the destination can have only incoming edges. As noted above, that's equivalent to saying that the DAG is destination-oriented.