Readings:
Ko, Vaidya LAR paper
Ko, Vaidya multicast paper (skim only)
Kranakis, Singh, Urrutia (optional) Bose et al. Routing with guaranteed delivery

1 Introduction

We continue our study of the problem of routing a message from a particular source to a named destination, in a mobile ad hoc network.

Today and next time we will consider algorithms that use real, Euclidean coordinates to assist in routing.
Such algorithms were mentioned in last lecture's papers—in fact, those papers claim to improve on the use of real geographical coordinates.
So in a way, we are going backwards by now studying the previous "unimproved" papers.
However, it is pretty clear that a characteristic feature of mobile ad hoc networks is that they exist in a real geographical context, the nodes can learn about that geography, and can use geographical information as an integral part of algorithms.
So it's worthwhile to see how geographical info can be used to solve an important problem—routing.

We will begin with two papers by Ko and Vaidya, which are really the same paper written twice.
Those papers describe a geography-based technique for restricting flooding protocols of the kind we have seen earlier, e.g., in DSR.
The difference between the two papers is that one uses the technique to perform *route discovery*, in case the geographical coordinates of the target are known, whereas the other uses the exact same method to *multicast* a message to a particular geographical region.
But the method is the same, and in fact, the papers are nearly isomorphic.

Then we will proceed to a series of related papers that don't involve any flooding: each of these simply forwards a single copy of a packet from one node to the next.
These papers use a range of different strategies, starting from simple *greedy forwarding*, where a node just passes the packet to the neighbor who is closest to the destination.
A similar technique is *compass routing* (the Kranakis paper), whereby a node passes the packet along the edge whose angle to the target is the smallest.

But these techniques do not guarantee that the packet will reach the target—they can reach dead ends, where no next move is possible.
The next strategy, *face routing*, also in the Kranakis paper, avoids this problem by traversing successive regions of the network graph, using the "right-hand-rule" for traversing mazes.
This always terminates, but is not always very efficient.

Then we will read two fundamental papers: the one by Bose et al. and the very-well-known Karp and Kung paper on GPSR.
These papers develop a strategy that combines face routing and greedy routing, thus getting guaranteed delivery and reasonably efficiency in practice.

Finally, we will consider two papers that explore improvements on the earlier combined protocols: the Barriere paper, which attempts to add robustness, removing some of the strong theoretical assumptions, and the Kuhn paper, which develops a combined strategy that has both excellent provable worst-case behavior, and also performs very well in practical simulations.

Some things to think about when considering all of these papers:
1. What assumptions are they making about the underlying network? Are they reasonable? If not, are they crucial to the results, or can they be weakened?
2. What happens in the presence of change? Many of these papers have a rather static flavor—can they be modified to adapt to joins, leaves, failures, mobility? How?
(Good homework questions.)

2 Paper 1: Ko, Vaidya, "Geocasting in mobile ad hoc networks: Location-based multicast algorithms"

A little paper with a few small ideas. First, they are considering a multicast problem, not a point-to-point routing problem, in a mobile ad hoc network. They talk about multicasting to a "group", but throughout the paper, the group consists simply of all the nodes in a designated geographical region. They assume that the nodes have exact knowledge of their own positions, from GPS for example. They ignore any issues of obstacles. The problem is simply to get the message to everyone in the specified geographical regions.

Unlike the papers we have been considering, they are not considering solutions that involve sending only a single message to a destination area. Instead, they are considering solutions that involve considerable flooding of packets. Of course, that will be more "complete" in getting the packet to its intended destinations. But it also produces a lot of overhead. There's some kind of tradeoff here.

They give two slightly different algorithms, both involving flooding.

2.1 Introduction

Multicast region = the geographical region that a sender might want to send a message to.

They don't say that the region has to be one-hop radius; thus, some kind of flooding would probably be needed anyway, in order to distributed the message throughout the region once it reaches it. But it's not clear that they need to broadcast in order to get there—why do they do this?

2.2 Location-based multicast protocols

Goal for Geocasting: to send a message from a source to a group of nodes specified by being within a particular destination region. Very similar to the problem of multicasting, except implicit grouping by location and now in a MANET setting. For multicast (MC), two solution:

flooding MC : very simple algorithm that essentially just sends the message to all of the nodes connected to the source. If the nodes in the destination region are connected to the source, then they will receive the message. Seems very inefficient since ALL nodes connected to source will receive the message. It seems like a very simple algorithm and also seems robust to many types of node failures (although they don't say comment on this in paper).

tree-base MC : according to authors, tree-based approaches do not work well in MANET because of excessive overhead in tree-reconfiguration because of mobility of node. Therefore they, really, just make changes to a very simple flooding MC algorithm to lessen the overhead in flooding MC.

2.2.1 Multicast flooding

For the most direct comparison, they describe a global flooding protocol, which simply floods the message throughout the entire network in the hopes of finding all the intended destination nodes. Each recipient of the packet (the first time it receives the packet), checks to see if it in the target multicast region. If so, accept the packet. In either case, rebroadcast it.

Simple, robust, not very efficient.

2.2.2 Preliminaries

Location info from GPS (error-free). Consider 2D only (graph and motion).

Multicast Region : closed polygon (rectangle, circle, etc.) determining the destination region of message.

Measure completeness (they term is "accuracy"): defined as the ratio of the number of nodes that WERE IN THE multicast region when the geocast initiated that receive the message and the total number of nodes within the multicast region when geocast initiated. –¿ What about the nodes that receive the message because they moved INTO the multicast region during geocast time but where not initially in the multicast region ? it seems like these might be considered in a completeness measurement, but they choose not to. Is this because the only point of reference between different algorithms (that each could take different amounts of time to complete the geocast) is when a geocast is initiated? What are other good models for completeness?

Forwarding zone: A zone in which the packet gets rebroadcast. The proposed algorithms simply define certain restricted forwarding zones in which nodes will rebroadcast. These are supposed to be regions that are "on the way" from the sender to the multicast region. The forwarding zone should include the entire multicast region, plus the sender.

Tradeoff : completeness vs. packet overhead (bigger forwarding zone increase completeness but incur overhead)

2.3 The protocols

The algorithms are essentially identical to multicast flooding, except that a node not in the forwarding zone doesn't rebroadcast. The two algorithms here define different kinds of forwarding zones, and also different ways in which nodes determine whether they are in the forwarding zone.

They mention testing whether the node was RECENTLY in the forwarding zone instead of whether it is there RIGHT NOW, but don't pursue this.

2.3.1 Algorithm 1

Forwarding zone is simply the smallest rectangle that includes both the sender S and the entire multicast region, and that has its sides parallel to the x-axis and y-axis.

Here, the sender can determine the four corners of the forwarding zone, and send them along in the message. Easy for nodes to determine if they are in the region.

They tune this a bit by adding a buffer zone of width delta around the forwarding zone.

2.3.2 Algorithm 2

Forwarding Zone : determined by the nodes themselves when they receive a message The sender includes in a message : the multicast region, the center of the multicast region, and the coordinate of the sender (this last changes as the message gets forwarded)

When a node i receives a packet from node j, i determines if it's in the multicast region as follows: It calculates its distance from the center of the multicast region, and compares it to the same calculated distance for j. If i's distance is smaller (or, not too much larger), it decrees that it is in the forwarding region and behaves accordingly, rebroadcasting (but including its own coordinates in place of j's in the packet).

What is the forwarding region here? It seems to be the entire circle centered at the center of the multicast region, with radius equal to the distance from the sender to that center! By broadcasting, it seems that the packets can creep all the way around such a circle

Neither of these sounds like a very efficient way to get a multicast packet to just the particular multicast region of interest. Why flood at all? Why not just send a unicast until it reaches the multicast region, then start flooding through that region only? However, the idea of "controlled flooding" has been brought up before in other routing papers as ways to avoid failures in certain instances. These algorithms could be used as fallbacks.

2.4 Experimental results

They do a lot of runs of a simple simulation. They don't compare their two algorithms to each other, but rather, to basic multicast flooding. Unsurprisingly, both algorithms succeed in getting completeness that is comparable to basic flooding, but with much less packet overhead.

2.5 Optimizations

They quickly state some optimizations they think might work:

- They thought of various ways to dynamically change the forwarding zone for Alg 1 as a node received a message.
- Use a rectangle parallel to the line between source and center of multicast region rather than parallel to XY axis.
- Conic sections rather than rectangle for Alg 1
- directional antennae use

3 Paper 2: Ko, Vaidya, "Location-Aided Routing (LAR) in mobile ad hoc networks"

Hmm, they wrote the same paper twice! This paper is virtually identical to the previous one. The main difference is that they apply exactly the same techniques to a slightly different problem: route discovery in a MANET rather than geocasting.

Because of the different problems, there are a few little differences all the way through. And, this paper is longer, so has room for more experimental studies. But they aren't all that enlightening.

The protocols in this paper are called "Location-Aided Routing" protocols (LAR).

3.1 Introduction

Assume GPS. LAR for route discovery: Use location info, which may be out-of-date by the time it is used, to reduce the search space for a route. Limiting the search space results in fewer route discovery messages, compared to unconstrained flooding.

New issues in this paper, not discussed in the other paper, involve handling of time, velocity, etc. Thus, they try to take account of the fact that info may be out-of-date, by predicting where a node might be RIGHT NOW based on where it was some time ago, and knowledge about velocities, etc.

They review previous routing algorithms: DSR, AODV, TORA, ZRP. None of these take into account the physical locations of the nodes—so this paper claims to be the first one that does so. Note that this paper isn't suggesting it for data-delivery, but just for route discovery.

3.2 LAR Protocols

3.2.1 Route discovery using flooding

Standard flooding, without any constraints. Now, since this is route discovery, the packet also records the path it follows. When destination receives RREQ, responds with RREP on the reverse route. If sender doesn't receive RREP by a timeout, it reinitiates route discovery.

Sender initiates route discovery in the first place only if it is trying to actually reach the destination with a data message and either it doesn't know any route or it finds out the route is broken. It finds out a route is broken while using it if it gets a route-error message from some node along the way.

3.2.2 Preliminaries

Location is known without error (later they reduce this to an error margin and show that their alg can handle such cases)

Expected zone: The analogue of the multicast region. An area that is likely to contain the position of node D at time t1. E.g., t1 can be the time at which a sender S expects its flooded messages, from a route-discovery initiation, to arrive at D. S can determine the expected zone based on knowledge that D was at a certain position at time t0, using knowledge of the maximum (or average) velocity of D. Requires synchronized clocks to determine this—the info that originates from D about its position at time t0 must be correlated with a time on S's clock. The more info S has, the smaller the expected zone can be. Note that this estimate uses possibly stale location information for the node.

Forwarding zone (request zone): As before, used to constrain who forwards the message: a node forwards it exactly if it is in the forwarding zone. Forwarding zone includes S and the expected zone, plus enough other area to allow a path between them.

If route discovery fails, S will retry with a larger forwarding zone—perhaps the entire plane (unconstrained flooding).

The location information for a node is updated whenever a route-reply message is sent since the destination node will send it's current location (and other info such as velocity, etc.) with the reply.

3.2.3 Algorithm 1

Forwarding zone is the smallest axis-aligned rectangular region containing S and the expected zone. S calculates the expected zone based on prior knowledge of D's location and on an estimate of D's velocity. This defines the forwarding zone, sends it in the RREQ packet.

When D sends a RREP, it includes its current position, which S can use for future route discoveries.

Size of expected zone (and hence, of forwarding zone) depends on average speed of motion, and on age of the information about D's position. Circle centered at stale location of destination with radius = v($t_0 - t_1$). t_0 = time of location info of destination, t_1 = current time, v = average or maximal velocity of destination. Since they assume that position info only comes as a result of route discovery, this age depends directly on the time between route discoveries. Since slow speeds may mean infrequent new route discoveries, this says that either slow or fast speeds can lead to large expected zones. Can reduce the size for slow speeds by piggybacking position info on other messages.

Errors in position readings: Can be taken into account in the calculation of expected zone—just add a buffer zone of width e, where e is an upper bound on the error.

Needs clock synch for circle radius computation (although in section 5 they state that it can handle unsynch clocks). –¿ unsych clocks are "handled" by having a node timstamp another node's location using ITS own local clock. The thought here is that message delivery delays are small. I don't particularly think this is true, especially if a node's location was received over multiple hops.

source sends corners of request zone in message so a node can determine if it's within the zone and forward message or not.

Tradeoff: size of zone vs. overhead.

3.2.4 Algorithm 2

This doesn't use any clock synch. Destination is regarded as the last known position of D. Node S, any any other node along the way, calculates its distance to that position, and includes it in the message for the next hop.

When a node receives the packet, it compares its distance to the destination position with the distance already in the packet. If its distance is "essentially better", it forwards, else discards. In the forwarded packet, the new node's distance replaces the previous one. The determination of whether the distance is "essentially better" is tunable, based on parameters alpha and beta.

Errors in position readings: Don't need to modify the algorithm; however, performance may degrade with increasing error.

3.3 Performance evaluation

Similar to before—they compare these algorithms to unconstrained flooding and observe a substantial improvement. Here, the improvement is in the amount of overhead needed to achieve successful route discovery.

Thus, they show: Number of routing packets per data packet is consistently lower for both LAR schemes than unconstrained flooding. Improvement is most pronounced with large transmission ranges—with small ranges, connectivity is decreased, which reduces the probability of route discovery within the timeout interval (so then they fall back on unconstrained route discovery anyway). Similarly, improvement most pronounced with more nodes. They say that algorithm 2 is the best as speeds grow—I don't see why.

Effects of error: Can go both ways. Increasing error increases the size of the forwarding zone, so increases the number of packets. But it also increases the likelihood of succeeding on the first try, without defaulting to unconstrained route discovery.

3.4 Optimizations, etc.

Adaptation of forwarding zone, in scheme 1: Intermediate node i may have more recent info about destination's position than the source S did.

Can use this to modify the forwarding zone, to decide whether i itself should retransmit.

Local search for broken routes rather than a full rediscovery plan.

using TTL information to have route discovery within expanding rings.

Directional antennae

Imprecisely synchronized clocks, in algorithm 1: If they aren't too far off, can still use as before (?)

4 Paper 3: Kranakis, Singh, Urrutia, "Compass Routing on Geometric Networks"

4-page version, from Kranakis' page, not the version on our course web page.

4.1 Introduction

Geometric graph: Presumably, this is in 2D, though they don't say. Vertices are just points in the plane, edges are straight-line segments between them.

Compass routing, simplest form: From any vertex, always move across the adjacent edge that points most closely in the direction of the target t.

More generally, they consider "local routing algorithms". The traveling packet has state, including:

The coordinates of the starting position and destination.

Some extra storage to keep a constant number of identifiers of vertices in the graph. (Cannot store entire topology.)

Each vertex also has some information, but this doesn't change during the routing algorithm. The information in an arriving packet, plus the info at the vertex, are used to choose the next edge.

Why isn't the vertex' info allowed to change? Because we are thinking that the algorithm is being used concurrently for many traveling packets. We don't want the vertex to become a repository for info about lots of packets—that would cost too much space.

If we didn't have actual coordinates, but just an arbitrary graph, then this problem would not be solvable with deterministic rules. (To solve this version of the problem, we would need to leave markers of some kind at the visited nodes.)

They want local routing algorithms that work in a large class of geometric graphs—those that are planar (no crossing edges). Goal isn't necessarily to find shortest path connecting two vertices, but just to make sure the packet reaches the destination.

4.2 Compass routing (Greedy)

A geometric graph G "supports compass routing" if it always works, for any s and t.

Not every planar geometric graph supports compass routing. Even when the graph we are using is a triangulation, the algorithm can still fail to find an existing path. They give a nice example, based on a triangulated square plus some nearby w points.

Then they describe a particular kind of geometric graph, called the "Delaunay triangulation" of a set of n points P_n. This means to take the convex hull (smallest convex set) containing all the points in P_n, and then divide that into a set of triangles whose vertices are exactly the points in P_n. Do this in such a way that each circle defined by the vertices of a triangle in the triangulation has no other point of P_n in its interior. A theorem says that such a triangulation can always be constructed, provided that no four positions lie on the same circle ("general circular position").

Theorem 2.1: The Delaunay triangulation of any set of points P_n supports compass routing.

Proof: Based on the fact that if compass routing chooses to traverse edge (s,v) then distance(v,t) < distance (s,t), strictly decreasing. The argument is based on simple geometry—a little ad hoc, but it seems convincing.

4.3 Compass II (Faces)

Now they consider more general local information routing algorithms again, not just simple compass routing, although for some reason they call the section "compass routing II".

A geometric graph is convexly embedded if all the faces of the graph are convex except for the unbounded one. First, they describe their algorithm when their graphs are convexly embedded, they later remove this restriction.

Alg for convexly embedded graphs from s to t:

Start at s and choose face F_0 that intersect the line st.

Pick any of the two edges in F_0 incident of s and start traversing the face until we find the second edge (say uv) that intersects st.

Update the current face F to be other face containing uv on its boundary.

Traverse F until we find a second edge (say xy) which intersects st.

We update F to now be the other face containing xy in boundary.

Iterate until we find t.

Why does this work? because the graph is convexly embedded, we can traverse the faces arbitrarily and as soon as we hit another edge that intersects st we can move on to the next face since all the faces are convex. However, in a general geometric graph (not convexly embedded) the algorithm needs to be modified.

Compass Routing II: We work with the line s-t from source to target. That line passes through a "path" consisting of convex polygons, F0, F1, F2,...Fr. Normally, it will pass through successive edges, but in special cases it may pass through vertices instead of edges, and may even pass along some edges. (These last two cases are ignored in the paper—it seems the algorithm could be extended to handle them, though.) In the algorithm, the packet just moves systematically around the successive polygons in this path, until it reaches t. It seems that it can do this easily enough, with the limited available information:

The packet could always move clockwise around the faces. (using this “right-hand-rule” to traverse the faces makes this algorithm exactly like the Face-I algorithm in the Bose et al. paper.)

The packet itself contains s and t.

It can check whether an edge crosses s-t.

Now, what about arbitrary planar geometric graphs? Such a graph induces a partitioning of the plane into polygonal regions, but the faces needn't be disjoint. Well, it can be a little more complicated than that—some edges could even appear twice—see Figure 6.

To route on such a graph, we again work with the line s-t. Here they are again traversing faces one by one, only now, some of the faces may in fact be the external area. The algorithm will ALWAYS traverse the entire face, keeping track of all the intersection points between the line s-t and the face being traversed. After traversing the entire face, choose the intersection point farthest from the starting node for this face. Then traverse the next face that also included the edge containing the chosen point. In this way, we eventually reach the target, although each edge might be traversed at most four times ??? the paper states twice, but I cannot see how this is done and they did not prove it – the Bose paper states four times, which is what I can get ??? . To see this, notice that we traverse each face we 'choose' once fully to find all intersection points with s-t. Now, we never traverse a face more than once. However, after traversing a face to find the intersection points, we must then traverse the edges of that face to get to the chosen intersection point (farthest from starting node of face) on the next face. This will at most traverse each edge of the face again, therefore we traverse each edge of the faces at most twice, and each face at most once. Now, each edge is shared by two faces, so we can traverse each edge at most 4 times (similar reasoning is in the Bose et al. paper for their Face-I algorithm, which is nearly identical, except that the Bose papers forces the traversals to be done using the “right-hand-rule”).

4.3.1 The “Right-Hand-Rule”

This is not used as such in this paper, but later on in the lecture it will com up. Also, it seems like a standard way to traverse a face (which is used in this paper). The rule states that when arriving at a node x from a node y, the next edge traversed is the one in the counterclockwise direction about x from edge xy. The rule traverses the interior of a closed polygonal region (a face) in clockwise edge order. See Figure 4 in GPSR paper). The rule traverses the exterior region in a counterclockwise edge order.

4.4 Further work

They state a theorem near the end which seems to be rather disconnected from the rest of the paper. However, they are merely saying that their compass-II algorithm will only traverse a linear number of edges (see the proof above) and will always reach its destination if there exists a path. The algorithm is a “local routing algorithm” because it keeps minimal state, does not leave markings on any node, and traverses the graph based on choosing a particular neighbor to visit next.

Conclusion: This is written as a pure math paper. The results apply to planar graphs. Mobile ad hoc network communication graphs aren't necessarily planar, in fact, they would typically have many criss-crossed edges. So, although the math results are interesting, I don't see what to do with this in a practical setting.

Is there some obvious generalization to dense, not necessarily planar, networks? I suppose that would be just to use simple compass routing. It probably works pretty well.

Note that this is not quite the same as saying that we pass the packet to the neighbor who is closest to the target. But that seems like it could be a more useful strategy.

Later on (in the Bose et al. and later papers) we will see how we can create a planar graph and use such algorithms to route on the generated graph.

5 Planar Graphs

A Planar graph is a graph that can be draw without having edges intersect. These graphs are used in many of the algorithms used for routing using location information.

There are in fact two main planar graphs used (because they can be generated using simple distributed algorithms): Gabriel Graphs (GG) and Relative Neighborhood Graphs (RNG). The basic concept if to define a "witness" for an edge uv in a graph. If such a witness exists, then that edge can be removed and still maintain the same connectivity as in the original graph.

"witnesses" for planar graphs:

RNG : See Figure 5 in the GPSR paper. The idea is to remove an edge (u,v) if there exists another node w for which d(u,w) and d(w,v) are both $< d(u, v)$; that is, if there is a "witness" node w in the interior of the region in Figure 5 of GPSR.

GG : See Figure 6 in the GPSR paper. Now remove an edge (u,v) if there is another node w for which $d(u, v)^2 < d(u, w)^2 + d(w, v)^2$ That is, in the triangle uvw, angle w is $> \frac{pi}{2}$.

Note that this edge removal rule depends only on distances—it describes, for a given graph G, which edges should be removed, based only on distance info (not on edge info). So, an algorithm to construct this graph has some flexibility in the order in which it processes information about different edges.

Distributed algorithm that allows a node u to remove its adjacent links, producing the RNG: u just looks for pairs w,v where d(u,w) and d(v,w) are both $< d(u, v)$, and removes the edge (u,v) in this case. It can do this for all pairs (v,w) in any order.

Distributed algorithm that allows a node u to remove some adjacent links, producing the GG: u looks for pairs w,v where $d(w, m) < d(u, m)$, where m is the midpoint of the line segment u-v, and removes the edge (u,v) in this case. It can do this for all pairs (v,w) in any order.

Notice that this determination of edges (detection of "witness") is made separately by each endpoint. In this paper we assume a disk model of communication, so both endpoints will see that same witness. In later papers (Barrierre et al.), we will study a relaxation of this model and see how this affects the algorithms.

Eliminating edges that aren't part of the RNG or GG can't disconnect a connected graph: An edge is only removed if there is some node within range of both in both RNG and GG. Use an inductive argument, based on removing one edge at a time.

RNG is a subgraph of GG. RNG's connectivity is less than that of GG. We can see this from noticing that the "witness" region for RNG is larger than the region for GG, hence the connectivity if less than GG and RNG is a subgraph of GG.

6 Bose, Morin, Stojmenovic, Urrutia, "Routing with Guaranteed Delivery in ad hoc Wireless Networks"

This paper later evolved into GPSR. They consider stationary unit graphs. They give algorithms for three routing problems:

routing to a known geographical location
geocast to everyone in a known geographical region
broadcast to everyone in the network

They try to do this while having only one packet running around at a time, and with no memory maintained at the nodes. Still they want guaranteed delivery (under connectivity assumptions).

6.1 Introduction

Local information at each node: own location, location of neighbors and destination.
Unit disk graphs: Edge (u,v) in the graph iff $dist(u, v) \leq 1$.

The routing problems:

1. Routing:
   s wants to send to t, knowing t's coordinates (exactly)

2. Geocasting:
   s wants to send to everyone in a region r. Here they take r to be a disk, but the algorithms generalize to arbitrary convex regions.

3. Broadcasting:
   Broadcasting can be considered a special case of geocasting, in which r is a disk with infinite radius (essentially a radius equal to the diameter of the graph should be adequate).

Their algorithms are local—no global info is needed anywhere. Each node maintains information about itself and its neighbors—who the neighbors are, and what the locations are for itself and the neighbors.

Previous algorithms were:
Greedy: These don't guarantee that a packet reaches its destinations.
Flooding: Controlled packet duplication, like LAR. These don't terminate unless nodes remember what packets they have seen.

In their algorithm: Nodes do not maintain any info about packets. They guarantee to deliver the packet to all or its intended recipients, assuming the graph is static and connected. They don't make duplicates of packets as flooding algorithms do.

Basic idea: Algorithms find a connected planar subgraph of the unit disk graph and then apply routing algorithms for planar graphs on this subgraph.

6.2 Extracting a connected planar subgraph

Lemma 1 says that, if the original unit graph G is connected, then so is the Gabriel graph obtained from G by removing edges as before. Proof: They use [14] to claim that MST(S) is a subgraph of GG(S), so GG(S) is connected too. Show the proof based on contradiction.

Lemma 2 says that if an edge $(u, v) \in U(S)$, but $\notin GG(S)$ and w is a witness to this, then (u, w) and $(v, w) \in U(S)$.

Theorem 3. The cost of computing GG(S) is $O(dlogd)$ (but the algorithm shown in the paper is actually $O(d^2)$).

6.3 Routing in planar graphs

They give the algorithms for planar graphs, then say they can be applied to unit disk graphs too, after "planarizing" to the Gabriel Graph.

6.3.1 Routing

FACE-1, is essentially the face routing algorithm from Kranakis paper.
A connected planar graph G partitions the plane into "faces", each of which is a "polygon" bounded by edges of G. The external area is also counted as a "polygon".

¿From a vertex v, we can traverse the boundary of a face using the "right-hand rule". Actually, here the "right-hand rule" is described backwards from the Kranakis and GPSR papers. Essentially, it does the same thing, only counter-clockwise, instead of clockwise (GPSR). Anyway, they basically traverse a path around all the faces intersected by the imaginary line segment s-t. For each face, they travel all the way around in order to determine another edge that intersects s-t; they do this because there might be several, in general (these faces don't have to be convex), and they want to be sure they find the best one—the one closest to t. So, after they travel all the way around the face, they return to the best edge, then continue on the next face.
Worst-case 4|E| hops altogether. A nice proof for that has already been provided in the Compass II section.

FACE-2, a modification of FACE-1 that performs better in practice. This one doesn't search for the best edge on each face—it just looks for SOME OTHER edge on the same face that intersects with the line s-t. See Figure 2 on paper—this seems to just hug the line segement s-t, seems pretty direct. However, they claim in bad cases this can visit Omega(n2) edges, and give an example. In this case, the algorithm actually goes back once it finds an edge that intersects with s-t, because of following the "right-hand-rule" and there is no other edge on the right, except for going backwards. See Figure 3 in paper.

6.3.2 Broadcasting

They have an algorithm, from a previous paper, of enumerating all the faces, edges, vertices of a planar graph, without using any mark bits, or stack. The algorithm assumes some total order on the edges of G. They showed how to obtain a spanning tree of the faces: Just put an edge between faces f1 and f2 if the minimum edge on f1 is also an edge of f2. (This idea is the same as the one used in standard construction of minimum spanning trees)

Now, they use this spanning tree structure as the basis for traversing all the vertices of G:

Traverse the spanning tree of the faces, and when each face is reached, just traverse all the edges of that face. Apply this easily to perform a broadcast.

The hardest part of this is apparently in the previous paper—how to identify the minimum edge on each face. This is a standard leader-election problem, and in fact, the algorithm they use is essentially the standard Hirshberg-Sinclair leader election algorithm (in my book).

6.3.3 Geocasting

They can use FACE-1, for example, to reach the center of the target regions, and then use the traversal algorithm above to visit all the nodes in the target region. Notice that their algorithms don't make any use of greedy routing anywhere—just face routing!

Theorem 7. Total cost of GEOCASTING: $O(n + klogk)$ (n for FACE-1, klogk for BROADCASTing in region r, where k is the complexity of all faces (#nodes in region r?) that intersect with r).

The above time can be improved at the cost of having multiple copies of the packet in the network.

6.4 Experimental results

Random connected unit graphs
Stationary graphs

Measure:
success rate, taken as an average over all pairs of nodes
path stretch, again an average over all pairs of nodes

The algorithms don't seem to perform very well! Then they observe that it might be a good idea to use greedy routing or something else for the normal case, and just use their face routing ideas for when the normal algorithm gets stuck.

Two ways of combining greedy and face routing:
Figure 7, try the normal algorithm, and if it fails, change over to face routing.
Figure 8, try the normal algorithm, and if it fails, change over temporarily to face routing, until face routing produces something that the normal algorithm would consider an improvement. Then go back to the normal algorithm.

6.5 Conclusions

Open problems:
Extend the work to dynamic networks.
Develop good math models and simulation methods to evaluate these dynamic versions.