Geo-Routing
Chapter 3

Rating

- Area maturity
  - First steps
  - Text book

- Practical importance
  - No apps
  - Mission critical

- Theoretical importance
  - Not really
  - Must have

Overview

- Classic routing overview
- Geo-routing
- Greedy geo-routing

- Euclidean and planar graphs
- Face Routing
- Greedy and Face Routing
- 3D Geo-Routing
- Geometric Routing without Geometry

Classic Routing 1: Flooding

- What is Routing?
  - “Routing is the act of moving information across a network from a source to a destination.” (CISCO)

- The simplest form of routing is “flooding”: a source \( s \) sends the message to all its neighbors; when a node other than destination \( t \) receives the message the first time it re-sends it to all its neighbors.
  + simple (sequence numbers)
    - a node might see the same message more than once. (How often?)
    - what if the network is huge but the target \( t \) sits just next to the source \( s \)?
  - We need a smarter routing algorithm
Classic Routing 2: Link-State Routing Protocols

- Link-state routing protocols are a preferred iBGP method (within an autonomous system) in the Internet.
- Idea: periodic notification of all nodes about the complete graph.
- Routers then forward a message along (for example) the shortest path in the graph.
- Message follows shortest path.
  - Every node needs to store the whole graph, even links that are not on any path.
  - Every node needs to send and receive messages that describe the whole graph regularly.

Classic Routing 3: Distance Vector Routing Protocols

- The predominant method for wired networks.
- Idea: each node stores a routing table that has an entry to each destination (destination, distance, neighbor).
- If a router notices a change in its neighborhood or receives an update message from a neighbor, it updates its routing table accordingly and sends an update to all its neighbors.
- Message follows shortest path.
  - Only send updates when topology changes.
  - Most topology changes are irrelevant for a given source/destination pair.
  - Every node needs to store a big table.
  - Count-to-infinity problem.

Discussion of Classic Routing Protocols

- **Proactive Routing Protocols**
  - Both link-state and distance vector are “proactive,” that is, routes are established and updated even if they are never needed.
  - If there is almost no mobility, proactive algorithms are superior because they never have to exchange information and find optimal routes easily.

- **Reactive Routing Protocols**
  - Flooding is “reactive,” but does not scale.
  - If mobility is high and data transmission rare, reactive algorithms are superior; in the extreme case of almost no data and very much mobility the simple flooding protocol might be a good choice.

There is no “optimal” routing protocol; the choice of the routing protocol depends on the circumstances. Of particular importance is the mobility/data ratio.

Routing in Ad-Hoc Networks

- **Reliability**
  - Nodes in an ad-hoc network are not 100% reliable.
  - Algorithms need to find alternate routes when nodes are failing.

- **Mobile Ad-Hoc Network (MANET)**
  - It is often assumed that the nodes are mobile (“Moteran”).

- 10 Tricks → $2^{16}$ routing algorithms.
- In reality there are almost that many proposals!

- Q: How good are these routing algorithms? Any hard results?
  - A: Almost none! Method-of-choice is simulation…
  - “If you simulate three times, you get three different results.”
Geometric (geographic, directional, position-based) routing

- ...even with all the tricks there will be flooding every now and then.

- In this chapter we will assume that the nodes are location aware (they have GPS, Galileo, or an ad-hoc way to figure out their coordinates), and that we know where the destination is.

- Then we simply route towards the destination.

Geometric routing

- Problem: What if there is no path in the right direction?

- We need a guaranteed way to reach a destination even in the case when there is no directional path...

- Hack: as in flooding nodes keep track of the messages they have already seen, and then they backtrack* from there

*backtracking? Does this mean that we need a stack?!?

Geo-Routing: Strictly Local

Greedy Geo-Routing?
Greedy Geo-Routing?

What is Geographic Routing?

- A.k.a. geometric, location-based, position-based, etc.

- Each node knows its own position and position of neighbors
- Source knows the position of the destination
- No routing tables stored in nodes!

Geographic routing makes sense
- Own position: GPS/Galileo, local positioning algorithms
- Destination: Geocasting, location services, source routing++
- Learn about ad-hoc routing in general

Greedy routing

- Greedy routing looks promising.
- Maybe there is a way to choose the next neighbor and a particular graph where we always reach the destination?

Examples why greedy algorithms fail

- We greedily route to the neighbor which is closest to the destination: But both neighbors of x are not closer to destination D

- Also the best angle approach might fail, even in a triangulation: if, in the example on the right, you always follow the edge with the narrowest angle to destination t, you will forward on a loop $v_0, w_0, v_1, w_1, ..., v_3, w_3, v_0, ...$
Euclidean and Planar Graphs

- Euclidean: Points in the plane, with coordinates
- Planar: can be drawn without “edge crossings” in a plane
- Euclidean planar graphs (planar embeddings) simplify geometric routing.

Unit disk graph

- We are given a set $V$ of nodes in the plane (points with coordinates).
- The unit disk graph $UDG(V)$ is defined as an undirected graph (with $E$ being a set of undirected edges). There is an edge between two nodes $u,v$ iff the Euclidean distance between $u$ and $v$ is at most 1.
- Think of the unit distance as the maximum transmission range.
- We assume that the unit disk graph $UDG$ is connected (that is, there is a path between each pair of nodes)
- The unit disk graph has many edges.
- Can we drop some edges in the $UDG$ to reduce complexity?

Planar graphs

- Definition: A planar graph is a graph that can be drawn in the plane such that its edges only intersect at their common end-vertices.
- Kuratowski’s Theorem: A graph is planar iff it contains no subgraph that is edge contractible to $K_5$ or $K_{3,3}$.
- Euler’s Polyhedron Formula: A connected planar graph with $n$ nodes, $m$ edges, and $f$ faces has $n - m + f = 2$.
- Right: Example with 9 vertices, 14 edges, and 7 faces (the yellow “outside” face is called the infinite face)
- Theorem: A simple planar graph with $n$ nodes has at most $3n-6$ edges, for $n \geq 3$.

Gabriel Graph

- Let disk($u,v$) be a disk with diameter $(u,v)$ that is determined by the two points $u,v$.
- The Gabriel Graph $GG(V)$ is defined as an undirected graph (with $E$ being a set of undirected edges). There is an edge between two nodes $u,v$ iff the disk($u,v$) including boundary contains no other points.
- As we will see the Gabriel Graph has interesting properties.
Delaunay Triangulation

- Let \( \text{disk}(u,v,w) \) be a disk defined by the three points \( u, v, w \).
- The Delaunay Triangulation (Graph) \( \text{DT}(V) \) is defined as an undirected graph (with \( E \) being a set of undirected edges). There is a triangle of edges between three nodes \( u, v, w \) iff the \( \text{disk}(u,v,w) \) contains no other points.
- The Delaunay Triangulation is the dual of the Voronoi diagram, and widely used in various CS areas; the DT is planar; the distance of a path \((s, \ldots, t)\) on the DT is within a constant factor of the s-t distance.

Other planar graphs

- Relative Neighborhood Graph \( \text{RNG}(V) \)
  - An edge \( e = (u,v) \) is in the RNG(V) iff there is no node \( w \) with \( (u,w) < (u,v) \) and \( (v,w) < (u,v) \).
- Minimum Spanning Tree \( \text{MST}(V) \)
  - A subset of \( E \) of \( G \) of minimum weight which forms a tree on \( V \).

Routing on Delaunay Triangulation?

- Let \( d \) be the Euclidean distance of source \( s \) and destination \( t \)
- Let \( c \) be the sum of the distances of the links of the shortest path in the Delaunay Triangulation
  - It was shown that \( c = \Theta(d) \)
- Three problems:
  1) How do we find this best route in the DT? With flooding?!?
  2) How do we find the DT at all in a distributed fashion?
  3) Worse: The DT contains edges that are not in the UDG, that is, nodes that cannot receive each other are “neighbors” in the DT
Breakthrough idea: route on faces

- Remember the faces...

- Idea: Route along the boundaries of the faces that lie on the source–destination line.

Face Routing

0. Let $f$ be the face incident to the source $s$, intersected by $(s,t)$.

1. Explore the boundary of $f$, remember the point $p$ where the boundary intersects with $(s,t)$ which is nearest to $t$; after traversing the whole boundary, go back to $p$, switch the face, and repeat 1 until you hit destination $t$.

Face Routing Works on Any Graph

Face Routing Properties

- All necessary information is stored in the message
  - Source and destination positions
  - Point of transition to next face

- Completely local:
  - Knowledge about direct neighbors’ positions sufficient
  - Faces are implicit

- Planarity of graph is computed locally (not an assumption)
  - Computation for instance with Gabriel Graph
Face routing is correct

- Theorem: Face routing terminates on any simple planar graph in $O(n)$ steps, where $n$ is the number of nodes in the network

- Proof: A simple planar graph has at most $3n - 6$ edges. You leave each face at the point that is closest to the destination, that is, you never visit a face twice, because you can order the faces that intersect the source—destination line on the exit point. Each edge is in at most 2 faces. Therefore each edge is visited at most 4 times. The algorithm terminates in $O(n)$ steps.

Is there something better than Face Routing?

- How to improve face routing? A proposal called “Face Routing 2”

- Idea: Don’t search a whole face for the best exit point, but take the first (better) exit point you find. Then you don’t have to traverse huge faces that point away from the destination.

- Efficiency: Seems to be practically more efficient than face routing. But the theoretical worst case is worse – $O(n^2)$.

- Problem: if source and destination are very close, we don’t want to route through all nodes of the network. Instead we want a routing algorithm where the cost is a function of the cost of the best route in the unit disk graph (and independent of the number of nodes).

Face Routing

- Theorem: Face Routing reaches destination in $O(n)$ steps

- But: Can be very bad compared to the optimal route

Bounding Searchable Area

- Theorem: Face Routing reaches destination in $O(n^2)$ steps

- But: Can be very bad compared to the optimal route
Adaptive Face Routing (AFR)

- Idea: Use face routing together with ad hoc routing trick 1!!
- That is, don't route beyond some radius \( r \) by branching the planar graph within an ellipse of exponentially growing size.

AFR Example Continued

- We grow the ellipse and find a path

AFR Pseudo-Code

0. Calculate \( G = GG(V) \cap UDG(V) \)
   - Set \( c \) to be twice the Euclidean source—destination distance.

1. Nodes \( w \in W \) are nodes where the path \( s-w-t \) is larger than \( c \). Do face routing on the graph \( G \), but without visiting nodes in \( W \). (This is like pruning the graph \( G \) with an ellipse.) You either reach the destination, or you are stuck at a face (that is, you do not find a better exit point.)

2. If step 1 did not succeed, double \( c \) and go back to step 1.
   - Note: All the steps can be done completely locally, and the nodes need no local storage.

The \( \Omega(1) \) Model

- We simplify the model by assuming that nodes are sufficiently far apart; that is, there is a constant \( d_0 \) such that all pairs of nodes have at least distance \( d_0 \). We call this the \( \Omega(1) \) model.
- This simplification is natural because nodes with transmission range 1 (the unit disk graph) will usually not "sit right on top of each other".
- Lemma: In the \( \Omega(1) \) model, all natural cost models (such as the Euclidean distance, the energy metric, the link distance, or hybrids of these) are equal up to a constant factor.
- Remark: The properties we use from the \( \Omega(1) \) model can also be established with a backbone graph construction.
Analysis of AFR in the $\Omega(1)$ model

- Lemma 1: In an ellipse of size $c$ there are at most $O(c^2)$ nodes.
- Lemma 2: In an ellipse of size $c$, face routing terminates in $O(c^2)$ steps, either by finding the destination, or by not finding a new face.
- Lemma 3: Let the optimal source—destination route in the UDG have cost $c^*$. Then this route $c^*$ must be in any ellipse of size $c^*$ or larger.
- Theorem: AFR terminates with cost $O(c^*^2)$.
- Proof: Summing up all the costs until we have the right ellipse size is bounded by the size of the cost of the right ellipse size.

Lower Bound

- The network on the right constructs a lower bound.
- The destination is the center of the circle, the source any node on the ring.
- Finding the right chain costs $\Omega(c^*)$, even for randomized algorithms.
- Theorem: AFR is asymptotically optimal.

Non-geometric routing algorithms

- In the $\Omega(1)$ model, a standard flooding algorithm enhanced with trick 1 will (for the same reasons) also cost $O(c^*^2)$.
- However, such a flooding algorithm needs $O(1)$ extra storage at each node (a node needs to know whether it has already forwarded a message).
- Therefore, there is a trade-off between $O(1)$ storage at each node or that nodes are location aware, and also location aware about the destination. This is intriguing.

GOAFR – Greedy Other Adaptive Face Routing

- Back to geometric routing...
- AFR Algorithm is not very efficient (especially in dense graphs)
- Combine Greedy and (Other Adaptive) Face Routing
  - Route greedily as long as possible
  - Circumvent "dead ends" by use of face routing
  - Then route greedily again

Other AFR: In each face proceed to node closest to destination.
GOAFR+ improvements:
- Early fallback to greedy routing
  - (Circle centered at destination instead of ellipse)

Early Fallback to Greedy Routing?
- We could fall back to greedy routing as soon as we are closer to \( t \) than the local minimum
- But:

```
\Omega(c^2)\text{ nodes} \quad \Omega(c^*)\text{ local minima}
```

- "Maze" with \( \Omega(c^2) \) edges is traversed \( \Omega(c^*) \) times → \( \Omega(c^3) \) steps

GOAFR – Greedy Other Adaptive Face Routing
- Early fallback to greedy routing:
  - Use counters \( p \) and \( q \). Let \( u \) be the node where the exploration of the current face \( F \) started
    - \( p \) counts the nodes closer to \( t \) than \( u \)
    - \( q \) counts the nodes not closer to \( t \) than \( u \)
  - Fall back to greedy routing as soon as \( p > \sigma \cdot q \) (constant \( \sigma > 0 \))

Theorem: GOAFR is still asymptotically worst-case optimal… …and it is efficient in practice, in the average-case.

- What does "practice" mean?
  - Usually nodes placed uniformly at random

Average Case
- Not interesting when graph not dense enough
- Not interesting when graph is too dense
- Critical density range ("percolation")
  - Shortest path is significantly longer than Euclidean distance

```
too sparse \quad \text{critical density} \quad \text{too dense}
```
Critical Density: Shortest Path vs. Euclidean Distance

- Shortest path is significantly longer than Euclidean distance

- Critical density range mandatory for the simulation of any routing algorithm (not only geographic)

Randomly Generated Graphs: Critical Density Range

Simulation on Randomly Generated Graphs

A Word on Performance

- What does a performance of 3.3 in the critical density range mean?

- If an optimal path (found by Dijkstra) has cost $c$, then $\text{GOAFR}^+$ finds the destination in $3.3 \cdot c$ steps.

- It does not mean that the path found is 3.3 times as long as the optimal path! The path found can be much smaller…

- Remarks about cost metrics
  - In this lecture "cost" $c = c$ hops
  - There are other results, for instance on distance/energy/hybrid metrics
  - In particular: With energy metric there is no competitive geometric routing algorithm
Energy Metric Lower Bound

Example graph: k "stalks", of which only one leads to t
- any deterministic (randomized) geometric routing algorithm $A$ has to visit all $k$ (at least $k/2$) "stalks"
- optimal path has constant cost $c^*$ (covering a constant distance at almost no cost)

$$\lim_{k \to \infty} \frac{c(A)}{c^*} = \infty$$

$\rightarrow$ With energy metric there is no competitive geometric routing algorithm

GOAFR: Summary

3D Geo-Routing

- The world is not flat. We can certainly envision networks in 3D, e.g. in a large office building. Can we geo-route in three dimensions? Are the same techniques possible?
- Certainly, if the node density is high enough (and the node distribution is kind to us), we can simply use greedy routing. But what about those local minima?!
- Is there something like a face in 3D?
- How would you do 3D routing?
- The picture on the right is the 3D equivalent of the 2D lower bound, proving that we need at least $OPT^3$ steps.

Deterministic Routing in 3-Dimensional Networks

We will prove that

There is no deterministic $k$-local routing algorithm for 3D UDGs

- Deterministic: Whenever a node $n$ receives a message from node $m$, $n$ determines the next hop as a function $f(n,m,s,t,N(n))$, where $s$ and $t$ are the source and the target nodes and $N(n)$ the neighborhood of $n$.
- $k$-local: A node only knows its $k$-hop neighborhood

- Proof Outline:
  (A) We show that an arbitrary graph $G$ can be translated to a 3D UDG $G'$
  (B) Assume for contradiction that there is a $k$-local algorithm $A_k$ for 3D UDGs,
  (C) We show that there must also be a 1-local algorithm $A_1$ for 3D UDGs
  (D) The translation from $G$ to $G'$ is strictly local, therefore, we could simulate $A_1$ on $G$ and obtain a 1-local routing for arbitrary graphs
  (E) We show that there is no such algorithm, disproving the existence of $A_k$. 
Transforming a general graph to a 3D UDG (1/2)

- Main idea: Build the 3D UDG similar to an electronic circuit on three layers, and add chains of virtual nodes (the conductors)

Transforming a general graph to a 3D UDG (2/2)

- Virtual nodes on the middle layer establish the connections
- The resulting graph is a 3D UDG

1-local Routing for 3D UDGs

- Assume that there is a $k$-local routing algorithm $A_k$ for 3D UDG
- Adapt the transformation s.t. the connecting lines contain at least $2k$ virtual nodes
- As a result, $A_k$ cannot see more than 1 hop of the original graph
- The stretching of the paths introduces ‘dummy’ information of no use, but the algorithm $A_k$ still has to work
- Therefore, there must also be a 1-local algorithm $A_1$ for 3D UDG

1-local Routing for Arbitrary Graphs

- The transformation to the 3D UDG $G'$ can be determined strictly locally from any graph $G$
- The nodes of any graph $G$ can simulate $A_1$ by simulating $G'$
- Therefore, $A_1$ can be used to build a 1-local routing algorithm for arbitrary graphs
1-local Routing for Arbitrary Graphs is \textit{impossible} (1/2)

- A \textit{deterministic} routing algorithm can be described as a function $f(n, m, s, t, N(n))$, which returns the next hop.
- $n$: current node, $m$: previous node, $s$: source, $t$: target, $N(n)$: neighborhood of $n$.
- Node $n$ has no means to determine \textit{locally} which of its neighbors has a connection to $t \rightarrow n$ must try all of them before returning to $m$.
- Even the position of $t$ or $s$ can't help.
- The function $f$ must be a cycle over the $i+1$ neighbors.
- If not, we miss some neighbors of $n$, which may connect to $t$.

1-local Routing for Arbitrary Graphs is \textit{impossible} (2/2)

- Node 2 and 7 have to decide on one forwarding function.
- There are 4 combinations possible. For all of them, forwarding fails either in the left or the right network.
- Conclusion 1: 1-local routing algorithms do not exist.
- Conclusion 2: There is no $k$-local routing algorithm for 3D UDG.
- Conclusion 3: There is no $k$-local routing algorithm for 3D graphs.

Routing with and without position information

- \textbf{Without} position information:
  - Flooding \textbf{does not scale}.
  - Distance Vector Routing \textbf{does not scale}.
  - Source Routing
    - increased per-packet overhead.
    - no theoretical results, only simulation.

- \textbf{With} position information:
  - Greedy Routing \textbf{may fail}: message may get stuck in a "dead end".
  - Geometric Routing
    - It is assumed that each node \textit{knows its position}.

Obtaining Position Information

- Attach GPS to each sensor node.
  - Often undesirable or impossible.
  - GPS receivers clumsy, expensive, and energy-inefficient.

- Equip only a few designated nodes with a GPS.
  - \textbf{Anchor} (landmark) nodes have GPS.
  - Non-anchors derive their position through communication (e.g., count number of hops to different anchors).

\textbf{Anchor density determines quality of solution.}
What about no GPS at all?

- In absence of GPS-equipped anchors...
- ...nodes are clueless about real coordinates.
- For many applications, real coordinates are not necessary
- Virtual coordinates are sufficient

‡ Given the connectivity information for each node and knowing the underlying graph is a UDG find virtual coordinates in the plane such that all connectivity requirements are fulfilled, i.e. find a realization (embedding) of a UDG:
  - each edge has length at most 1
  - between non-neighbored nodes the distance is more than 1

‡ Finding a realization of a UDG from connectivity information only is NP-hard...
  - [Breu, Kirkpatrick, Comp.Geom.Theory 1998]
‡ ...and also hard to approximate
  - [Kuhn, Moscibroda, Wattenhofer, DIALM 2004]

What are „good“ virtual coordinates?

‡ For many applications, like routing, finding a realization of a UDG is not mandatory
‡ Virtual coordinates merely as infrastructure for geometric routing

⇒ Pseudo geometric coordinates:
  - Select some nodes as anchors: \( a_1, a_2, \ldots, a_k \)
  - Coordinate of each node \( u \) is its hop-distance to all anchors:
    \( (d(u,a_1), d(u,a_2), \ldots, d(u,a_k)) \)

‡ Requirements:
  - each node uniquely identified: Naming Problem
  - routing based on (pseudo geometric) coordinates possible: Routing Problem

Geometric Routing without Geometry

‡ For many applications, like routing, finding a realization of a UDG is not mandatory
‡ Virtual coordinates merely as infrastructure for geometric routing

⇒ Pseudo-geometric routing in the grid: Naming

Lemma: The naming problem in the grid can be solved with two anchors.

Pseudo-geometric routing in the grid: Routing

Rule: pass message to neighbor which is closest to destination

Lemma: The routing problem in the grid can be solved with two anchors.

Problem: UDG is usually not a grid

- Recursive construction of a unit dist tree (UDT) which needs $\Omega(n)$ anchors

Pseudo-geometric routing in the UDT: Naming

- Leaf-siblings can only be distinguished if one of them is an anchor:

Lemma: in a unit disk tree with $n$ nodes there are up to $O(n)$ leaf-siblings. That is, we need to $O(n)$ anchors.

Pseudo-geometric routing in the ad hoc networks

- Naming and routing in grid quite good, in previous UDT example very bad
- Real-world ad hoc networks are very probable neither perfect grids nor naughty unit disk trees

Truth is somewhere in between...
Summary of Results

- If position information is available geo-routing is a feasible option.
- Face routing guarantees to deliver the message.
- By restricting the search area the efficiency is $\text{OPT}^2$.
- Because of a lower bound this is asymptotically optimal.
- Combining greedy and face gives efficient algorithm.
- 3D geo-routing is impossible.
- Even if there is no position information, some ideas might be helpful.

- Geo-routing is probably the only class of routing that is well understood.
- There are many adjacent areas: topology control, location services, routing in general, etc.

Open problem

- One of the most-understood topics. In that sense it is hard to come up with a decent open problem. Let's try something wishy-washy.

- For a 2D UDG the efficiency of geo-routing can be quadratic to an optimal algorithm (with routing tables). However, the worst-case example is quite special. Open problem: How much information does one need to store in the network to guarantee only constant overhead?
  - Variant: Instead of UDG some more realistic model
  - How can one maintain this information if the network is dynamic?