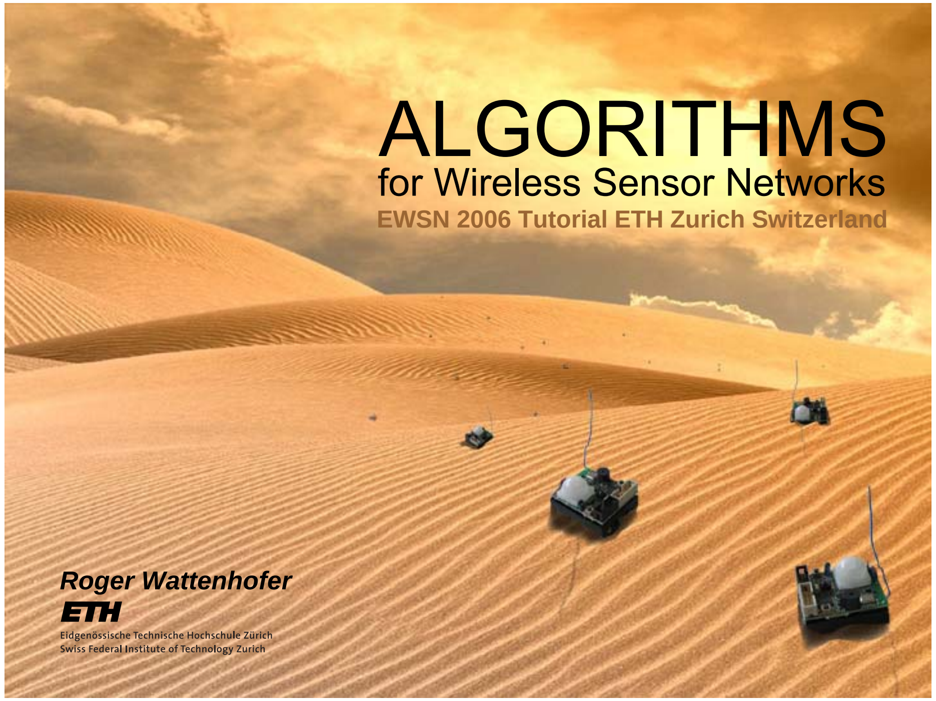


Chapter 0
INTRODUCTION

Distributed Computing Group

EWSN 2006

Introductory comments

- Way too many slides...
  - But don't worry, we won't do all of them
- Heterogeneous audience
  - Some students, some industry folks, some famous professors, …
  - I assume everybody knows 101 of sensor networking
  - Instead of a real introduction, I will show some "opinion" slides
- This tutorial has a quite narrow definition of the term "algorithm"
- An algorithm is an algorithm only if it features an analytical proof of efficiency.
- If performance is proved by simulation only, we call it a heuristic.
- We look a distributed algorithms mostly.

My Own Private View on Networking Research

Class	Analysis	Communication model	Node distribution	Other drawbacks	Popularity
Imple-mentation	Testbed	Reality	Reality(?)	"Too specific"	5%
Heuristic	Simulation	UDG to SINR	Random, and more	Many…! (no benchmarks)	80%
Scaling law	Theorem/ proof	SINR, and more	Random	Existential (no protocols)	10%
Algorithm	Theorem/ proof	UDG, and more	Any (worst-case)	Worst-case unusual	5%

Algorithm Classes

Global Algorithm → Distributed Algorithm → Local / Localized / Unstructured

- For some problems we don't even understand the non-distributed case
- "Reiceive msg X → Transmit msg Y"
- Every algo can be made distributed

Local

+ Node can only communicate with neighbors k times.
+ Strict time bounds
– Often synchronous

Localized

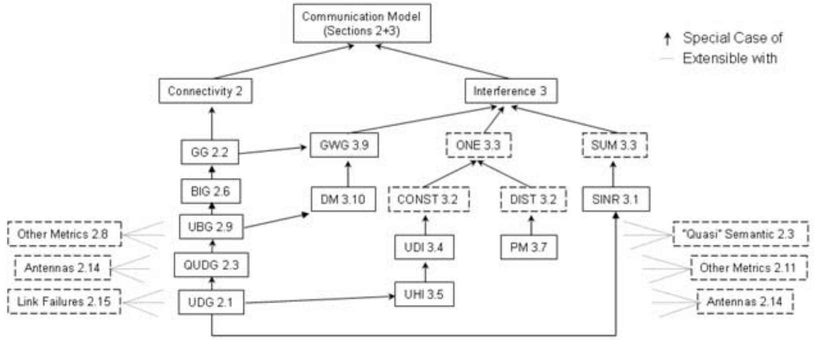
+ Often simple
– Nodes can wait for neighbor actions
– Often linear chain of causality

Unstructured

+ Implement MAC layer yourself; you control everything
– Often complicated
– Argumentation overhead

Some algorithmic communication models

- Some of them we will see in this lecture, most of them not…



What has been studied?

Link Layer

- MAC Layer and Coloring
- Topology and Power Control
- Interference and Signal-to-Noise-Ratio
- Clustering (Dominating Sets, etc.)
- Deployment (Unstructured Radio Networks)

Network Layer

- New Routing Paradigms (e.g. Link Reversal)
- Geo-Routing
- Broadcast and Multicast
- Data Gathering

Services

- Location Services and Positioning
- Time Synchronization

Theory/Models

- Models and Mobility
- Lower Bounds for Message Passing
- Selfish Agents, Economic Aspects, (Security)

What has received most attention?

- MAC Layer and Coloring
- Topology and Power Control
- Interference and Signal-to-Noise-Ratio
- Clustering (Dominating Sets, etc.)
- Deployment (Unstructured Radio Networks)
- New Routing Paradigms (e.g. Link Reversal)
- Geo-Routing
- Broadcast and Multicast ("energy-efficient BC")
- Data Gathering
- Location Services and Positioning
- Time Synchronization
- Models and Mobility
- Lower Bounds for Message Passing
- Selfish Agents, Economic Aspects, Security

What is really important?!? (Opinion…)

Philosophy

- Understand algorithmic fundamentals of sensor networks.
  - See some algorithms with implementation appeal
- Find models that capture reality
  - No random distribution
  - No random mobility
- Show a few examples
  - Mix between well-studied and "important" topics
- More material
  - Reading list on www.dcg.ethz.ch

Chapter 1
GEOMETRIC ROUTING

Distributed Computing Group

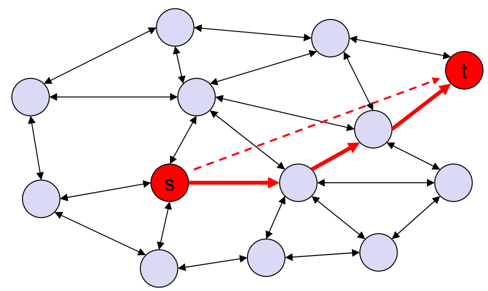
EWSN 2006

Overview – Geometric Routing

- Geometric routing
- Greedy geometric routing
- Euclidean and planar graphs
- Unit disk graph
- Gabriel graph and other planar graphs
- Face Routing
- Greedy and Face Routing
- Geometric Routing without Geometry

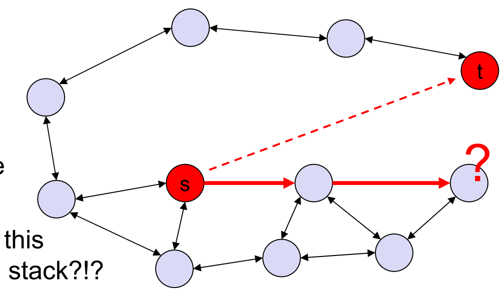
Geometric (geographic, directional, position-based) routing

- …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



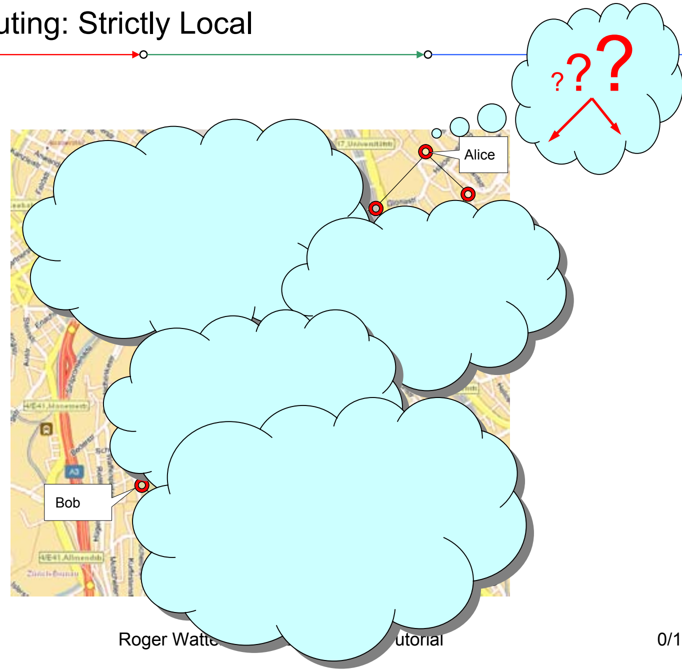
Geometric routing

- 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

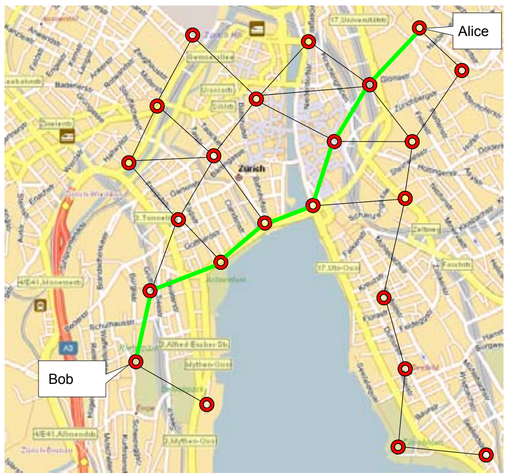


*backtracking? Does this mean that we need a stack?!?

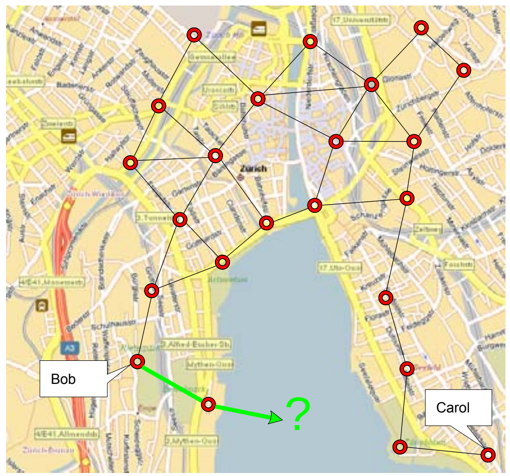
Geo-Routing: Strictly Local



Greedy Geo-Routing?



Greedy Geo-Routing?



What is Geographic Routing?

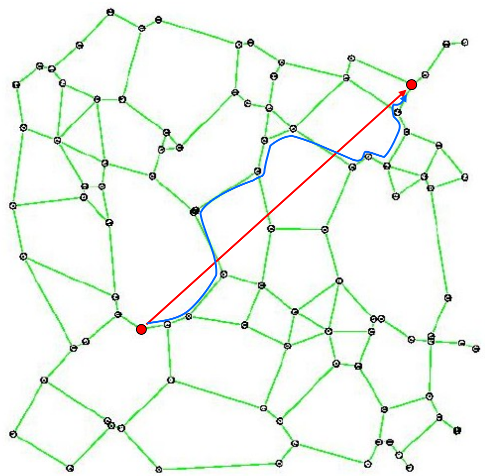
- A.k.a. geometric, location-based, position-based, etc.

- 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

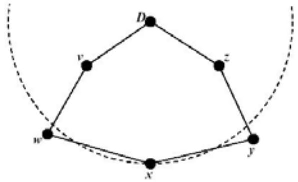
Greedy routing

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

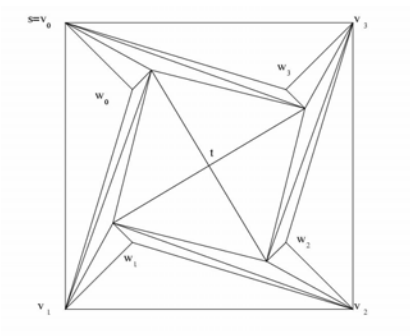


Examples why greedy algorithms fail

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

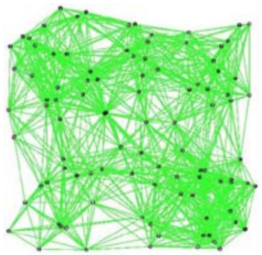


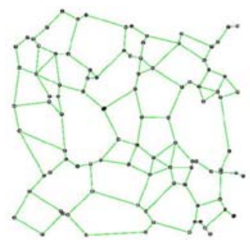
- Also the best angle approach might fail, even in a triangulation: if, in the example on the right, you always follow the edge with the narrowest angle to destination t, you will forward on a loop $v_0, w_0, v_1, w_1, \ldots, v_3, w_3, v_0, \ldots$



Euclidean and Planar Graphs

- Euclidean: Points in the plane, with coordinates
- Planar: can be drawn without "edge crossings" in a plane



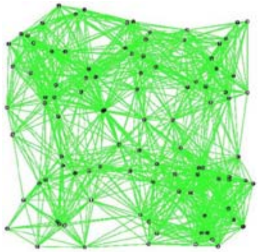


- Euclidean planar graphs (planar embeddings) simplify geometric routing.

Unit disk graph

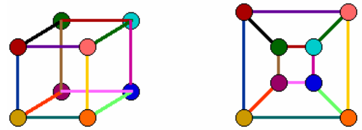
- We are given a set *V* of nodes in the plane (points with coordinates).
- The unit disk graph *UDG*(*V*) is defined as an undirected graph (with *E* being a set of undirected edges). There is an edge between two nodes *u*,*v* iff the Euclidean distance between *u* and *v* is at most 1.
- Think of the 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 reduced complexity and interference?

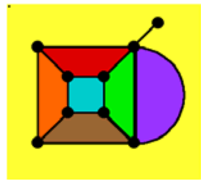


Planar graphs

- 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.

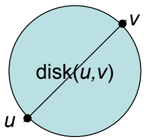


- Kuratowski's Theorem: A graph is planar iff it contains no subgraph that is edge contractible to K_5 or $K_{3,3}$.
- Euler's Polyhedron Formula: A connected planar graph with *n* nodes, *m* edges, and *f* faces has $n - m + f = 2$.
- Right: Example with 9 vertices,14 edges, and 7 faces (the yellow "outside" face is called the infinite face)
- Theorem: A simple planar graph with n nodes has at most 3n–6 edges, for n≥3.

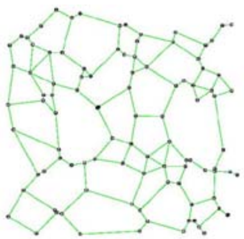


Gabriel Graph

- 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.

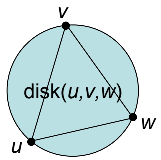


- As we will see the Gabriel Graph has interesting properties.

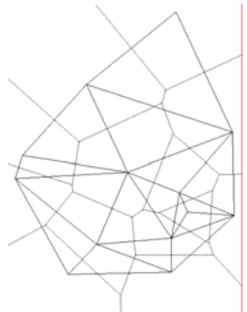


Delaunay Triangulation

- Let disk(*u,v,w*) be a disk defined by the three points *u*,*v*,*w*.
- The Delaunay Triangulation (Graph) 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 disk(*u*,*v*,*w*) contains no other points.

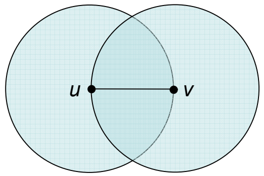


- The Delaunay Triangulation is the dual of the Voronoi diagram, and widely used in various CS areas; the DT is planar; the distance of a path (s,…,t) on the DT is within a constant factor of the s-t distance.

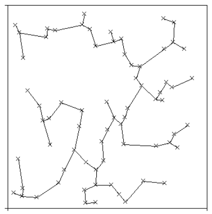


Other planar graphs

- Relative Neighborhood Graph 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 MST(V)
- A subset of *E* of *G* of minimum weight which forms a tree on *V*.

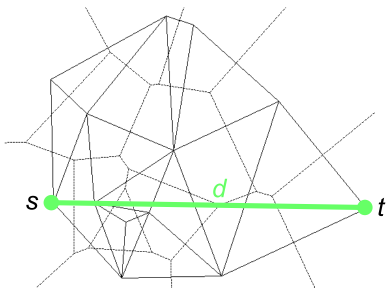


Properties of planar graphs

- Theorem 1:
  $\mathrm{MST}(V) \subseteq \mathrm{RNG}(V) \subseteq \mathrm{GG}(V) \subseteq \mathrm{DT}(V)$
- Corollary:
  Since the MST(V) is connected and the DT(V) is planar, all the planar graphs in Theorem 1 are connected and planar.
- Theorem 2:
  The Gabriel Graph contains the Minimum Energy Path (for any path loss exponent $\alpha \geq 2$)
- Corollary:
  GG(V) $\cap$ UDG(V) contains the Minimum Energy Path in UDG(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* = Θ(*d*)

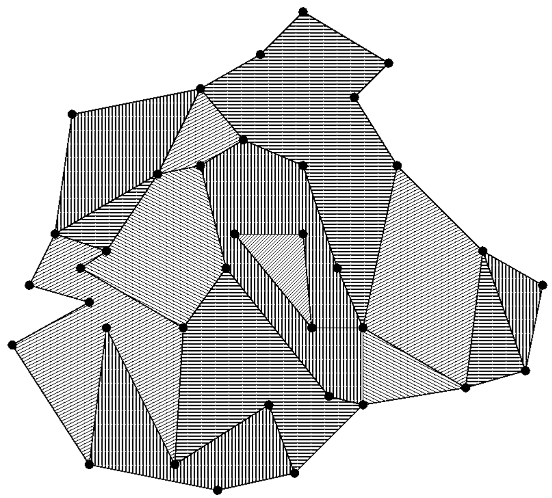


- Three problems:

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

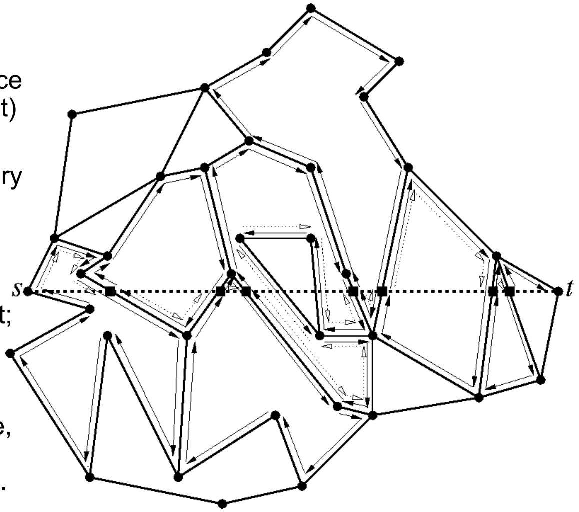
- Remember the faces…
- Idea:
  Route along the boundaries of the faces that lie on the source–destination line



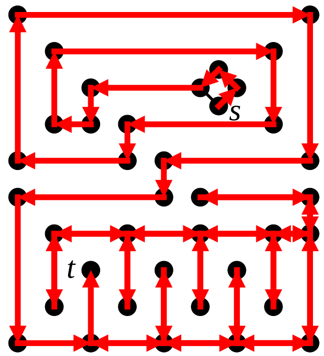
Face Routing

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.

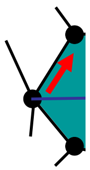


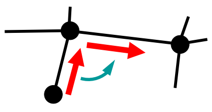
Face Routing Works on *Any* Graph



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





"Right Hand Rule"

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

Face routing is correct

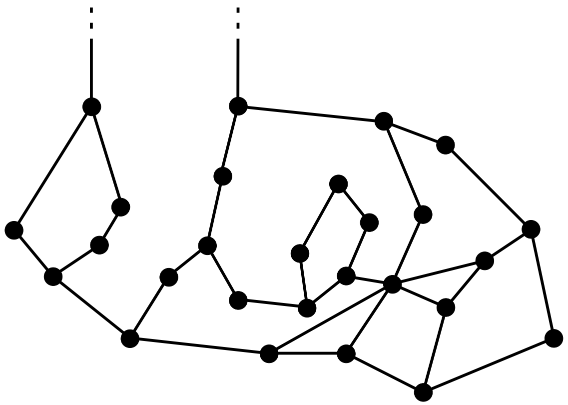
- Theorem: Face routing terminates on any simple planar graph in O(n) steps, where n is the number of nodes in the network
- Proof: A simple planar graph has at most 3n–6 edges. You leave each face at the point that is closest to the destination, that is, you never visit a face twice, because you can 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?

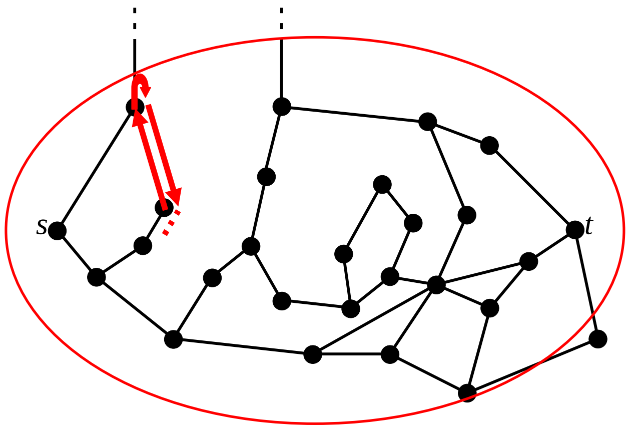
- How to improve face routing? A proposal called “Face Routing 2”
- Idea: Don’t search a whole face for the best exit point, but take the first (better) exit point you find. Then you don’t have to traverse huge faces that point away from the destination.
- Efficiency: Seems to be practically more efficient 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

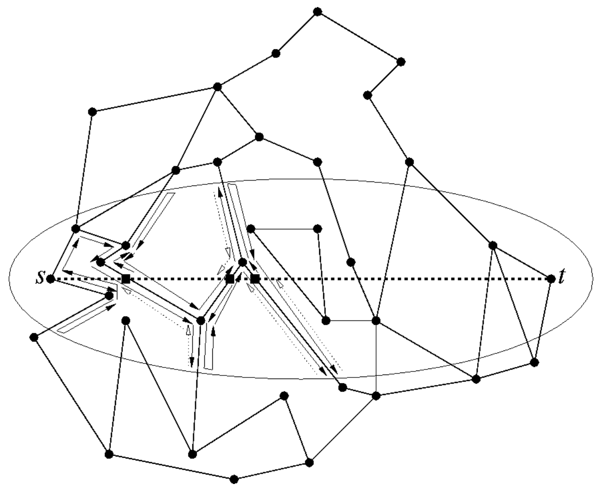


Bounding Searchable Area



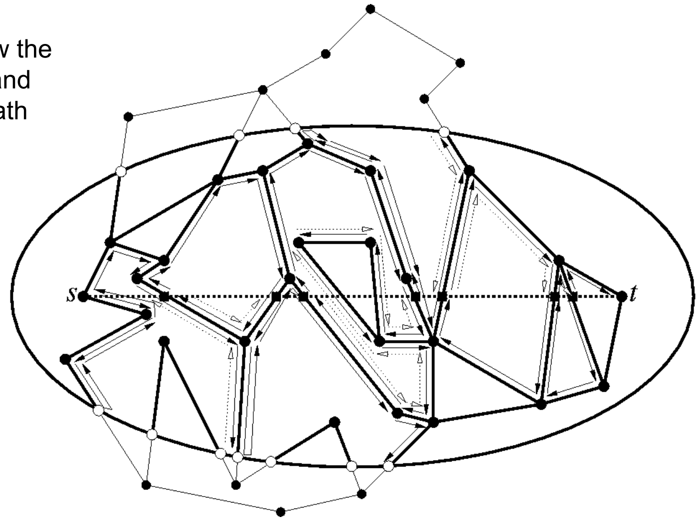
Adaptive Face Routing (AFR)

- 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) ∩ UDG(V)
   Set c to be twice the Euclidean source—destination distance.

1. Nodes w ∈ 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.

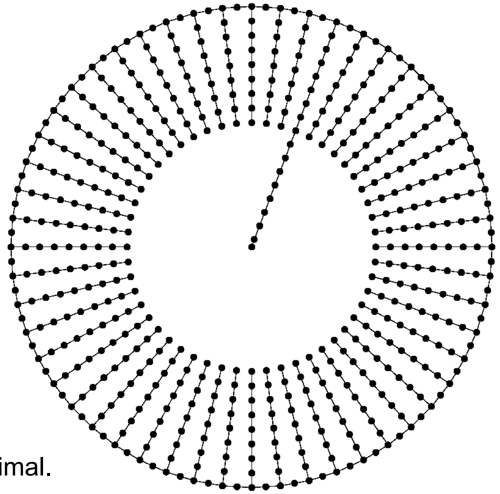
- Lemma 2: In an ellipse of size c, face routing terminates in $O(c^2)$ steps, either by finding the destination, or by not finding a new face.

- 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 $\Omega(c^{*2})$, even for randomized algorithms
- Theorem:
  AFR is asymptotically optimal.



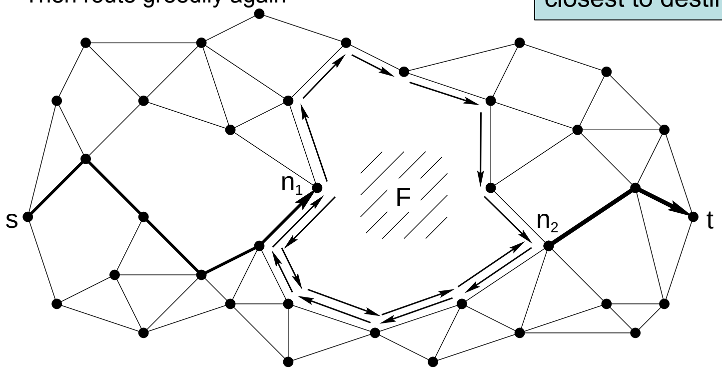
Non-geometric routing algorithms

- 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

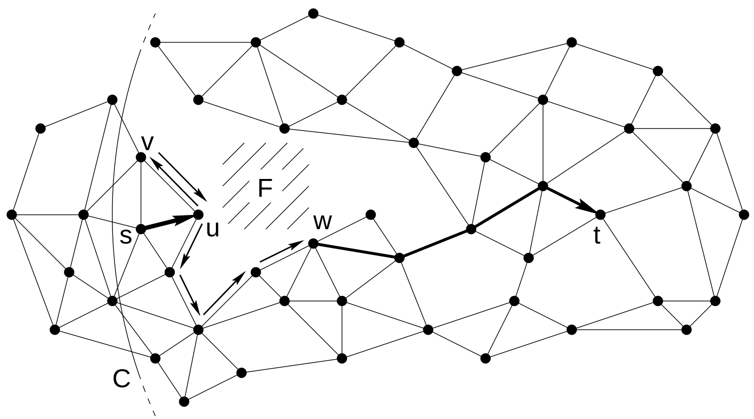
- Back to geometric routing…
- AFR Algorithm is not very efficient (especially in dense graphs)
- Combine Greedy and (Other Adaptive) Face Routing
  - Route greedily as long as possible
  - Circumvent "dead ends" by use of face routing
  - Then route greedily again

Other AFR: In each face proceed to node closest to destination



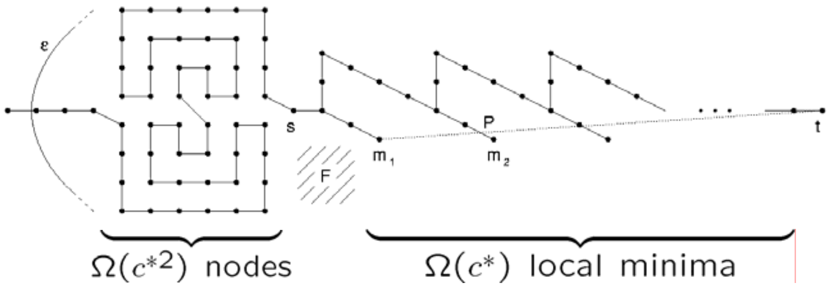
GOAFR+

- 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:



- "Maze" with $\Omega(c^{*2})$ edges is traversed $\Omega(c^*)$ times $\rightarrow \Omega(c^{*3})$ steps

GOAFR – Greedy Other Adaptive Face Routing

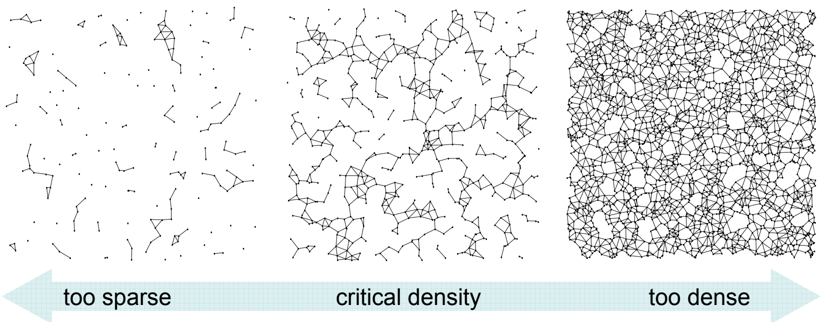
- Early fallback to greedy routing:
  - Use counters p and q. Let u be the node where the exploration of the current face F started
    - p counts the nodes closer to t than u
    - q counts the nodes *not* closer to t than u
  - Fall back to greedy routing as soon as $p > \sigma \cdot q$ (constant $\sigma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

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

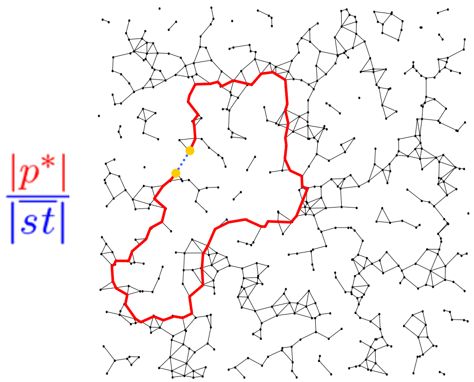
Average Case

- 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



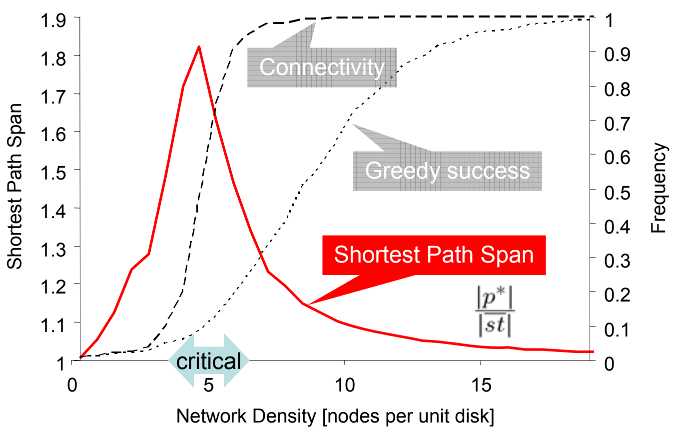
Critical Density: Shortest Path vs. Euclidean Distance

- Shortest path is significantly longer than Euclidean distance

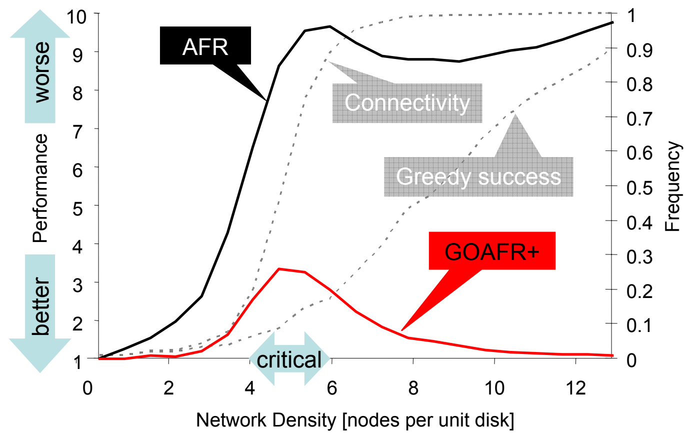


- Critical density range mandatory for the simulation of any routing algorithm (not only geographic)

Randomly Generated Graphs: Critical Density Range



Simulation on Randomly Generated Graphs



A Word on Performance

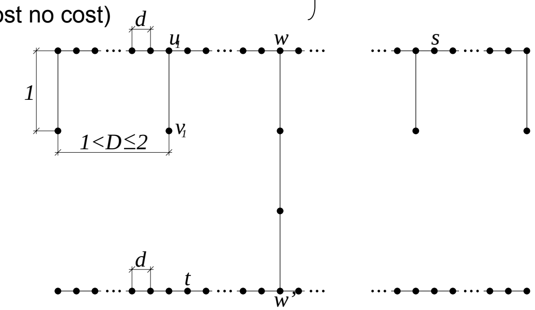
- What does a performance of 3.3 in the critical density range mean?
- If an optimal path (found by Dijkstra) has cost c, then GOAFR+ finds the destination in 3.3·c steps.
- It does *not* mean that the *path* found is 3.3 times as long as the optimal path! The path found can be much smaller…
- Remarks 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

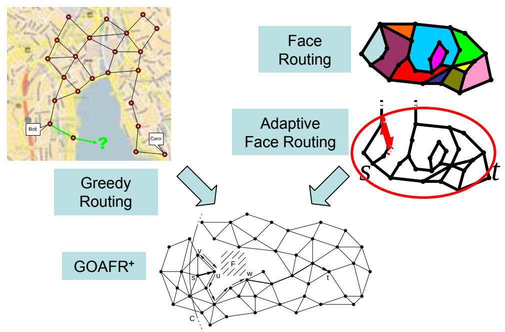
- any deterministic (randomized) geometric routing algorithm A has to visit all k (at least k/2) "stalks"
- optimal path has constant cost c* (covering a constant distance at almost no cost)

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



→ With energy metric there is no competitive geometric routing algorithm

GOAFR: Summary



Average-case efficiency | Worst-case optimality

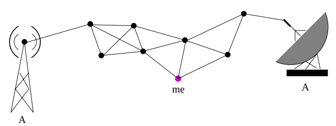
"Practice" | "Theory"

Routing with and without position information

- Without position information:
  - Flooding
    → does not scale
  - Distance Vector Routing
    → does not scale
  - Source Routing
    - increased per-packet overhead
    - no theoretical results, only simulation
- With position information:
  - Greedy Routing
    → may fail: message may get stuck in a "dead end"
  - Geometric Routing
    → It is assumed that each node knows its position

Obtaining Position Information

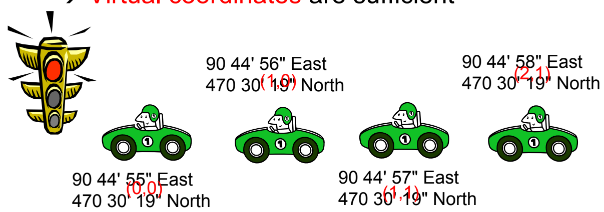
- Attach GPS to each sensor node
  - Often undesirable or impossible
  - GPS receivers clumsy, expensive, and energy-inefficient
- Equip only a few designated nodes with a GPS
  - Anchor (landmark) nodes have GPS
  - Non-anchors derive their position through communication (e.g., count number of hops to different anchors)

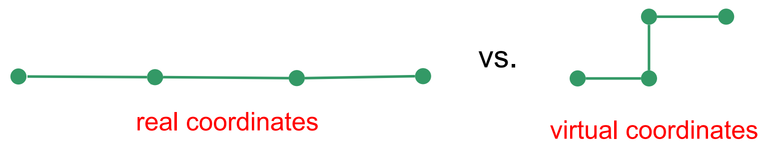


Anchor density determines quality of solution

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





What are „good“ virtual coordinates?

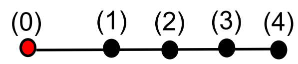
- Given the connectivity information for each node and knowing the underlying graph is a UDG find virtual coordinates in the plane such that all connectivity requirements are fulfilled, i.e. find a realization (embedding) of a UDG:
  - each edge has length at most 1
  - between non-neighbored nodes the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

Geometric Routing without Geometry

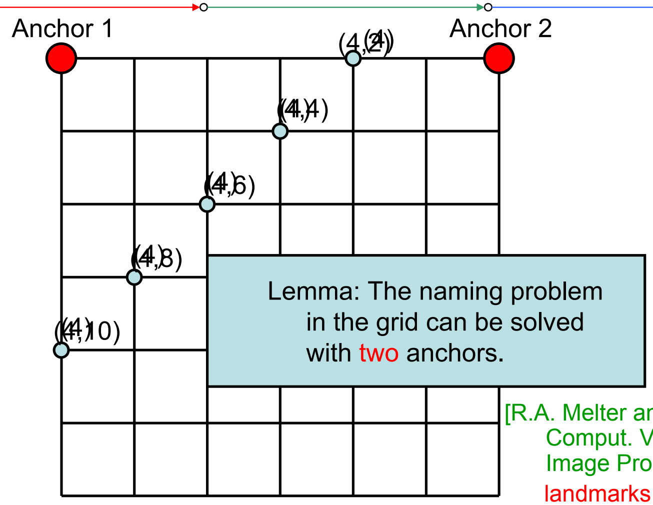
- For many applications, like routing, finding a realization of a UDG is not mandatory
- Virtual coordinates merely as infrastructure for geometric routing

→ Pseudo geometric 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

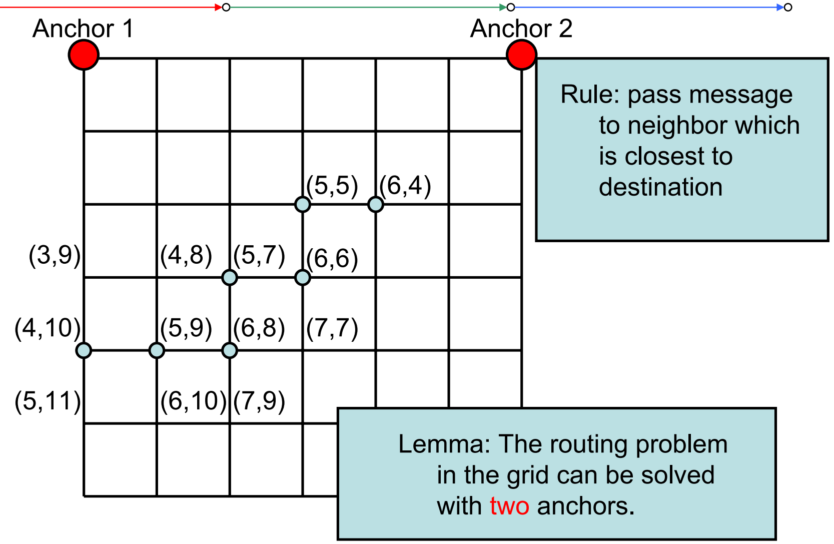
Pseudo-geometric routing in the grid: Naming



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

[R.A. Melter and I. Tomescu, Comput. Vision, Graphics. Image Process., 1984]: landmarks in graphs

Pseudo-geometric routing in the grid: Routing

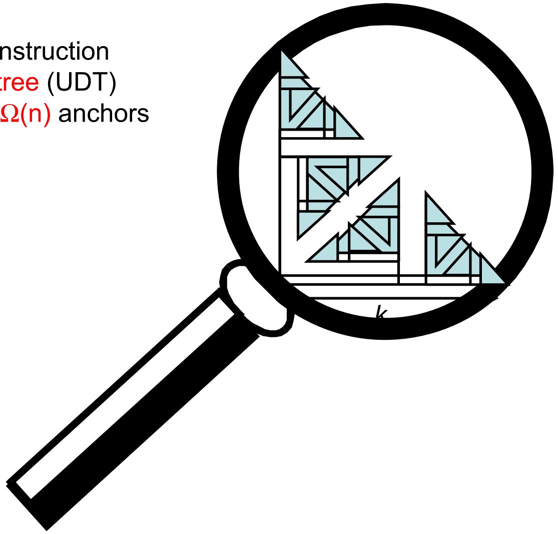


Rule: pass message to neighbor which is closest to destination

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

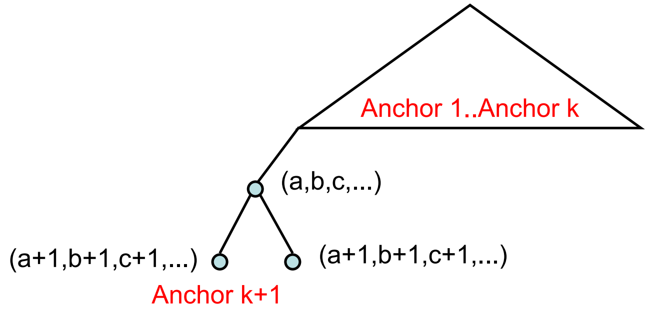
Problem: UDG is usually not a grid

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



Pseudo-geometric routing in the UDT: Naming

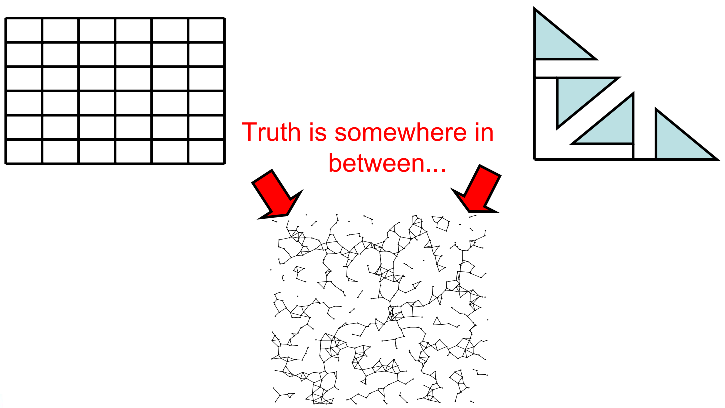
- Leaf-siblings can only be distinguished if one of them is an anchor:



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



Chapter 2
LOCATION SERVICES

Distributed Computing Group

EWSN 2006

Overview

- **Location Services & Routing**
  - Classification of location services
  - Home based
  - GLS
  - MLS

Location services

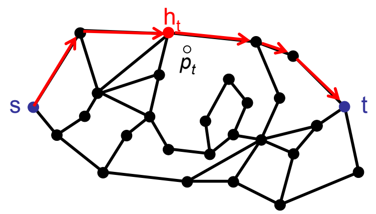
- **Service that maps node names to (geographic) coordinates**
  - Should be distributed (no require for specialized hardware)
  - Should be efficient
- **Lookup of the position (or COA) of a mobile node**
  - Mobile IP: Ask home agent
  - Home agent is determined through IP (unique ID) of MN
  - Possibly long detours even though sender and receiver are close
  - OK for Internet applications, where latency is (normally) low
- **Other application: Routing in a MANET**
  - MANET: mobile ad hoc network
  - No dedicated routing hardware
  - Limited memory on each node: cannot store huge routing tables
  - Nodes are mostly battery powered and have limited energy
  - Nodes route messages, e.g. using georouting

Home based georouting in a MANET

- **How can the sender learn the current position of another node?**
  - Flooding the entire network is undesirable (traffic and energy overhead)
- **Home based approach**
  - Similar to Mobile IP, each node has a *home* node, where it stores and regularly updates its current position
  - The home is determined by the unique ID of the node *t*. One possibility is to hash the ID to a position p_t and use the node closest to p_t as home.
  - Thus, given the ID of a node, every node can determine the position of the corresponding home.

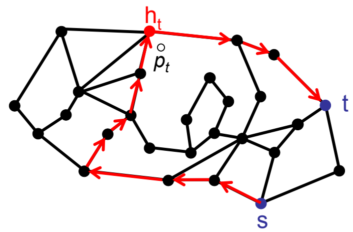
Home based routing

1. Route packet to h_t, the home of the destination *t*
2. Read the current position of *t*
3. Route to *t*



Home based location service – how good is it?

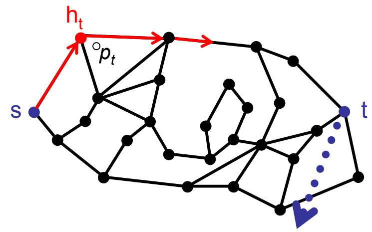
- Visiting the home of a node might be wasteful if the sender and receiver happen to be close, but the home far away



- The routing stretch is defined as

$$\text{stretch} := \frac{\text{length of route}}{\text{length of optimal route}}$$

  We want routing algorithms with low stretch.
- Simultaneous message routing and node movement might cause problems
- Can we do better?



Classification of location services

- **Proactive**
  - Mobile node divulges its position to all nodes whenever it moves
  - E.g. through flooding
- **Reactive**
  - Sender searches mobile host only when it wants to send a message
  - E.g. through flooding
- **Hybrid**
  - Both, proactive and reactive.
  - Some nodes store information about where a node is located
  - Arbitrarily complicated storage structures
  - Support for simultaneous routing and node mobility

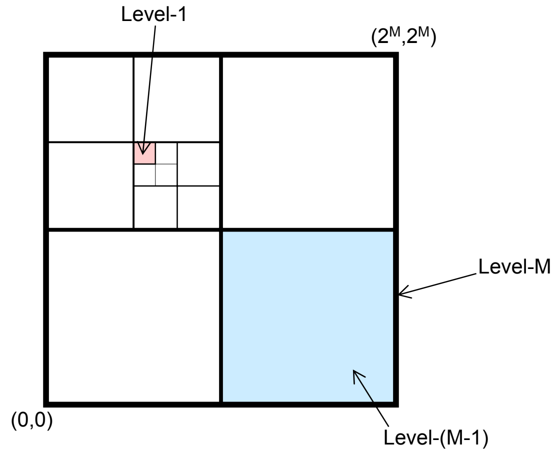
Location services: Lookup & Publish

- **Any node *A* can invoke to basic operations:**
  - Lookup(*A, B*): *A* asks for the position of B
  - Publish(*A*, *x*, *y*): *A* announces its move from position *x* to *y*
- **Open questions**
  - How often does a node publish its current position?
  - Where is the position information stored?
  - How does the lookup operation find the desired information?

The Grid Location Service (GLS), Li et. al (2000)

- Cannot get reasonable stretch with one single home. Therefore, use several homes (**location servers)** where the node publishes its position.
- The location servers are selected based on a grid structure:
  - The area in which the nodes are located is divided into squares
  - All nodes agree on the lower left corner (0,0) and upper right corner (2^M, 2^M), which forms the square called **level-M**
  - Recursively, each level-N square is split into 4 level-(N-1) squares
  - The recursion stops for level-1

The Grid



Addressing of nodes

- **Unique IDs** are generated for each node (e.g. by using a hash-function)
- ID space (all possible hash values) is **circular**
- Every node can find a **least greater** node w.r.t. the ID space (the **closest node**)

- Example:

  Let the ID space range from 1 to 99 and consider the IDs {3, 43, 80, 92}. Then, the least greater node with respect to the given ID space is

  3 → 43; 43 → 80; 80 → 92; 90 → 3

Selecting location servers

- Each node *A* recruits location servers using the underlying grid:
  - In each of the 3 level-1 squares that, along with *A*, make up a level-2 square, *A* chooses the node closest to its own ID as location server.
  - The same selection process is repeated on higher level squares.

92 87 23	92 17 53	92 31	
92 11	92	59 84	
62		49 73	92 2
92 3		33	42

Example for node 92, which selects the nodes {23, 17, 11} on the level-1 and {2, 3, 31} on level-2.

Complete example

	70, 72, 76, 81, 82, 84, 87 90	1, 5, 6, 10, 12, 14, 37, 62, 70, 90, 91 16				19, 35, 37, 45, 50, 51, 82 39	
1, 5, 16, 37, 62, 63, 90, 91 70			16, 17, 19, 21, 23, 26, 28, 31, 32, 35 37	19, 35, 39, 45, 51, 82 50		39, 41, 43 45	
1, 62, 70, 90 91	1, 5, 16, 37, 39, 41, 43, 45, 50, 51, 55, 61, 91 62	1, 2, 16, 37, 62, 70, 90, 91 5			35, 39, 45, 50 51		19, 35, 39, 45, 50, 51, 55, 61, 62, 63, 70, 72, 76, 81 82
	62, 91, 98 1				19, 20, 21, 23, 26, 28, 31, 32, 51, 82 35	1, 2, 5, 6, 10, 12, 14, 16, 17, 82, 84, 87, 90, 91, 98 19	
14, 17, 19, 20, 21, 23, 87 26		2, 17, 20, 63 23	2, 17, 23, 26, 31, 32, 43, 55, 61, 62 63	28, 31, 32, 35, 37, 39 41		10, 20, 21, 28, 41, 43, 45, 50, 51, 55, 61, 62, 63, 70 72	
14, 23, 26, 31, 32, 43, 55, 61, 63, 81, 82, 84 87	2, 12, 26, 87, 98 14	1, 17, 23, 63, 81, 87, 98 2	2, 12, 14, 16, 23, 63 17		6, 10, 20, 21, 23, 26, 41, 72, 76, 84 28	6, 72, 76, 84 10	
31, 81, 98 32	31, 32, 81, 87, 90, 91 98	12, 43, 45, 50, 51, 61 55	12, 43, 55 61	1, 2, 5, 21, 76, 84, 87, 90, 91, 98 6	6, 10, 20 , 76 21		6, 10, 12, 14, 16, 17, 19, 84 20
31, 32, 43, 55, 61, 63, 70, 72, 76, 98 81	2, 12, 14, 17, 23, 26, 28, 32, 81, 98 31	12, 14, 17, 23, 26, 31, 32, 35, 37, 39, 41, 55, 61 43	2, 5, 6, 10, 43, 55, 61, 63, 81, 87, 98 12		6, 21, 28, 41, 72 76	20, 21, 28, 41, 72, 76, 81, 82 84	

Querying location of other nodes

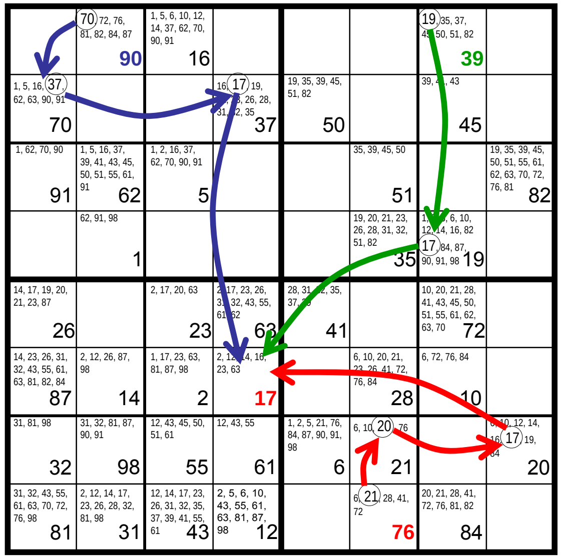
- **Lookup(*A*, *B*): Find a location server of node B**
  1. Node *A* sends the request (with georouting) to the node with ID closest to B for which *A* has location information
  2. Each node on the way forwards the request in the same way
  3. Eventually, the query reaches a location server of *B*, which forwards it to *B*.

Example: Send packet from 81 to 23

14, 17, 19, 20, 21, 23, 87 26		2, 17, 20, 63 23	2, 17, 23, 26, 31, 32, 43, 55, 61, 62 63
14, 23, 26, 31, 32, 43, 55, 61, 63, 81, 82, 84 87	2, 12, 26, 87, 98 14	1, 17, 23, 63, 81, 87, 98 2	2, 12, 14, 16, 23, 63 17
31, 81, 98 32	31, 32, 81, 87, 90, 91 98	12, 43, 45, 50, 51, 61 55	12, 43, 55 61
31, 32, 43, 55, 61, 63, 70, 72, 76, 98 81	2, 12, 14, 17, 23, 26, 28, 32, 81, 98 31	12, 14, 17, 23, 26, 31, 32, 35, 37, 39, 41, 55, 61 43	2, 5, 6, 10, 43, 55, 61, 63, 81, 87, 98 12

Lookup Example

Lookup for 17 from 76, 39 and 90

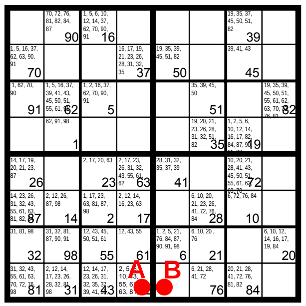


Analysis of GLS

- **Theorem 1:** A query needs no more than *k* location query steps to reach a location server of the destination when the sender and receiver are colocated in a level-*k* square.
- **Theorem 2:** The query never leaves the level-*k* square in which the sender and destination are colocated.

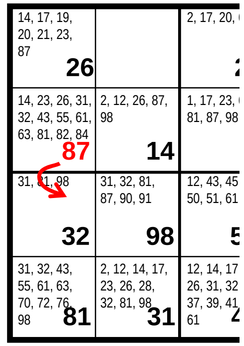
GLS has no worst case guarantees

- The lookup cost between two nodes might be arbitrarily high even though the nodes are very close
- The publish cost might be arbitrarily high even though a node only moved a very short distance
- In sparse networks, routing to the location server may have worst case cost, while routing directly can be more efficient



GLS and mobility

- Node crosses boundary line: what happens to the node's role as location server?
  - Must redistribute all information in the old level
  - Gather new information in the new level
  - Publish cost is arbitrarily high compared to the moved distance



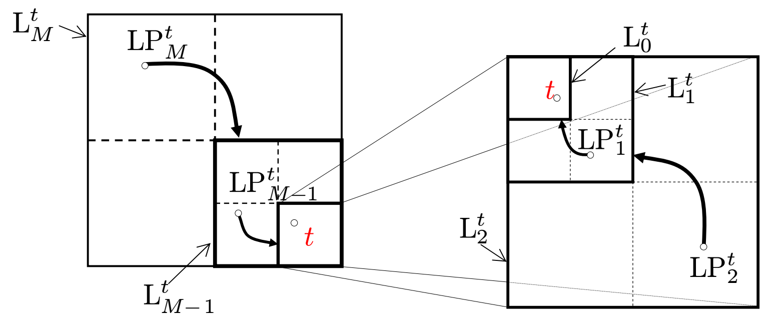
- A lookup happening in parallel with node movement might fail. Thus, GLS does not guarantee delivery for real concurrent systems, where nodes might move independently at any time.

Improving GLS

- Goals for MLS
  - Publish cost only depends on moved distance
  - Lookup cost only depends on the distance between the sender and receiver
  - Nodes might move arbitrarily at any time, even while other nodes issue lookup requests
  - Determine the maximum allowed node speed under which MLS still guarantees delivery

Location pointers (aka location servers)

- Difference to GLS:
  - Only *one* location pointer ($\mathbf{LP}$) per level (L) (GLS: 3 location servers)
  - The location pointer only knows in which sub-level the node is located (GLS: the location server knows the exact position)



Location pointer & Notation

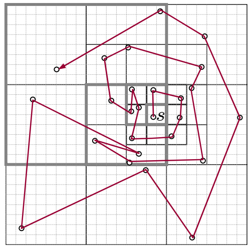
- Notation:
  - LP_k^t Location pointer for node *t* on level-*k*
  - L_k^t Level-*k* that contains node *t*
- The location pointers are placed depending on their ID, as in the home-based lookup system.
- The position of LP_k^t is obtained by hashing the ID of node *t* to a position in L_k^t. The location pointer is stored on the nearest nodes.

Routing in MLS

- Routing from a node *s* to a node *t* consists of two phases:
  1. Find a location pointer LP_k^t
  2. Once a first location pointer is found on level-*k*, we know in which of the 4 sub-squares *t* is located and thus in which L_{k-1} *t* has published another location pointer LP_{k-1}^t.
     Recursively, the message is routed towards location pointers on lower levels until it reaches the lowest level, from where it can be routed directly to *t*.

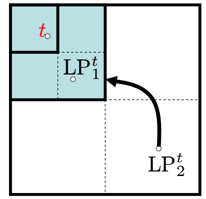
Routing in MLS (2)

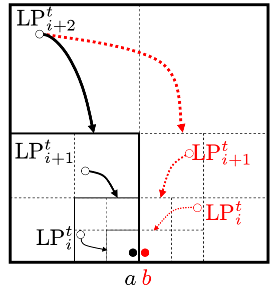
- When a node *s* wants to find a location pointer of a node *t*, it first searches in its immediate neighborhood and then extends the search area with exponential growing coverage.
  - First, try to find a location pointer LP_0^t in L_0^s or one of its 8 neighboring levels.
  - Repeat this search on the next higher level until a LP_k^t is found
- The lookup path draws a spiral-like shape with exponentially increasing radius until it finds a location pointer of *t*.
- Once a location pointer is found, the lookup request knows in which sub-square it can find the next location pointer of *t*.



Support for mobility in MLS

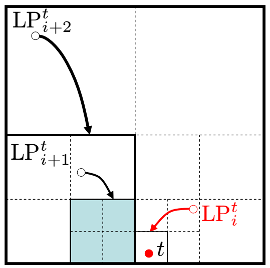
- A location pointer only needs to be updated when the node leaves the corresponding sub-square.
  - LP_2^t is OK as long as t remains in the shaded area.
  - Most of the time, only the closest few location pointers need to be updated due to mobility.
- Not enough: If a node moves across a level boundary, many pointers need to be updated. E.g. a node oscillates between the two points a and b.





Lazy publishing

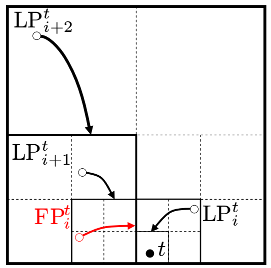
- Idea: Don't update a level pointer LP_k^t as long as *t* is still somewhat close to the level L_k where LP_k^t points.



- Breaks the lookup: LP_{i+1}^t points to a level that does not contain LP_i^t

Lazy publishing with forwarding pointers

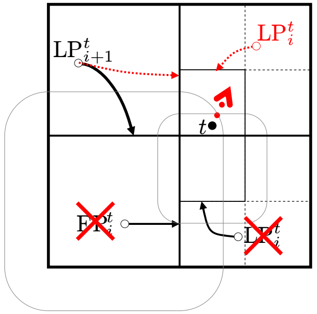
- No problem, add a **forwarding pointer** that indicates in which neighboring level the location pointer can be found.



Concurrency in MLS

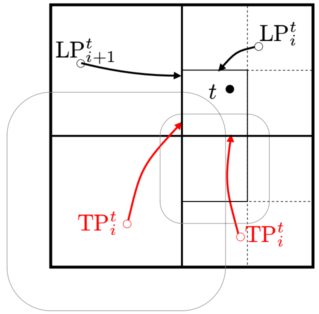
- Allowing for concurrent lookup requests and node mobility is somewhat tricky, especially the deletion of pointers.
- Note that a lookup request needs some time to travel between location pointers. The same holds for requests to create or delete location (or forwarding) pointers.

- Example:
  - A lookup request follows LP_{i+1}^t, and node *t* moves as indicated
  - *t* updates its LP_i^t and LP_{i+1}^t and removes the FP_i^t and the old LP_i^t
  - The lookup request fails if it arrives after the FP_i^t has been removed



Concurrency in MLS (2)

- No problem either: Instead of removing a location pointer or forwarding pointer, replace it with a **temporary pointer** that remains there for a *short time* until we are sure that no lookup request might arrive anymore on this outdated path.
- Similar to the forwarding pointer, a temporary pointer redirects a lookup to the neighbor level where the node is located.



Properties of MLS

- Constant lookup stretch
  - The length of the chosen route is only a constant longer than the optimal route
- Publish cost is O(*d* log *d*) where moved distance is *d*
  - Even if nodes move considerably, the induced message overhead due to publish requests is moderate.
- Works in a concurrent setup
  - Lookup requests and node movement might interleave arbitrarily
- Nodes might not move faster than 1/15 of the underlying routing speed
  - We can determine the maximum node speed that MLS supports. Only if nodes move faster, there might arise situations where a lookup request fails.

MLS Conclusions

- It's somewhat tricky to handle concurrency properly
  - Use of temporary forwarding pointers
- MLS is the first location service that determines the maximum speed at which nodes might move
  - Without the speed limitation, no delivery guarantees can be made!
- Drawbacks
  - MLS utilizes an underlying routing algorithm that can deliver messages with constant stretch given the position of the destination
  - MLS requires a relatively dense node population

Chapter 3
POSITIONING

EWSN 2006

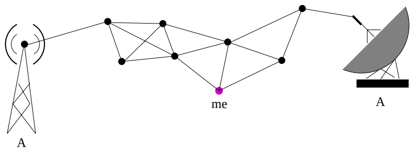
Distributed Computing Group

Overview

- Motivation
- Measurements
- Anchors
- Virtual Coordinates
- Heuristics
- Practice

Motivation

- Why positioning?
  - Sensor nodes without position information is often meaningless
  - Heavy and/or costly positioning hardware
  - Geo-routing



- Why not GPS (or Galileo)?
  - Heavy, large, and expensive (as of yet)
  - Battery drain
  - Not indoors
  - Accuracy?
- Solution: equip small fraction with GPS (anchors)

Measurements

Distance estimation

- Received Signal Strength Indicator (RSSI)
  - The further away, the weaker the received signal.
  - Mainly used for RF signals.
- Time of Arrival (ToA) or Time Difference of Arrival (TDoA)
  - Signal propagation time translates to distance.
  - RF, acoustic, infrared and ultrasound.

Angle estimation

- Angle of Arrival (AoA)
  - Determining the direction of propagation of a radio-frequency wave incident on an antenna array.
- Directional Antenna
- Special hardware, e.g., laser transmitter and receivers.

Positioning (a.k.a. Localization)

- Task: Given distance or angle measurements or mere connectivity information, find the locations of the sensors.

- Anchor-based
  - Some nodes know their locations, either by a GPS or as pre-specified.
- Anchor-free
  - Relative location only. Sometimes called virtual coordinates.
  - Theoretically cleaner model (less parameters, such as anchor density)

- Range-based
  - Use range information (distance estimation).
- Range-free
  - No distance estimation, use connectivity information such as hop count.
  - It was shown that bad measurements don't help a lot anyway.

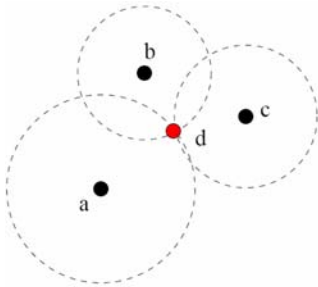
Trilateration and Triangulation

- Use geometry, measure the distances/angles to three anchors.

- Trilateration: use distances
  - Global Positioning System (GPS)

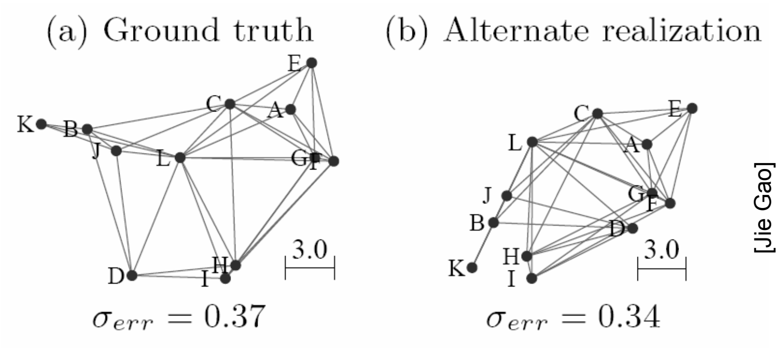
- Triangulation: use angles
  - Some cell phone systems

- How to deal with inaccurate measurements?
  - Least squares type of approach
  - What about strictly more than 3 (inaccurate) measurements?



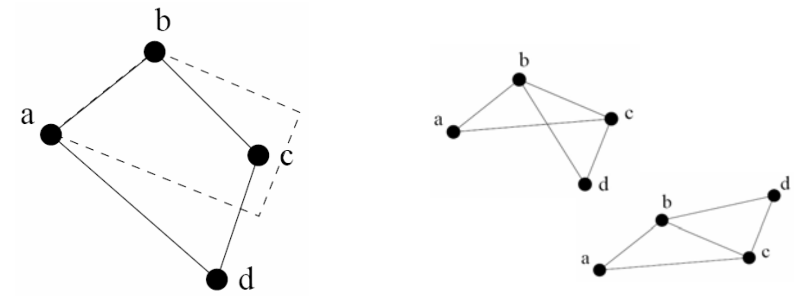
Ambiguity Problems

- Same distances, different realization.



[Jie Gao]

Continuous deformation, flips, etc.

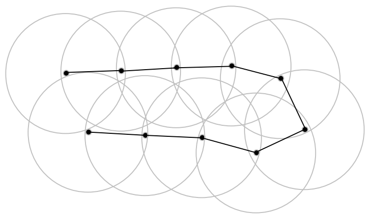


[Jie Gao]

- Rigidity theory: Given a set of rigid bars connected by hinges, rigidity theory studies whether you can move them continuously.

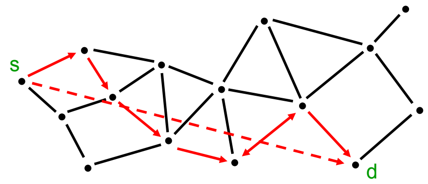
Simple hop-based algorithms

- Algorithm
  - Get graph distance h to anchor(s)
  - Intersect circles around anchors
    - radius = distance to anchor
  - Choose point such that maximum error is minimal
    - Find enclosing circle (ball) of minimal radius
    - Center is calculated location
- In higher dimensions: $1 < d \leq h$
  - Rule of thumb: Sparse graph
    → bad performance



How about no anchors at all...?

- In absence of anchors...
  → ...nodes are clueless about real coordinates.
- For many applications, real coordinates are not necessary
  → Virtual coordinates are sufficient
  → Geometric Routing requires only virtual coordinates
    - Require no routing tables
    - Resource-frugal and scalable



Virtual Coordinates

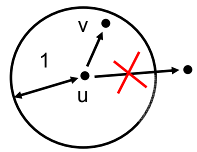
- Idea:
  Close-by nodes have similar coordinates
  Distant nodes have very different coordinates

→ Similar coordinates imply physical proximity!

- Applications
  - Geometric Routing
  - Locality-sensitive queries
  - Obtaining meta information on the network
  - Anycast services („*Which of the service nodes is closest to me?*“)
  - Outside the sensor network domain: e.g., Internet mapping

Model

- Unit Disk Graph (UDG) to model wireless multi-hop network
  - Two nodes can communicate iff Euclidean distance is at most 1



- Sensor nodes may not be capable of
  - Sensing directions to neighbors
  - Measuring distances to neighbors
- Goal: Derive topologically correct coordinate information from connectivity information only.
  - Even the simplest nodes can derive connectivity information

Context

	Distance/Angle information	Connectivity information only
With Anchors	Positioning (Solution quality depends on anchor density)	
No Anchors	Distance/Angle based Virtual Coordinates	Connectivity based Virtual Coordinates

next

Virtual Coordinates ⟷ UDG Embedding

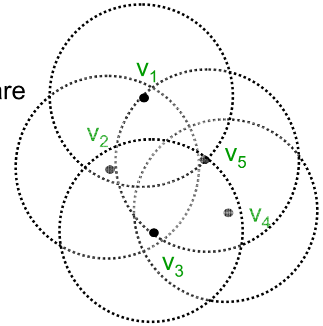
- Given the connectivity information for each node...

v_1	v_2	v_3	v_4	v_5
v_2	v_1	v_2	v_3	v_1
v_5	v_3	v_4	v_5	v_3
		v_5		v_4

...and knowing the underlying graph is a UDG...

- ...find a UDG embedding in the plane such that all connectivity requirements are fulfilled! (→ Find a realization of a UDG)

This problem is NP-hard!
(Simple reduction to *UDG-recognition* problem, which is NP-hard)
[Breu, Kirkpatrick, Comp.Geom.Theory 1998]



UDG Approximation – Quality of Embedding

- Finding an exact realization of a UDG is NP-hard.
  → Find an embedding r(G) which approximates a realization.
- Particularly,
  → Map adjacent vertices (edges) to points which are close together.
  → Map non-adjacent vertices („non-edges") to far apart points.
- Define quality of embedding q(r(G)) as:

Ratio between longest edge to shortest non-edge in the embedding.

Let ρ(u,v) be the distance between points u and v in the embedding.

$$q(r(G)) := \frac{\max_{\{u,v\}\in E}\rho(u,v)}{\min_{\{u',v'\}\notin E}\rho(u',v')}$$

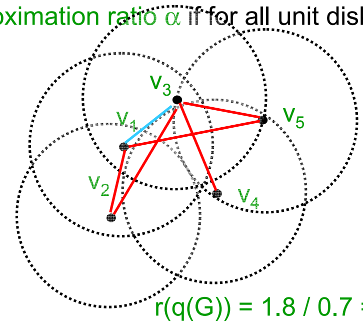
UDG Approximation

- For each UDG G, there exists an embedding r(G), such that, $q(r(G)) \leq 1$.
  (a realization of G)

$$q(r(G)) := \frac{\max_{\{u,v\}\in E}\rho(u,v)}{\min_{\{u',v'\}\notin E}\rho(u',v')}$$

- Finding such an embedding is NP-hard
- An algorithm ALG achieves approximation ratio α if for all unit disk graphs G, $q(r_{ALG}(G)) \leq \alpha$.
- Example:

v_1	v_2	v_3	v_4	v_5
v_2	v_1	v_2	v_3	v_1
v_5	v_3	v_4	v_5	v_3
		v_5		v_4



r(q(G)) = 1.8 / 0.7 = 2.6

Some Results

- There are a few virtual coordinates algorithms
  All of them evaluated only by simulation on random graphs
- In fact there is only one provable approximation algorithm

> There is an algorithm which achieves an approximation ratio of $O(\log^{2.5} n \sqrt{\log \log n})$, n being the number of nodes in G.

- Plus there are lower bounds on the approximability.

> There is no algorithm with approximation ratio better than $\sqrt{3/2} - \epsilon$, unless P=NP.

Approximation Algorithm: Overview

- Four major steps

1. Compute metric on MIS of input graph → Spreading constraints
   (Key conceptual difference to previous approaches!)
2. Volume-respecting, high dimensional embedding
3. Random projection to 2D
4. Final embedding

UDG Graph G with MIS M.

↓

Approximate pairwise distances between nodes such that, MIS nodes are neatly spread out.

↓

Volume respecting embedding of nodes in $\boldsymbol{R}^n$ with small distortion.

↓

Nodes spread out fairly well in $\boldsymbol{R}^2$.

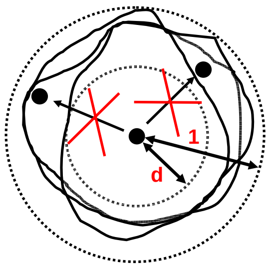
↓

Final *embedding of G in* $\boldsymbol{R}^2$.

Lower Bound: Quasi Unit Disk Graph

- Definition Quasi Unit Disk Graph:

> Let V∈ $\mathbf{R}^2$, and d ∈ [0,1]. The symmetric Euclidean graph G=(V,E), such that for any pair u,v ∈ V
> - dist(u,v) ≤ d ⇒ {u,v} ∈ E
> - dist(u,v) > 1 ⇒ {u,v} ∉ E
>
> is called *d-quasi unit disk graph*.



- Note that between d and 1, the existence of an edge is unspecified.

Reduction

- We want to show that finding an embedding with $q(r(G)) \leq \sqrt{3/2} - \epsilon$, where ε goes to 0 for n → ∞ is NP-hard.
- We prove an equivalent statement:

> Given a unit disk graph G=(V,E), it is NP-hard to find a realization of G as a d-quasi unit disk graph with $d \geq \sqrt{2/3} + \epsilon$, where ε tends to 0 for n→∞.

→ Even when allowing non-edges to be smaller than 1, embedding a unit disk graph remains NP-hard!

→ It follows that finding an approximation ratio better than $\sqrt{3/2} - \epsilon$ is also NP-hard.

Reduction

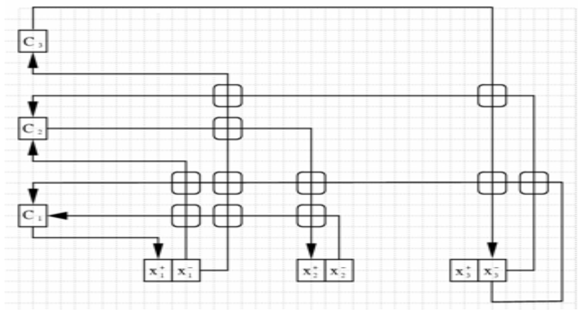
- Reduction from 3-SAT (each variable appears in at most 3 clauses)
- Given a instance C of this 3-SAT, we give a polynomial time construction of $G_C=(V_C, E_C)$ such that the following holds:

> – C is satisfiable ⇒ G_C is realizable as a unit disk graph
> – C is not satisfiable ⇒ G_C is not realizable as a d-quasi unit disk graph with $d \geq \sqrt{2/3+\epsilon}$

- Unless P=NP, there is no approximation algorithm with approximation ratio better than $\sqrt{3/2-\epsilon}$.

Proof idea

- Construct a grid drawing of the SAT instance.
- Grid drawing is *orientable* iff SAT instance is satisfiable.
- Grid components (clauses, literals, wires, crossings,...) are composed of nodes → Graph G_C.
- G_C is realizable as a d-quasi unit disk graph with $d \geq \sqrt{2/3}+\epsilon$ iff grid drawing is orientable.



Summary

- Virtual coordinates problem is important!
- Natural formulation as unit disk graph embedding.
  → Clear-cut optimization problem.

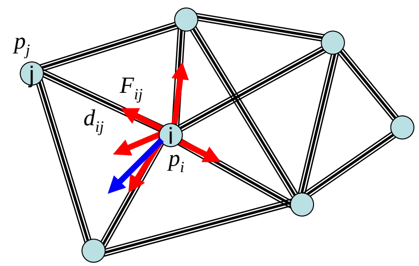
> Upper Bound : $\alpha \in O(\log^{2.5} n \sqrt{\log\log n})$
> Lower Bound : $\alpha \geq \sqrt{3/2-\epsilon}$

→ Gap between upper and lower bound is huge!

> **Open Problems:**
> • Diminish gap between upper and lower bound
> • Distributed Algorithm

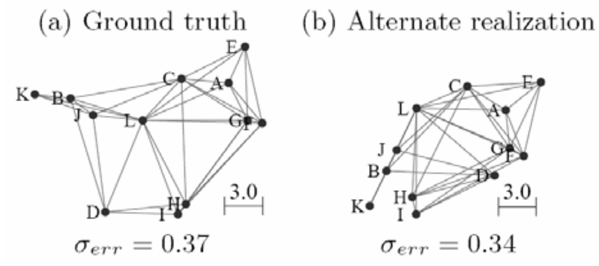
Heuristics: Spring embedder

- Nodes are "masses", edges are "springs".
- Length of the spring equals the distance measurement.
- Springs put forces to the nodes, nodes move, until stabilization.
- Force: $F_{ij}=d_{ij}-r_{ij}$, along the direction $p_i p_j$.
- Total force on n_i: $F_i=\Sigma F_{ij}$.
- Move the node n_i by a small distance (proportional to F_i).



Spring Embedder Discussion

- Problems:
  - may deadlock in local minimum
  - may never converge/stabilize (e.g. just two nodes)
- Solution: Need to start from a reasonably good initial estimation.



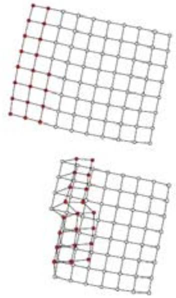
[Jie Gao]

Heuristics: Priyantha et al.

N.B. Priyantha, H. Balakrishnan, E. Demaine, S. Teller: **Anchor-Free Distributed Localization in Sensor Networks**, *SenSys*, 2003.

iterative process minimizes the layout energy

$$E(p) = \sum_{\{i,j\}\in E} \left(||p_i - p_j|| - \ell_{ij}\right)^2$$



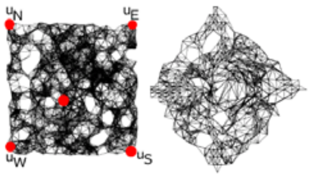
- fact: layouts can have *foldovers* without violating the distance constraints
- problem: optimization can converge to such a local optimum
- solution: find a good initial layout *fold-free* → already close to the global optimum (="real layout")

[Fleischer & Pich]

Continued

Phase 1: compute initial layout

- determine periphery nodes u_N, u_S, u_W, u_E
- determine central node u_C
- use polar coordinates



$$\rho_v = d(v, u_C) \quad \theta_v = \arctan\left(\frac{d(v,u_N) - d(v,u_S)}{d(v,u_W) - d(v,u_E)}\right)$$

as positions of node v

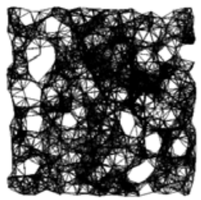
Phase 2: Spring Embedder

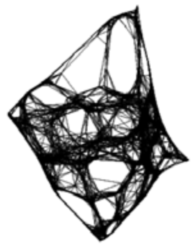
[Fleischer & Pich]

Heuristics: Gotsman et al.

C. Gotsman, Y. Koren [5]. **Distributed Graph Layout for Sensor Networks**, *GD*, 2004.

- initial placement: spread sensors $\frac{\sum_{\{i,j\}\in E} \exp(-\ell_{ij})||p_i - p_j||^2}{\sum_{i<j} ||p_i - p_j||^2} \to \min$
- linear algebra: minimized by second highest eigenvector v_2 of A where
  $a_{ij} = -\frac{\exp(-\ell_{ij})}{\sum_{j:\{i,j\}\in E} \exp(-\ell_{ij})}$
  $a_{ii} = 1$
- $x, Ax, A^2x, A^3x, \ldots$ converges to v_2
- $x_i \leftarrow \frac{1}{2}\left(x_i + \frac{\sum_{j:\{i,j\}\in E} \exp(-\ell_{ij} x_j)}{\sum_{j:\{i,j\}\in E} \exp(-\ell_{ij})}\right)$
- compute third eigenvector v_3, use v_2, v_3 as coordinates

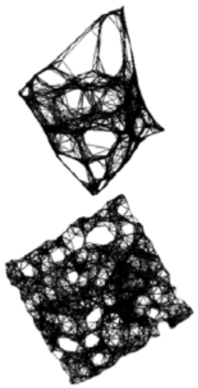




[Fleischer & Pich]

Continued

- distributed optimization (spring model)
- alternative: *majorization*
- compute sequence of layouts $p^{(0)}, p^{(1)}, p^{(2)}, \ldots$ with $E(p^{(0)}) \geq E(p^{(1)}) \geq E(p^{(2)}) \geq \ldots$
  - solve linear equation $L^{(t+1)} p^{(t+1)} = L^{(t)} p^{(t)}$ in distributed manner

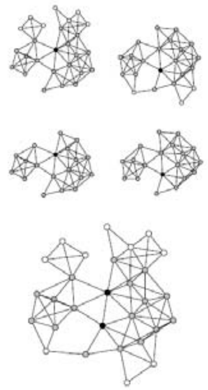


[Fleischer & Pich]

Heuristics: Shang et al.

Y. Shang, W. Ruml [7].
Improved MDS-based Localization, *IEEE Infocom*, 2004.

- compute a local map for each node (local MDS of the 2-hop neighborhood)
- merge local map patches into a global map (use incremental or binary-tree strategy)
- apply distributed optimization to the result

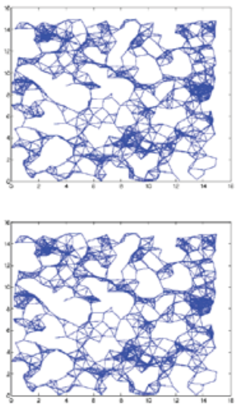


[Fleischer & Pich]

Heuristics: Bruck et al.

J. Bruck, J. Gao, A. Jiang [8]. **Localization and Routing in Sensor Networks by Local Angle Information**, *Mobile Ad Hoc Networking & Computing*, 2005.

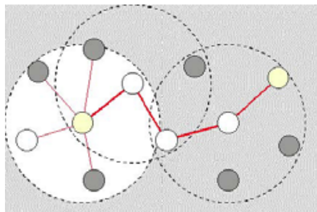
- Choose an edge e as x-axis to obtain absolute angles.
- Form an LP whose variables are the edge lengths $\ell(e)$.
- For all edges $0 \leq \ell(e) \leq 1$.
- For any cycle $e_1, \ldots, e_p$: $\sum_{i=1}^{p} \ell(e_i) \cos \theta_i = 0$ and $\sum_{i=1}^{p} \ell(e_i) \sin \theta_i = 0$.
- Non-adjacent node pair constraints.
- Crossing-edge constraints.



[Fleischer & Pich]

Practical lessons

Theory



Practice

- RSSI in sensor networks: good, but not for “reasonable” localization
- For exact indoor localization
  - Buy special hardware (e.g., UWB)
  - Place huge amount of short range anchors for single-hop localization

Chapter 4
DATA GATHERING

Distributed
Computing
Group

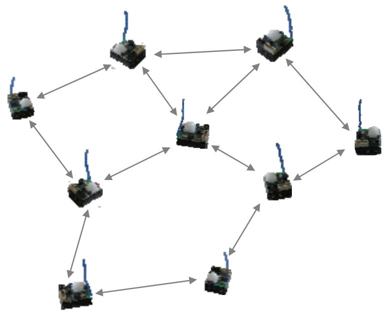
EWSN 2006

Overview

- Motivation
- Data gathering with coding
  - Self-coding
    - Excursion: Shallow Light Tree
  - Foreign coding
  - Multicoding
- Universal data gathering tree
  - Max, Min, Average, Median, Count Distinct, …
- Energy-efficient broadcasting

Sensor networks

- Sensor nodes
  - Processor & memory
  - Short-range radio
  - Battery powered
- Requirements
  - Monitoring geographic region
  - Unattended operation
  - Long lifetime

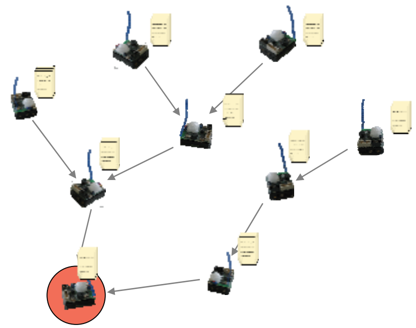


Data gathering

- All nodes produce relevant information about their vicinity periodically.
- Data is conveyed to an information sink for further processing.

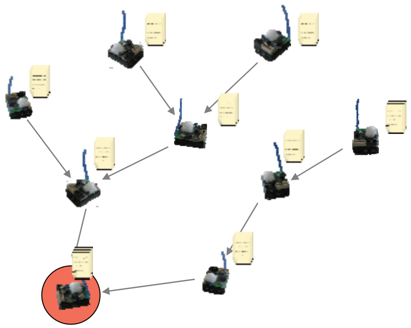
➡ Routing scheme

On which path is node u's data forwarded to the sink?



Time coding

- The simplest trick in the book: If the sensed data of a node changes not too often (e.g. temperature), the node only needs to send a new message when its data changes.
- Improvement: Only send change of data, not actual data (similar to video codecs)



More than one sink?

- Use the anycast approach, and send to the closest sink.
- In the simplest case, a source wants to minimize the number of hops. To make anycast work, we only need to implement the regular distance-vector routing algorithm.
- However, one can imagine more complicated schemes where e.g. sink load is balanced, or even intermediate load is balanced.

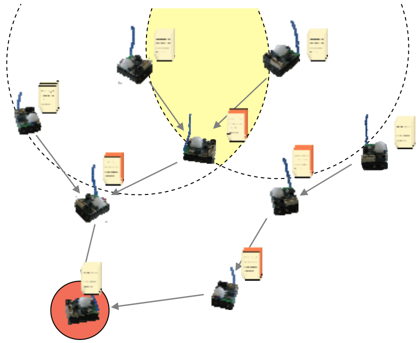
Correlated Data

- Different sensor nodes partially monitor the same spatial region.

  ➡ Data correlation

- Data might be processed as it is routed to the information sink.

  ➡ In-network coding



At which node is node u's data encoded?

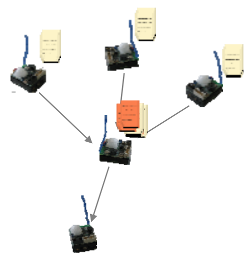
Find a routing scheme and a coding scheme to deliver data packets from all nodes to the sink such that the overall energy consumption is minimal.

Coding strategies

- Multi-input coding
  - Exploit correlation among several nodes.
  - Combined aggregation of all incoming data.

  ➡ Recoding at intermediate nodes

  ➡ Synchronous communication model



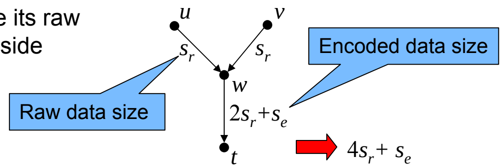
- Single-input coding
  - Encoding of a nodes data only depends on the side information of one other node.

  ➡ No recoding at intermediate nodes

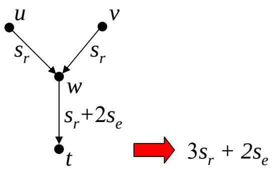
  ➡ No waiting for belated information at intermediate nodes

Single-input coding

- Self-coding
  - A node can only encode its raw data in the presence of side information.



- Foreign coding
  - A node can use its raw data to encode data it is relaying.



Self-coding

- Lower-bound the cost of an optimal

$$c_{opt} = \sum_{u \in B} \left(s_r \cdot \mathsf{ST}(S_u, u, t) + \sum_{v \in S_u} s_e \cdot \mathsf{SP}(v, t) \right).$$

Set of nodes that encode with data from u

Set of nodes with no side information

Steiner tree

Shortest path

- Two ways to lower-bound this equation:
  - $c_{opt} \geq \sum_{u \in V} s_e \cdot \mathsf{SP}(u, t)$
  - $c_{opt} \geq s_r \cdot c(\mathsf{MST})$

Algorithm

- LEGA (Low Energy Gathering Algorithm)
- Based on the shallow light tree (SLT)
- Compute SLT rooted at the sink t.
- The sink t transmits its packet p_t
- Upon reception of a data packet p_j at node v_i
  - Encode p_i with $p_j \rightarrow p_i^j$
  - Transmit p_i^j to the sink t
  - Transmit p_i to all children

Size = s_r

Size = s_e

Excursion: Shallow-Light Tree (SLT)

- Introduced by [Awerbuch, Baratz, Peleg, PODC 1990]
- Improved by [Khuller, Raghavachari, Young, SODA 1993]
  - new name: Light-Approximate-Shortest-Path-Tree (LAST)
- Idea: Construct a spanning tree for a given root r that is both a MST-approximation as well as a SPT-approximation for the root r. In particular, for any $\gamma > 0$
  - $c(\mathsf{SLT}) \leq (1 + \sqrt{2}/\gamma) \cdot c(\mathsf{MST})$
  - $d_{SLT}(v_i, r) \leq (1 + \sqrt{2}\gamma) \cdot \mathsf{SP}(v_i, r)$
- Remember:
  - MST: Easily computable with e.g. Prim's greedy edge picking algorithm
  - SPT: Easily computable with e.g. Dijkstra's shortest path algorithm

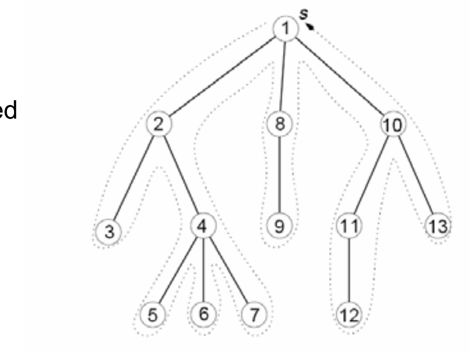
MST vs. SPT

- Is a good SPT not automatically a good MST (or vice versa)?

Result & Preordering

- Main Theorem: Given an $\alpha > 1$, the algorithm returns a tree T rooted at r such that all shortest paths from r to u in T have cost at most α the shortest path from r to u in the original graph (for all nodes u). Moreover the total cost of T is at most $\beta = 1+2/(\alpha-1)$ the cost of the MST.

- We need an ingredient: A preordering of a rooted tree is generated when ordering the nodes of the tree as visited by a depth-first search algorithm.

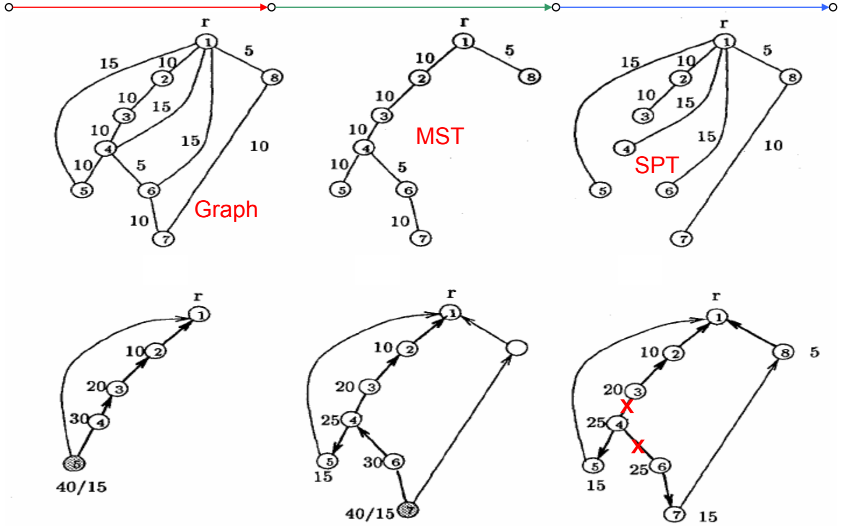


The SLT Algorithm

1. Compute MST H of Graph G;
2. Compute all shortest paths (SPT) from the root r.
3. Compute preordering of MST with root r.
4. For all nodes v in order of their preordering do
   - Compute shortest path from r to u in H. If the cost of this shortest path in H is more than a factor α more than the cost of the shortest path in G, then just add the shortest path in G to H.
5. Now simply compute the SPT with root r in H.

- Sounds crazy… but it works!

An example, $\alpha = 2$



Proof of Main Theorem

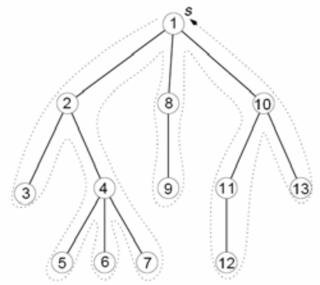
- The SPT α-approximation is clearly given since we included all necessary paths during the construction and in step 5 only removed edges which were not in the SPT.
- We need to show that our final tree is a β-approximation of the MST. In fact we show that the graph H before step 5 is already a β-approximation!
- For this we need a little helper lemma first…

A preordering lemma

- Lemma: Let T be a rooted spanning tree, with root r, and let z_0, z_1, …, z_k be arbitrary nodes of T in preorder. Then,

$$\sum_{i=1}^{k} d_T(z_{i-1}, z_i) \leq 2 \cdot cost(T).$$

- "Proof by picture": Every edge is traversed at most twice.
- Remark: Exactly like the 2-approximation algorithm for metric TSP.



Proof of Main Theorem (2)

- Let $z_1, z_2, \ldots, z_k$ be the set of k nodes for which we added their shortest paths to the root r in the graph in step 4. In addition, let z_0 be the root r. The node z_i can only be in the set if (for example) $d_G(r,z_{i-1}) + d_{MST}(z_{i-1},z_i) > \alpha d_G(r,z_i)$, since the shortest path (r,z_{i-1}) and the path on the MST (z_{i-1},z_i) are already in H when we study z_i.
- We can rewrite this as $\alpha d_G(r,z_i) - d_G(r,z_{i-1}) < d_{MST}(z_{i-1},z_i)$. Summing up:

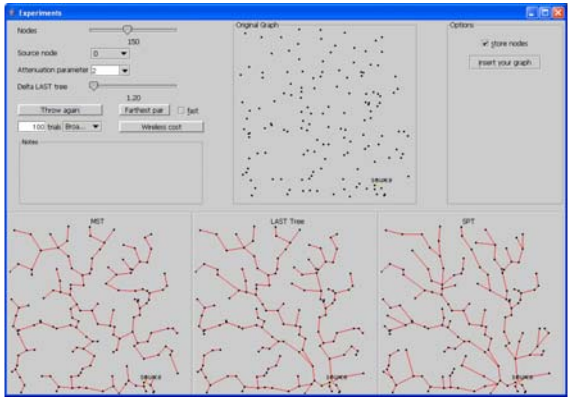
$\alpha d_G(r,z_1) - d_G(r,z_0)$	$<$	$d_{MST}(z_0,z_1)$	(i=1)
$\alpha d_G(r,z_2) - d_G(r,z_1)$	$<$	$d_{MST}(z_1,z_2)$	(i=2)
…	…	…	
$\alpha d_G(r,z_k) - d_G(r,z_{k-1})$	$<$	$d_{MST}(z_{k-1},z_k)$	(i=k)
$\Sigma_{i=1\ldots k}(\alpha-1)\, d_G(r,z_i) \quad + \cancel{d_G(r,z_k)}$	$<$	$\Sigma_{i=1\ldots k}\, d_{MST}(z_{i-1},z_i)$	

Proof of Main Theorem (3)

- In other words, $(\alpha-1)\, \Sigma_{i=1\ldots k}\, d_G(r,z_i) < \Sigma_{i=1\ldots k}\, d_{MST}(z_{i-1},z_i)$
- All we did in our construction of H was to add exactly at most the cost $\Sigma_{i=1\ldots k}\, d_G(r,z_i)$ to the cost of the MST. In other words, $cost(H) \leq cost(MST) + \Sigma_{i=1\ldots k}\, d_G(r,z_i)$.
- Using the inequality on the top of this slide we have $cost(H) < cost(MST) + 1/(\alpha-1)\, \Sigma_{i=1\ldots k}\, d_{MST}(z_{i-1},z_i)$.
- Using our preordering lemma we have $cost(H) \leq cost(MST) + 1/(\alpha-1)\, 2cost(MST) = 1+2/(\alpha-1)\, cost(MST)$
- That's exactly what we needed: $\beta = 1+2/(\alpha-1)$.

How the SLT can be used

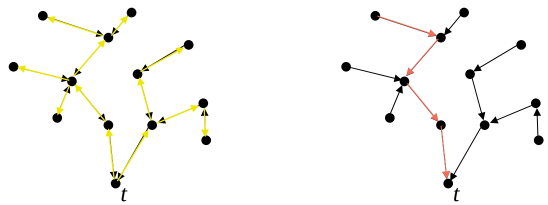
- The SLT has many applications in communication networks.
- Essentially, it bounds the cost of unicasting (using the SPT) and broadcasting (using the MST).
- Remark: If you use $\alpha = 1+\sqrt{2}$, then $\beta = 1+2/(\alpha-1) = \alpha$.



www.dia.unisa.it/~ventre

Analysis of LEGA

Theorem: LEGA achieves a $2(1+\sqrt{2})$-approximation of the optimal topology. (We use $\alpha = 1+\sqrt{2}$.)



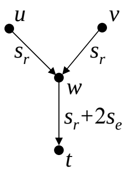
$s_r \cdot c(\mathrm{SLT})$ $\qquad$ $\sum_{v_i \in V} s_e \cdot d_{SLT}(v_i, t)$

$$c_{LEGA} \leq s_r \cdot (1+\sqrt{2})c(\mathrm{MST}) + (1+\sqrt{2}) \sum_{v_i \in V} s_e \cdot \mathrm{SP}(v_i, t)$$
$$\leq 2(1+\sqrt{2})c_{opt}$$

Slide 9/10

Foreign coding

- MEGA (Minimum-Energy Gathering Algorithm)
  - Superposition of two tree constructions.

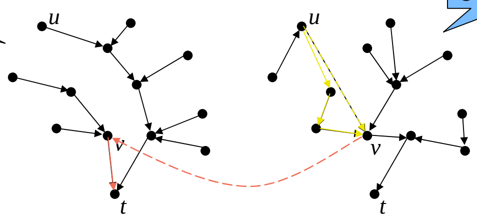


- Compute the shortest path tree (SPT) rooted at t.
- Compute a coding tree.
  - Determine for each node u a corresponding encoding node v.

Encoding must not result in cyclic dependencies.

SPT

Coding tree



Coding tree construction

- Build complete directed graph
- Weight of an edge $e=(v_i,v_j)$

$$w(e) = s_i \cdot \mathrm{SP}(v_i, v_j) + s_i^j \cdot \mathrm{SP}(v_j, t)$$

Cost from v_j to the sink t.

Cost from v_i to the encoding node v_j.

Number of bits when encoding v_i's info at v_j

- Compute a directed minimum spanning tree (arborescence) of this graph. (This is not trivial, but possible.)

Theorem: MEGA computes a minimum-energy data gathering topology for the given network.

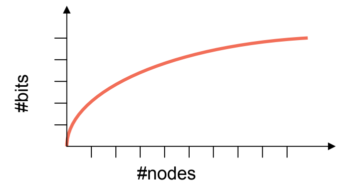
All costs are summarized in the edge weights of the directed graph.

Summary

- Self-coding:
  - The problem is NP-hard [Cristescu et al, INFOCOM 2004]
  - LEGA uses the SLT and gives a $2(1+\sqrt{2})$-approximation.
  - Attention: We assumed that the raw data resp. the encoded data always needs s_r resp. s_e bits (no matter how far the encoding data is!). This is quite unrealistic as correlation is usually regional.
- Foreign coding
  - The problem is in P, as computed by MEGA.
- What if we allow both coding strategies at the same time?
- What if multicoding is still allowed?

Multicoding

- Hierarchical matching algorithm [Goel & Estrin SODA 2003].
- We assume to have concave, non-decreasing aggregation functions. That is, to transmit data from k sources, we need f(k) bits with f(0)=0, f(k) $\geq$ f(k-1), and f(k+1)/f(k) $\leq$ f(k)/f(k-1).



- The nodes of the network must be a metric space*, that is, the cost of sending a bit over edge (u,v) is c(u,v), with
  - Non-negativity: c(u,v) $\geq$ 0
  - Zero distance: c(u,u) = 0 (*we don't need the identity of indescernibles)
  - Symmetry: c(u,v) = c(v,u)
  - Triangle inequality: c(u,w) $\leq$ c(u,v) + c(v,w)

The algorithm

- Remark: If the network is not a complete graph, or does not obey the triangle inequality, we only need to use the cost of the shortest path as the distance function, and we are fine.
- Let S be the set of source nodes. Assume that S is a power of 2. (If not, simply add copies of the sink node until you hit the power of 2.) Now do the following:

1. Find a min-cost perfect matching in S.
2. For each of the matching edges, remove one of the two nodes from S (throw a regular coin to choose which node).
3. If the set S still has more than one node, go back to step 1. Else connect the last remaining node with the sink.

The result

- Theorem: For any concave, non-decreasing aggregation function f, and for [optimal] total cost C[*], the hierarchical matching algorithm guarantees

$$E\left[\max_{\forall f} \frac{C(f)}{C^*(f)}\right] \leq 1 + \log k.$$

- That is, the expectation of the worst cost overhead is logarithmically bounded by the number of sources.
- Proof: Too intricate to be featured in this lecture.

Remarks

- For specific concave, non-decreasing aggregation functions, there are simpler solutions.
  - For f(x) = x the SPT is optimal.
  - For f(x) = const (with the exception of f(0) = 0), the MST is optimal.
  - For anything in between it seems that the SLT again is a good choice.
  - For any a priori known f one can use a deterministic solution by [Chekuri, Khanna, and Naor, SODA 2001]
  - If we only need to minimize the maximum expected ratio (instead of the expected maximum ratio), [Awerbuch and Azar, FOCS 1997] show how it works.

- Again, sources are considered to aggregate equally well with other sources. A correlation model is needed to resemble the reality better.

Other work using coding

- LEACH [Heinzelman et al. HICSS 2000]: randomized clustering with data aggregation at the clusterheads.
  - Heuristic and simulation only.
  - For provably good clustering, see the next chapter.

- Correlated data gathering [Cristescu et al. INFOCOM 2004]:
  - Coding with Slepian-Wolf
  - Distance independent correlation among nodes.
  - Encoding only at the producing node in presence of side information.
  - Same model as LEGA, but heuristic & simulation only.
  - NP-hardness proof for this model.

TinyDB and TinySQL

- Use paradigms familiar from relational databases to simplify the "programming" interface for the application developer.

```
SELECT roomno, AVERAGE(light), AVERAGE(volume)
FROM sensors
GROUP BY roomno
HAVING AVERAGE(light) > l AND AVERAGE(volume) > v
EPOCH DURATION 5min
```

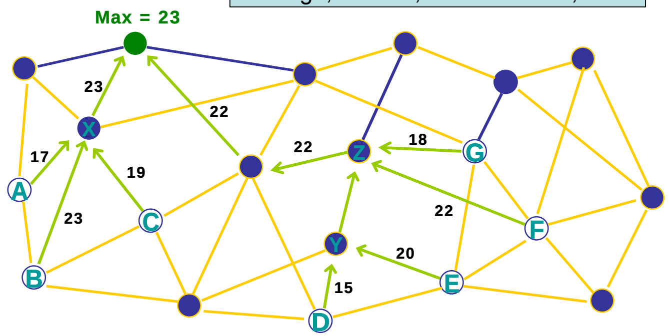
- TinyDB then supports in-network aggregation to speed up communication.

SELECT <aggregates>, <attributes>
[FROM {sensors | <buffer>}]
[WHERE <predicates>]
[GROUP BY <exprs>]
[SAMPLE PERIOD <const> | ONCE]
[INTO <buffer>]
[TRIGGER ACTION <command>]

Data Aggregation: N-to-1 Communication

- SELECT MAX(temp) FROM sensors WHERE node_id < "H".

Average, Median, Count Distinct, ...?!



Selective data aggregation

- In sensor network applications
  - Queries can be frequent
  - Sensor groups are time-varying
  - Events happen in a dynamic fashion
- Option 1: Construct aggregation trees for each group
  - Setting up a good tree incurs communication overhead
- Option 2: Construct a single spanning tree
  - When given a sensor group, simply use the induced tree

Group-Independent (a.k.a. Universal) Spanning Tree

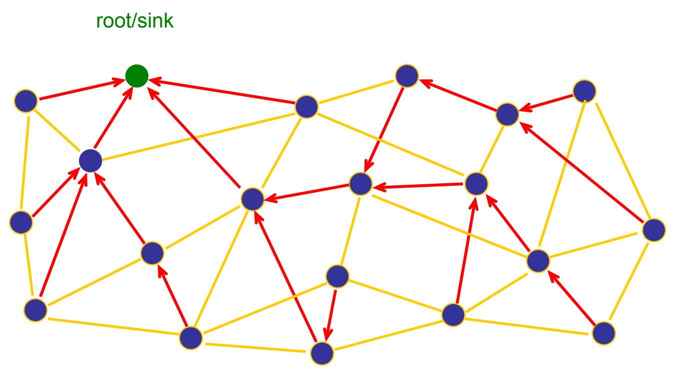
- Given
  - A set of nodes V in the Euclidean plane (or forming a metric space)
  - A root node r ∈ V
  - Define stretch of a universal spanning tree T to be

$$\max_{S\subseteq V} \frac{\text{cost(induced tree of S+r on T)}}{\text{cost(minimum Steiner tree of S+r)}}.$$

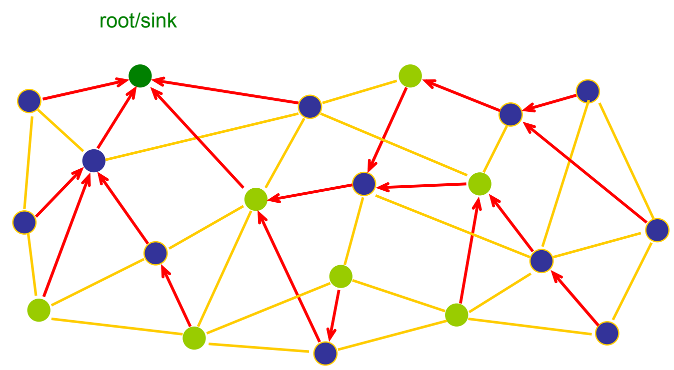
- We're looking for a spanning tree T on V with minimum stretch.

Example

- The red tree is the universal spanning tree. All links cost 1.

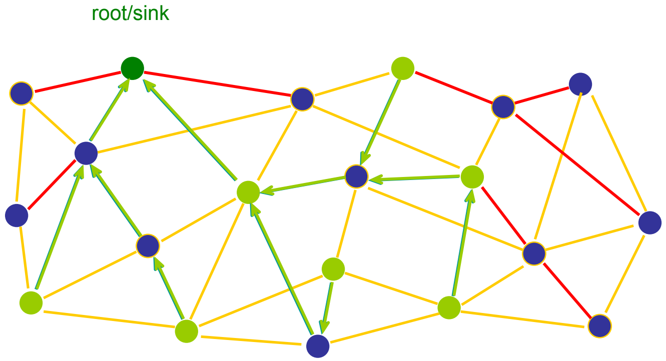


Given the lime subset…



Induced Subtree

- The cost of the induced subtree for this set S is 11. The optimal was 8.

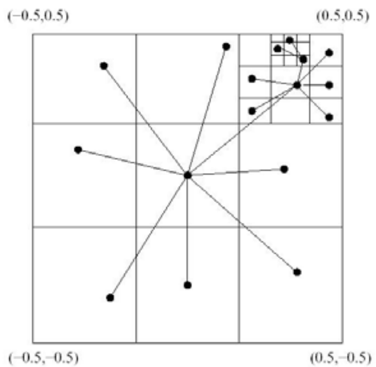


Main results

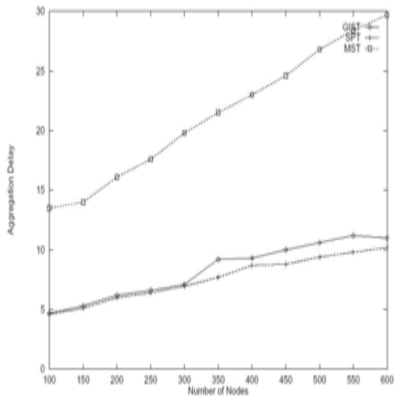
- [Jia, Lin, Noubir, Rajaraman and Sundaram, STOC 2005]
- Theorem 1: (Upper bound)
  For the minimum UST problem on Euclidean plane, an approximation of O(log n) can be achieved within polynomial time.
- Theorem 2: (Lower bound)
  No polynomial time algorithm can approximate the minimum UST problem with stretch better than $\Omega(\log n / \log \log n)$.
- Proofs: Not in this lecture.

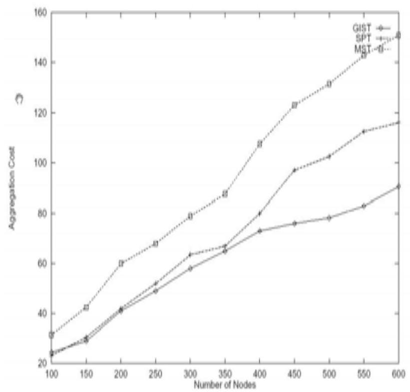
Algorithm sketch

- For the simplest Euclidean case:
- Recursively divide the plane and select random node.
- Results: The induced tree has logarithmic overhead. The aggregation delay is also constant.



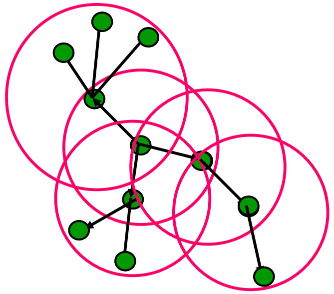
Simulation with random node distribution & random events





Minimum Energy Broadcasting

- First step for data gathering, sort of.
- Given a set of nodes in the plane
- Goal: Broadcast from a source to all nodes
- In a single step, a node may transmit within a range by appropriately adjusting transmission power.
- Energy consumed by a transmission of radius r is proportional to r^{α}, with $\alpha \geq 2$.
- Problem: Compute the sequence of transmission steps that consume minimum total energy, even in a centralized way.



[Rajomohan Rajaraman]

Three natural greedy heuristics

- In a tree, power for each parent node proportional to α'th exponent of distance to farthest child in tree:

- Shortest Paths Tree (SPT)
- Minimum Spanning Tree (MST)
- Broadcasting Incremental Power (BIP)
  - "Node" version of Dijkstra's SPT algorithm
  - Maintains an arborescence rooted at source
  - In each step, add a node that can be reached with minimum increment in total cost.

- Results:
  - NP, not even PTAS, there is a constant approximation. [Clementi, Crescenzi, Penna, Rossi, Vocca, STACS 2001]
  - Analysis of the three heuristics. [Wan, Calinescu, Li, Frieder, Infocom 2001]
  - Optimal MST approximation constant, e.g. [Ambühl, ICALP 2005]

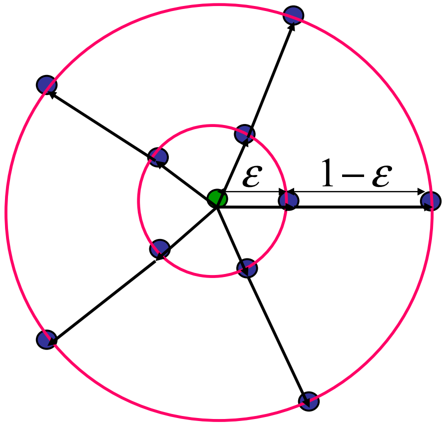
Lower Bound on SPT

- Assume (n-1)/2 nodes per ring

- Total energy of SPT:
  $(n-1)(\varepsilon^{\alpha}+(1-\varepsilon)^{\alpha})/2$

- Better solution:
- Broadcast to all nodes
- Cost 1

- Approximation ratio $\Omega(n)$.



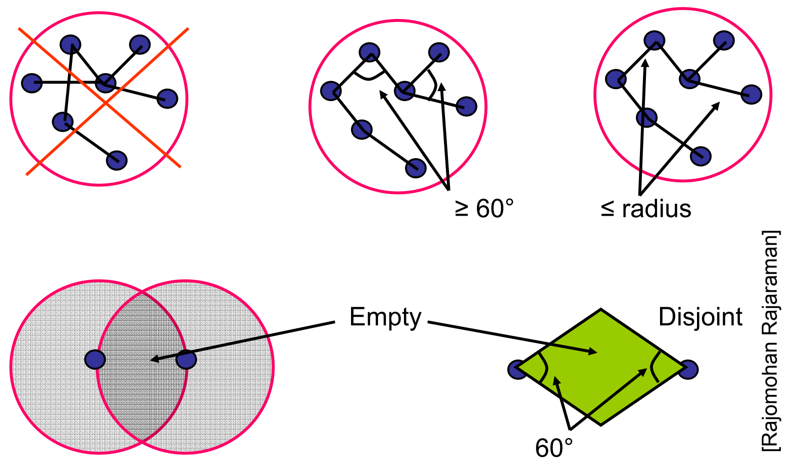
Performance of the MST Heuristic

- Weight of an edge (u,v) equals $d(u,v)^{\alpha}$.

- MST for these weights same as Euclidean MST
  - Weight is an increasing function of distance
  - Follows from correctness of Prim's algorithm

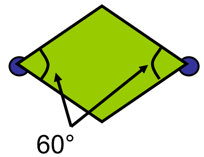
- Upper bound on total MST weight
- Lower bound on optimal broadcast tree

Structural Properties of MST



Upper Bound on Weight of MST

- Assume $\alpha = 2$
- For each edge e, its diamond accounts for an area of exactly $\frac{|e|^2}{2\sqrt{3}}$



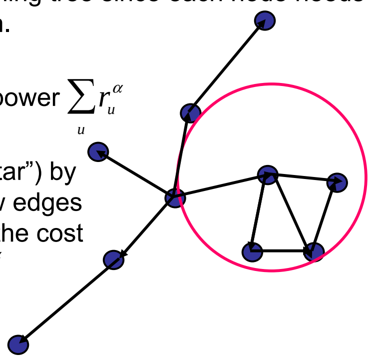
- Diamonds for edges in circle can be slightly outside circle, but not too much: The radius factor is at most $2/\sqrt{3}$, hence the total area accounted for is at most $\pi(2/\sqrt{3})^2 = 4\pi/3$
- Now we can bound the cost of the MST in a unit disk with

$$\text{cost(MST)} \le \sum_e |e|^2 = 2\sqrt{3}\sum_e \frac{|e|^2}{2\sqrt{3}} \le 2\sqrt{3}\frac{4\pi}{3} = \frac{8\pi}{\sqrt{3}} \approx 14.51.$$

- This analysis can be extended to $\alpha > 2$, and improved to 12.

Lower Bound on Optimal and Conclusion of Proof

- Also the optimal algorithm needs a few transmissions. Let $u_0, u_1, \ldots, u_k$ be the nodes which need to transmit, each u_i with radius r_i. These transmissions need to form a spanning tree since each node needs to receive at least one transmission.
- Then the optimal algorithm needs power $\sum_u r_u^\alpha$
- Now replace each transmission ("star") by an MST of the nodes. Since all new edges are part of the transmission circle, the cost of the new graph is at most $12\sum_u r_u^\alpha$
- Since the cost of the global MST is at most the cost of this spanner, the MST is 12-competitive.



Chapter 5
TIME SYNC

EWSN 2006

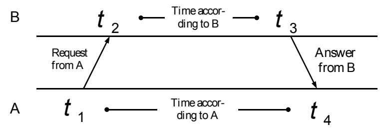
Distributed Computing Group

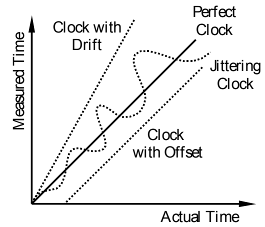
Overview

- Motivation
- Reference-Broadcast Synchronization (RBS)
- Time-sync Protocol for Sensor Networks (TSPN)
- Gradient Clock Synchronization

Motivation

- Time synchronization is essential for many applications
  - Coordination of wake-up and sleeping times
  - TDMA schedules
  - Ordering of sensed events in habitat environments
  - Estimation of position information
  - …
- Scope of a Clock Synchronization Algorithm
  - *Packet delay / latency*
  - *Offset* between clocks
  - *Drift* between clocks





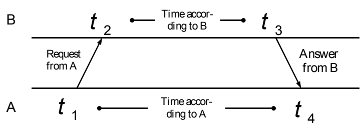
Disturbing Influences on Packet Latency

- Influences
  - Sending Time *S*
  - Medium Access Time *A*
  - Propagation Time $P_{A,B}$
  - Reception Time *R*
- Asymmetric packet delays due to *non-determinism*
- Example: RTT-based synchronization

$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}$$

$$\theta = \frac{(t_2 - (t_1 + \delta)) - (t_4 - (t_3 + \delta))}{2}$$

$$= \frac{(t_2 - t_1) + (t_3 - t_4)}{2}$$



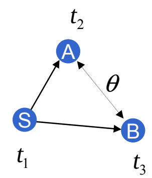
Reference-Broadcast Synchronization (RBS)

- A sender synchronizes a set of receivers with one another
- Point of reference: beacon's arrival time

$$t_2 = t_1 + S_S + A_S + P_{S,A} + R_A$$

$$t_3 = t_1 + S_S + A_S + P_{S,B} + R_B$$

$$\theta = t_2 - t_3 = (P_{S,A} - P_{S,B}) + (R_A - R_B)$$



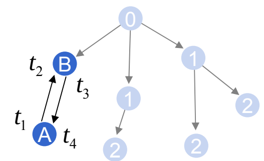
- Only sensitive to the difference in propagation and reception time
- Time stamping at the interrupt time when a beacon is received
- After a beacon is sent, all receivers exchange their reception times to calculate their clock offset
- Post-synchronization possible
- Least-square linear regression to tackle clock drifts

Time-sync Protocol for Sensor Networks (TSPN)

- Traditional sender-receiver synchronization (RTT-based)
- *Initialization phase: Breadth-first-search flooding*
  - Root node at level 0 sends out a *level discovery* packet
  - Receiving nodes which have not yet an assigned level set their level to +1 and start a random timer
  - After the timer is expired, a new level discovery packet will be sent
- *Synchronization phase*
  - Root node issues a *time sync* packet which triggers a random timer at all level 1 nodes
  - After the timer is expired, the node asks its parent for synchronization using a *synchronization pulse*
  - The parent node answers with an *acknowledgement*
  - Thus, the requesting node knows the round trip time and can calculate its clock offset
  - Child nodes receiving a synchronization pulse also start a random timer themselves to trigger their own synchronization

Time-sync Protocol for Sensor Networks (TSPN)

$$t_2 = t_1 + S_A + A_A + P_{A,B} + R_B$$
$$t_4 = t_3 + S_B + A_B + P_{B,A} + R_A$$
$$\theta = \frac{(S_A - S_B) + (A_A - A_B) + (P_{A,B} - P_{B,A}) + (R_B - R_A)}{2}$$



- Time stamping packets at the MAC layer
- In contrast to RBS, the signal propagation time might be negligible
- About “two times” better than RBS
- Again, clock drifts are taken into account using periodical synchronization messages

- Problem: What happens in a ring?!?
  - Two neighbors will have exceptionally badly synchronization

Theoretical Bounds for Clock Synchronization

- Network Model:
  - Each node has a private clock
  - *n* node network, with diameter $\Delta \leq n$.
  - Reliable point-to-point communication with minimal delay μ
  - Jitter ε is the uncertainty in message delay

- Two neighboring nodes u, v cannot distinguish whether message is faster from u to v and slower from v to u, or vice versa. Hence clocks of neighboring nodes can be up to ε off.
- Hence, two nodes at distance Δ might have clocks which are $\varepsilon\Delta$ off.

- This can be achieved by a simple flooding algorithm: Whenever a node receives a new minimum value, it sets its clock to the new value and forwards its new clock value to all its neighbors.

Gradient Clock Synchronization

- It could happen that a clock has to jump back to a much lower value
  - Think again about a ring example, assume that in one leg of the ring messages are forwarded fast all of a sudden.

- Problem: At a node, you don’t want a clock to jump back all of a sudden.
  - You don’t want new events to be registered earlier than older events.
  - Instead, you want your clock always to move forward. Sometimes faster, sometimes slower is OK. But there should be a minimum and a maximum speed.
  - This is called “gradient” clock synchronization in [Fan and Lynch, PODC 2004] .

- In [Fan and Lynch, PODC 2004] it is shown that when logical clocks need to obey minimum/maximum speed rules, the skew of two neighboring clocks can be up to

$$\Omega\left(\frac{\log \Delta}{\log\log \Delta}\right)$$

Chapter 6
CLUSTERING

Distributed
Computing
Group

EWSN 2006

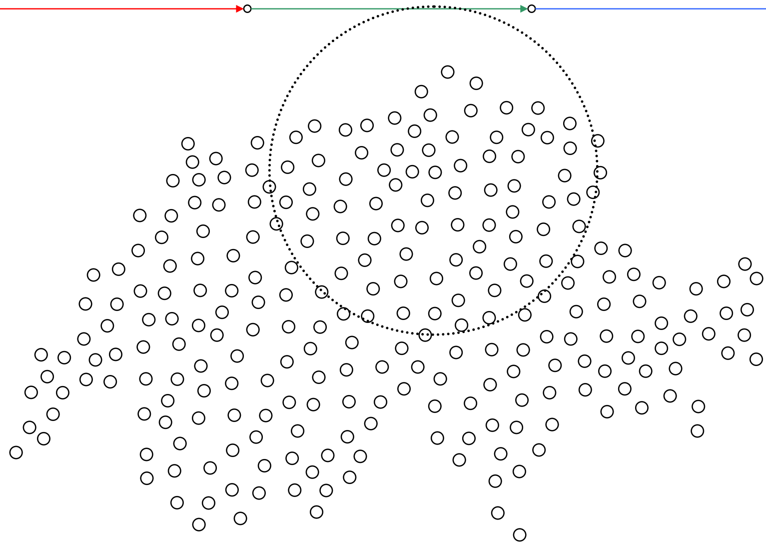
Overview

- Motivation
- Dominating Set
- Connected Dominating Set
- General Algorithms:
  - The “Greedy” Algorithm
  - The “Tree Growing” Algorithm
  - The “Marking” Algorithm
  - The “k-Local” Algorithm
- Algorithms for Special Models:
  - Unit Ball Graphs: The “Largest ID” Algorithm
  - Independence-Bounded Graphs: The “MIS” Algorithm
  - Unstructured Radio Network Model

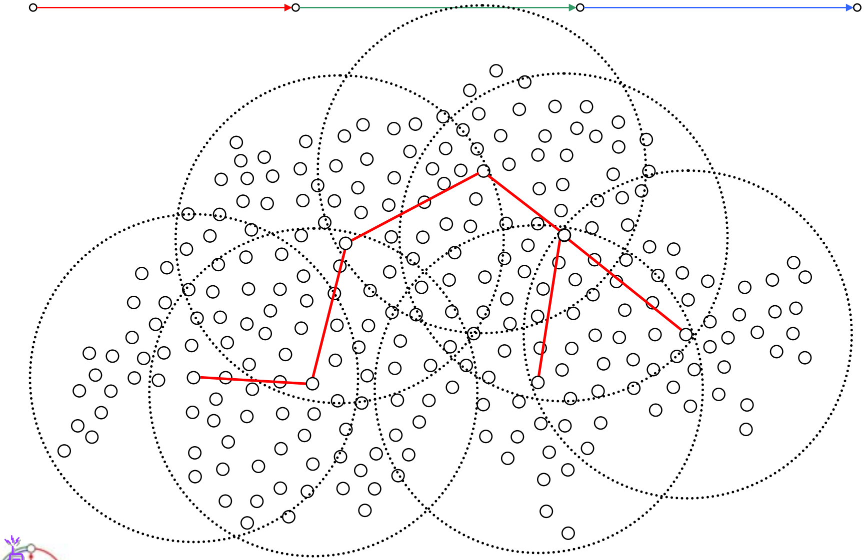
Discussion

- We have seen: 10 Tricks → 2^{10} routing algorithms
- In reality there are almost that many!
- Q: How good are these routing algorithms?!? Any hard results?
- A: Almost none! Method-of-choice is simulation…
- Perkins: “if you simulate three times, you get three different results”
- Flooding is key component of (many) proposed algorithms, including most prominent ones (AODV, DSR)
- At least flooding should be efficient

Finding a Destination by Flooding



Finding a Destination *Efficiently*

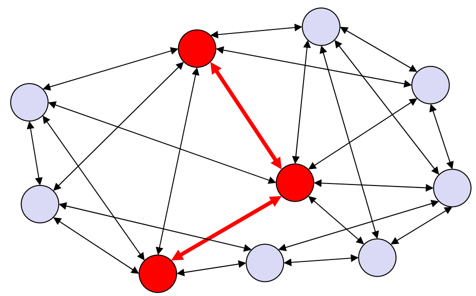


Backbone

- Idea: Some nodes become backbone nodes (gateways). Each node can access and be accessed by at least one backbone node.

- Routing:

1. If source is not a gateway, transmit message to gateway
2. Gateway acts as proxy source and routes message on backbone to gateway of destination.
3. Transmission gateway to destination.



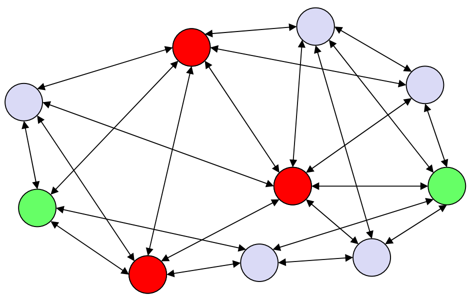
(Connected) Dominating Set

- A Dominating Set DS is a subset of nodes such that each node is either in DS or has a neighbor in DS.

- A Connected Dominating Set CDS is a connected DS, that is, there is a path between any two nodes in CDS that does not use nodes that are not in CDS.

- A CDS is a good choice for a backbone.

- It might be favorable to have few nodes in the CDS. This is known as the Minimum CDS problem



Formal Problem Definition: M(C)DS

- Input: We are given an (arbitrary) undirected graph.

- Output: Find a Minimum (Connected) Dominating Set, that is, a (C)DS with a minimum number of nodes.

- Problems
  - M(C)DS is NP-hard
  - Find a (C)DS that is “close” to minimum (approximation)
  - The solution must be local (global solutions are impractical for mobile ad-hoc network) – topology of graph “far away” should not influence decision who belongs to (C)DS

Greedy Algorithm for Dominating Sets

- Idea: Greedy choose "good" nodes into the dominating set.

- Black nodes are in the DS
- Grey nodes are neighbors of nodes in the DS
- White nodes are not yet dominated, initially all nodes are white.

- Algorithm: Greedily choose a node that colors most white nodes.

- One can show that this gives a log Δ approximation, if Δ is the maximum node degree of the graph. (The proof is similar to the "Tree Growing" proof on 6/13ff.)
- One can also show that there is no polynomial algorithm with better performance unless P≈NP.

CDS: The "too simple tree growing" algorithm

- Idea: start with the root, and then greedily choose a neighbor of the tree that dominates as many as possible new nodes

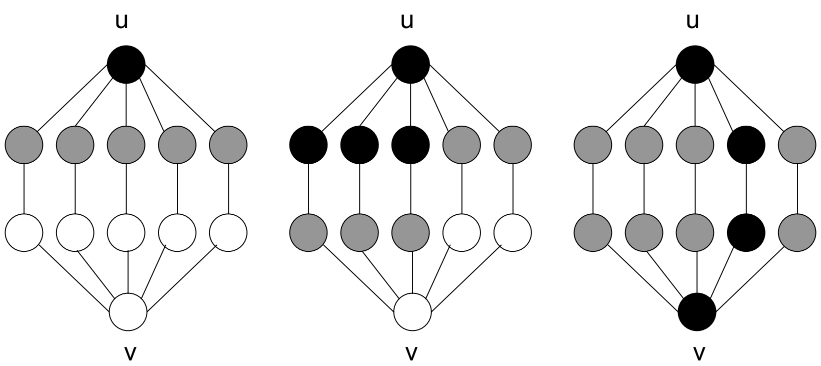
- Black nodes are in the CDS
- Grey nodes are neighbors of nodes in the CDS
- White nodes are not yet dominated, initially all nodes are white.

- Start: Choose a node with maximum degree, and make it the root of the CDS, that is, color it black (and its white neighbors grey).

- Step: Choose a grey node with a maximum number of white neighbors and color it black (and its white neighbors grey).

Example of the "too simple tree growing" algorithm

Graph with 2n+2 nodes; tree growing: |CDS|=n+2; Minimum |CDS|=4

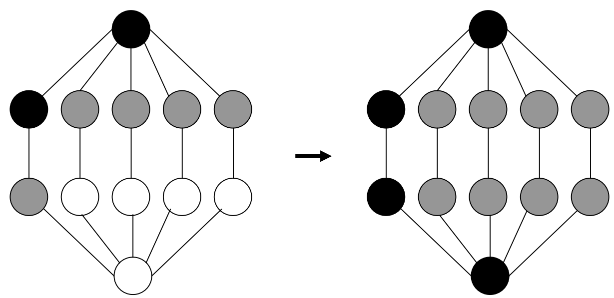


tree growing: start … Minimum CDS

Tree Growing Algorithm

- Idea: Don't scan one but two nodes!

- Alternative step: Choose a grey node and its white neighbor node with a maximum sum of white neighbors and color both black (and their white neighbors grey).



Analysis of the tree growing algorithm

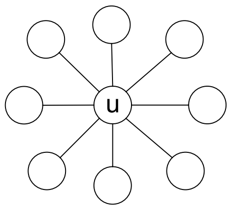
- Theorem: The tree growing algorithm finds a connected set of size $|CDS| \leq 2(1+H(\Delta)) \cdot |DS_{OPT}|$.
- DS_{OPT} is a (not connected) minimum dominating set
- Δ is the maximum node degree in the graph
- H is the harmonic function with $H(n) \approx \log(n)+0.7$
- In other words, the connected dominating set of the tree growing algorithm is at most a $O(\log(\Delta))$ factor worse than an optimum minimum dominating set (which is NP-hard to compute).
- With a lower bound argument (reduction to set cover) one can show that a better approximation factor is impossible, unless P≈NP.

Proof Sketch

- The proof is done with amortized analysis.
- Let S_u be the set of nodes dominated by $u \in DS_{OPT}$, or u itself. If a node is dominated by more than one node, we put it in one of the sets.
- We charge the nodes in the graph for each node we color black. In particular we charge all the newly colored grey nodes. Since we color a node grey at most once, it is charged at most once.
- We show that the total charge on the vertices in an S_u is at most $2(1+H(\Delta))$, for any u.

Charge on S_u

- Initially $|S_u| = u_0$.
- Whenever we color some nodes of S_u, we call this a step.
- The number of white nodes in S_u after step i is u_i.
- After step k there are no more white nodes in S_u.



- In the first step $u_0 - u_1$ nodes are colored (grey or black). Each vertex gets a charge of at most $2/(u_0 - u_1)$.
- After the first step, node u becomes eligible to be colored (as part of a pair with one of the grey nodes in S_u). If u is not chosen in step i (with a potential to paint u_i nodes grey), then we have found a better (pair of) node. That is, the charge to any of the new grey nodes in step i in S_u is at most $2/u_i$.

Adding up the charges in S_u

$$C \leq \frac{2}{u_0 - u_1}(u_0 - u_1) + \sum_{i=1}^{k-1} \frac{2}{u_i}(u_i - u_{i+1})$$

$$= 2 + 2\sum_{i=1}^{k-1} \frac{u_i - u_{i+1}}{u_i}$$

$$\leq 2 + 2\sum_{i=1}^{k-1} \left(H(u_i) - H(u_{i+1})\right)$$

$$= 2+2(H(u_1)-H(u_k)) = 2(1+H(u_1)) = 2(1+H(\Delta))$$

Discussion of the tree growing algorithm

- We have an extremely simple algorithm that is asymptotically optimal unless P≈NP. And even the constants are small.
- Are we happy?
- Not really. How do we implement this algorithm in a real mobile network? How do we figure out where the best grey/white pair of nodes is? How slow is this algorithm in a distributed setting?
- We need a fully distributed algorithm. Nodes should only consider local information.

The Marking Algorithm

- Idea: The connected dominating set CDS consists of the nodes that have two neighbors that are not neighboring.

1. Each node u compiles the set of neighbors N(u)
2. Each node u transmits N(u), and receives N(v) from all its neighbors
3. If node u has two neighbors v,w and w is not in N(v) (and since the graph is undirected v is not in N(w)), then u marks itself being in the set CDS.

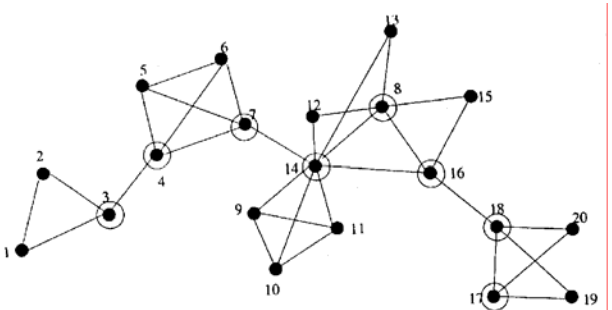
\+ Completely local; only exchange N(u) with all neighbors

\+ Each node sends only 1 message, and receives at most Δ

\+ Messages have size $O(\Delta)$

- Is the marking algorithm really producing a connected dominating set? How good is the set?

Example for the Marking Algorithm



[J. Wu]

Correctness of Marking Algorithm

- We assume that the input graph G is connected but not complete.
- Note: If G was complete then constructing a CDS would not make sense. Note that in a complete graph, no node would be marked.
- We show:

  The set of marked nodes CDS is
  a) a dominating set
  b) connected
  c) a shortest path in G between two nodes of the CDS is in CDS

Proof of a) dominating set

- Proof: Assume for the sake of contradiction that node u is a node that is not in the dominating set, and also not dominated. Since no neighbor of u is in the dominating set, the nodes $N^+(u) := u \cup N(u)$ form:

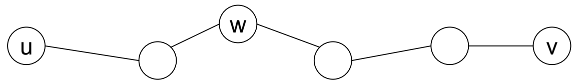
- a complete graph
  - if there are two nodes in N(u) that are not connected, u must be in the dominating set by definition
- no node $v \in N(u)$ has a neighbor outside N(u)
  - or, also by definition, the node v is in the dominating set

- Since the graph G is connected it only consists of the complete graph $N^+(u)$. We precluded this in the assumptions, therefore we have a contradiction

Proof of b) connected, c) shortest path in CDS

- Proof: Let p be any shortest path between the two nodes u and v, with $u,v \in$ CDS.

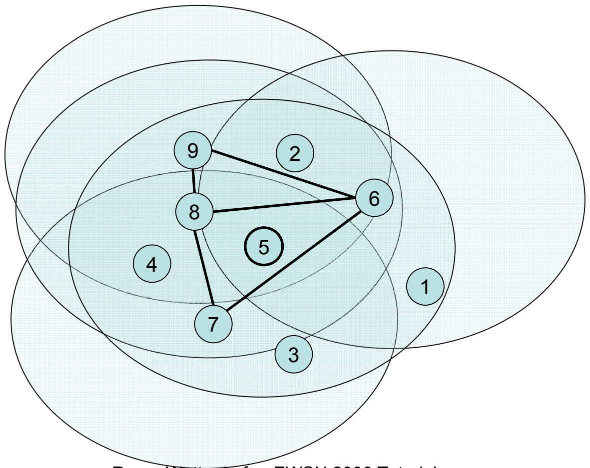
- Assume for the sake of contradiction that there is a node w on this shortest path that is not in the connected dominating set.



- Then the two neighbors of w must be connected, which gives us a shorter path. This is a contradiction.

Improved Marking Algorithm

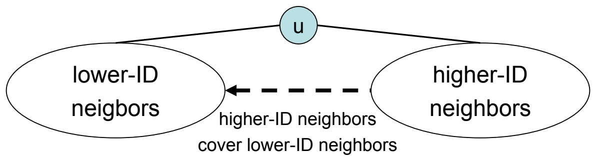
- If neighbors with larger ID are connected and cover all other neighbors, then don't join CDS, else join CDS



Correctness of Improved Marking Algorithm

- Theorem: Algorithm computes a CDS S

- Proof (by induction of node IDs):
  - assume that initially all nodes are in S
  - look at nodes u in increasing ID order and remove from S if higher-ID neighbors of u are connected
  - S remains a DS at all times: (assume that u is removed from S)



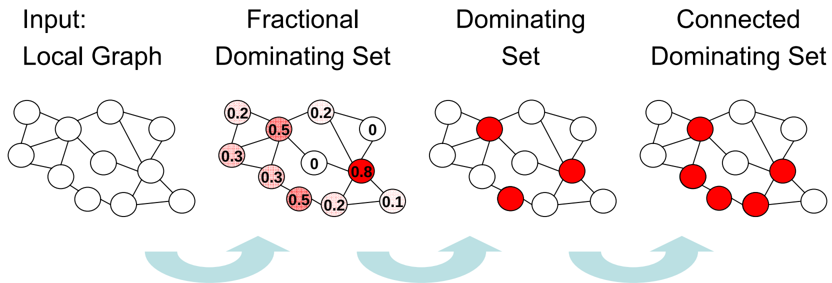
  - S remains connected:
    replace connection v-u-v' by $v\text{-}n_1,\ldots,n_k\text{-}v'$ (n_i: higher-ID neighbors of u)

Quality of the (Improved) Marking Algorithm

- Given an Euclidean chain of n homogeneous nodes
- The transmission range of each node is such that it is connected to the k left and right neighbors, the id's of the nodes are ascending.

- An optimal algorithm (and also the tree growing algorithm) puts every k'th node into the CDS. Thus $|CDS_{OPT}| \approx n/k$; with $k = n/c$ for some positive constant c we have $|CDS_{OPT}| = O(1)$.
- The marking algorithm (also the improved version) does mark all the nodes (except the k leftmost ones). Thus $|CDS_{Marking}| = n – k$; with $k = n/c$ we have $|CDS_{Marking}| = \Omega(n)$.
- The worst-case quality of the marking algorithm is worst-case! ☺

Algorithm Overview



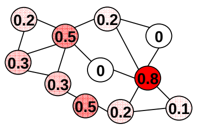
Phase A: Distributed linear program
rel. high degree gives high value

Phase B: Probabilistic algorithm

Phase C: Connect DS by "tree" of "bridges"

Phase A is a Distributed Linear Program

- Nodes 1, …, *n*: Each node *u* has variable x_u with $x_u \geq 0$
- Sum of *x*-values in each neighborhood at least 1 (local)
- Minimize sum of all *x*-values (global)



0.5+0.3+0.3+0.2+0.2+0 = 1.5 ≥ 1

Linear Program

$$\min \sum_{i=1}^{n} x_i$$

$$\text{subject to} \quad N \cdot \underline{x} \geq \underline{1}, \quad \underline{x} \geq \underline{0}$$

Adjacency matrix with 1's in diagonal

- Linear Programs can be solved optimally in polynomial time
- But not in a distributed fashion! That's what we need here…

Phase A Algorithm

LP Approximation Algorithm for Primal Node $v_i^{(p)}$:

```
x_i := 0;
for e_p := k_p − 2 to −f − 1 by −1 do
    for 1 to h do
        (* γ_i := (c_max/c_i) Σ_j a_ji r_j *)
        for e_d := k_d − 1 to 0 by −1 do
            γ̃_i := (c_max/c_i) Σ_j a_ji r̃_j;
            if γ̃_i ≥ 1/Γ_p^{e_p/k_p} then
                x_i^+ := 1/Γ_d^{e_d/k_d}; x_i := x_i + x_i^+;
            fi;
            send x_i^+, γ̃_i to dual neighbors;




            receive r̃_j from dual neighbors
        od;

        receive r_j from dual neighbors
    od
od;
x_i := x_i / min_{j∈N_i^(p)} Σ_ℓ a_jℓ x_ℓ
```

LP Approximation Algorithm for Dual Node $v_i^{(d)}$:

```
y_i := y_i^+ := w_i := f_i := 0; r_i := 1;
for e_p := k_p − 2 to −f − 1 by −1 do
    for 1 to h do
        r̃_i := r_i;
        for e_d := k_d − 1 to 0 by −1 do




            receive x_j^+, γ̃_j fro
            y_i^+ := y_i^+ + r̃_i Σ_j
            w_i^+ := Σ_j a_ij x_j^+;
            w_i := w_i + w_i^+; f_i
            if w_i ≥ 1 then r̃_i :
            send r̃_i to primal n
        od;
        increase_duals();
        send r_i to primal nei
    od
od;
y_i := y_i / max_{j∈N_i^(d)} (1/c_j) Σ
```

procedure increase_duals():

```
if w_i ≥ 1 then
    if f_i ≥ f then
        y_i := y_i + y_i^+; y_i^+ := 0;
        r_i := 0; w_i := 0
    else if w_i ≥ 2 then
        y_i := y_i + y_i^+; y_i^+ := 0;
        r_i := r_i/Γ_p^{⌊w_i⌋/k_p}
    else
        λ := max{Γ_d^{1/k_d}, Γ_p^{1/k_p}};
        y_i := y_i + min{y_i^+, r_i λ/Γ_p^{e_p/k_p}};
        y_i^+ := y_i^+ − min{y_i^+, r_i λ/Γ_p^{e_p/k_p}};
        r_i := r_i/Γ_p^{1/k_p}
    fi;
    w_i := w_i − ⌊w_i⌋
fi
```

Result after Phase A

- Distributed Approximation for Linear Program
- Instead of the optimal values x_i^* at nodes, nodes have $x_i^{(\alpha)}$, with

$$\sum_{i=1}^{n} x_i^{(\alpha)} \leq \alpha \cdot \sum_{i=1}^{n} x_i^*$$

- The value of α depends on the number of rounds k (the locality)

$$\alpha \leq (\Delta + 1)^{c/\sqrt{k}}$$

- The analysis is rather intricate… ☺

Phase B Algorithm

Each node applies the following algorithm:

1. *Calculate* $\delta_i^{(2)}$ *(= maximum degree of neighbors in distance 2)*
2. *Become a dominator (i.e. go to the dominating set) with probability*

$$p_i := \min\{1, \underbrace{x_i^{(\alpha)}}_{\text{From phase A}} \cdot \ln(\underbrace{\delta_i^{(2)}}_{\text{Highest degree in distance 2}} + 1)\}$$

3. *Send status (dominator or not) to all neighbors*
4. *If no neighbor is a dominator, become a dominator yourself*

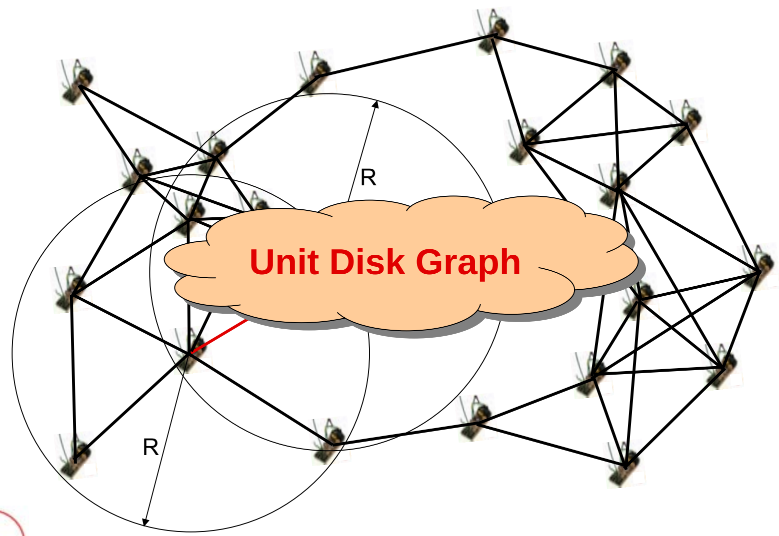
Result after Phase B

- Randomized rounding technique
- Expected number of nodes joining the dominating set in step 2 is bounded by $\alpha \log(\Delta+1) \cdot |DS_{OPT}|$.
- Expected number of nodes joining the dominating set in step 4 is bounded by $|DS_{OPT}|$.

Theorem: $E[|DS|] = O\left((\Delta+1)^{c/\sqrt{k}} \log \Delta \cdot |DS_{OPT}|\right)$

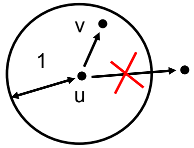
- Phase C → essentially the same result for CDS

A better algorithm?

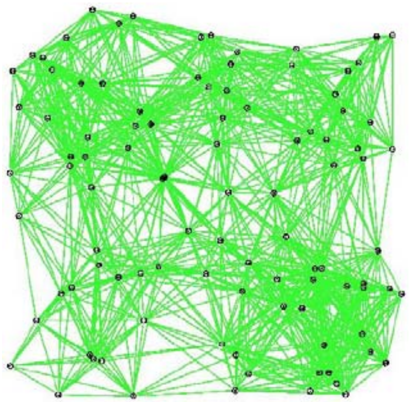


Better and faster algorithm

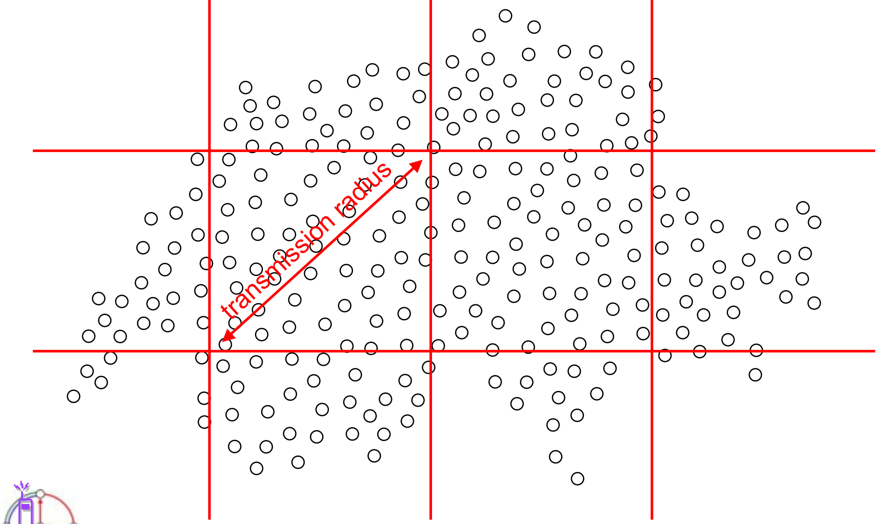
- Assume that graph is a unit disk graph (UDG)



- Assume that nodes know their positions (GPS)



Then…



Grid Algorithm

1. Beacon your position
2. If, in your virtual grid cell, you are the node closest to the center of the cell, then join the DS, else do not join.
3. That's it.

- 1 transmission per node, O(1) approximation.
- If you have mobility, then simply "loop" through algorithm, as fast as your application/mobility wants you to.

Comparison

<u>k-local algorithm</u>	<u>Grid algorithm</u>
• Algorithm computes DS	• Algorithm computes DS
• k^2+O(1) transmissions/node	• 1 transmission/node
• O($\Delta^{O(1)/k}$ log Δ) approximation	• O(1) approximation
• General graph	• Unit disk graph (UDG)
• No position information	• Position information (GPS)

The model determines the distributed complexity of clustering

Let's talk about models…

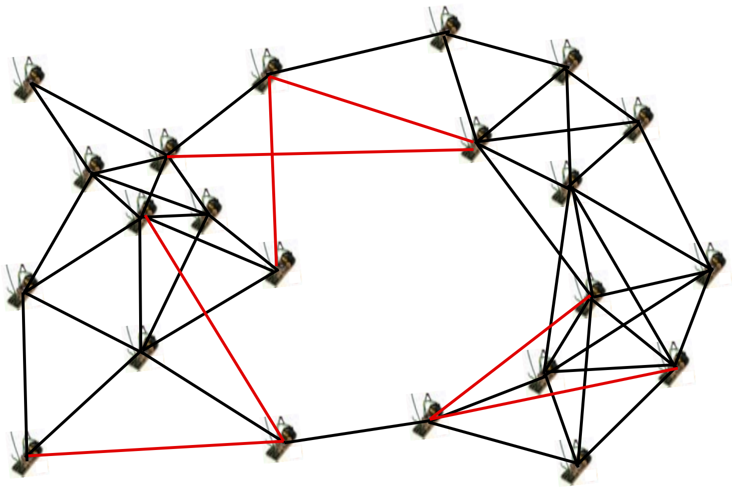
- General Graph
- Captures obstacles
- Captures directional radios
- Often too pessimistic

- UDG & GPS
- UDG is not realistic
- GPS not always available
  - Indoors
- 2D → 3D?
- Often too optimistic

too pessimistic too optimistic

Let's look at models in between these extremes!

Real Networks



Models

General Graph

UDG No GPS

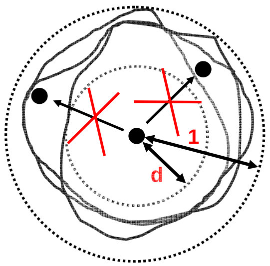
UDG GPS

too pessimistic too optimistic

Bounded Independence

Unit Ball Graph

Quasi UDG



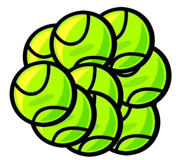
Unit Ball Graphs

- ∃ metric (V,d) describing distances between nodes u,v ∈ V

  such that: $d(u,v) \leq 1 : (u,v) \in E$
  $d(u,v) \geq 1 : (u,v) \notin E$

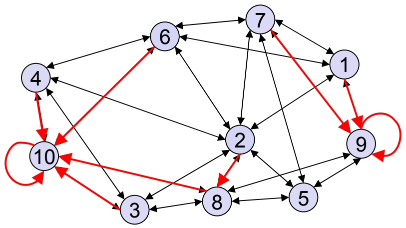
- Assume that doubling dimension of metric is constant
  - Doubling dimension: log(#balls of radius r/2 to cover ball of radius r)

UBG based on underlying doubling metric.



The "Largest-ID" Algorithm

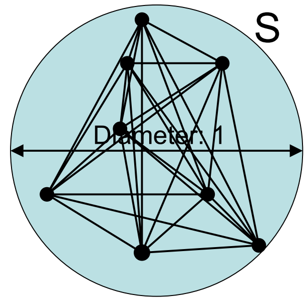
- All nodes have unique IDs, chosen at random.
- Algorithm for each node:
  1. Send ID to all neighbors
  2. Tell node with largest ID in neighborhood that it has to join the DS
- Algorithm computes a DS in 2 rounds (extremely local!)



"Largest ID" Algorithm, Analysis I

- To simplify analysis: assume graph is UDG
  (same analysis works for UBG based on doubling metric)
- We look at a disk S of diameter 1:

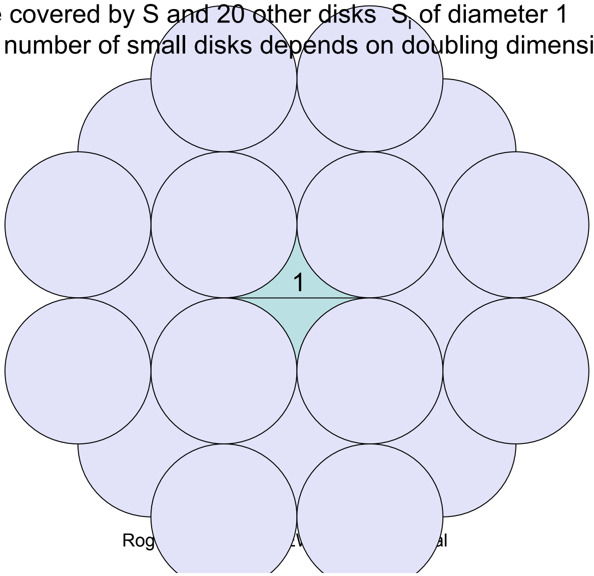
Nodes inside S have distance at most 1.
→ they form a clique



How many nodes in S are selected for the DS?

"Largert ID" Algorithm, Analysis II

- Nodes which select nodes in S are in disk of radius 3/2 which can be covered by S and 20 other disks S_i of diameter 1
  (UBG: number of small disks depends on doubling dimension)



"Largest ID" Algorithm: Analysis III

- How many nodes in S are chosen by nodes in a disk S_i?
- x = # of nodes in S, y = # of nodes in S_i:
- A node u∈S is only chosen by a node in S_i if $\mathrm{ID}(u) > \max_{v \in S_i}\{\mathrm{ID}(v)\}$
  (all nodes in S_i see each other).
- The probability for this is: $\frac{1}{1+y}$
- Therefore, the expected number of nodes in S chosen by nodes in S_i is at most:

$$\min\left\{y, \frac{x}{1+y}\right\}$$

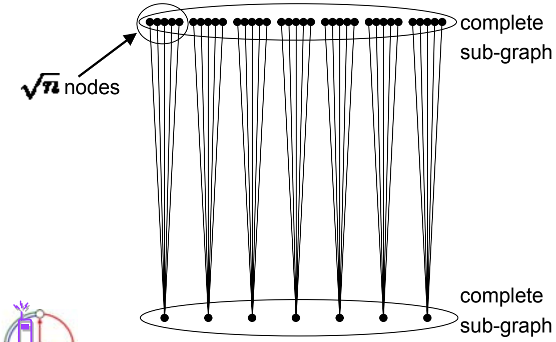
Because at most y nodes in S_i can choose nodes in S
and because of linearity of expectation.

“Largest ID” Algorithm, Analysis IV

- From x≤n and y≤n, it follows that: $\min\left\{y, \frac{x}{1+y}\right\} \le \sqrt{n}$
- Hence, in expectation the DS contains at most $20\sqrt{n}$ nodes per disk with diameter 1.
- An optimal algorithm needs to choose at least 1 node in the disk with radius 1 around any node.
- This disk can be covered by a constant (9) number of disks of diameter 1.
- The algorithm chooses at most $O(\sqrt{n})$ times more disks than an optimal one

“Largest ID” Algorithm, Remarks

- For typical settings, the “Largest ID” algorithm produces very good dominating sets (also for non-UDGs)
- There are UDGs where the “Largest ID” algorithm computes an $\Theta(\sqrt{n})$-approximation (analysis is tight).



Optimal DS: size 2

“Largest ID” alg:

- bottom nodes choose top nodes with probability≈1/2
- 1 node every 2nd group
- $\Theta(\sqrt{n})$ nodes

Iterative “Largest ID” Algorithm

- Assume that nodes know the distances to their neighbors:

  all nodes are active;
  for i := k to 1 do
  ∀ act. nodes: select act. node with largest ID in dist. ≤ $1/2^i$;
  selected nodes remain active
  od;
  DS = set of active nodes

- Set of active nodes is always a DS (computing CDS also possible)
- Number of rounds: k
- Approximation ratio $n^{(1/2^k)}$
- For k=O(loglog n), approximation ratio = O(1)

Iterative “Largest ID” Algorithm, Remarks

- Possible to do everything in O(1) rounds
  (messages get larger, local computations more complicated)
- If we slightly change the algorithm such that largest radius is 1/4:
  - Sufficient to know IDs of all neighbors, distances to neighbors, and distances between adjacent neighbors
  - Every node can then locally simulate relevant part of algorithm to find out whether or not to join DS

Doubling UBG: O(1) approximation in O(1) rounds

Models

General Graph

UDG No GPS

UDG GPS

too pessimistic **too optimistic**

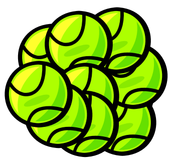
Bounded Independence

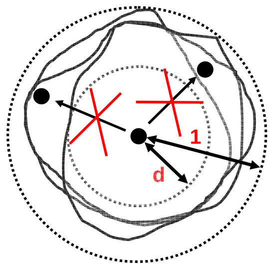
Unit Ball Graph

Quasi UDG

Number of independent neighbors is bounded (UDG: 5)

In a doubling metric:





Real Networks

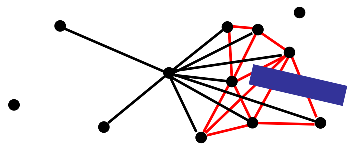
Wireless Networks are not unit disk graphs, but:

- No links between far-away nodes
- Close nodes tend to be connected
- In particular: Densely covered area → many connections

Bounded Independence:
Bounded neighborhoods have bounded independent sets

Bounded Independence

- Def.: A graph G has bounded independence if there is a function $f(r)$ such that every r-neighborhood in G contains at most $f(r)$ independent nodes.
  - Note: $f(r)$ does not depend on size of the graph!
  - Polynomially Bounded Independence: $f(r) = poly(r)$, e.g. $O(r^3)$



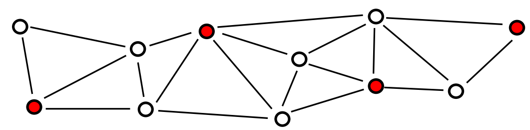
f(1) = 6

1) A node can have many neighbors
2) But not all of them can be independent!
3) Can model obstacles, walls, ...

- Definition includes:
  - (Quasi) Unit Disk Graphs, Doubling Unit Ball Graphs
  - Coverage Area Graphs, Bounded Disk Graphs, ...

Maximal Independent Set I

- Maximal Independent Set (MIS):
  (non-extendable set of pair-wise non-adjacent nodes)



- An MIS is also a dominating set:
  - assume that there is a node v which is not dominated
  - v∉MIS, (u,v)∈E → u∉MIS
  - add v to MIS

Maximal Independent Set II

- Lemma:

  On independence-bounded graphs: **$|MIS| \le O(1) \cdot |DS_{OPT}|$**

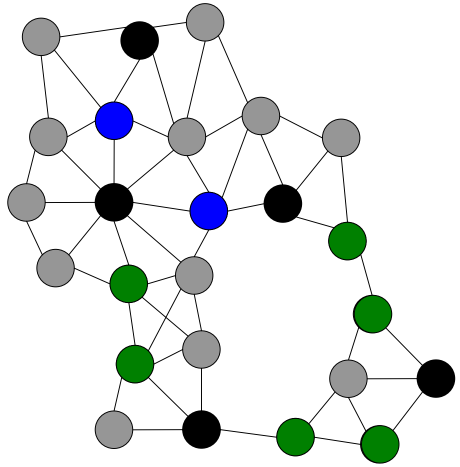
- Proof:
  1. Assign every MIS node to an adjacent node of DS_{OPT}
  2. $u \in DS_{OPT}$ has at most f(1) neighbors $v \in MIS$
  3. At most f(1) MIS nodes assigned to every node of DS_{OPT}

  → $|MIS| \le f(1) \cdot |DS_{OPT}|$

- Time to compute MIS on independence-bounded graphs:

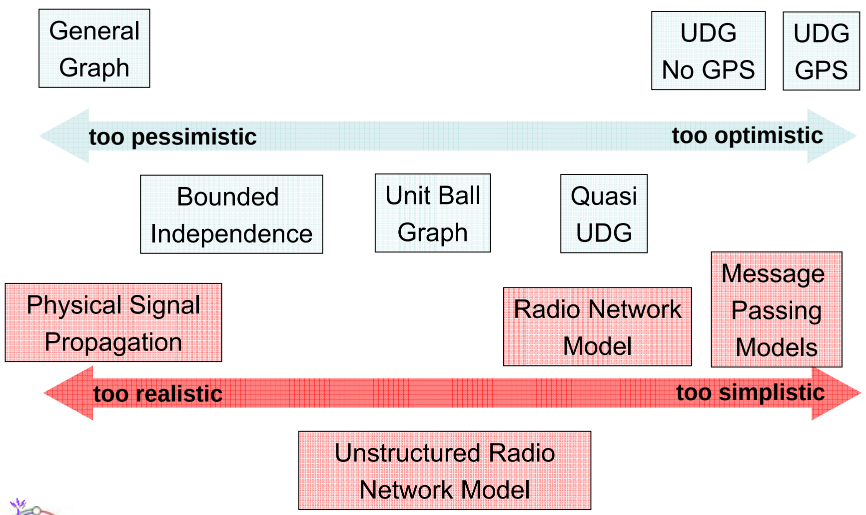
$$\boxed{O(\log \Delta \cdot \log^* n)}$$

MIS (DS) → CDS

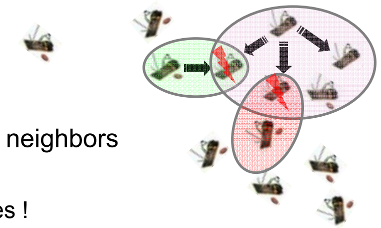


- MIS gives a dominating set.
- But it is not connected.
- Connect any two MIS nodes which can be connected by one additional node.
- Connect unconnected MIS nodes which can be conn. by two additional nodes.
- This gives a CDS!
- #2-hop connectors$\le$f(2)·|MIS|
  #3-hop connectors$\le$2f(3)·|MIS|
- **|CDS| = O(|MIS|)**

Models



Unstructured Radio Network Model



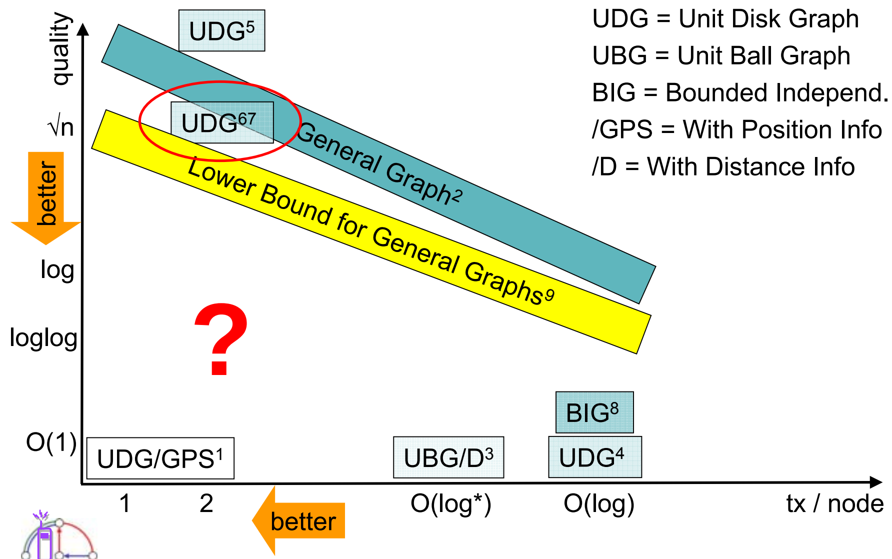
- Multi-Hop
- No collision detection
  - Not even at the sender!
- No knowledge about (the number of) neighbors
- Asynchronous Wake-Up
  - Nodes are not woken up by messages !
- Unit Disk Graph (UDG) to model wireless multi-hop network
  - Two nodes can communicate iff Euclidean distance is at most 1
- Upper bound n for number of nodes in network is known
  - This is necessary due to $\Omega(n / \log n)$ lower bound
    [Jurdzinski, Stachowiak, ISAAC 2002]

Unstructured Radio Network Model

- Can MDS and MIS be solved efficiently in such a harsh model?

There is a MIS algorithm with running time $O(\log^2 n)$ with high probability.

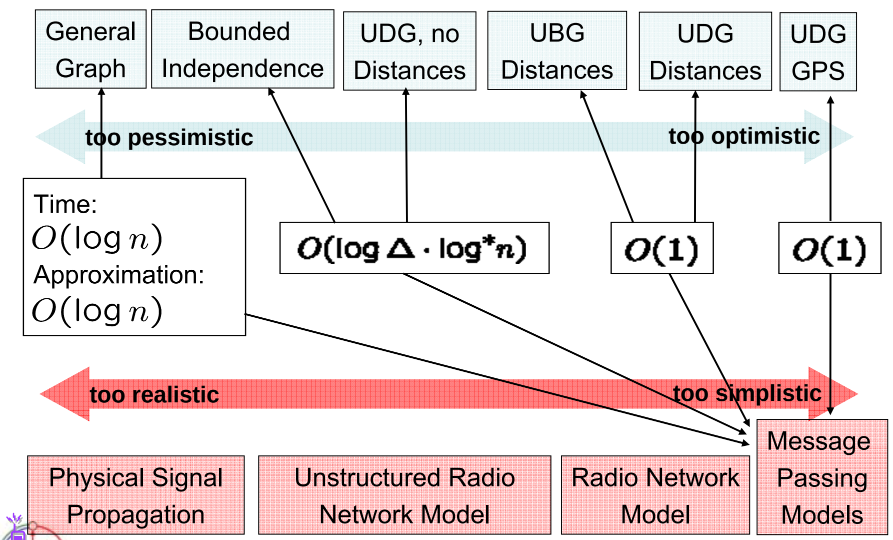
Result Overview



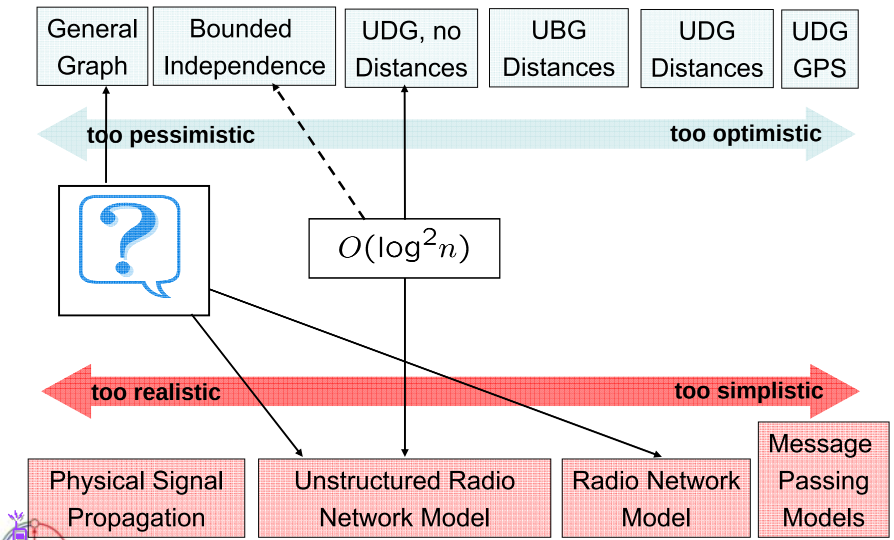
References

1. Folk theorem, e.g. Kuhn, Wattenhofer, Zhang, Zollinger, PODC 2003
2. Kuhn, Wattenhofer, PODC 2003
   - CDS improvement by Dubhashi et al, SODA 2003
   - Improved version: Kuhn, Moscibroda, Wattenhofer, SODA 2006
3. Kuhn, Moscibroda, Wattenhofer, PODC 2005
4. Alzoubi, Wan, Frieder, MobiHoc 2002
5. Wu and Li, DIALM 1999
6. Gao, Guibas, Hershberger, Zhang, Zhu, SCG 2001
7. Wu and Li (several papers, improved here)
8. Kuhn, Moscibroda,Nieberg, Wattenhofer, DISC 2005
9. Kuhn, Moscibroda, Wattenhofer, PODC 2004

Summary Dominating Set I



Summary Dominating Set II



Chapter 7
INTERFERENCE

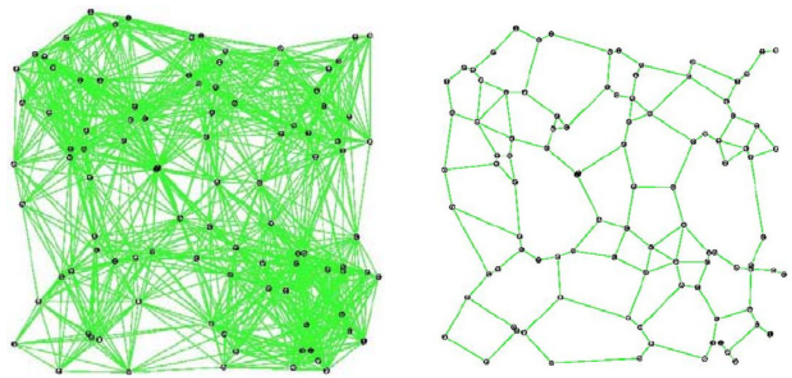
Distributed Computing Group

EWSN 2006

Overview – Topology Control

- Gabriel Graph et al.
- XTC
- Interference
- SINR & Scheduling Complexity

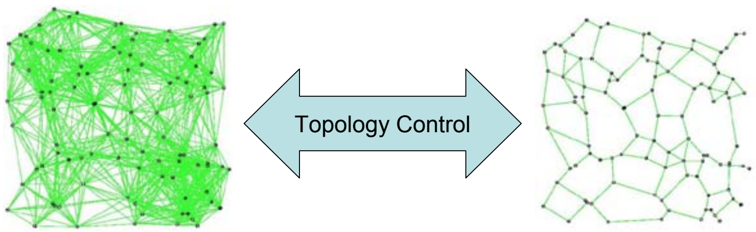
Topology Control



- Drop long-range neighbors: Reduces interference and energy!
- But still stay connected (or even spanner)

Topology Control as a Trade-Off

Sometimes also clustering, Dominating Set construction (See later)



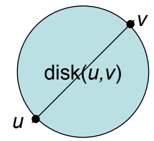
Network Connectivity
Spanner Property

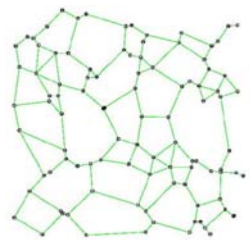
$d(u,v) \cdot t \geq d_{TC}(u,v)$

Conserve Energy
Reduce Interference
Sparse Graph, Low Degree
Planarity
Symmetric Links
Less Dynamics

Gabriel Graph

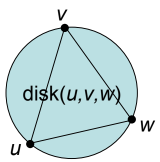
- Let disk(*u,v*) be a disk with diameter (*u,v*) that is determined by the two points *u*,*v*.
- The Gabriel Graph GG(*V*) is defined as an undirected graph (with *E* being a set of undirected edges). There is an edge between two nodes *u*,*v* iff the disk(*u,v*) including boundary contains no other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

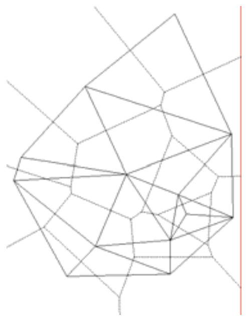




Delaunay Triangulation

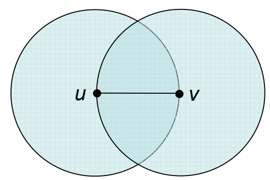
- Let disk(*u,v,w*) be a disk defined by the three points *u*,*v*,*w*.
- The Delaunay Triangulation (Graph) DT(*V*) is defined as an undirected graph (with *E* being a set of undirected edges). There is a triangle of edges between three nodes *u*,*v*,*w* iff the disk(*u,v,w*) contains no other points.
- 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,…,t) on the DT is within a constant factor of the s-t distance.



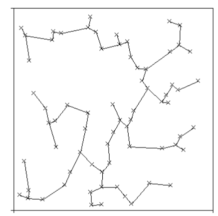


Other planar graphs

- Relative Neighborhood Graph 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 MST(V)
- A subset of *E* of *G* of minimum weight which forms a tree on *V*.

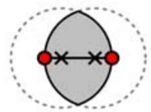


Properties of planar graphs

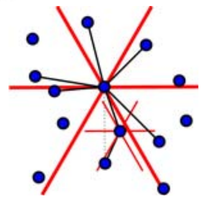
- Theorem 1:
  $\mathrm{MST}(V) \subseteq \mathrm{RNG}(V) \subseteq \mathrm{GG}(V) \subseteq \mathrm{DT}(V)$
- Corollary:
  Since the MST(V) is connected and the DT(V) is planar, all the planar graphs in Theorem 1 are connected and planar.
- Theorem 2:
  The Gabriel Graph contains the Minimum Energy Path (for any path loss exponent $\alpha \geq 2$)
- Corollary:
  $\mathrm{GG}(V) \cap \mathrm{UDG}(V)$ contains the Minimum Energy Path in UDG(V)

More examples

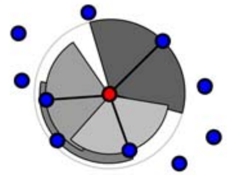
- β-Skeleton
  – Generalizing Gabriel (β = 1) and Relative Neighborhood (β = 2) Graph



- Yao-Graph
  – Each node partitions directions in k cones and then connects to the closest node in each cone



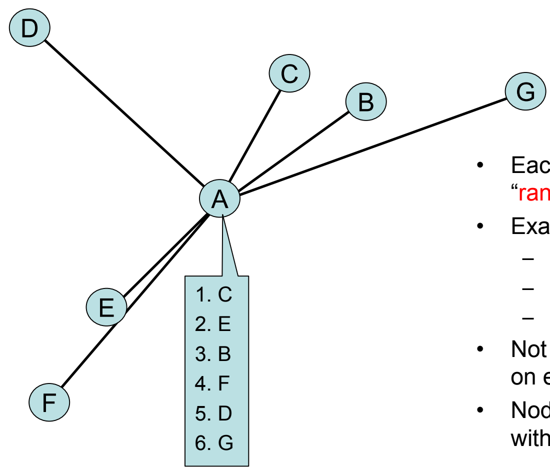
- Cone-Based Graph
  – Dynamic version of the Yao Graph. Neighbors are visited in order of their distance, and used only if they cover not yet covered angle



XTC: Lightweight Topology Control

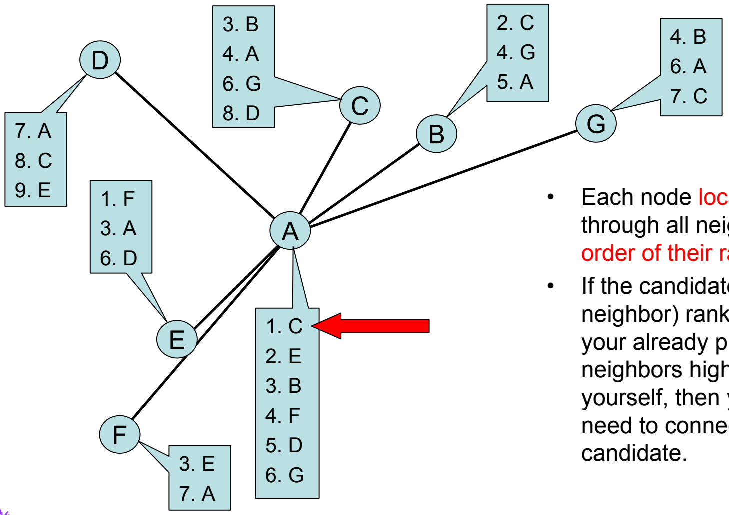
- Topology Control commonly assumes that the node positions are known.
- What if we do not have access to position information?
- XTC algorithm
- XTC analysis
  – Worst case
  – Average case

XTC: lightweight topology control without geometry



- Each node produces "ranking" of neighbors.
- Examples
  – Distance (closest)
  – Energy (lowest)
  – Link quality (best)
- Not necessarily depending on explicit positions
- Nodes exchange rankings with neighbors

XTC Algorithm (Part 2)



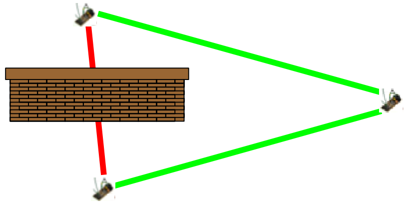
- Each node locally goes through all neighbors in order of their ranking
- If the candidate (current neighbor) ranks any of your already processed neighbors higher than yourself, then you do not need to connect to the candidate.

XTC Analysis (Part 1)

- Symmetry: A node u wants a node v as a neighbor if and only if v wants u.
- Proof:
  - Assume 1) $u \rightarrow v$ and 2) $u \not\leftarrow v$
  - Assumption 2) $\Rightarrow \exists w$: $\underbrace{\text{(i) } w \prec_v u \text{ and (ii) } w \prec_u v}_{\text{Contradicts Assumption 1)}}$

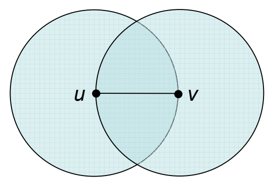
XTC Analysis (Part 1)

- Symmetry: A node u wants a node v as a neighbor if and only if v wants u.
- Connectivity: If two nodes are connected originally, they will stay so (provided that rankings are based on symmetric link-weights).
- If the ranking is energy or link quality based, then XTC will choose a topology that routes around walls and obstacles.

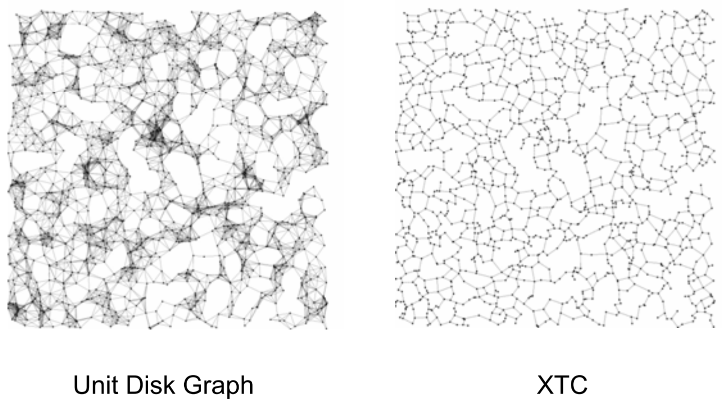


XTC Analysis (Part 2)

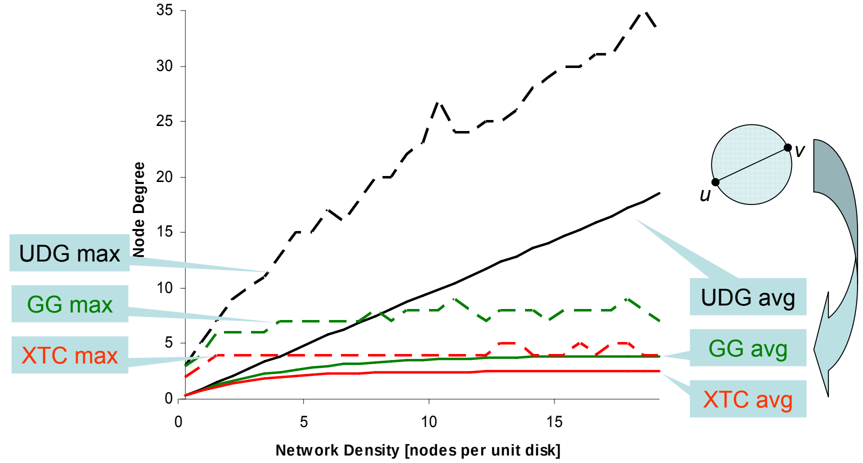
- If the given graph is a Unit Disk Graph (no obstacles, nodes homogeneous, but not necessarily uniformly distributed), then …
- The degree of each node is at most 6.
- The topology is planar.
- The graph is a subgraph of the RNG.
- Relative Neighborhood Graph RNG(V):
- An edge e = (u,v) is in the RNG(V) iff there is no node w with (u,w) < (u,v) and (v,w) < (u,v).



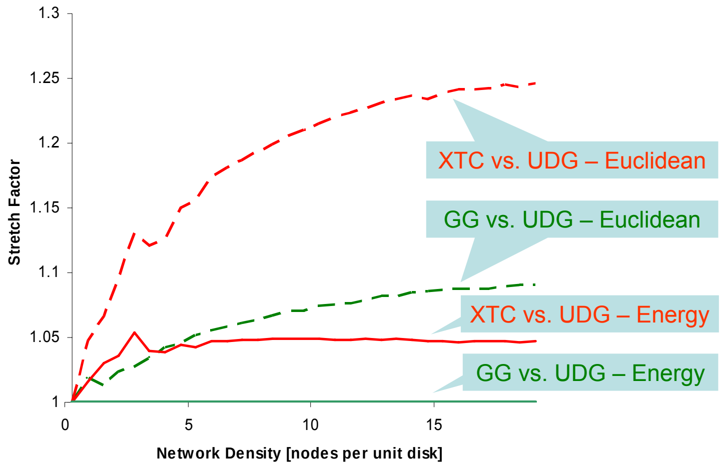
XTC Average-Case



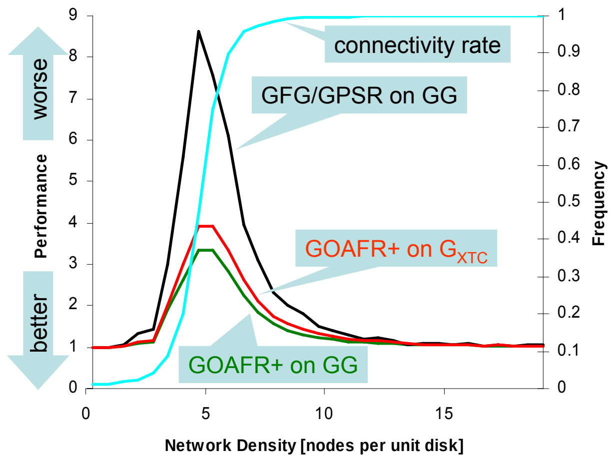
XTC Average-Case (Degrees)



XTC Average-Case (Stretch Factor)



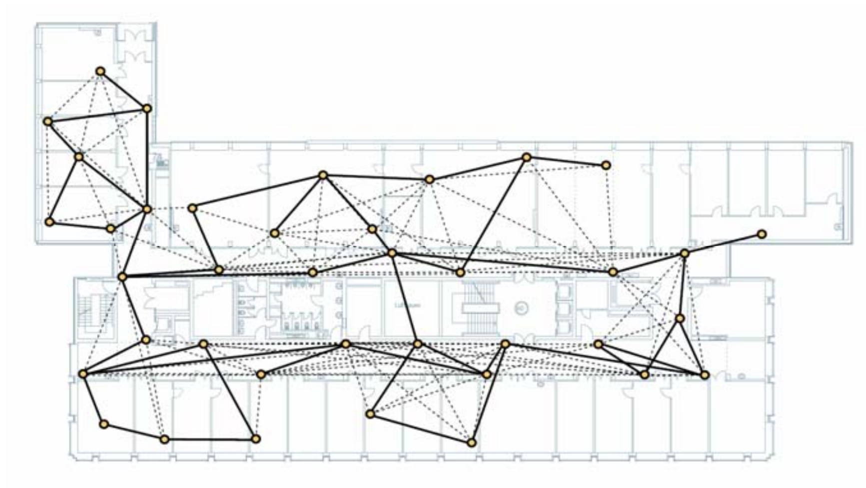
XTC Average-Case (Geometric Routing)



k-XTC: More connectivity

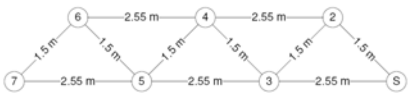
- A graph is k-(node)-connected, if k-1 arbitrary nodes can be removed, and the graph is still connected.
- In k-XTC, an edge (u,v) is only removed if there exist k nodes w_1, …, w_k such that the 2k edges (w_1, u), …, (w_k, u), (w_1,v), …, (w_k,v) are all better than the original edge (u,v).
- Theorem: If the original graph is k-connected, then the pruned graph produced by k-XTC is as well.
- Proof: Let (u,v) be the best edge that was removed by k-XTC. Using the construction of k-XTC, there is at least one common neighbor w that survives the slaughter of k-1 nodes. By induction assume that this is true for the j best edges. By the same argument as for the best edge, also the j+1st edge (u',v'), since at least one neighbor survives w' survives and the edges (u',w') and (v',w') are better.

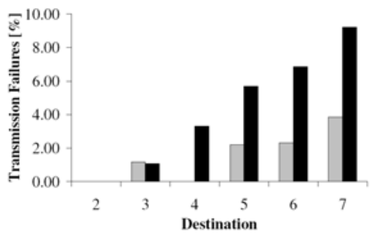
Implementing XTC, e.g. BTnodes v3

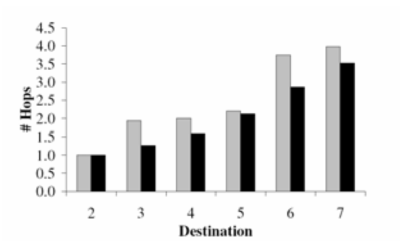


Implementing XTC, e.g. on mica2 motes

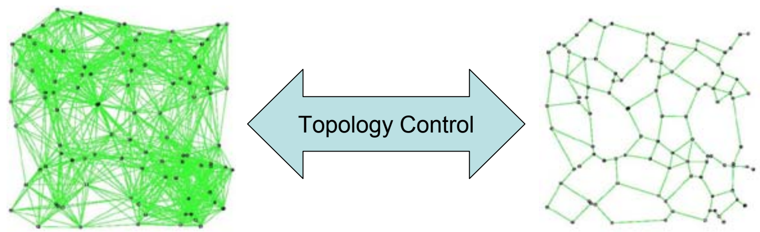
- Idea:
  - XTC chooses the reliable links
  - The quality measure is a moving average of the received packet ratio
  - Source routing: route discovery (flooding) over these reliable links only







Topology Control as a Trade-Off



Network Connectivity
Spanner Property

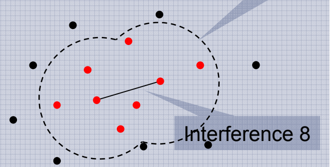
Conserve Energy
Reduce Interference
Sparse Graph, Low Degree
Planarity
Symmetric Links
Less Dynamics

Really?!?

What is Interference?

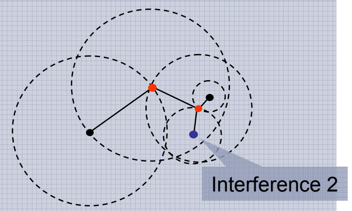
Exact size of interference range does not change the results

Link-based Interference Model



„How many nodes are affected by communication over a given link?"

Node-based Interference Model

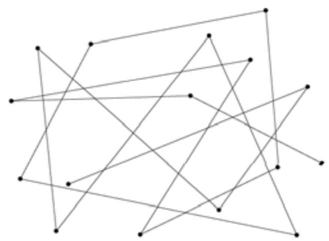


„By how many other nodes can a given network node be disturbed?"

- Problem statement
  - We want to minimize maximum interference
  - At the same time topology must be connected or a spanner etc.

Low Node Degree Topology Control?

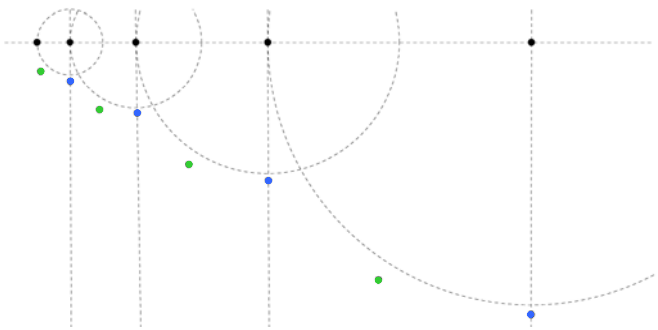
Low node degree does not necessarily imply low interference:



Very low node degree but huge interference

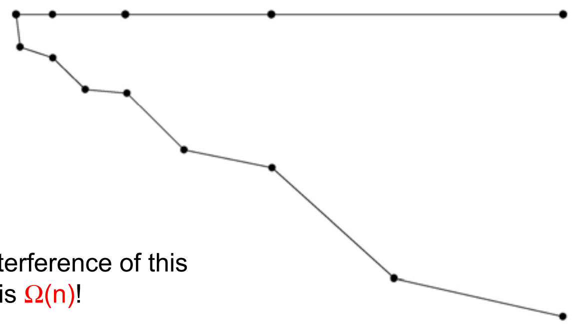
Let's Study the Following Topology!

…from a worst-case perspective



Topology Control Algorithms Produce…

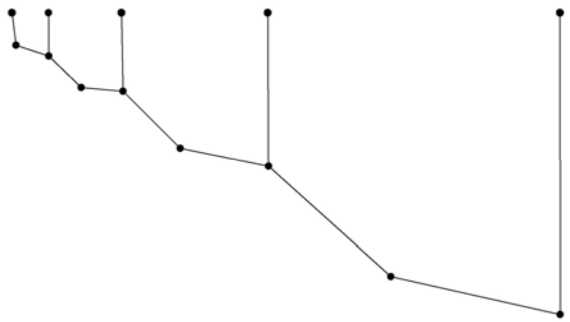
- All known topology control algorithms (with symmetric edges) include the nearest neighbor forest as a subgraph and produce something like this:



- The interference of this graph is $\Omega(n)$!

But Interference…

- Interference does not need to be high…



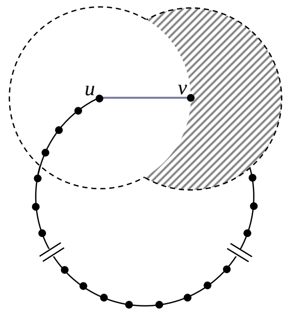
- This topology has interference O(1)!!

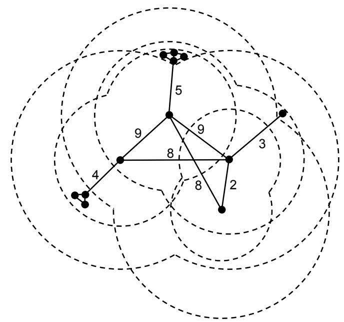
Link-based Interference Model

- Interference-optimal topologies:

There is no local algorithm that can find a good interference topology

The optimal topology will not be planar





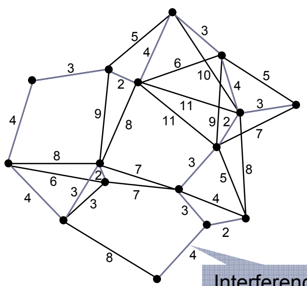
Link-based Interference Model

- LIFE (Low Interference Forest Establisher)
  - Preserves Graph Connectivity

LIFE

- Attribute interference values as weights to edges
- Compute minimum spanning tree/forest (Kruskal's algorithm)

LIFE constructs a minimum-interference forest



Interference 4

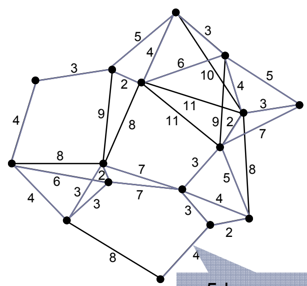
Link-based Interference Model

- LISE (Low Interference Spanner Establisher)
  - Constructs a spanning subgraph

LISE

- Add edges with increasing interference until spanner property fulfilled

LISE constructs a minimum-interference t-spanner



5-hop spanner with Interference 7

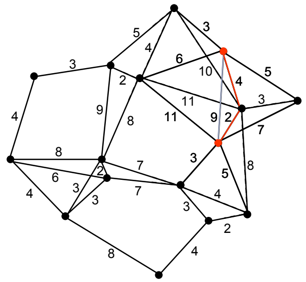
Link-based Interference Model

- LocaLISE
  - Constructs a spanner locally

Scalability

LocaLISE

- Nodes collect (t/2)-neighborhood
- Locally compute interference-minimal paths guaranteeing spanner property
- Only request that path to stay in the resulting topology



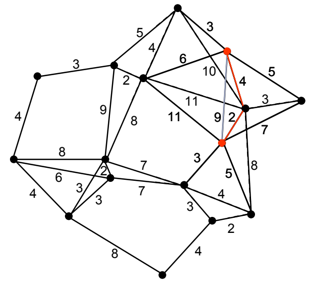
LocaLISE constructs a minimum-interference t-spanner

Link-based Interference Model

- LocaLISE (Low Interference Spanner Establisher)
  - Constructs a spanner locally

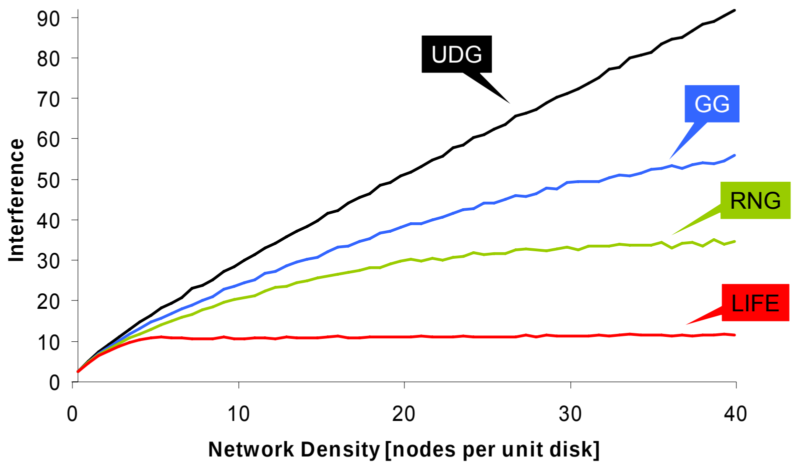
LocaLISE

- Nodes collect (t/2)-neighborhood
- Locally compute interference-minimal paths guaranteeing spanner property
- Only request that path to stay in the resulting topology

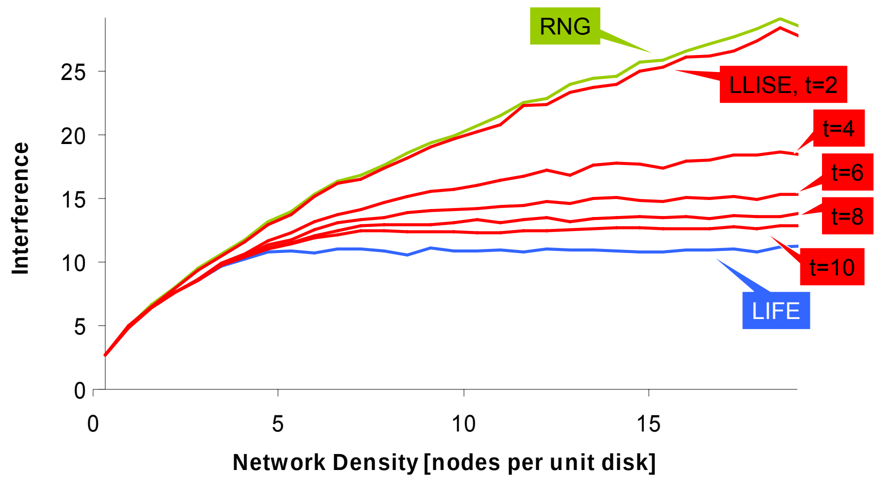


LocaLISE constructs a minimum-interference t-spanner

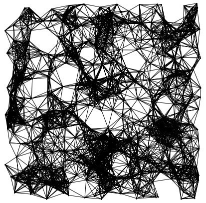
Average-Case Interference: Preserve Connectivity



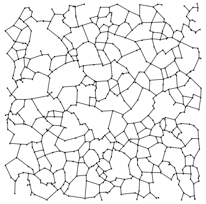
Average-Case Interference: Spanners



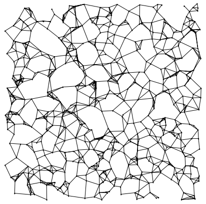
Link-based Interference Model



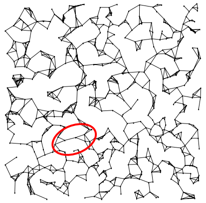
UDG, I = 50



RNG, I = 25



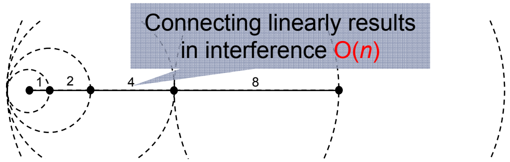
LocaLISE$_2$, I = 23



LocaLISE$_{10}$, I = 12

Node-based Interference Model

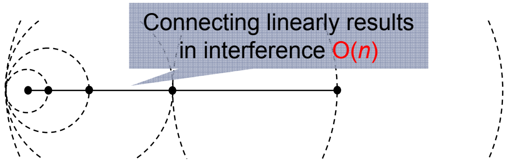
- Already 1-dimensional node distributions seem to yield inherently high interference...



- ...but the exponential node chain can be connected in a better way

Node-based Interference Model

- Already 1-dimensional node distributions seem to yield inherently high interference...



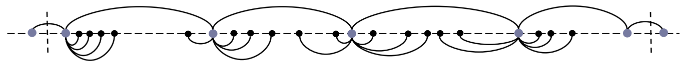
- ...but the exponential node chain can be connected in a better way

Interference $\in O(\sqrt{n})$

Matches an existing lower bound

Node-based Interference Model

- Arbitrary distributed nodes in one dimension
  - Approximation algorithm with approximation ratio in O($\sqrt[4]{n}$)



- Two-dimensional node distributions
  - Randomized algorithm resulting in interference O($\sqrt{n \log n}$)
  - No deterministic algorithm so far...

Towards a More Realistic Interference Model...

- Signal-to-interference and noise ratio (SINR)

$$\frac{\frac{P_u}{d(u,v)^{\alpha}}}{N+\sum_{w\in V\setminus\{u\}}\frac{P_w}{d(w,v)^{\alpha}}} \geq \beta$$

Power level of node u

Path-loss exponent

Noise

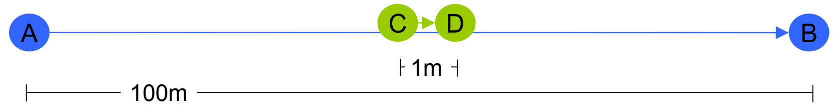
Distance between two nodes

Minimum signal-to-interference ratio

- Problem statement
  - Determine a power assignment and a schedule for each node such that all message transmissions are successful

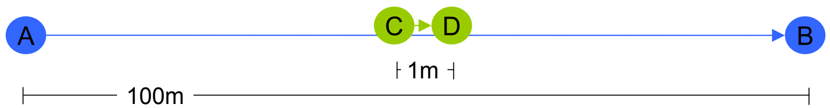
SINR is always assured

Quiz: Can these two links transmit simultaneously?



- Graph-theoretical models: No!
  - Neither in- nor out-interference
- SINR model: constant power: No!
  - Node B will receive the transmission of node C
- SINR model: power according to distance-squared: No!
  - Node D will receive the transmission of node A

Let's try harder…



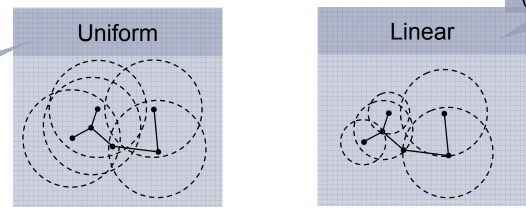
A Simple Problem

$$\frac{\frac{P_u}{d(u,v)^{\alpha}}}{N+\sum_{w\in V\setminus\{u\}}\frac{P_w}{d(w,v)^{\alpha}}} \geq \beta$$

- Each node in the network wants to send a message to an arbitrary other node
  - Commonly assumed power assignment schemes

Proportional to (receiver distance)$^{\alpha}$

Constant power level



➡ Both lead to a schedule of length $\in \Theta(n)$

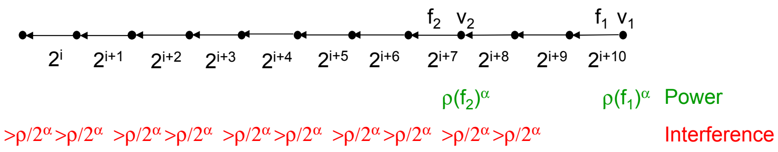
Asymptotically worst possible!

  - A clever power assignment results in a schedule of length $\in O(\log^2 n)$

This has strong implications to MAC layer protocols

Example: Linear Power Assignment

- Consider again the exponential chain:



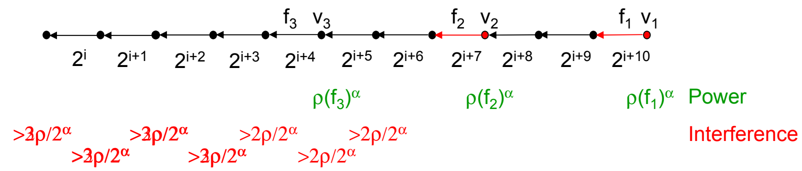
- How many links can we schedule simultaneously?
- Let us start with the first node v_1...

  → its power is $P_1 \geq \rho 2^{\alpha(i+10)}$ for some constant ρ

  Why?
- This creates interference of at least $\rho/2^\alpha$ at every other node!
- The second node v_2 also sends with power $P_2=\rho 2^{\alpha(i+7)}$
- Again, this creates an additional interference of at least $\rho/2^\alpha$ at every other node!

Example: Linear Power Assignment

- Consider again the exponential chain:



- How many links can we schedule simultaneously?
- Let us start with the first node v_1...

  → its power is $P_1 \geq \rho 2^{\alpha(i+10)}$ for some constant ρ

  Why?
- This creates interference of at least $\rho/2^\alpha$ at every other node!
- The second node v_2 also sends with power $P_2=\rho 2^{\alpha(i+7)}$
- Again, this creates an additional interference of at least $\rho/2^\alpha$ at every other node!

And so on…

Example: Linear Power Assignment

- Assume we can schedule *R* nodes in parallel.
- The left-most receiver x_r faces an interference of $R\cdot \rho/2^\alpha$

  → yet, x_r receives the message, say from x_s.
- How large can *R* be?
- The SINR at x_r must be at least β, and hence

$$\frac{\frac{\rho\cdot d(x_s,x_r)^\alpha}{d(x_s,x_r)^\alpha}}{N+R\cdot\frac{\rho}{2^\alpha}} \geq \frac{\rho 2^\alpha}{2^\alpha N+\rho R} \geq \beta$$

- From this, it follows that *R* is at most $2^\alpha/\beta$
- And therefore....

  .... at least $n\cdot \min\{1,\beta/2^\alpha\}$ time slots are required for all links!

A clever power assignment solves this instance in a constant number of time slots!