Ad hoc and Sensor Networks
Topology Control

Slides taken from
Holger Karl

(Protocols and Architectures for Wireless Sensor Networks)

Goals

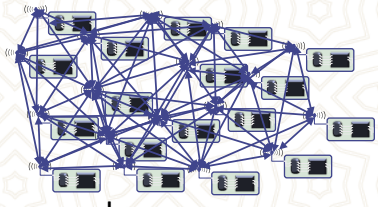
- Networks can be too dense – too many nodes in close (radio) vicinity
- This chapter looks at methods to deal with such networks by
  - Reducing/controlling transmission power
  - Deciding which links to use
  - Turning some nodes off
- Focus is on basic ideas, some algorithms
  - Complexity results are only very superficially covered

Overview

- **Motivation, basics**
- Power control
- Backbone construction
- Clustering
- Adaptive node activity

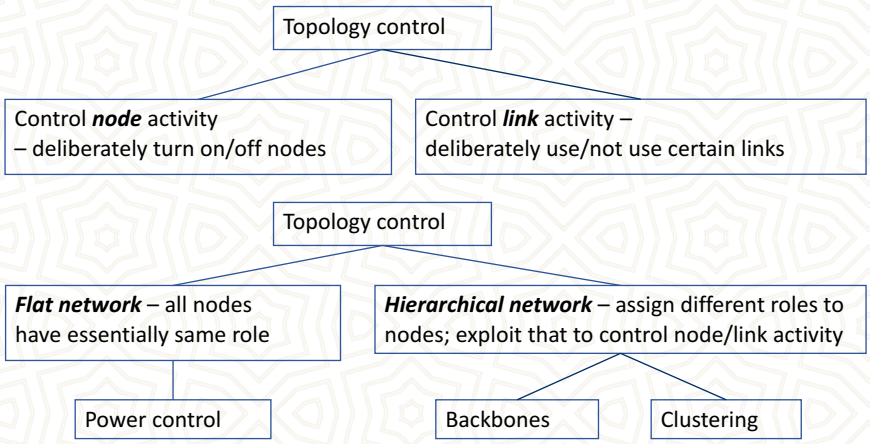
Motivation: Dense Networks

- In a very dense networks, too many nodes might be in range for an efficient operation
  - Too many collisions/too complex operation for a MAC protocol, too many paths to chose from for a routing protocol, …



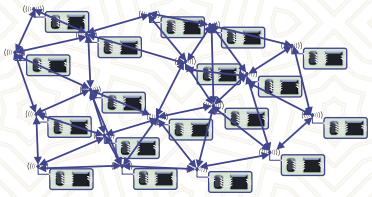
- Idea: Make **topology** less complex
  - **Topology**: Which node is able/allowed to communicate with which other nodes
  - Topology control needs to maintain invariants, e.g., connectivity

Options For Topology Control



Flat Networks

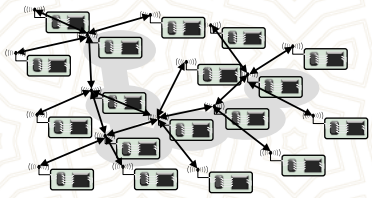
- Main option: Control transmission power
  - Do not always use maximum power
  - Selectively for some links or for a node as a whole
  - Topology looks "thinner"
  - Less interference, …



- Alternative: Selectively discard some links
  - Usually done by introducing hierarchies

Hierarchical Networks – Backbone

- Construct a backbone network
  - Some nodes "control" their neighbors – they form a (minimal) dominating set
  - Each node should have a controlling neighbor
  - Controlling nodes have to be connected (backbone)
  - Only links within backbone and from backbone to controlled neighbors are used

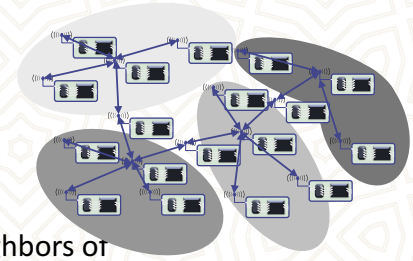


- Formally: Given graph G=(V,E), construct D ½ V such that

$$\forall v \in V : v \in D \vee \exists d \in D : (v, d) \in E$$

Hierarchical Network – Clustering

- Construct **clusters**
  - Partition nodes into groups ("clusters")
  - Each node in exactly one group
    - Except for nodes "bridging" between two or more groups
  - Groups can have clusterheads
  - Typically: all nodes in a cluster are direct neighbors of their clusterhead
  - Clusterheads are also a dominating set, but should be separated from each other – they form an independent set



- Formally: Given graph G=(V,E), construct C ½ V such that

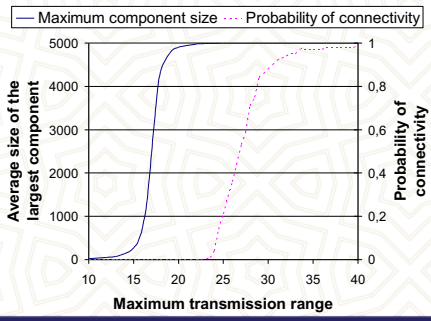
$$\forall v \in V - C : \exists c \in C : (v, c) \in E \qquad \forall c_1, c_2 \in C : (c_1, c_2) \notin E$$

Aspects Of Topology-Control Algorithms

- **Connectivity** – If two nodes connected in G, they have to be connected in G0 resulting from topology control
- **Stretch factor** – should be small
  - **Hop stretch factor**: how much longer are paths in G0 than in G?
  - **Energy stretch factor**: how much more energy does the most energy-efficient path need?
- **Throughput** – removing nodes/links can reduce throughput, by how much?
- Robustness to mobility
- Algorithm overhead

Example: Price For Maintaining Connectivity

- Maintaining connectivity can be very "costly" for a power control approach
- Compare power required for connectivity compared to power required to reach a very big maximum component



Overview

- Motivation, basics
- **Power control**
- Backbone construction
- Clustering
- Adaptive node activity

Power Control – Magic Numbers?

- Question: What is a good power level for a node to ensure "nice" properties of the resulting graph?
- Idea: Controlling transmission power corresponds to controlling the number of neighbors for a given node
- Is there an "optimal" number of neighbors a node should have?
  - Is there a "magic number" that is good irrespective of the actual graph/network under consideration?
- Historically, k=6 or k=8 had been suggested as such "magic numbers"
  - However, they optimize progress per hop – they do not guarantee connectivity of the graph!!
  - ! Needs deeper analysis

Controlling Transmission Range

- Assume all nodes have identical transmission range r=r(|V|), network covers area A, V nodes, uniformly distr.
- Fact: Probability of connectivity goes to zero if:

$$r(|V|) \leq \sqrt{\frac{(1-\epsilon)A\log|V|}{\pi|V|}}, \text{ for any } \epsilon > 0$$

- Fact: Probability of connectivity goes to 1 for

$$r(|V|) \geq \sqrt{\frac{A(\log|V|+\gamma_{|V|})}{\pi|V|}}$$

if and only if ρ |V| ! 1 with |V|

- Fact (uniform node distribution, density ρ):

$$P(\text{G is } k\text{-connected}) \approx \left(1 - \sum_{l=0}^{k-1} \frac{(\rho\pi r^2)^l}{l!} e^{-\rho\pi r^2}\right)$$

Controlling Number Of Neighbors

- Knowledge about range also tells about number of neighbors
  - Assuming node distribution (and density) is known, e.g., uniform
- Alternative: directly analyze number of neighbors
  - Assumption: Nodes randomly, uniformly placed, only transmission range is controlled, identical for all nodes, only symmetric links are considered
- Result: For connected network, required number of neighbors per node is **Θ (log |V|)**
  - It is **not a constant**, but depends on the number of nodes!
  - For a larger network, nodes need to have more neighbors & larger transmission range! – Rather inconvenient
  - Constants can be bounded

Some Example Constructions For Power Control

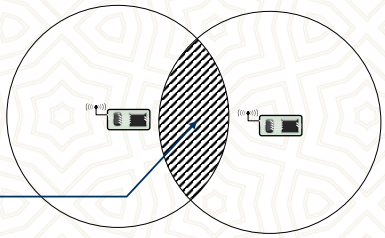
- Basic idea for most of the following methods:
  Take a graph G=(V,E), produce a graph G0=(V,E0) that maintains connectivity with fewer edges
  - Assume, e.g., knowledge about node positions
  - Construction should be local (for distributed implementation)

Example 1: Relative Neighborhood Graph (RNG)

- Edge between nodes u and v if and only if there is no other node w that is closer to either u or v
- Formally:

$$\forall u, v \in V : (u, v) \in E' \text{ iff}$$
$$\nexists w \in V : \max\{d(u, w), d(v, w)\} < d(u, v)$$

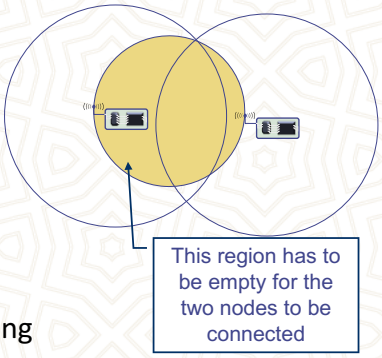
- RNG maintains connectivity of the original graph
- Easy to compute locally
- But: Worst-case spanning ratio is ▶ (|V|)
- Average degree is 2.6



This region has to be empty for the two nodes to be connected

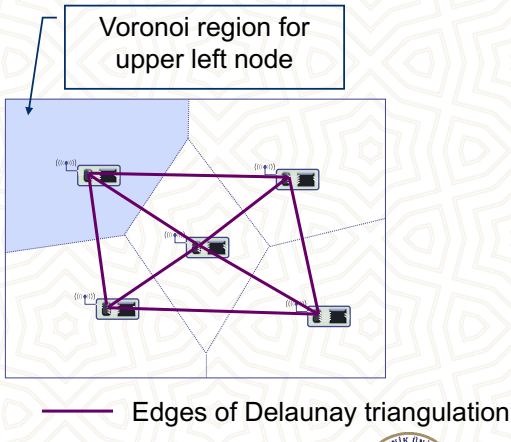
Example 2: Gabriel Graph

- Gabriel graph (GG) similar to RNG
- Difference: Smallest circle with nodes u and v on its circumference must only contain node u and v for u and v to be connected
- Formally: $\forall u, v \in V : (u, v) \in E'$ iff
  $\nexists w \in V : d^2(u, w) + d^2(v, w) < d^2(u, v)$
- Properties: Maintains connectivity, Worst-case spanning ratio $\Omega(|V|1/2)$, energy stretch O(1) (depending on consumption model!), worst-case degree $\Omega(|V|)$



Example 3: Delaunay Triangulation

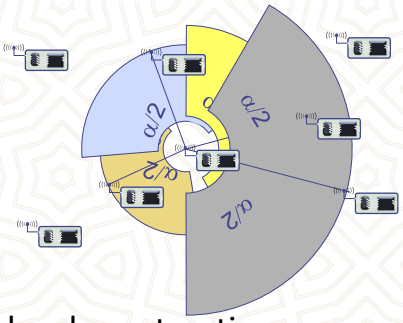
- Assign, to each node, all points in the plane for which it is the closest node
  ! Voronoi diagram
  - Constructed in O(|V| log |V|) time
- Connect any two nodes for which the Voronoi regions touch
  ! Delaunay triangulation
- Problem: Might produce very long links; not well suited for power control



Example: Cone-Based Topology Control

- Assumption: Distance and angle information between nodes is available
- Two-phase algorithm
- Phase 1
  - Every node starts with a small transmission power
  - Increase it until a node has sufficiently many neighbors
  - What is "sufficient"? – When there is at least one neighbor in each **cone** of angle α
  - $\alpha = 5/6\ \pi$ is necessary and sufficient condition for connectivity!
- Phase 2
  - Remove redundant edges: Drop a neighbor w of u if there is a node v of w and u such that sending from u to w directly is less efficient than sending from u via v to w
  - Essentially, a local Gabriel graph construction

Example: Cone-Based Topology Control (2)



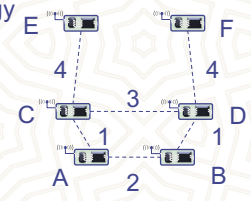
- Properties: simple, local construction
- Extensions for k-connectivity (Yao graph)
- Little exercise: What happens when $\alpha <$ or $> 5/6\ \pi$?

Centralized Power Control Algorithm

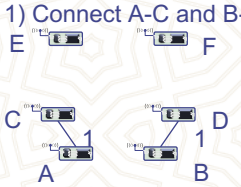
- Goal: Find topology control algorithm minimizing the **maximum** power used by any node
  - Ensuring simple or bi-connectivity
  - Assumptions: Locations of all nodes and path loss between all node pairs are known; each node uses an individually set power level to communicate with all its neighbors
- Idea: Use a centralized, greedy algorithm
  - Initially, all nodes have transmission power 0
  - Connect those two components with the shortest distance between them (raise transmission power accordingly)
- Second phase: Remove links (=reduce transmission power) not needed for connectivity
- Exercise: Relation to Kruskal's MST algorithm?

Centralized Power Control Algorithm

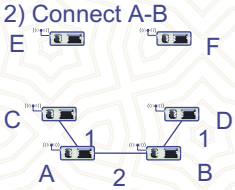
Topology



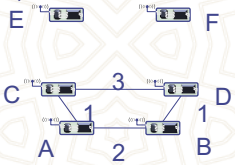
1) Connect A-C and B-D



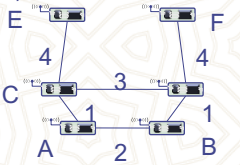
2) Connect A-B



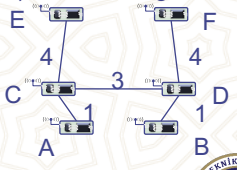
3) Connect C-D



4) Connect C-E and D-F



5) Remove edge A-B



Overview

- Motivation, basics
- Power control
- **Backbone construction**
- Clustering
- Adaptive node activity

Hierarchical Networks – Backbones

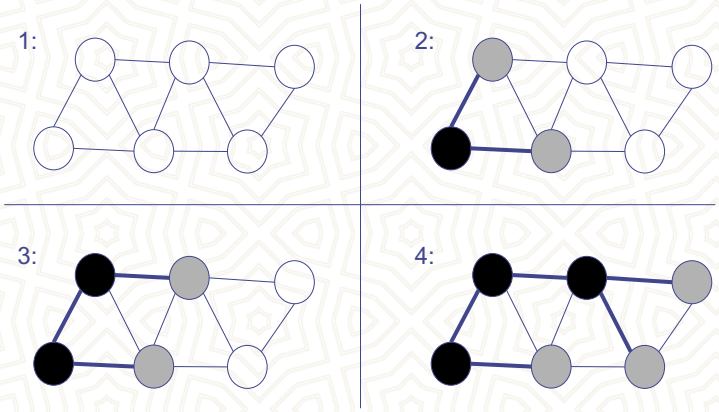
- Idea: Select some nodes from the network/graph to form a **backbone**
  - A connected, minimal, dominating set (MDS or MCDS)
  - Dominating nodes control their neighbors
  - Protocols like routing are confronted with a simple topology – from a simple node, route to the backbone, routing in backbone is simple (few nodes)
- Problem: MDS is an NP-hard problem
  - Hard to approximate, and even approximations need quite a few messages

Backbone By Growing a Tree

- Construct the backbone as a tree, grown iteratively

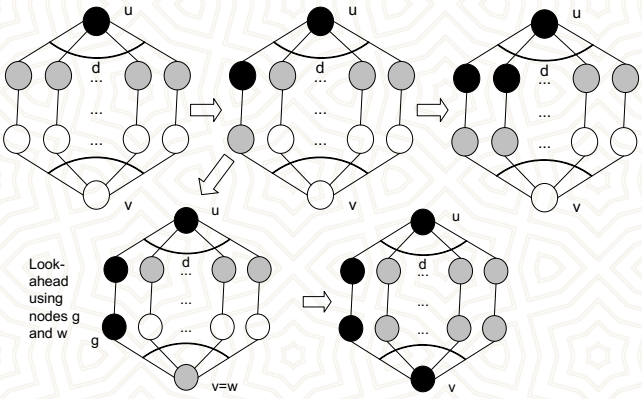
```
initialize all nodes' color to white
pick an arbitrary node and color it grey

while (there are white nodes) {
  pick a grey node v that has white neighbors
  color the grey node v black
  foreach white neighbor u of v {
    color u grey
    add (v,u) to tree T
  }
}
```


Backbone By Growing a Tree – Example



Problem: Which Gray Node To Pick?

- When blindly picking any gray node to turn black, resulting tree can be very bad



Solution:

Look ahead!

One step suffices

Performance Of Tree Growing With Look Ahead

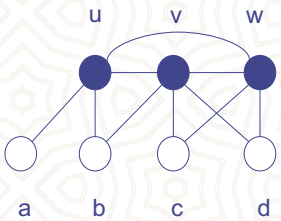
- Dominating set obtained by growing a tree with the look ahead heuristic is at most a factor 2(1+ H(Δ)) larger than MDS
  - H(¢) harmonic function, H(k) = $\sum_{i=1}^{k} 1/i <= \ln k + 1$
  - Δ is maximum degree of the graph
- It is automatically connected
- Can be implemented in a distributed fashion as well

Start Big, Make Lean

- Idea: start with some, possibly large, connected dominating set, reduce it by removing unnecessary nodes
- Initial construction for dominating set
  - All nodes are initially white
  - Mark any node black that has two neighbors that are not neighbors of each other (they might need to be dominated)
  - ! Black nodes form a connected dominating set (proof by contradiction); shortest path between ANY two nodes only contains black nodes
- Needed: Pruning heuristics

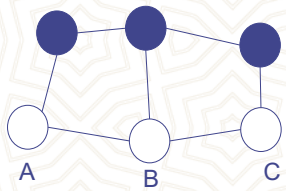
Pruning Heuristics

- Heuristic 1: Unmark node v if
  - Node v and its neighborhood are included in the neighborhood of some node marked node u (then u will do the domination for v as well)
  - Node v has a smaller unique identifier than u (to break ties)
- Heuristic 2: Unmark node v if
  - Node v's neighborhood is included in the neighborhood of two
  - marked neighbors u and w
  - Node v has the smallest identifier of the tree nodes
- Nice and easy, but only linear approximation factor



One More Distributed Backbone Heuristic: Span

- Construct backbone, but take into account need to carry traffic – preserve capacity
  - Means: If two paths could operate without interference in the original graph, they should be present in the reduced graph as well
  - Idea: If the stretch factor (induced by the backbone) becomes too large, more nodes are needed in the backbone
- Rule: Each node observes traffic around itself
  - If node detects two neighbors that need three hops to communicate with each other, node joins the backbone, shortening the path
  - Contention among potential new backbone nodes handled using random backoff

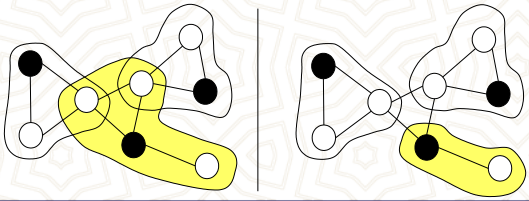


Overview

- Motivation, basics
- Power control
- Backbone construction
- **Clustering**
- Adaptive node activity

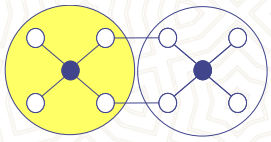
Clustering

- Partition nodes into groups of nodes – **clusters**
- Many options for details
  - Are there **clusterheads**? – One controller/representative node per cluster
  - May clusterheads be neighbors? If no: clusterheads form an **independent set** C: Typically: clusterheads form a **maximum independent set**
  - May clusters overlap? Do they have nodes in common?



Clustering

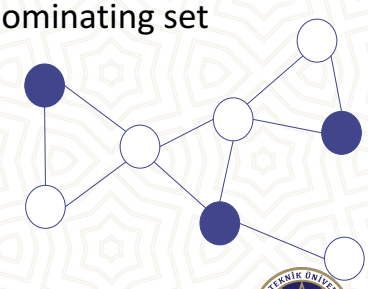
- Further options
  - How do clusters communicate? Some nodes need to act as **gateways** between clusters
    If clusters may not overlap, two nodes need to jointly act as a **distributed gateway**



  - How many gateways exist between clusters? Are all active, or some standby?
  - What is the maximal diameter of a cluster? If more than 2, then cluster heads are not necessarily a maximum independent set
  - Is there a hierarchy of clusters?

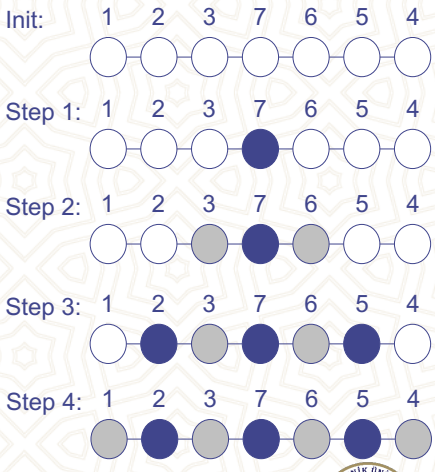
Maximum Independent Set

- Computing a maximum independent set is NP-complete
- Can be approximate within (Δ +3)/5 for small Δ, within O(Δ log log Δ / log Δ) else; Δ bounded degree
- Show: A maximum independent set is also a dominating set
- Maximum independent set not necessarily intuitively desired solution
  - Example: Radial graph, with only (v0,vi) 2 E



A Basic Construction Idea For Independent Sets

- Use some attribute of nodes to break local symmetries
  - Node identifiers, energy reserve, mobility, weighted combinations… - matters not for the idea as such (all types of variations have been looked at)
- Make each node a clusterhead that locally has the largest attribute value
- Once a node is dominated by a clusterhead, it abstains from local competition, giving other nodes a chance



Determining Gateways To Connect Clusters

- Suppose: Cluster heads have been found
- How to connect the clusters, how to select gateways?
- It suffices for each clusterhead to connect to all other cluster heads that are at most three hops
- Resulting backbone (!) is connected
- Formally: Steiner tree problem
  - Given: Graph G=(V,E), a subset C ½ V
  - Required: Find another subset T ½ V such that S [T is connected and S [T is a cheapest such set
  - Cost metric: number of nodes in T, link cost
  - Here: special case since C are an independent set

Rotating Clusterheads

- Serving as a clusterhead can put additional burdens on a node
  - For MAC coordination, routing, …
- Let this duty rotate among various members
  - Periodically reelect – useful when energy reserves are used as discriminating attribute
  - LEACH – determine an optimal percentage P of nodes to become clusterheads in a network
    - Use 1/P rounds to form a period
    - In each round, nP nodes are elected as clusterheads
    - At beginning of round r, node that has not served as clusterhead in this period becomes clusterhead with probability P/(1-p(r mod 1/P))

Multi-Hop Clusters

- Clusters with diameters larger than 2 can be useful, e.g., when used for routing protocol support
- Formally: Extend "domination" definition to also dominate nodes that are at most d hops away
- Goal: Find a smallest set D of dominating nodes with this extended definition of dominance
- Only somewhat complicated heuristics exist

- Different tilt: Fix the **size** (not the diameter) of clusters
  - Idea: Use **growth budgets** – amount of nodes that can still be adopted into a cluster, pass this number along with broadcast adoption messages, reduce budget as new nodes are found

Passive Clustering

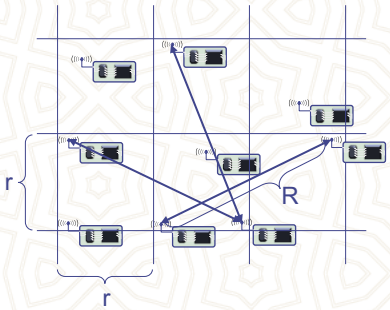
- Constructing a clustering structure brings overheads
  - Not clear whether they can be amortized via improved efficiency
- Question: Eat cake and have it?
  - Have a clustering structure without any overhead?
  - Maybe not the best structure, and maybe not immediately, but benefits at zero cost are no bad deal…
- ! Passive clustering
  - Whenever a broadcast message travels the network, use it to construct clusters on the fly
  - Node to start a broadcast: Initial node
  - Nodes to forward this first packet: Clusterhead
  - Nodes forwarding packets from clusterheads: ordinary/gateway nodes
  - And so on… ! Clusters will emerge at low overhead

Overview

- Motivation, basics
- Power control
- Backbone construction
- Clustering
- **Adaptive node activity**

Adaptive Node Activity

- Remaining option: Turn some nodes off deliberately
- Only possible if other nodes remain on that can take over their duties
- Example duty: Packet forwarding
  - Approach: Geographic Adaptive Fidelity (GAF)
- Observation: Any two nodes within a square of length $r < R/5^{1/2}$ can replace each other with respect to forwarding
  - R radio range
- Keep only one such node active, let the other sleep



Conclusion

- Various approaches exist to trim the topology of a network to a desired shape
- Most of them bear some non-negligible overhead
  - At least: Some distributed coordination among neighbors, or they require additional information
  - Constructed structures can turn out to be somewhat brittle – overhead might be wasted or even counter-productive
- Benefits have to be carefully weighted against risks for the particular scenario at hand

Ad hoc and Sensor Networks
Routing protocols

Goals

- In any network of diameter > 1, the routing & forwarding problem appears
- We will discuss mechanisms for constructing routing tables in ad hoc/sensor networks
  - Specifically, when nodes are mobile
  - Specifically, for broadcast/multicast requirements
  - Specifically, with energy efficiency as an optimization metric
  - Specifically, when node position is available

Overview

- **Unicast routing in MANETs**
- Energy efficiency & unicast routing
- Multi-/broadcast routing
- Geographical routing

Unicast, Id-Centric Routing

- Given: a network/a graph
  - Each node has a unique identifier (ID)
- Goal: Derive a mechanism that allows a packet sent from an arbitrary node to arrive at some arbitrary destination node
  - The routing & forwarding problem
  - Routing: Construct data structures (e.g., tables) that contain information how a given destination can be reached
  - Forwarding: Consult these data structures to forward a given packet to its next hop
- Challenges
  - Nodes may move around, neighborhood relations change
  - Optimization metrics may be more complicated than "smallest hop count" – e.g., energy efficiency

Ad-Hoc Routing Protocols

- Because of challenges, standard routing approaches not really applicable
  - Too big an overhead, too slow in reacting to changes
  - Examples: Dijkstra's link state algorithm; Bellman-Ford distance vector algorithm
- Simple solution: Flooding
  - Does not need any information (routing tables) – simple
  - Packets are usually delivered to destination
  - But: overhead is prohibitive

  ! Usually not acceptable, either

! Need specific, **ad hoc routing protocols**

Ad Hoc Routing Protocols – Classification

- Main question to ask: **When** does the routing protocol operate?
- Option 1: Routing protocol **always** tries to keep its routing data up-to-date
  - Protocol is **proactive** (active before tables are actually needed) or **table-driven**
- Option 2: Route is only determined when actually needed
  - Protocol operates **on demand**
- Option 3: Combine these behaviors
  - **Hybrid** protocols

Ad Hoc Routing Protocols – Classification

- Is the network regarded as flat or hierarchical?
  - Compare topology control, traditional routing
- Which data is used to identify nodes?
  - An arbitrary identifier?
  - The **position** of a node?
    - Can be used to assist in ***geographic*** routing protocols because choice of next hop neighbor can be computed based on destination address
  - Identifiers that are not arbitrary, but carry some structure?
    - As in traditional routing
    - Structure akin to position, on a logical level?

Proactive Protocols

- Idea: Start from a +/- standard routing protocol, adapt it
- Adapted distance vector: **Destination Sequence Distance Vector (DSDV)**
  - Based on distributed Bellman Ford procedure
  - Add **aging** information to route information propagated by distance vector exchanges; helps to avoid routing loops
  - Periodically send full route updates
  - On topology change, send incremental route updates
  - Unstable route updates are delayed
  - … + some smaller changes

Proactive Protocols – OLSR

- Combine link-state protocol & topology control
- **Optimized Link State Routing (OLSR)**
- Topology control component: Each node selects a minimal dominating set for its two-hop neighborhood
  - Called the ***multipoint relays***
  - Only these nodes are used for packet forwarding
  - Allows for efficient flooding
- Link-state component: Essentially a standard link-state algorithms on this reduced topology
  - Observation: Key idea is to reduce flooding overhead (here by modifying topology)

Proactive Protocols – Combine LS & DS: Fish Eye

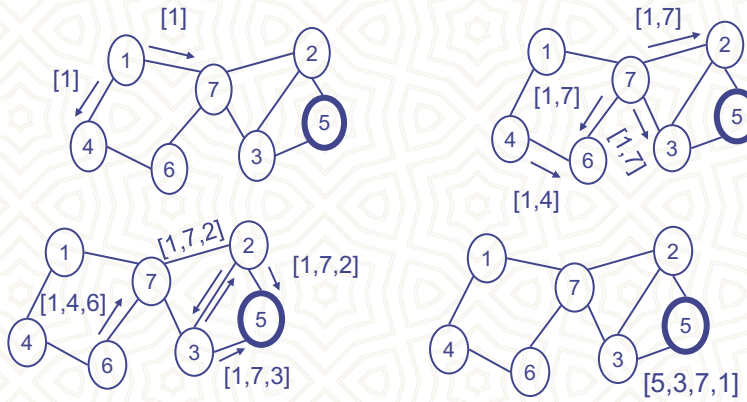
- Fisheye State Routing (FSR) makes basic observation: When destination is far away, details about path are not relevant – only in vicinity are details required
  - Look at the graph as if through a fisheye lens
  - Regions of different accuracy of routing information
- Practically:
  - Each node maintains topology table of network (as in LS)
  - Unlike LS: only distribute link state updates locally
  - More frequent routing updates for nodes with smaller scope

Reactive Protocols – DSR

- In a reactive protocol, how to forward a packet to destination?
  - Initially, no information about next hop is available at all
  - One (only?) possible recourse: Send packet to all neighbors – flood the network
  - Hope: At some point, packet will reach destination and an answer is sent pack – use this answer for backward learning the route from destination to source
- Practically: **Dynamic Source Routing (DSR)**
  - Use separate **route request/route reply** packets to discover route
    - Data packets only sent once route has been established
    - Discovery packets smaller than data packets
  - Store routing information in the discovery packets

DSR Route Discovery Procedure

- Search for route from 1 to 5



Node 5 uses route information recorded in RREQ to send back, via ***source routing***, a route reply

DSR Modifications, Extensions

- Intermediate nodes may send route replies in case they already know a route
  - Problem: stale route caches
- Promiscuous operation of radio devices – nodes can learn about topology by listening to control messages
- Random delays for generating route replies
  - Many nodes might know an answer – reply storms
  - NOT necessary for medium access – MAC should take care of it
- Salvaging/local repair
  - When an error is detected, usually sender times out and constructs entire route anew
  - Instead: try to locally change the source-designated route
- Cache management mechanisms
  - To remove stale cache entries quickly
  - Fixed or adaptive lifetime, cache removal messages, …

Reactive Protocols – AODV

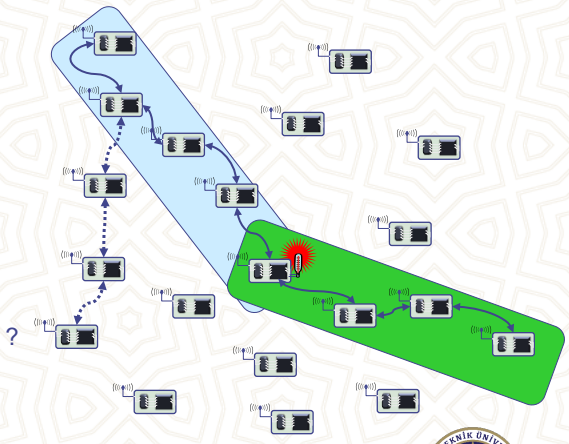
- **Ad hoc On Demand Distance Vector** routing (AODV)
  - Very popular routing protocol
  - Essentially same basic idea as DSR for discovery procedure
  - Nodes maintain routing tables instead of source routing
  - Sequence numbers added to handle stale caches
  - Nodes remember from where a packet came and populate routing tables with that information

Reactive Protocols – TORA

- Observation: In hilly terrain, routing to a river's mouth is easy – just go downhill
- Idea: Turn network into hilly terrain
  - Different "landscape" for each destination
  - Assign "heights" to nodes such that when going downhill, destination is reached – in effect: orient edges between neighbors
  - Necessary: resulting directed graph has to be cycle free
- Reaction to topology changes
  - When link is removed that was the last "outlet" of a node, reverse direction of all its other links (increase height!)
  - Reapply continuously, until each node except destination has at least a single outlet – will succeed in a connected graph!

Alternative Approach: Gossiping/Rumor Routing

- Turn routing problem around: Think of an "agent" wandering through the network, looking for data (events, …)
- Agent initially perform random walk
- Leave "traces" in the network
- Later agents can use these traces to find data
- Essentially: works due to high probability of line intersections

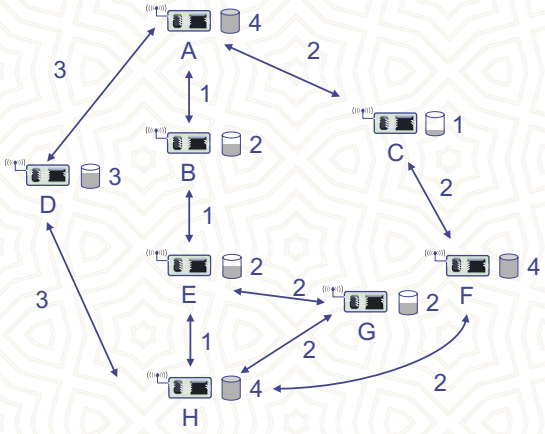


Overview

- Unicast routing in MANETs
- **Energy efficiency & unicast routing**
- Multi-/broadcast routing
- Geographical routing

Energy-Efficient Unicast: Goals

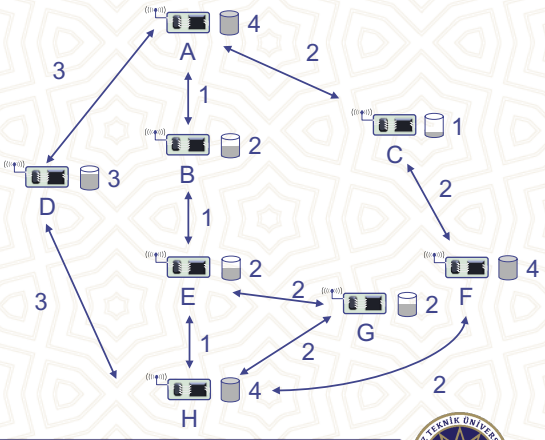
- Particularly interesting performance metric: Energy efficiency
- Goals
  - Minimize energy/bit
    - Example: A-B-E-H
  - Maximize network lifetime
    - Time until first node failure, loss of coverage, partitioning
- Seems trivial – use proper link/path metrics (not hop count) and standard routing



Example: Send data from node A to node H

Basic Options For Path Metrics

- Maximum total available battery capacity
  - Path metric: Sum of battery levels
  - Example: A-C-F-H
- Minimum battery cost routing
  - Path metric: Sum of reciprocal battery levels
  - Example: A-D-H
- Conditional max-min battery capacity routing
  - Only take battery level into account when below a given level
- Minimize variance in power levels
- Minimum total transmission power

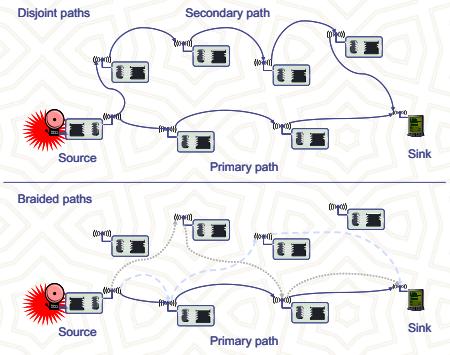


A Non-Trivial Path Metric

- Previous path metrics do not perform particularly well
- One non-trivial link weight: $w_{ij} = e_{ij}(\lambda^{\alpha_i} - 1)$
  - wij weight for link node i to node j
  - eij required energy, ✦ some constant, ✔ i fraction of battery of node i already used up
- Path metric: Sum of link weights
  - Use path with smallest metric
- Properties: Many messages can be send, high network lifetime
  - With admission control, even a competitive ratio logarithmic in network size can be shown

Multipath Unicast Routing

- Instead of only a single path, it can be useful to compute multiple paths between a given source/destination pair
- Multiple paths can be **disjoint** or **braided**
- Used simultaneously, alternatively, randomly, …



Overview

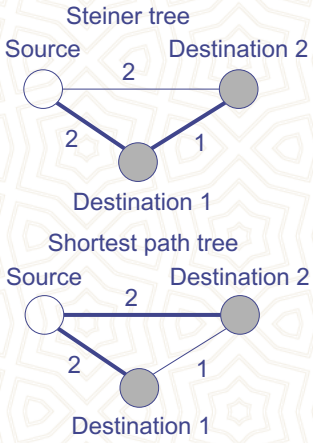
- Unicast routing in MANETs
- Energy efficiency & unicast routing
- **Multi-/broadcast routing**
- Geographical routing

Broadcast & Multicast (Energy-Efficient)

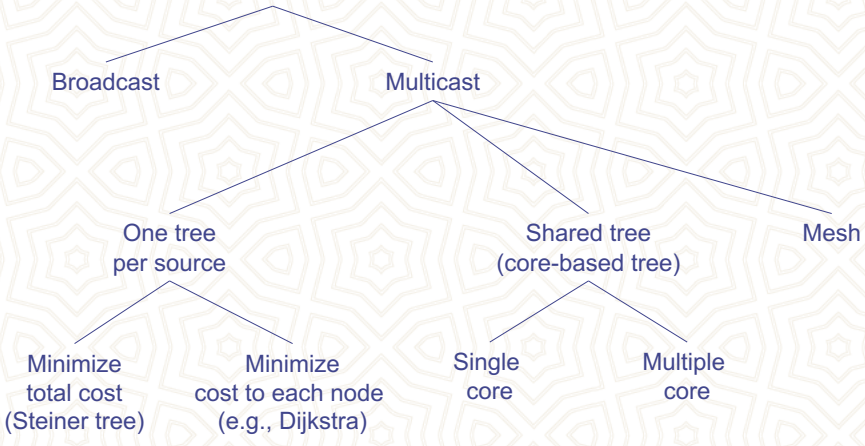
- Distribute a packet to all reachable nodes (**broadcast**) or to a somehow (explicitly) denoted subgroup (**multicast**)
- Basic options
  - Source-based tree: Construct a tree (one for each source) to reach all addressees
    - Minimize total cost (= sum of link weights) of the tree
    - Minimize maximum cost to each destination
  - Shared, core-based trees
    - Use only a single tree for all sources
    - Every source sends packets to the tree where they are distributed
  - Mesh
    - Trees are only 1-connected ! use meshes to provide higher redundancy and thus robustness in mobile environments

Optimization Goals For Source-Based Trees

- For each source, minimize **total cost**
  - This is the Steiner tree problem again
- For each source, minimize **maximum cost** to each destination
  - This is obtained by overlapping the **individual** shortest paths as computed by a normal routing protocol



Summary Of Options (Broadcast/Multicast)



Wireless Multicast Advantage

- Broad-/Multicasting in wireless is unlike broad-/multicasting in a wired medium
  - Wires: locally distributing a packet to n neighbors: n times the cost of a unicast packet
  - Wireless: sending to n neighbors can incur costs
    - As high as sending to a single neighbor – if receive costs are neglected completely
    - As high as sending once, receiving n times – if receives are tuned to the right moment
    - As high as sending n unicast packets – if the MAC protocol does not support local multicast
- ! If local multicast is cheaper than repeated unicasts, then **wireless multicast** advantage is present
  - Can be assumed realistically

Steiner Tree Approximations

- Computing Steiner tree is NP complete
- A simple approximation
  - Pick some arbitrary order of all destination nodes + source node
  - Successively add these nodes to the tree: For every next node, construct a shortest path to some other node already on the tree
  - Performs reasonably well in practice
- Takahashi Matsuyama heuristic
  - Similar, but let algorithm decide which is the next node to be added
  - Start with source node, add that destination node to the tree which has shortest path
  - Iterate, picking that destination node which has the shortest path to some node already on the tree
- Problem: Wireless multicast advantage not exploited!
  - And does not really fit to the Steiner tree formulation

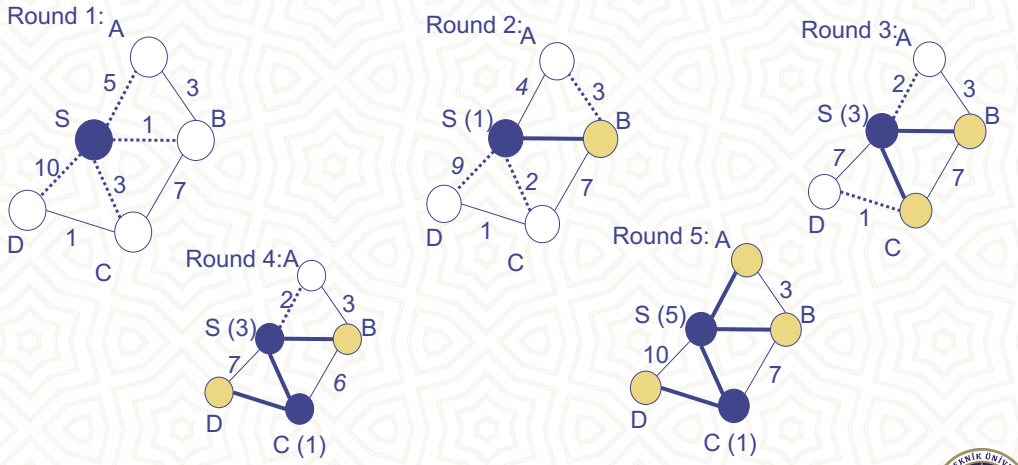
Broadcast Incremental Power (BIP)

- How to broadcast, using the wireless multicast advantage?
  - Goal: use as little transmission power as possible
- Idea: Use a minimum-spanning-tree-type construction (Prim's algorithm)
- But: Once a node transmits at a given power level & reaches some neighbors, it becomes cheaper to reach *additional* neighbors
- From BIP to multicast incremental power (MIP):
  - Start with broadcast tree construction, then prune unnecessary edges out of the tree

BIP – Algorithm

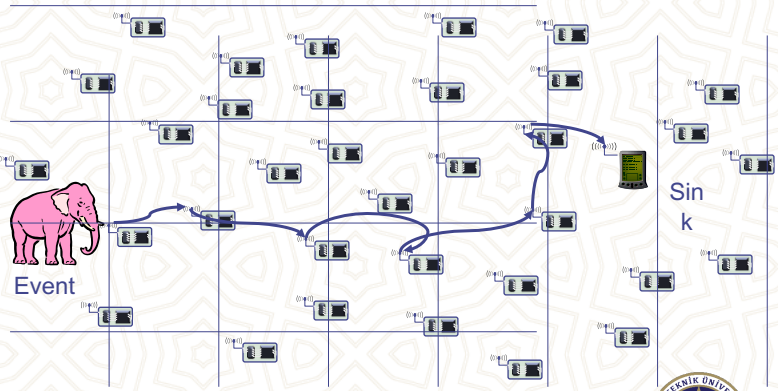
```
// Initialize
V_T = {source node}
P(source node) = 0 // transmission power assigned to a node
foreach (v in V \ V_T) {
  Set candidate edge to (source node,v)
  Set candidate edge weight to transmission power to
    reach v from source node
}
// Compute tree
while (V_T ≠ V) {
  Select v ∈ V \ V_T with smallest candidate edge weight
  Add v to V_T using its candidate edge (u,v)
  Increase P(u) to smallest power that reaches v
  // Recompute candidate edges and their weights
  foreach (v in V \ V_T) {
    Select u which minimizes P'(u) − P(u)
    // where P'(u) ≥ P(u) is smallest power to reach v from u
    Set candidate edge to (u,v)
    Set candidate edge weight to P'(u) − P(u)
  }
}
```

BIP – Example



Example For Mesh-Based Multicast

- Two-tier data dissemination
- Overlay a mesh, route along mesh intersections
- Broadcast within the quadrant where the destination is (assumed to be) located



Overview

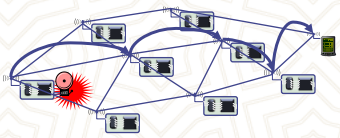
- Unicast routing in MANETs
- Energy efficiency & unicast routing
- Multi-/broadcast routing
- **Geographical routing**
  - **Position-based routing**
  - Geocasting

Geographic Routing

- Routing tables contain information to which next hop a packet should be forwarded
  - Explicitly constructed
- Alternative: Implicitly infer this information from physical placement of nodes
  - Position of current node, current neighbors, destination known – send to a neighbor in the right direction as next hop
  - **Geographic routing**
- Options
  - Send to any node in a given area – **geocasting**
  - Use position information to aid in routing – **position-based routing**
    - Might need a location service to map node ID to node position

Basics Of Position-Based Routing

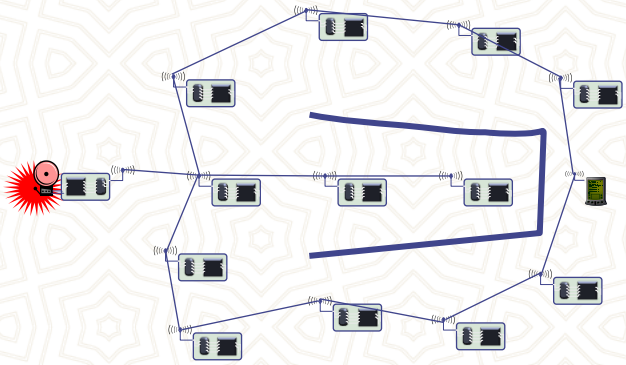
- "Most forward within range r" strategy
  - Send to that neighbor that realizes the most forward progress towards destination
  - NOT: farthest away from sender!



- Nearest node with (any) progress
  - Idea: Minimize transmission power
- Directional routing
  - Choose next hop that is angularly closest to destination
  - Choose next hop that is closest to the connecting line to destination
  - Problem: Might result in loops!

Problem: Dead Ends

- Simple strategies might send a packet into a dead end

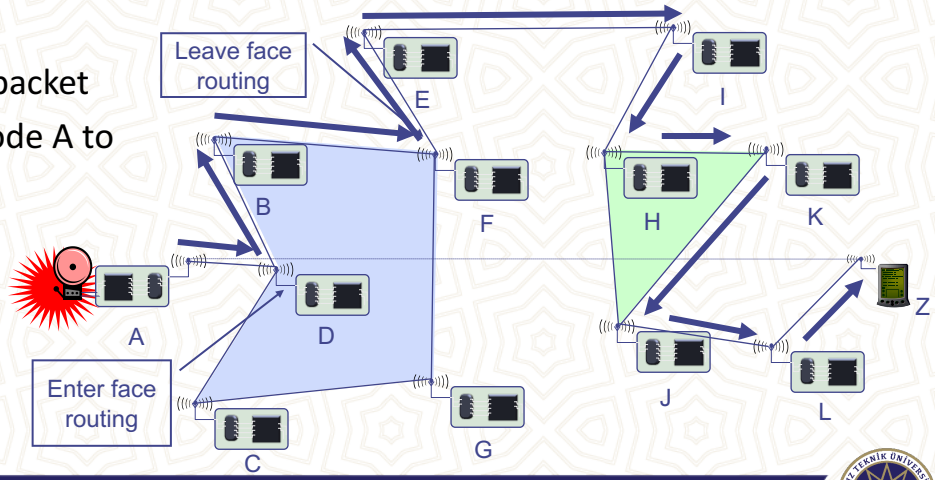


Right Hand Rule To Leave Dead Ends – GPSR

- Basic idea to get out of a dead end: Put right hand to the wall, follow the wall
  - Does not work if on some inner wall – will walk in circles
  - Need some additional rules to detect such circles
- **Geometric Perimeter State Routing (GPSR)**
  - Earlier versions: Compass Routing II, face-2 routing
  - Use greedy, "most forward" routing as long as possible
  - If no progress possible: Switch to "face" routing
    - Face: largest possible region of the plane that is not cut by any edge of the graph; can be exterior or interior
    - Send packet around the face using right-hand rule
    - Use position where face was entered and destination position to determine when face can be left again, switch back to greedy routing
  - Requires: planar graph! (topology control can ensure that)

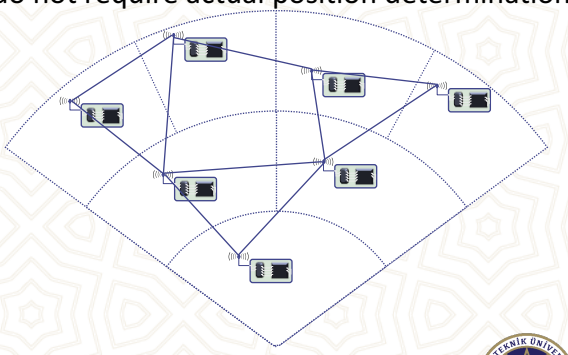
GPSR – Example

- Route packet from node A to node Z



Geographic Routing Without Positions – GEM

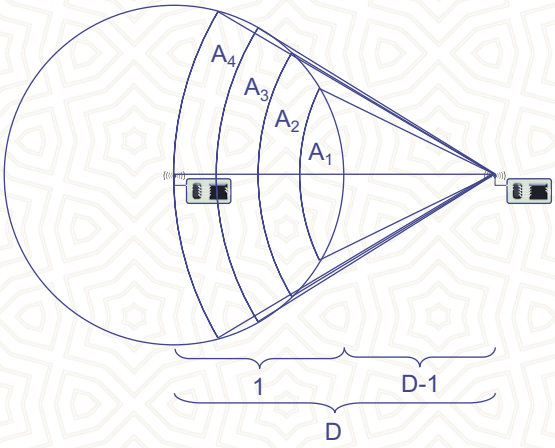
- Apparent contradiction: geographic, but no position?
- Construct **virtual coordinates** that preserve enough neighborhood information to be useful in geographic routing but do not require actual position determination
- Use polar coordinates from a center point
- Assign "virtual angle range" to neighbors of a node, bigger radius
- Angles are recursively redistributed to children nodes



GeRaF

- How to combine position knowledge with nodes turning on/off?
  - Goal: Transmit message over multiple hops to destination node; deal with topology constantly changing because of on/off node
- Idea: **Receiver-initiated** forwarding
  - Forwarding node S simply broadcasts a packet, without specifying next hop node
  - Some node T will pick it up (ideally, closest to the source) and forward it
- Problem: How to deal with multiple forwarders?
  - Position-informed randomization: The closer to the destination a forwarding node is, the shorter does it hesitate to forward packet
  - Use several annuli to make problem easier, group nodes according to distance (collisions can still occur)

GeRaF – Example



Overview

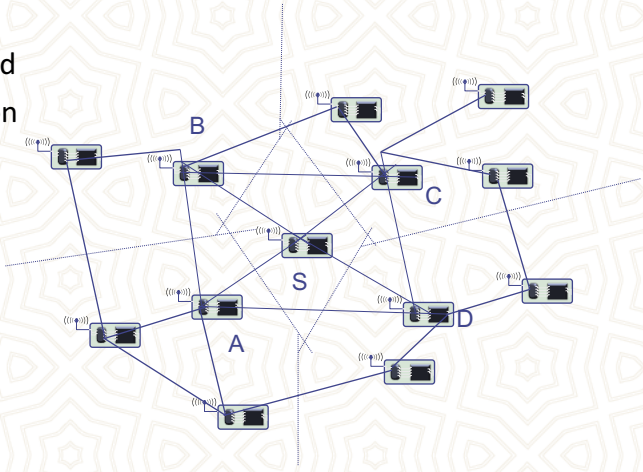
- Unicast routing in MANETs
- Energy efficiency & unicast routing
- Multi-/broadcast routing
- **Geographical routing**
  - Position-based routing
  - **Geocasting**

Location-Based Multicast (LBM)

- Geocasting by geographically restricted flooding
- Define a "forwarding" zone – nodes in this zone will forward the packet to make it reach the destination zone
  - Forwarding zone specified in packet or recomputed along the way
  - Static zone – smallest rectangle containing original source and destination zone
  - Adaptive zone – smallest rectangle containing forwarding node and destination zone
    - Possible dead ends again
  - Adaptive distances – packet is forwarded by node u if node u is closer to destination zone's center than predecessor node v (packet has made progress)
- Packet is always forwarded by nodes within the destination zone itself

Determining Next Hops Based On Voronoi Diagrams

- Goal: Use that neighbor to forward packet that is closest to destination among all the neighbors
- Use Voronoi diagram computed for the set of neighbors of the node currently holding the packet

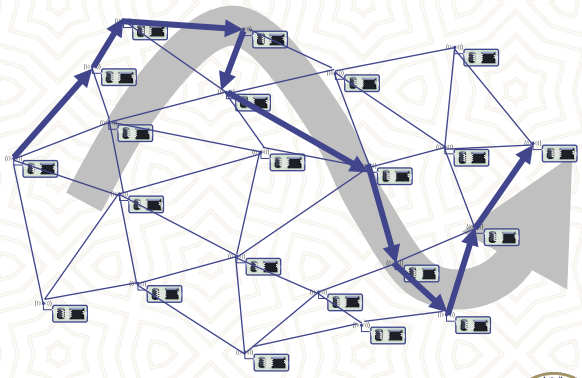


Geocasting Using Ad Hoc Routing – GeoTORA

- Recall TORA protocol: Nodes compute a DAG with destination as the only sink
- Observation: Forwarding along the DAG still works if multiple nodes are destination (graph has multiple sinks)
- GeoTORA: All nodes in the destination region act as sinks
  - Forwarding along DAG; all sinks also locally broadcast the packet in the destination region
- Remark: This also works for anycasting where destination nodes need not necessarily be neighbors
  - Packet is then delivered to some (not even necessarily closest) member of the group

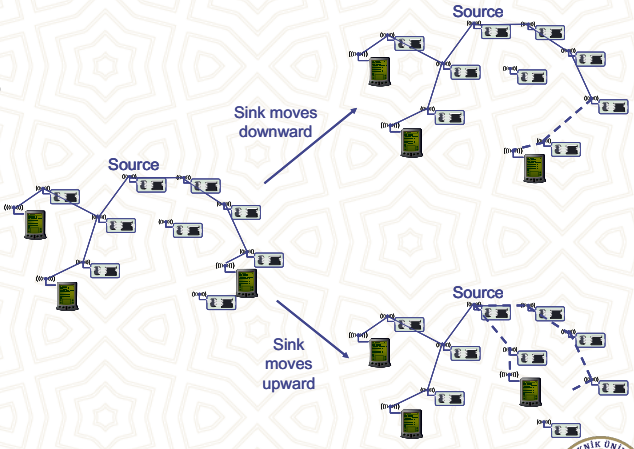
Trajectory-Based Forwarding (TBF)

- Think in terms of an "agent": Should travel around the network, e.g., collecting measurements
  - Random forwarding may take a long time
- Idea: Provide the agent with a certain trajectory along which to travel
  - Described, e.g., by a simple curve
  - Forward to node closest to this trajectory



Mobile Nodes, Mobile Sinks

- Mobile nodes cause some additional problems
  - E.g., multicast tree to distribute readings has to be adapted



Conclusion

- Routing exploit various sources of information to find destination of a packet
  - Explicitly constructed routing tables
  - Implicit topology/neighborhood information via positions
- Routing can make some difference for network lifetime
  - However, in some scenarios (streaming data to a single sink), there is only so much that can be done
  - Energy efficiency does not equal lifetime, holds for routing as well
- Non-standard routing tasks (multicasting, geocasting) require adapted protocols