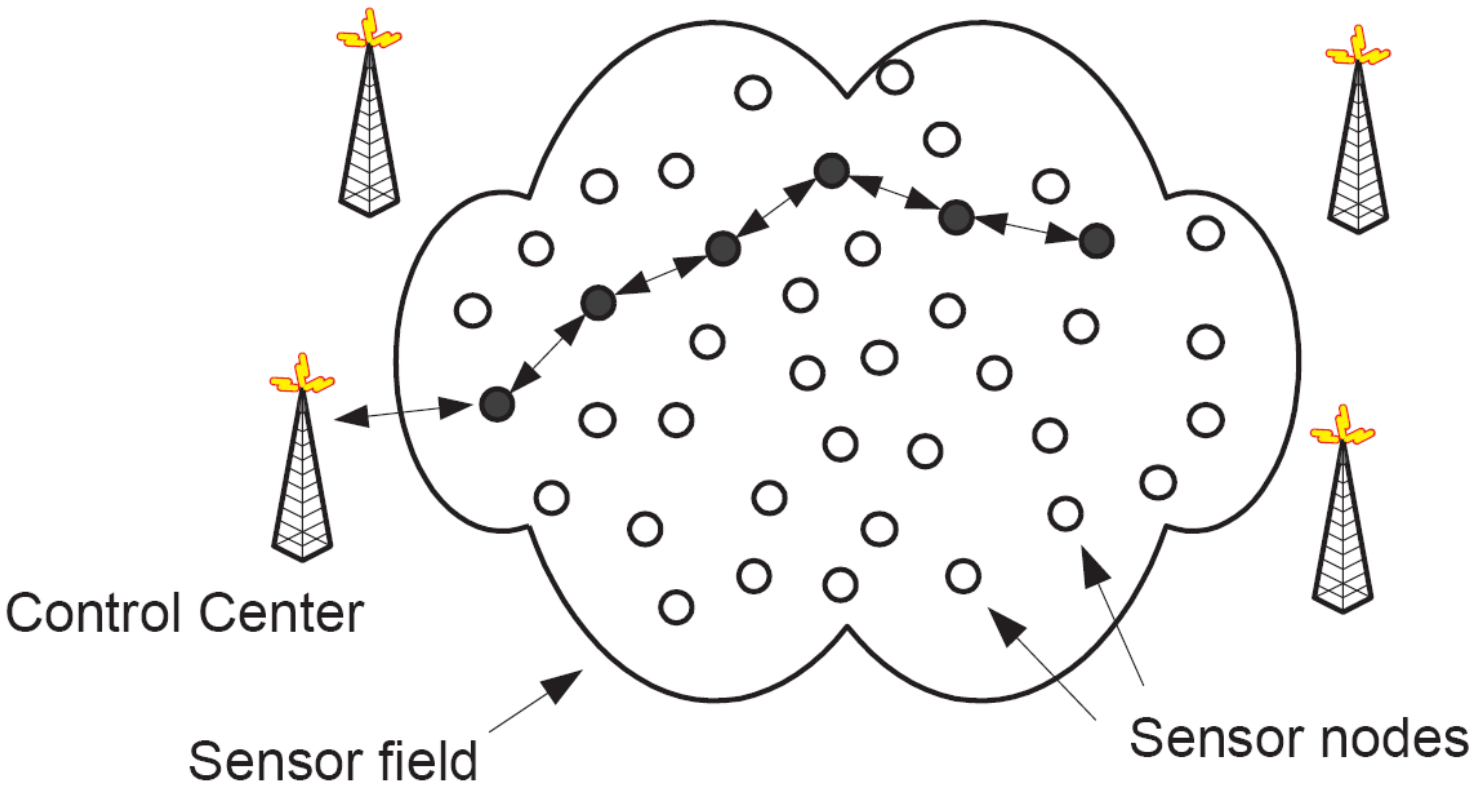
Course “Algorithmic Foundations of Sensor Networks”
Lecture 2: Data propagation algorithms part II

Sotiris Nikoletseas
Professor

Department of Computer Engineering and Informatics
University of Patras, Greece

The problem

"How can sensor p, via cooperation with the rest of the sensors in the network, propagate information about event $\mathcal{E}$ to the control center(s)?"



Lecture Overview

A. Data-centric networking - variants of DD
B. Data gathering with compression
C. Data discovery and querying
D. Detailed presentation of LEACH
E. Detailed presentation of Directed Diffusion

A. Data-Centric Networking

- A fundamental innovation of wireless sensor networking.
- *Basic idea*: routing, storage and querying can all be made more efficient if communication is *based directly on application specific data content* instead of the traditional IP-style addressing, which does not take into account the content requested.
- *Two great advantages* of data-centric networking in terms of energy efficiency:
  - *Communication overhead* for name binding *is minimized*.
  - *In-network processing* as content moves through the network is enabled, via *data aggregation* and *compression*.

Pull Vs Push Diffusion (I)

- The basic version of Directed Diffusion (DD) is actually a *two-phase pull mechanism*:
  - In phase 1, the sink *pulls information from sources* with relevant information via injecting interests
  - In phase 2, *actual data is pulled down* via reinforced path(s).
- An *one-phase pull* variant of DD
  - Eliminate reinforcement as a separate phase.
  - The sink propagates the interest along *multiple paths*.
  - The matching source *directly picks the best of its gradient links* to send data and so on up the reverse path back to the sink.
- Critique:
  - potentially more efficient than two-phase pull DD.
  - Assumes bidirectionality or sufficient knowledge of the links' properties/qualities in each direction.

Pull Vs Push Diffusion (II)

- A second variant: *push diffusion*
  - The sink *does not inject interests*, instead, sources with event detections send exploratory data along multiple paths.
  - The sink, if it has some relevant interests, reinforces one of these paths for data forwarding.
- Critique:
  - The pull and push variants are each most appropriate for different kinds of applications.
  - Pull is more efficient than push whenever there are many sources with high data generation rate but only few, rarely interested sinks.
  - Push is more efficient when there are *few sources which are highly active* but *many, frequently interested sinks*.

TEEN: Another Push-Based Data-Centric Protocol

- It is “threshold-sensitive energy efficient”.
- Nodes *react immediately to drastic changes* in the value of the sensed attribute.
- When this change exceeds a given threshold, the activated nodes communicate their changed value to a cluster-head for forwarding to the sink.

B. Data Gathering With Compression

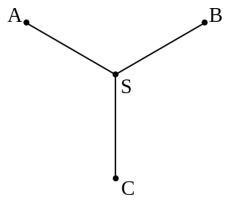
- Main idea: combine routing with *in-network compression*.
- Usual efficiency metric: *reduce total number of data transmissions* i.e. increase cumulative number of bits over each hop of transmission, per round of data-gathering from all sources.
- Compression of *correlated data* may be more efficient than simple clustering techniques (e.g. LEACH) that are not correlation aware.

On the Impact of Compression

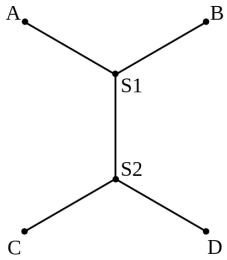
- An extreme case: the data from *any number of sources can be combined into a single packet* (e.g. suppression of duplicate data, when the sources generate identical data).
- If there are k sources, all located close to each other and far from the sink, then a route than combines their information close to the sources can achieve a $k-$*fold reduction* in transmissions, compared to each node sending its information separately without compression.
- In general, the optimal joint routing-compression structure for this case is a *minimum Steiner tree construction problem*, which is NP-hard. However, there exist polynomial solutions for special cases *where the sources are close to each other*.

The minimum Steiner tree problem (it generalises the minimum spanning tree problem)

Given an undirected graph with non-negative edge weights and a subset of vertices, usually referred to as terminals, the Steiner tree problem in graphs requires a tree of minimum weight that contains all terminals (but may include additional vertices).



Steiner tree for three points A, B, and C. The Steiner point S is located at the Fermat point of the triangle ABC, a point such that the sum of the three distances from each of the three vertices of the triangle to the point is the smallest possible.



Solution for four points—there are two Steiner points, S1 and S2.

Network Correlated Data Gathering (I)

- Consider the case when all nodes are sources but *the level of the correlation can vary*.
- When the data is *completely uncorrelated*, then clearly the *shortest path tree (SPT)* provides the best solution (minimizing the total transmission cost).
- In the general case, assume the following *correlation model*:
  - Only *nodes at the leaves* of the tree (whose root is the sink) need to provide *R bits*.
  - All *interior nodes*, which have side information from other nodes, need *only generate r bits* ($r \leq R$) of additional information.
  - The quantity

$$\rho = 1 - \frac{r}{R}$$

    is called the *correlation coefficient*.

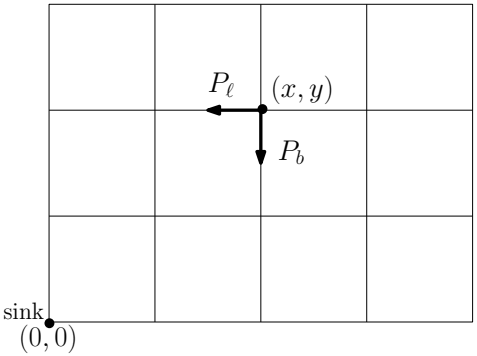
Network Correlated Data Gathering (II)

- As ρ increases, *a travelling salesman path (TSP)* provides an arbitrarily more efficient solution compared with SPT. However, TSP is NP-hard.
- *Good approximations* are given with the following *hybrid combinations* of STP and TSP:
  - All nodes *within some range from the sink* (larger the ρ, smaller this range) are connected *via SPTs*.
  - *Beyond them, an approximate TSP path is constructed* by adding nearby nodes to each strand of the SPT.

Scale-free Aggregation (I)

- In practice, *the degree of spatial correlation is a function of distance* and *nearby nodes are able to provide high compression* than nodes at a greater distance.
- A certain model for spatial correlation considers a *square grid of sensors*, assuming that grid nodes have information about all readings of all nodes within a $k-$hop radius.
- Grid nodes can communicate with any of their four grid neighbours, and aggregation/compression is performed by *suppressing redundant readings in the intermediate hops* towards the sink.

Scale-free Aggregation (II)



- The sink is located at the bottom-left corner and randomized routing of compressed data is performed at a (x, y) node as follows:

$$P_\ell = \frac{x}{x + y} \quad \text{forward to left grid node}$$

$$P_b = \frac{y}{x + y} \quad \text{forward to bottom grid node}$$

- It is shown that a constant factor approximation (in expectation) to the optimal solution is achieved.

C. Data Discovery and Querying

- Besides end-to-end routing, *data discovery and querying* form an important communication primitive in sensor networks.
- The goal is to design efficient alternatives to the high-overhead naive flooding based querying (FBQ), especially when it is not needed to provide continuously information to the sink but rather the sink is interested in a small part of the sensed data. In such cases, the sensed data may be stored locally and *only transmitted in response to a query* issued by the sink.
- Several querying techniques have been proposed, such as:
  - expanding ring search
  - rumor routing
  - the comb-needle technique

Types of Queries (I)

1. *Continuous vs one-shot queries*, (i.e. queries for a long duration data flow or a single piece of data).
2. *Simple vs complex queries*. Complex queries:
   - *combinations* of multiple simple subqueries (e.g. what are the locations of nodes where (a) the light intensity is at least x and the humidity is at least y or, (b) the light intensity is at least z)
   - aggregate queries requiring *information from several sources* e.g. “find the average temperature readings from all nodes in a certain region *R*".
3. Queries for *replicated data* (available at multiple nodes) *vs queries for unique data* at a single node only.
4. *Queries for historic vs current/future data*.

Types of Queries (II)

Note:

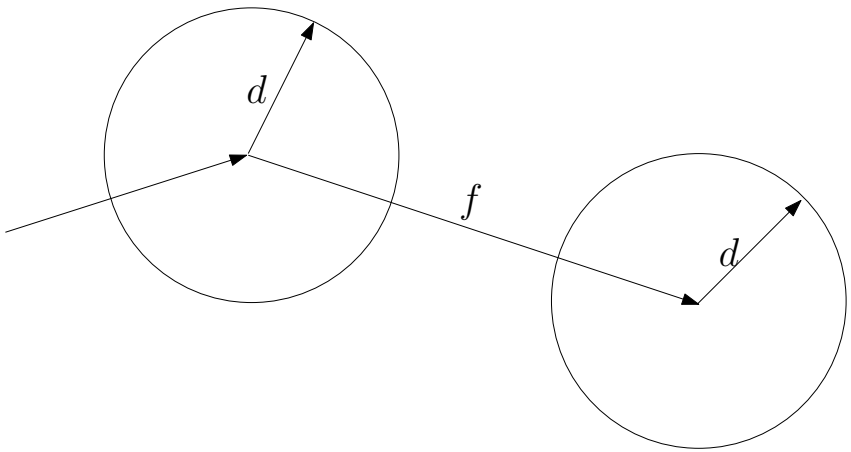
- In truly continuous queries, the cost of initial querying (even via naive flooding) may be relatively insignificant.
- However, for one-shot data the cost and overhead of flooding may be prohibitively expensive.
- Also, in queries for replicated data, flooding techniques may return multiple copies of the same data when only one copy is necessary.

Expanding Ring Search

- It proceeds as *a sequence of controlled floods*, with the radius of the flood increasing at each step *if the query has not been resolved* at the previous step.
- The choice of *the number of the hops to search at each step* can be optimized to minimize the expected search cost, using a dynamic programming technique.
- When there is *enough replicated/cached information*, the method is more likely to improve performance a lot. However, in the absence of replicated/cashed data, the energy saving achieved is marginal (less than 10%), while the delay increases a lot, and other methods are more relevant.

Active Query Forwarding (ACQUIRE) (I)

- It treats the query as an “intelligent entity" that moves in the network towards the desired response, as a repeated sequence of three parts.
  - *Examine cache*: Arriving at a node the query first checks its existing cache to see if it is valid/fresh enough to directly resolve the query there.
  - If not, the query performs a *Request update* from nodes within a $d-$hop neighbourhood (via controlled flood).
  - *Forward phase*: If not resolved, the query is forwarded to another active node, located a sufficient number f of hops away, so that the controlled flood phases (request updates) do not overlap a lot.



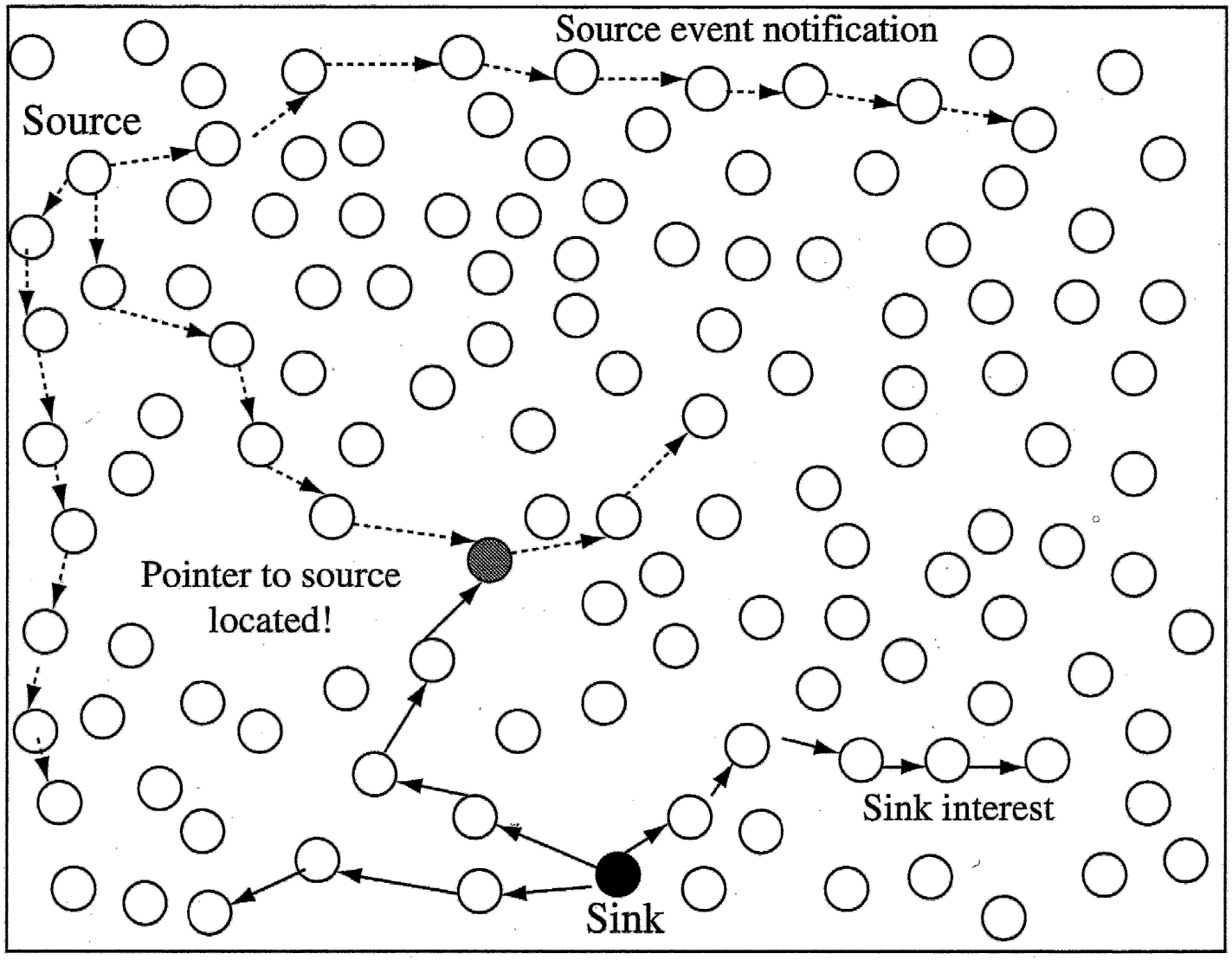
Active Query Forwarding (ACQUIRE) (II)

- The *look-ahead parameter d* offers a tunable trade-off: when *d* is small, the query is forwarded in a trajectory-based manner (like a random walk), but when *d* is large (in terms of the network diameter), the method resembles a flood. When *d* is small, the query needs to be forwarded more often but there are fewer update messages at each step, while for large *d*, fewer forwarding steps are needed but each one having more update messages.
- The optimal choice of *d* depends mostly *on the sensor data dynamics* captured e.g. by the ratio of the data change rate in the network to the query generation rate. When the data dynamics is low, caches remain valid for a long time and the cost of large *d* can be amortized over several queries. However, when the data dynamics is very high, repeated flooding is required and a small *d* should be chosen.

Rumor Routing (I)

- It provides an efficient *rendezvous mechanism* to combine push and pull approaches to get the desired information from the network.
- Sinks desiring information *send queries* through the network. Sources generating important events *send notifications* through the network.
- Both are treated as *mobile agents*. The event notifications *leave "trails"* for query agents visiting a node with a trail from an event notification agent to be able to *find pointers to the corresponding source*.
- The trajectories of both the events and the queries agents can be either random walks (with built-in loop-prevention) or more directed e.g. straight lines.
- Substantial energy savings can be obtained compared with the two extremes of query flooding (pull) and event flooding (push).

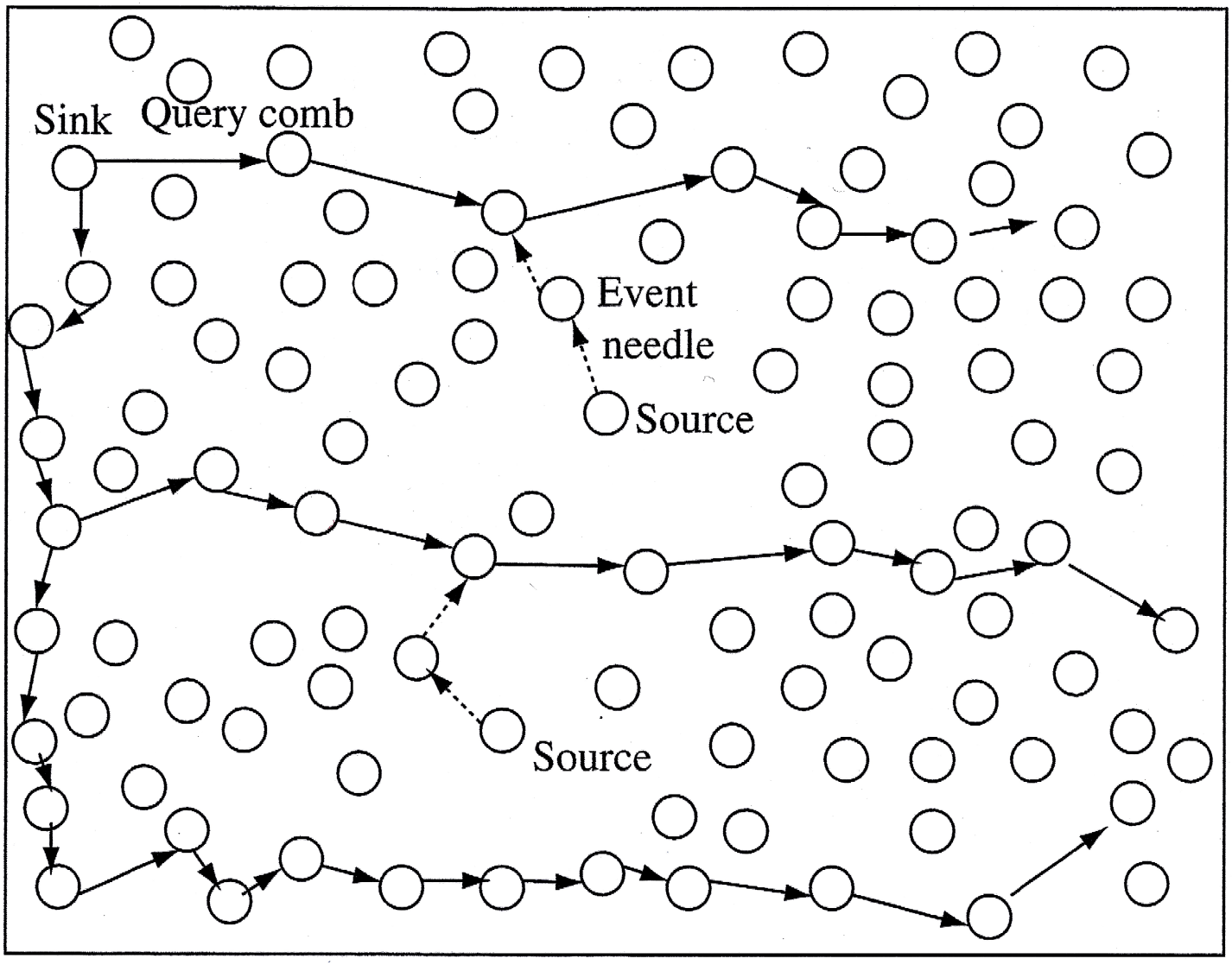
Rumor Routing (II)



The Comb-Needle Technique (I)

- Similar *combination of push and pull* based on intersections of queries and event notifications.
- Basic version: The *queries* build a *horizontal comb-like* routing structure, while events follow a vertical needle-like trajectory to meet the “teeth" of the comb.
- A tunable key parameter is the *spacing* between branches of the comb and correspondingly the length of the event notifications trajectory:
  - the spacing and length are chosen smaller when the event-to-query ratio is high ($\Rightarrow$ more pull, less push)
  - when the event-to-query ratio is low the spacing and length should be higher ($\Rightarrow$ less pull, more push)
  - in adaptive versions the spacing and length are *adjusted dynamically and distributively* by the sources and sinks based on local estimates of data and query frequencies to address their fluctuations over time.

The Comb-Needle Technique (II)



D. Flat vs Hierarchical Routing

- *Flat Routing*: All nodes in the network have similar role regarding the routing of data. No special nodes are used.
  - Example: Directed Diffusion
- *Hierarchical Routing*: Special nodes assume greater responsibility regarding routing of data than most nodes inside the network. Super nodes – cluster heads.
  - Example: LEACH

W.R. Heinzelman, **A. Chandrakasan** & **H. Balakrishnan**

“Energy-Efficient Communication Protocol for Wireless Microsensor Networks"

33rd Hawaii International Conference on System Sciences, HICCS-2000

Main features of LEACH

What is LEACH?

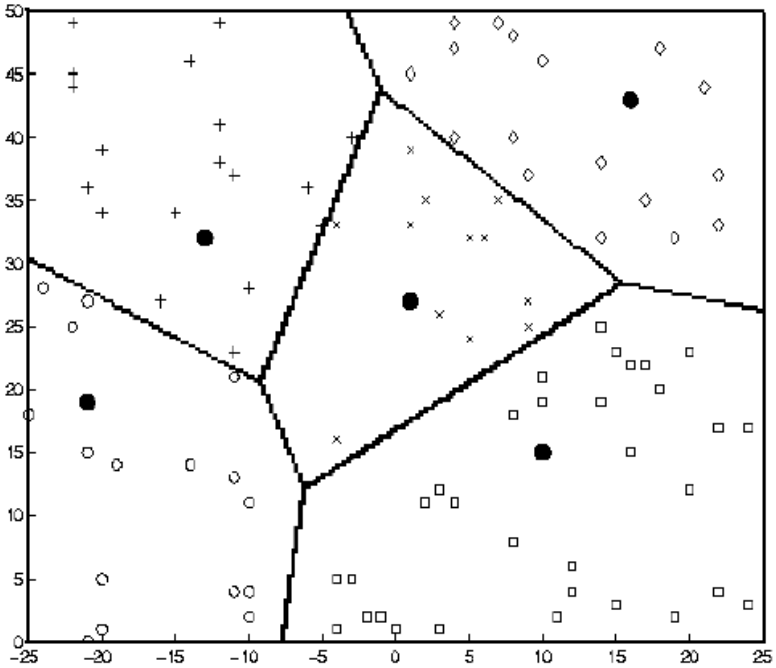
- *cluster-based* protocol that *minimizes* energy dissipation in sensor network.
- key features:
  - *localized* coordination and control for cluster set-up and orientation.
  - *randomized rotation* of the cluster “base station" or “cluster heads" and the corresponding clusters.
  - *local compression* to reduce global communication.

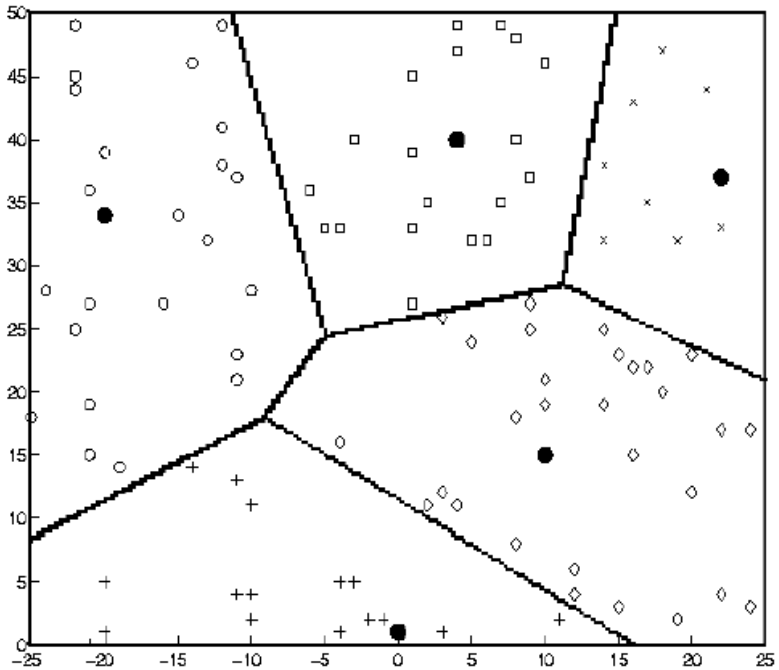
Intuitive description of LEACH

How does LEACH work:

- Network is partitioned in *clusters*.
- Each cluster has one *cluster-head*.
- Each non *cluster-node* sends data the head of the cluster it belongs.
- Cluster-heads gather the sent data, compress them and sends them to the base-station *directly*.

Dynamic Clusters





Operation of LEACH

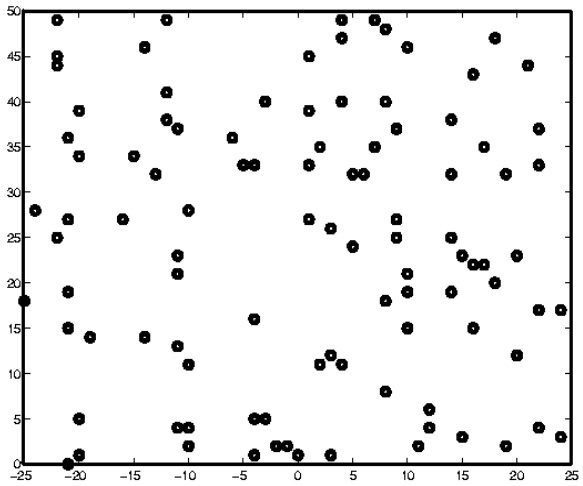
- LEACH operates in *rounds*.
- Each round in LEACH consists of *phases*.
  - Advertisement Phase.
  - Cluster Set-Up Phase.
  - Steady Phase.

Advertisement Phase (1/2)

Election: Node *n* decides with probability T(n) to elect itself *cluster-head*

$$T(n) = \begin{cases} \frac{P}{1-P*(r \bmod \frac{1}{P})} & \text{if } n \in G \\ 0 & \text{otherwise} \end{cases}$$

P: the desired percentage of cluster heads.
r: the current round.
G: the set of nodes that have not been cluster-heads in the last $\frac{1}{P}$ rounds.



UNIVERSITY OF PATRAS

Advertisement Phase (2/2)

Cluster-Head-Advertisement:

- Each cluster-head *broadcasts* an advertisement message to the rest nodes using CSMA-MAC protocol.
- Non cluster-head nodes hear the advertisements of all cluster-head nodes.
- Each non-cluster head node decides each cluster head by choosing the one that has the stronger signal.

Cluster Set-Up Phase

- Each cluster head is informed for the members of its cluster.
- The cluster head creates a *TDMA schedule*
- Cluster-head broadcasts the schedule back to the cluster members.

Steady Phase

- Non cluster-heads
  - *Senses* the environment.
  - *Send* their data to the cluster head during their transmission time.
- Cluster-heads
  - *Receives* data from non cluster-head nodes.
  - *Compresses* the data it has received.
  - *Send* its data to the base station.

The Duration of steady phase is “a priori” determined.

Hierarchical Clustering Extensions

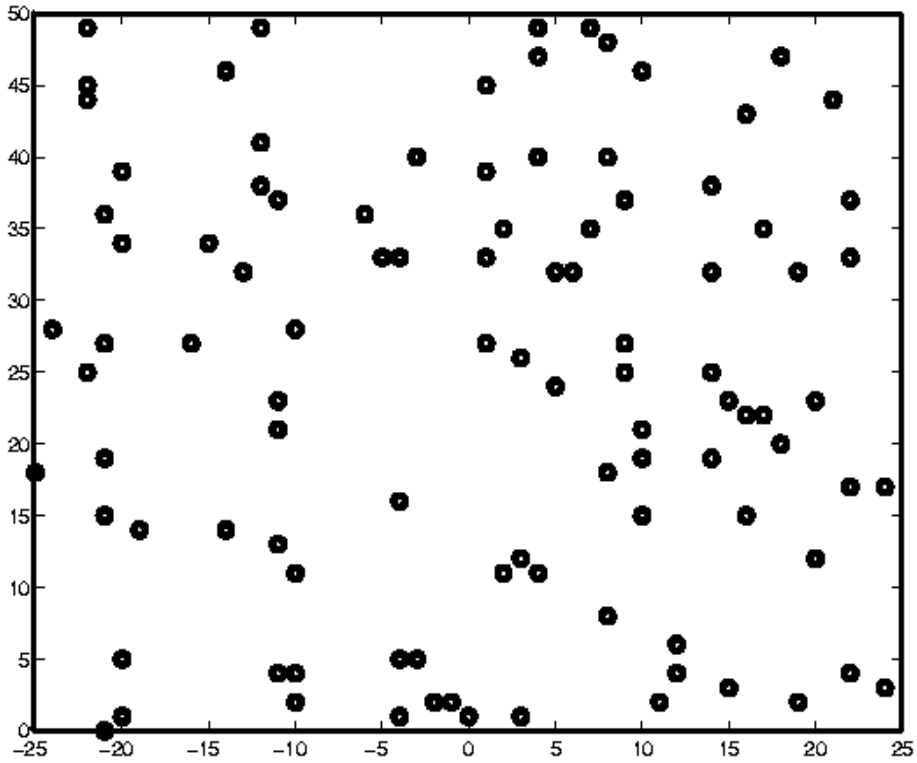
- Non cluster-heads communicate with their cluster-heads
- Cluster-heads communicate with *super-cluster-heads*
- And so on . . .

Advantages: Saves a lot of energy for larger networks, more realistic

Disadvantages: More complicated implementation, latency

Experimental Evaluation of LEACH

We are given a network of *100 nodes*.
In the area:



The *base station* is placed at $(0, 50)$.

Experimental Evaluation of LEACH

A comparative study between:

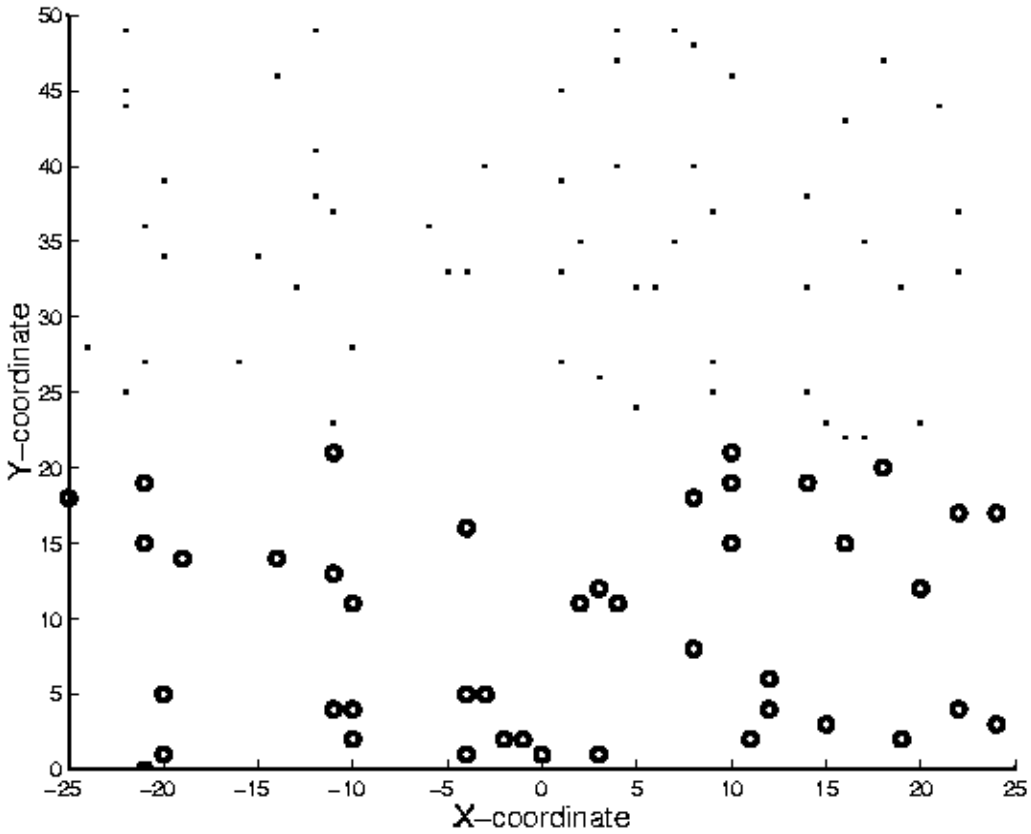
- Minimum-Energy-Transmission (muti-hop).
- Direct Transmission.
- LEACH.

The evaluation measures:

- Dead Nodes' *Distribution*.
- Total *Energy Dissipation*.
- System *Life-time* (time until first node dies).

Distribution of dead nodes (1/3)

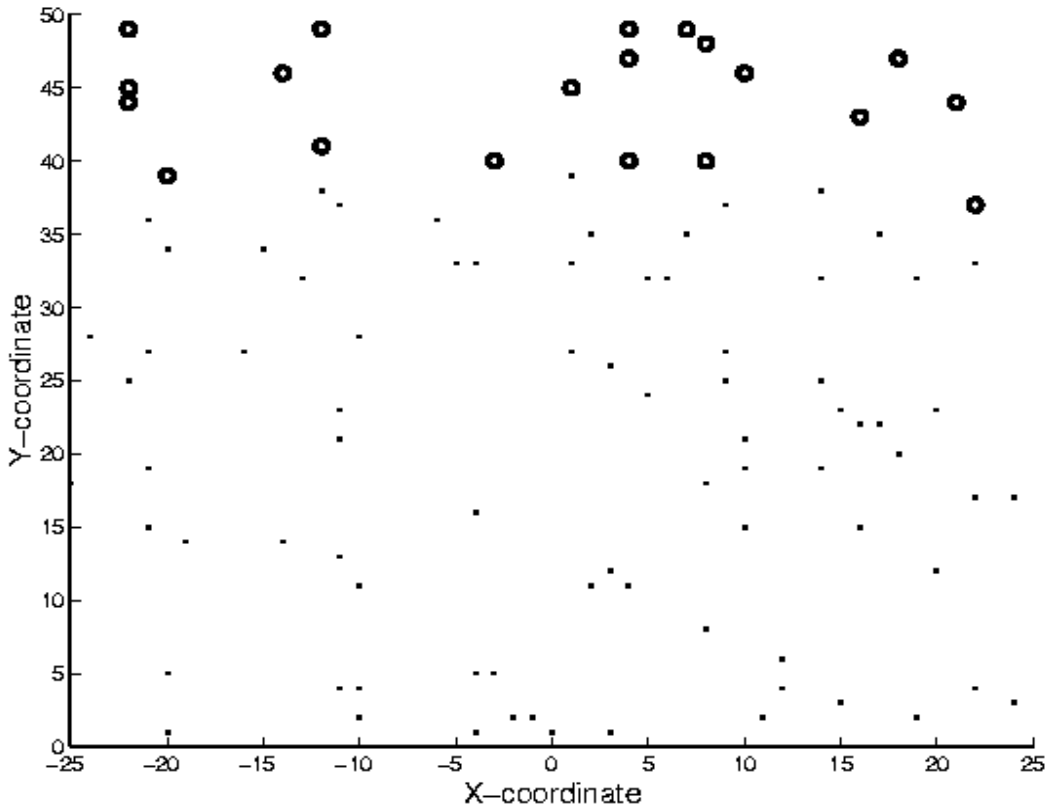
Minimum-Energy-Transmission



- After 180 rounds nodes *closer* to the sink die faster!

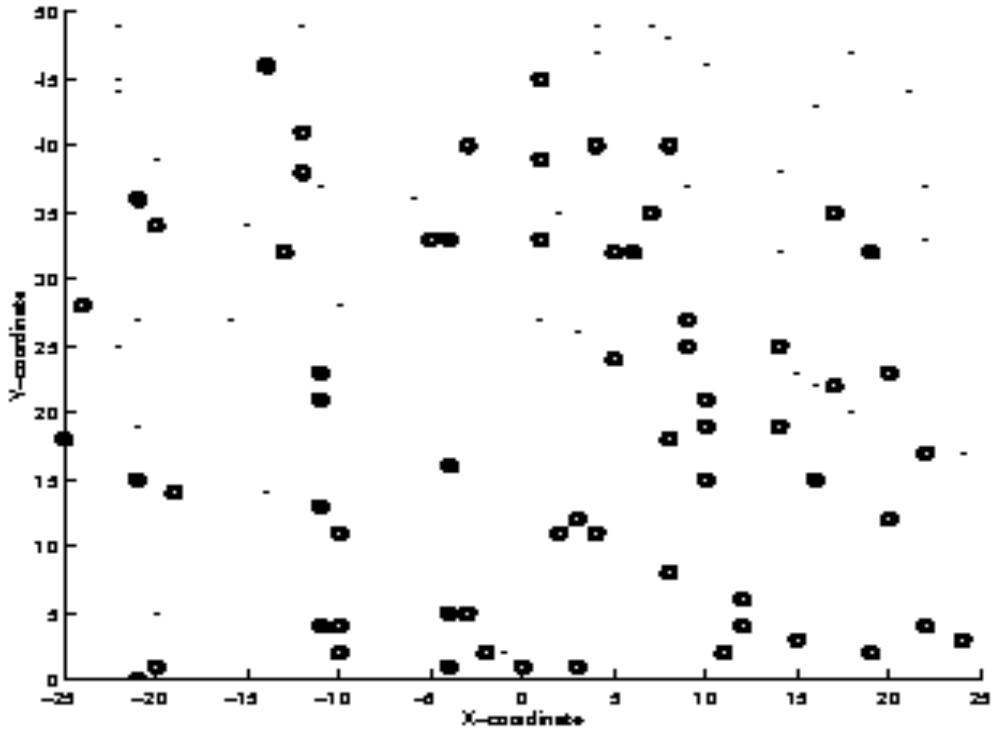
Distribution of dead nodes(2/3)

Direct-Transmission



- After 180 rounds nodes *further* to the sink die faster!

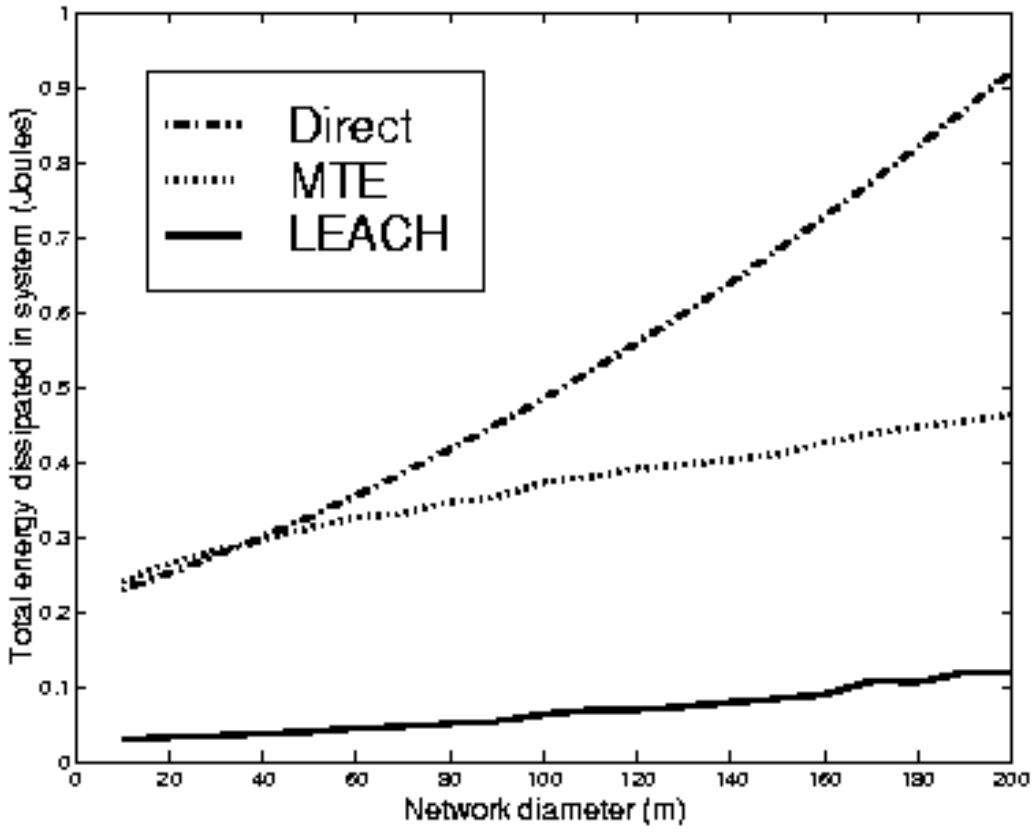
Distribution of dead nodes(3/3)



LEACH

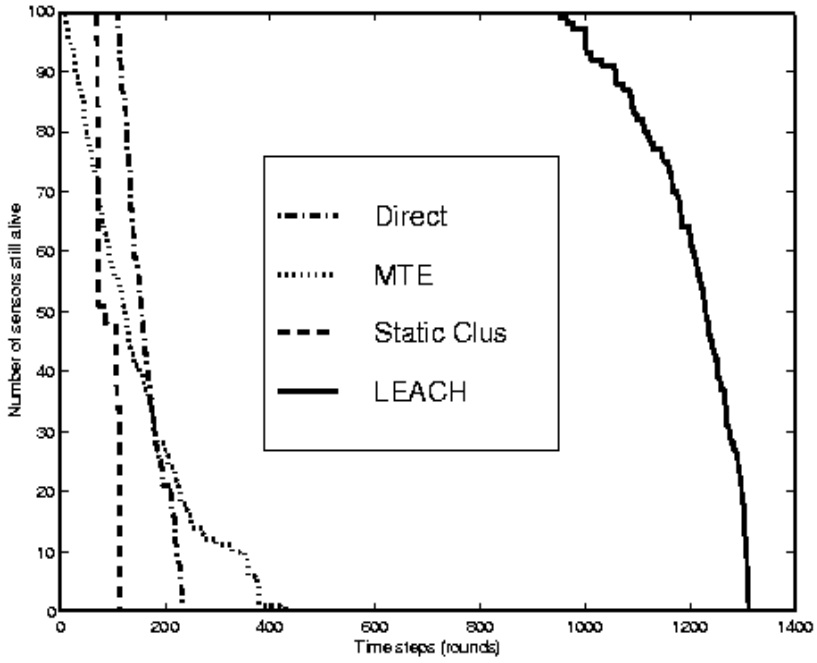
- After 1200 rounds nodes die in *uniform* manner.

Total energy dissipation - network diameter



- *LEACH* reduces 7x to 8x compared to *Direct-Transmission*.
- *LEACH* reduces 4x to 8x compared to *Minimum-Energy-Transmission*.

System Lifetime



- LEACH more than *doubles* the useful system lifetime.
- It takes *8-times* longer for the first node to die in LEACH.
- It takes *3-times* longer for the last node to die in LEACH.

E. Directed Diffusion

C. Intanagonwiwat, R. Govindan, D. Estrin

"Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks"

6th Annual International Conference on Mobile Computing and Networking, 2000

Directed Diffusion

Directed diffusion elements:

- Interest messages
- Data messages
- Gradients
- Reinforcements of gradients

Interests and tasks

- An interest contains the description of a sensing task.
- Task descriptions are named e.g. by a list of attribute-value pairs.
- The description specifies an interest for data matching the attributes.

```
type = wheeled vehicle        // detect vehicle location
interval = 100 ms              // send events every 100 ms
duration = 10 seconds         // for the next 10 seconds
rect = [-100, 100, 200, 400] // from nodes within rectangle
```

Example of an interest

Interests

Interests are injected into the network at some (possibly arbitrary) node, the sink.
The sink diffuses the interests through the sensor network.

- For each interest a task is generated.
- For each active task the sink generates an exploratory interest message.

```
type = wheeled vehicle
interval = 0.1s
rect = [-100, 100, 200, 400]
timestamp = 01:20:40
expiresAt = 01:30:40
```

Note that:

- An interest is periodically refreshed by the sink.
- Interests do not contain information about the sink.
- Interests can be aggregated e.g. interests with identical types and completely overlapping `rect` attributes.

Interests

Every node maintains an interest cache.
Upon interest reception a new entry is created in the cache.

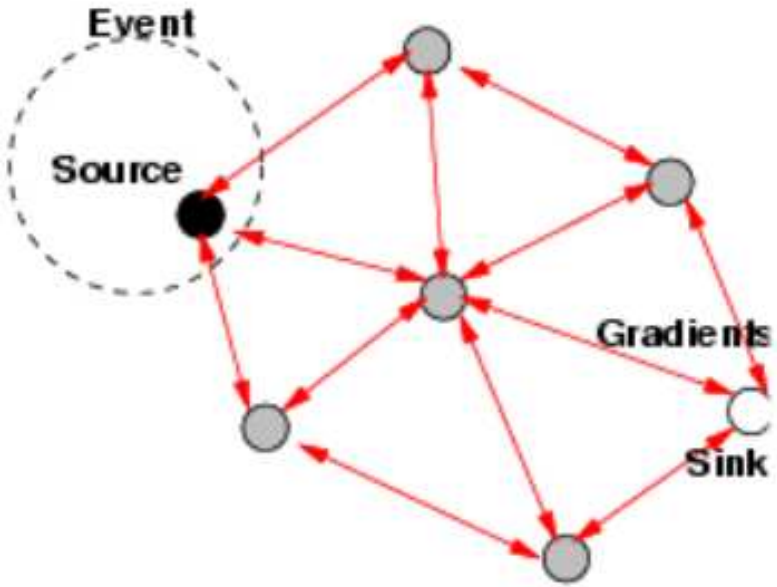
- A timestamp that stores the timestamp of the last received matching interest.
- A gradient entry, up to one per neighbor, is created. Each gradient stores:
  - Data rate.
  - Duration.

Note that:

- If an interest already exists in cache only a new gradient is created for this interest.
- An interest is erased from cache only when every gradient has expired.

Gradients

Gradients are formed by local interaction of neighboring nodes.
Neighboring nodes establish a gradient towards each other.
Gradients store a value and a direction.
Gradients facilitate "pulling down" data towards the sink.



Gradient establishment when flooding an interest.

Data propagation

A sensor that receives an interest it can serve, begins sensing. As soon as a matching event is detected:

- The node computes the highest requested event rate among it's gradients.
- The node generates event samples at this rate.

```
type = wheeled vehicle   // type of vehicle
instance = truck         // instance of this type
location = [125, 220]    // node location
intensity = 0.6          // signal amplitude measure
confidence = 0.85        // confidence in the match
timestamp = 01:20:40     // local event generation time
```

- Data messages are unicasted individually to the relevant neighbors (neighbors where gradients point to).

Data propagation

Every node maintains a data cache.
A node receives a data message.

- If the data message doesn't have a matching interest or data exists in cache, the message is dropped.
- Otherwise, the message is added to the data cache and is forwarded.

To forward a message:

- A node determines the data rate of received events by examining it's data cache.
- If the requested data rate on all gradients is greater or equal to the rate of incoming events, the message is forwarded.
- If some gradients have lower data rate then the data is down-converted to the appropriate rate.

Reinforcement for Path Establishment and Truncation

1. The sink initially repeatedly diffuses an interest for a low-rate event notification, the generated messages are called exploratory messages.
2. The gradients created by exploratory messages are called exploratory gradients and have low data rate.
3. As soon as a matching event is detected, exploratory events are generated and routed back to the sink.
4. After the sink receives those exploratory events, it reinforces one particular neighbor in order to “draw down" real data.
5. The gradients setup for receiving high quality tracking events are called data gradients.

Path Establishment Using Positive Reinforcement

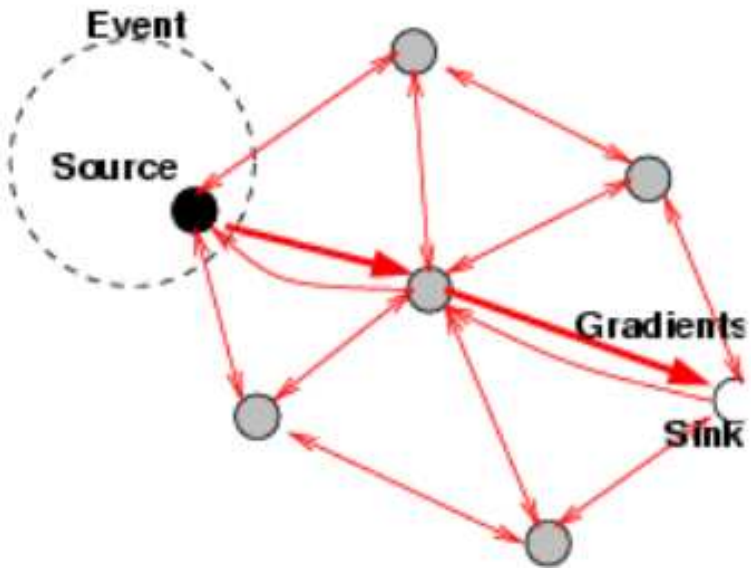
To reinforce a neighbor, the sink re-sends the original interest message with a smaller interval (higher rate).

```
type = wheeled vehicle
interval = 10ms
rect = [-100, 100, 200, 400]
timestamp = 01:22:35
expiresAt = 01:30:40
```

Upon receiption of this message a node updates the corresponding gradient to match the requested data rate.
If the requested data rate is higher than the rate of incoming events, the node reinforces one of it's neighbors.

Path Establishment Using Positive Reinforcement

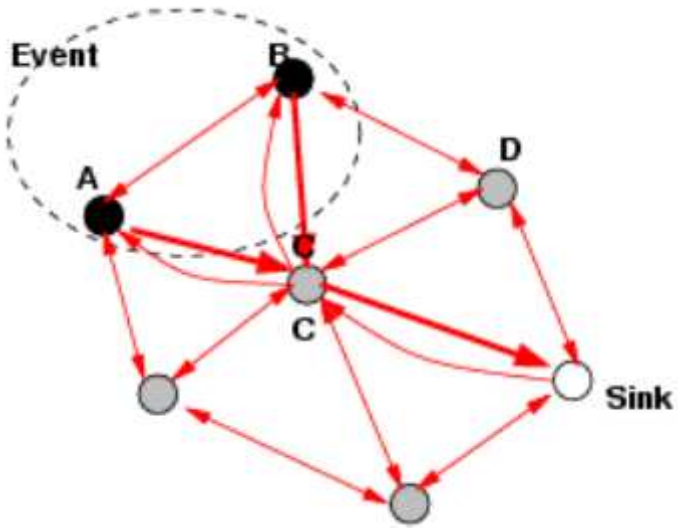
The selection of a neighbor for reinforcement is based on local criteria
i.e. the neighbor that reported first a new event is reinforced.
The data cache is used to determine which criteria are fulfilled.

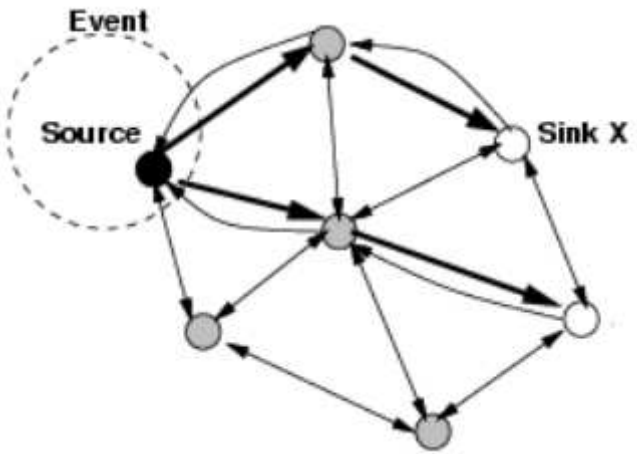


Gradient reinforcement.

Reinforcement for Path Establishment and Truncation

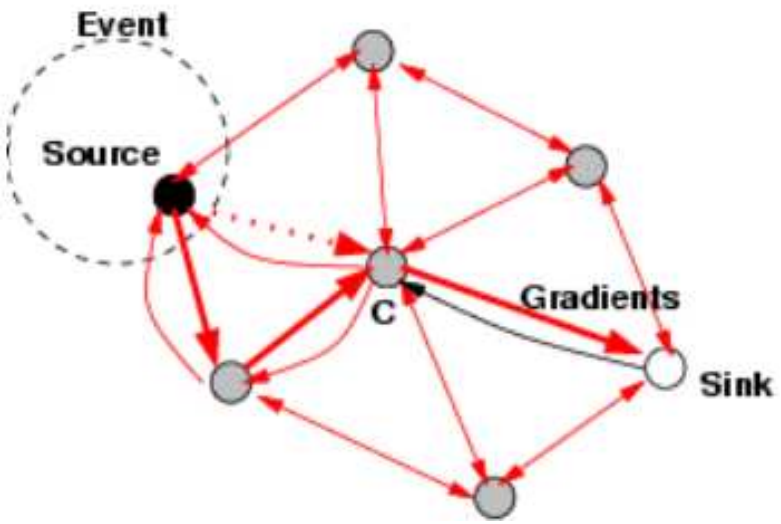
The above scheme is reactive to changes; whenever one path delivers an event faster than others, it is reinforced.
This scheme supports the existence of multiple sinks and multiple sources in the network.





Local Repair for Failed Paths

Paths may degrade over time.
A node can detect path degradation i.e. by noticing reduced event rates.
Intermediate nodes on a path can apply the reinforcement rules and repair the path.



Negative Reinforcement

A mechanism for truncating paths is required.

- Gradients timeout unless they are explicitly reinforced.
- Negative reinforcement of a path.

Negative reinforcement is achieved by sending an interest with exploratory data rate.
If all outgoing gradients of a node are exploratory the node negatively reinforces it's neighbors.
Negative reinforcement is applied when certain criteria are met i.e. a gradient doesn't deliver any new messages for an amount of time.

Performance Evaluation

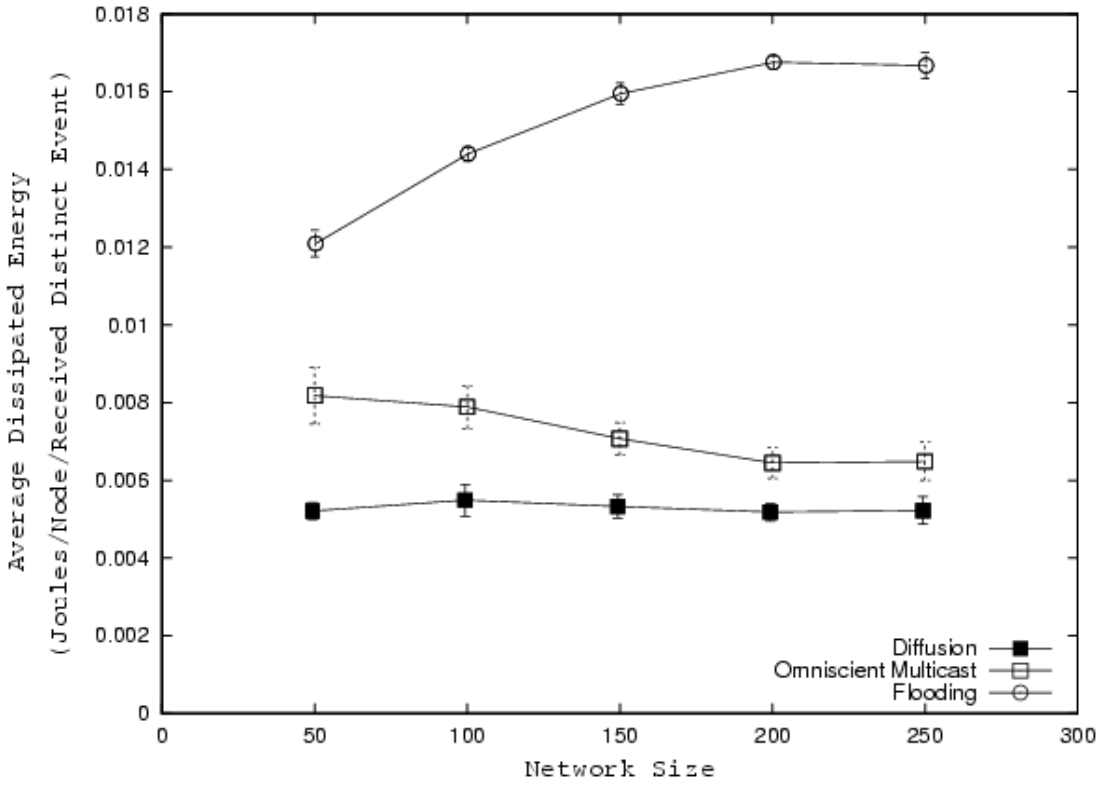
- Flooding
- Omniscient Multicast
  - Uses minimum height multicast trees
- Directed diffusion

Simulation

Flooding, Omniscient Multicast and Directed Diffusion were simulated on ns-2.
Metrics:

- Average dissipated energy.
- Average delay.
- Distinct-event delivery ratio.

Average dissipated energy

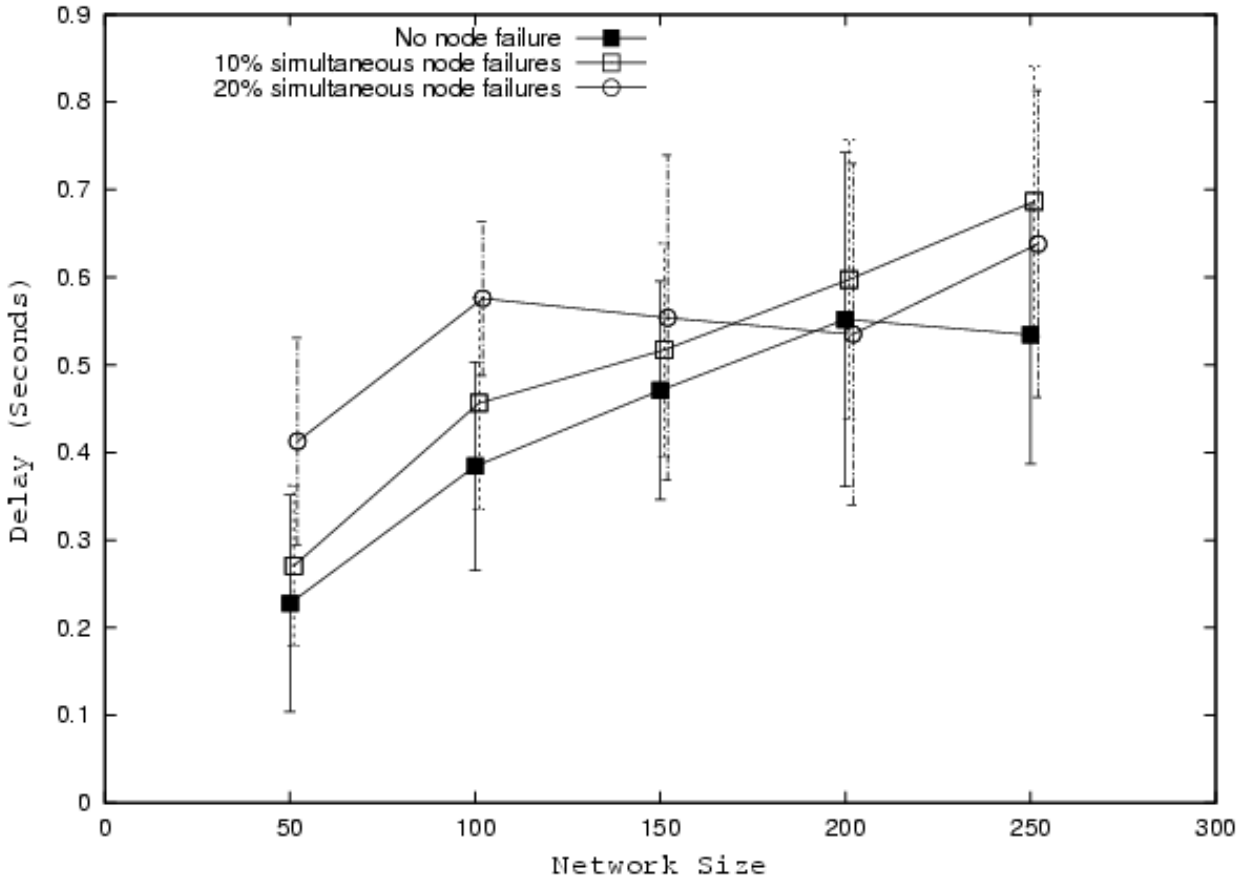


- Omniscient multicast dissipates significant less energy than flooding since events are delivered along a single path.
- Directed diffusion outperforms omniscient multicast as in-network aggregation suppresses duplicate messages.

Impact of dynamics

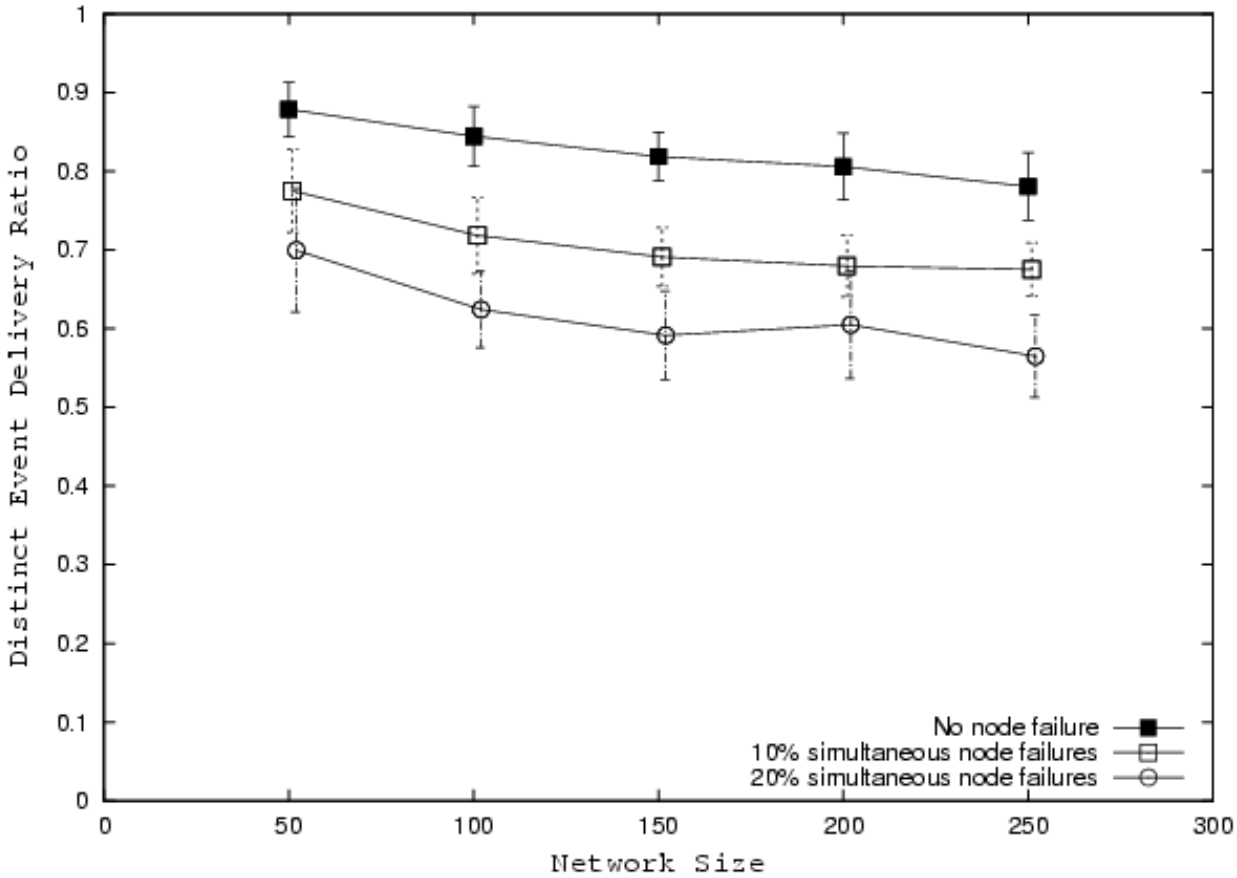
- Node failures occur randomly in the network.
- Half of node failures occur on nodes on the shortest path trees.
- Sources send different location estimates.

Average delay



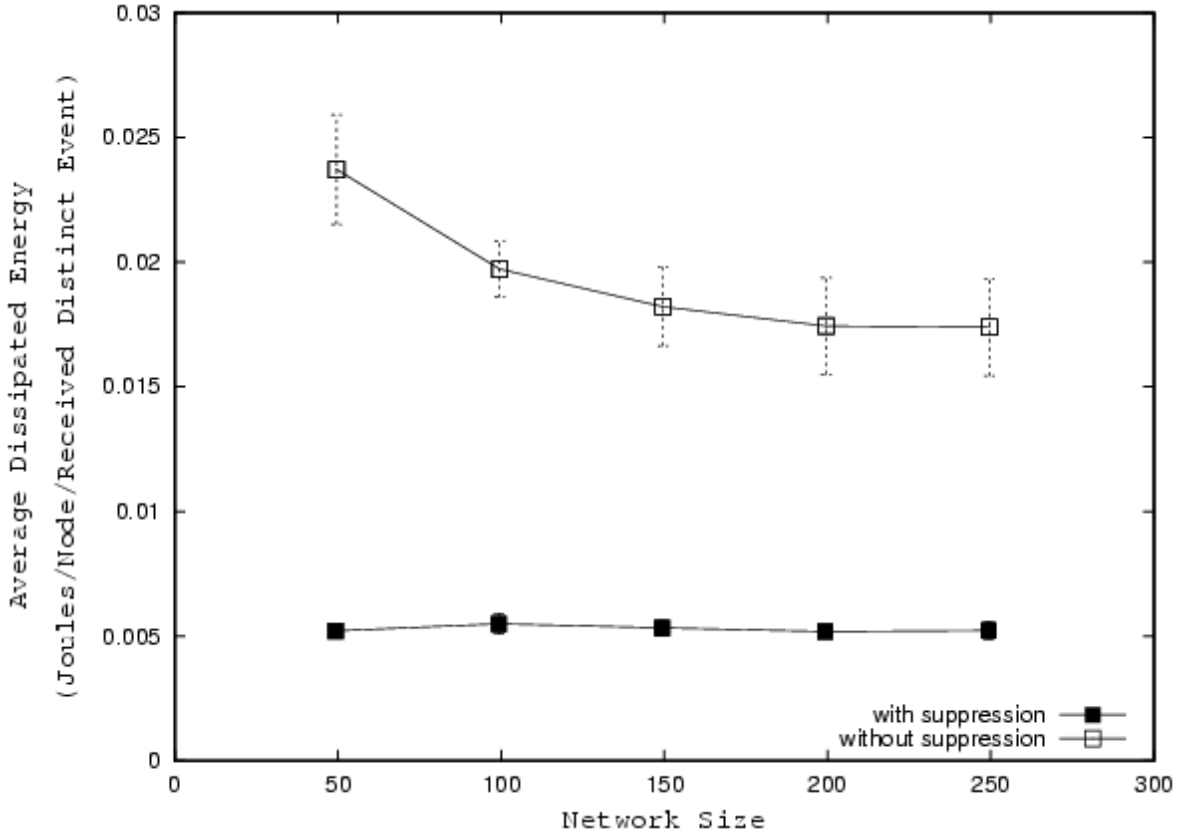
- Average delay increases but not more than 20%.

Event delivery ratio



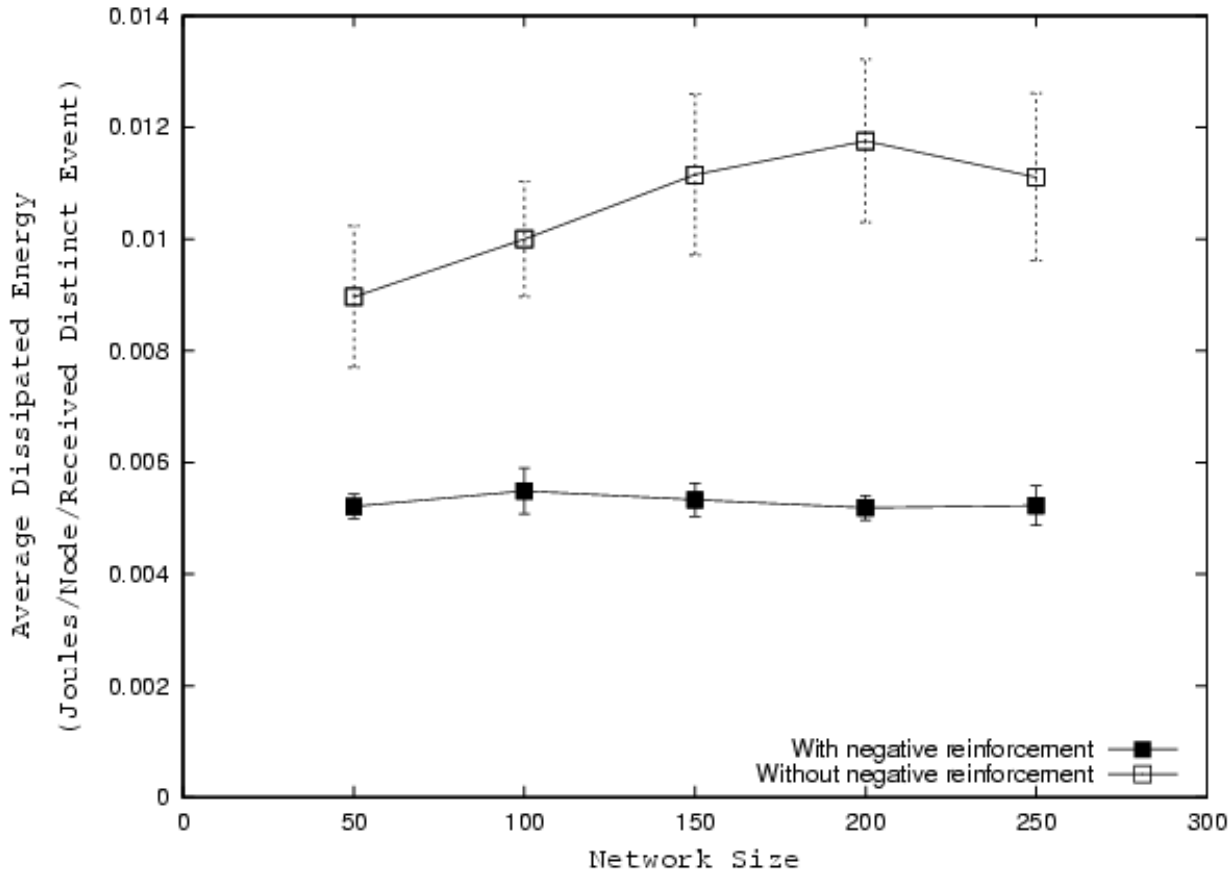
- Event delivery ratio reduces proportionally to node failure percentage.

Impact of Data Aggregation



- Without aggregation, diffusion dissipates 3 (for larger networks) to 5 times (small network sizes) more energy.
- Longer (higher latency) alternative paths form in large networks, which are pruned by negative reinforcement.

Impact of Negative reinforcement



- Without negative reinforcement 2 times more energy is dissipated.