Rumor Routing Algorithm For Sensor Networks.

David Braginsky (daveey@ucla.edu), Deborah Estrin.

Abstract

Advances in micro-sensor and radio technology will enable small but smart sensors to be deployed for a wide range of environmental monitoring applications. In order to constrain communication overhead, dense sensor networks call for new and highly efficient methods for distributing queries to nodes that have observed interesting events in the network. A highly efficient data-centric routing mechanism will offer significant power cost reductions [6], and improve network longevity. Moreover, because of the large amount of system and data redundancy possible, data becomes disassociated from specific node and resides in regions of the network [1][3][8]. This paper describes and evaluates through simulation a scheme we call Rumor Routing, which allows for queries to be delivered to events in the network. Rumor Routing is tunable, and allows for tradeoffs between setup overhead and delivery reliability. It's intended for contexts in which geographic routing criteria are not applicable because a coordinate system is not available or the phenomenon of interest is not geographically correlated.

1. Introduction

The emerging low-power and small form-factor processors, equipped with wireless communication capabilities and sensors allow for large-scale, extremely dense networks for environment monitoring. While most current sensing networks involve small numbers of sensors, supported by centralized processing and analysis hardware [4], these new networks will distribute computation among a high number of nodes. Applications for these networks must use algorithms that are highly distributed, since only short-ranged communication is preferred in the context of the stringent power constraints.[9][1] Furthermore, each node has limited high SNR sensing range, so sensing is best distributed and coordinated amongst a potentially large set of nodes. The algorithms these networks employ must be highly localized [18], as large distance transmissions are very expensive, and diminish the network's overall lifespan. Due to the size of these networks, they must be self-configuring, highly scalable, redundant, and robust in dealing with shifting topologies due to node failure and environment changes. [7] Applications utilizing these networks must be able to gather data from different parts of the network, without taxing the network's limited bandwidth and power. The communication channels are noisy, failure rates high, and routes ephemeral. Furthermore, ad-hoc deployment, required for dealing with networks of this size, may not provide global localization information to individual nodes.

One area in which these sensor-nets will be used is large scale environmental monitoring. [5] The goal is to enable the scattering of thousands of these nodes in areas that are difficult to access for study using conventional methods. The network could then monitor events [15], perform local computations on the data, and either, relay aggregated data, or configure local and global actuators.

In this paper we describe and analyze a method of routing queries to nodes that have observed a particular event. This allows retrieval of data keyed on the event, not the underlying network addressing scheme or geography.

An event is an abstraction, identifying anything from a set of sensor readings, to the node's processing capabilities. For the purpose of the simulation studies in this paper, events are assumed to be localized phenomenon, occurring in a fixed region of space. This assumption will hold for a wide variety of sensor-net applications, since many external events are localized them selves. A query can be a request for information, or orders to collect more data. Once the query arrives at its destination, data can begin to flow back to the query's originator. If the amount of returning data is significant, it makes sense to invest in discovering short paths from the source to the sink. Methods such as directed diffusion [1] resort to flooding the query throughout the entire network [11], in order to discover the best path. If geographic information is available, the best path is the greedy shortest path, and does not require flooding [17][2].

However, in many applications the quality of the path may not be very important, since the application may only request a small amount of data back, or simply needs to order the target node to initiate more

thorough sensing. In such cases, flooding every query may not be as efficient as delivering it by a non-optimal route.

Flooding does not have to be restricted to queries. For applications where there are few events and many queries, it makes sense to flood the event, and set up gradients towards it. [3] However, unless the number of queries per event and the amount of data generated by each event is quite high, the setup cost for event flooding cannot be effectively amortized.

This paper proposes *rumor routing*, a logical compromise between flooding queries and flooding event notifications. The idea is to create paths leading to each event; whereas event flooding creates a network-wide gradient field [3]. In this way, when a query is generated it can be sent on a random walk until it finds the event path; instead of flooding it throughout the network.

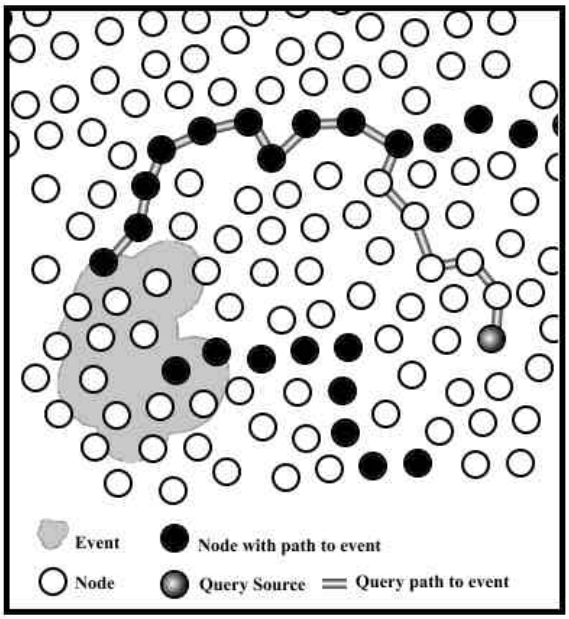

Figure 1: Query is originated from the query source and searches for a path to the event. As soon as it finds a node on the path, it's routed directly to the event.

As soon as the query discovers the event path, it can be routed directly to the event. If the path cannot be found, the application can try re-submitting the query, or as a last resort, flooding it. As this paper shows, under a wide range of conditions, it is possible to achieve an extremely high delivery rate. Monte-Carlo simulations show the probability of two lines intersecting in a bounded rectangular region to be approximately 69%. This means five paths leading to an event will have a 99.7% chance of being encountered by a query. Although neither the path nor the query is entirely straight, and the topology may not be rectangular, the heuristic should still hold. The number of paths and the number of query attempts increase the likelihood of delivery exponentially, making the Rumor Routing tunable to a wide variety of application requirements.

The method for setting up these paths to an event is the main focus of this paper. Again, we take advantage of the fact that two straight lines in a plane are likely to intersect. The algorithm employs a set of long-lived agents that create paths (in the form of state in nodes) directed towards the events they encounter. Whenever an agent crosses a path leading to an event it has not yet seen, it adapts its behavior and thenceforth creates path state that leads to both (or multiple) events.

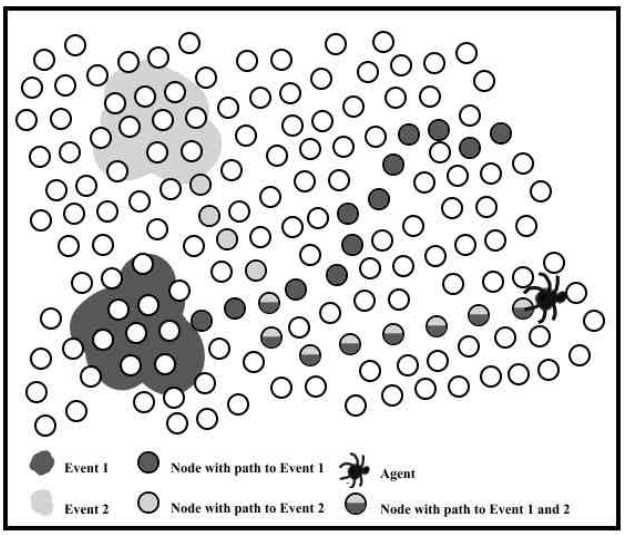

Figure 2: When agent prorogating the path to Event 2 comes across a path to Event 1, it begins to propagate the aggregate path to both.

In the diagram above, an agent has been creating path state leading to Event 2. When it crosses the path to Event 1, it begins to create aggregate path state, leading to both Event 1 and Event 2.

The agents also optimize the paths in the network if they find shorter ones. When an agent finds a node whose route to an event is more costly than its own, it will update the node's routing table to the more efficient path.

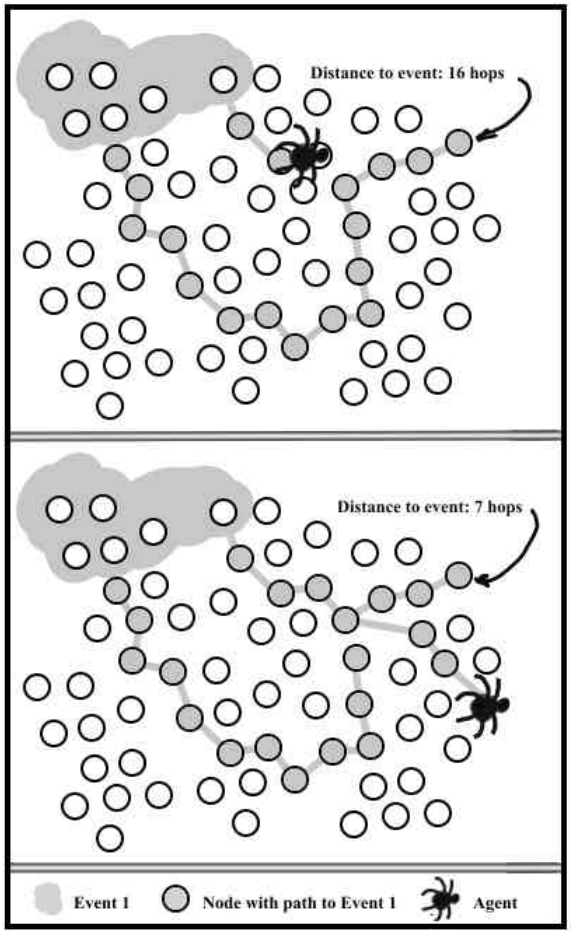

Figure 3: The agent modifies the exist path (top) to a more optimal one (bottom)

So it's not necessary to produce more than a few agents for each event, since the trail will be picked up and propagated by other agents.

2. Related Work

Sensor networks are the focus of a growing research effort. [1][3][8] Traditional routing schemes have been difficult to adopt, and as a result, many new algorithms have been developed.[1][2][17][3]

GRAdient Broadcast (GRAB) [3] - describes a way of building a cost field toward a particular node, and then reliably routing queries across a limited size mesh toward that node. It comes with the overhead of a network flood to set up the cost field, but queries are routed along an interleaved set of short paths, and can thus be delivered cheaply and reliably. GRAB was not designed specifically to support in network processing but significantly influenced the work presented in its use of event-centric routing state in the network.

Gossip Routing [11] – provides a scheme for performing reliable network broadcasts, probabilistically. Nodes flood by sending a message to some of the neighbors, instead of all, but due to the redundancy in the links, most nodes received the flooded packet. This scheme can be used to either deliver queries, or flood events for gradient setup, with less overhead than conventional flooding described in this paper. Thus far, GOSSIP routing has not been designed specifically for energy constrained contexts, but may be extended to be applicable in the area of sensor nets.

Ant Algorithms [10] – are a class of agent based routing algorithms modeled after ant behavior. Agents traverse the network encoding the quality of the path they have traveled, and leave it the encoded path as state in the nodes. At every node, an agent picks its next hop probabilistically, but biased toward already known good paths. [12] This results in faster and more thorough exploration of "good" regions, and a path for queries to follow. These algorithms are very effective in dealing with failure, since there is always some amount of exploration, especially around previously good solutions. However, due to the large number of nodes, the number of ant agents required to achieve good results tends to be very large, making them difficult to apply in sensor networks.

Directed Diffusion and Geo-Routing [1][2][17] – provide a mechanism for doing a limited flood of a query toward the event, and then setting up reverse gradients to send data back along the best route. GEAR/GPSR rely on localized nodes, and provides savings over a complete network flood by limiting the flooding to a geographical region. Diffusion results in high quality paths, but requires an initial flood of the query for exploration. One of its primary contributions is an architecture that names data and that is intended to support in network processing. *Rumor routing* is intended to work in conjunction with diffusion, bringing innovations from GRAB and GOSSIP routing to this context.

Data-Centric Storage in sensornets [15] – Allows access to named data by hashing the name to a geographic region in the network. This scheme can be used to efficiently deliver queries to named events by storing the location of the event, once known, in the region of the network to which the

name hashes. DCS relies on a global coordinate system, and an underlying geo-routing framework.

3. Overhead of Flooding Mechanisms

Before we present our *rumor routing* algorithm, we discuss the overhead of flooding mechanisms. As discussed earlier, if the application expects a large amount of data to be returned along the route of the query, either event flooding or query flooding could be used

If we assume uniform density in the network, and implicit broadcast of all transmissions, we can use the number of transmissions as a metric for comparing the energy requirements for these algorithms. Since every time a node transmits, all of its neighbors will receive the packet, and the number of neighbors is the same due to uniform density, the energy used in receiving is proportional to the number of transmissions. So the total energy used by the network is proportional to the number of transmissions.

3.1. Query Flooding

Assuming no localization information is available for use in geographic flooding, we resort to flooding the entire network with our query. So if we have N nodes, we must perform N transmissions per query, or N*Q transmissions total. This assumes no collisions, which, in a flood, can become a serious problem, and can make probabilistic flooding harder to implement due to the high amount of message loss. [13][11] The energy used is independent of the number of events tracked by the network. This scheme is useful if the number of events is very high, compared to the number of queries.

3.2. Event Flooding

Whenever a node witnesses an event, it can flood the network. All other nodes can form gradients toward the event, based on the number of hops to sink. An efficient way of setting up these gradients is discussed in the GRAB paper [3], and requires N transmissions per event. After the cost field is set up, queries can be reliably routed to the event along the shortest path. The cost of each query in terms of transmissions is negligible, and can be assumed to be zero for the scope of this paper. So the total energy expended by the network in event flooding is E*N, where E is the number of events. This is independent of the number of queries. So when the number of events is low, compared to the number of queries, event flooding can be efficient. The rest of this paper focuses on finding the threshold, below which *rumor routing* results in less energy use than event routing.

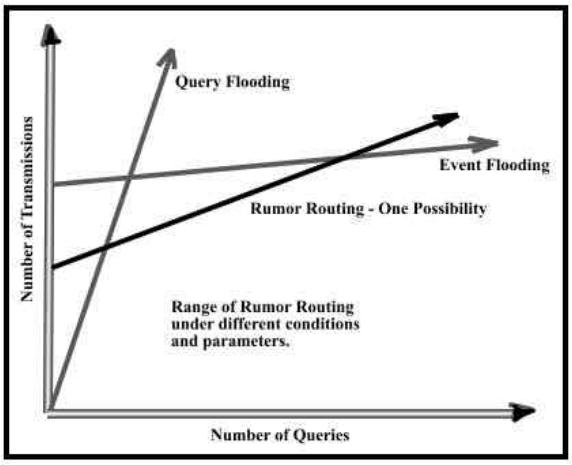

Figure 4: The gray region shows where a particularly configured instance of Rumor Routing fits in terms of setup and per-query cost. Obviously we are only interested in the region bellow query and event flooding.

3.3. Rumor Routing

The *Rumor Routing* algorithm is intended to fill the region between query flooding and event flooding. It is only useful if the number of queries compared to the number of events is between the two intersection points. An application aware of this ratio can use a hybrid of *Rumor Routing* and flooding to best utilize available power. Also, if reliable delivery is not a requirement, applications can tune the algorithm to trade off quality of service versus required energy.

4.1. Algorithm Overview

The network is modeled as a set of densely distributed wireless sensor nodes, with relatively short but symmetric radio ranges [16]. These nodes record unique events, and the application needs to be able to route queries to a node that has recorded a particular event. What follows is an informal description of the algorithm. The psudo-code is included in the appendix.

• Each node maintains a list of its neighbors, as well as an events table, with forwarding information to all the events it knows. The neighbor list can be actively created and maintained by actively broadcasting a request, or passively, through listening for other node broadcasts. Since the simulations were done in a static topology, each node simply broadcast its id at the beginning of the simulation.

• When a node witnesses an event, it adds it to its event table, with a distance of zero to the event. It also probabilistically generates an agent. The probability of generating an agent is an algorithm parameter, and is explored in the experiment section.

• An agent is a long-lived packet, which travels the network, propagating information about local events to distant nodes'. It contains an events table, similar to the nodes, which it synchronizes with every node it visits. The agent travels the network for some number of hops (**La**), and then dies.

• Any node may generate a query, which should be routed to a particular event. If the node has a route to the event, it will transmit the query. If it does not, it will forward the query in a random direction. This continues until the query TTL (**Lq**) expires, or until the query reaches a node that has observed the target event. In certain cases the node will not forward the query (loop detection).

• If the node that originated the query determines that the query did not reach a destination, it can try retransmitting, give up, or flood the query. Retransmission is a risk, but the chance of delivery is exponential with the number of transmissions. Hopefully only a very small percentage of queries would have to be flooded.

4.2. Agents

Each agent informs nodes it encounters of any events it has witnessed along its route. To do this, it carries a list of all the events it has encountered, along with the number of hops to that event. When it arrives at node A from its neighbor B, it will synchronize its list with the node's list.

In this case (Figure 5), A's route to event E1 is longer than the agent's. But the agent does not know how to route to E2. After the table synchronization completes, the event tables

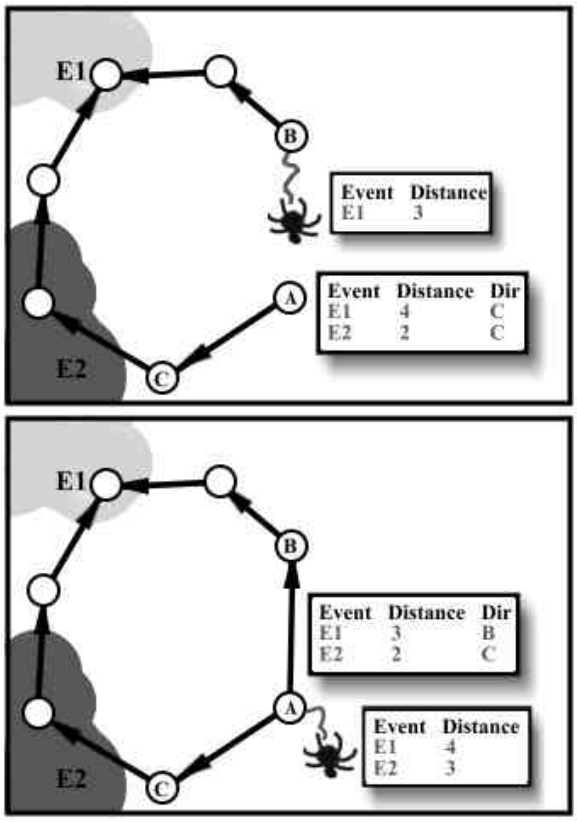

Figure 5: The agent having left node B contains a route to E1 of length 3. When it arrives at node A and performs a table sync, it will learn about the path to E2 and optimize A's path to E1.

will contain the best routes to each event.

Since all transmissions are effectively broadcasts in a wireless network, neighboring nodes can hear the agent as it moves along its path. They take advantage of this, and modify their event table based on the information the agent contains. This means the agent actually leaves a fairly thick path as it travels.

After performing the table synchronization, the agent's TTL is decremented, and if it is greater than zero, the agent is forwarded. In order to propagate directions to the event as far as possible in the network, a straightening algorithm is used when determining the agent's next hop. The agent maintains a list of recently seen nodes. When it arrives at a node, it adds all of the node's neighbors to the list. When picking its next hop, it will first try nodes not in the list. This eliminates most loops, and allows the agent to create fairly straight paths through the network. Although local looping will tend to generate more efficient paths, simulations show it is more important for a query to find a path to the event, regardless of quality.

Finally, a policy to generate agents is required. Although any node can generate an agent, it makes more sense for a node that has observed an event to do so. This way the agent starts out containing some useful information, and can start disseminating it immediately. The policy used for our simulations had a fixed probability that a node that has witnessed an event would generate an agent. The actual number of agents generated depends on the number of events, the event size, and the node density. A more optimal strategy for agent generation is left for future work.

For applications where events are temporal, the event table may have an expiration timestamp associated with each event. Agents may use this information in cases where packet size limits the number of events they can propagate.

4.3. Queries

A query can be generated at any time by any node, and is targeted to an event. If a node has a route (event path state) toward the target event, it forwards the query along the route. If it does not, it forwards the query to a random neighbor, assuming the query has not exceeded its TTL. Simulations show that forwarding queries along a straight path yields better results than random forwarding. The query employs the same mechanism as the agent, keeping a list of recently seen nodes, and avoiding visiting them. This only applies when the node is picking a random neighbor, since the query is always forwarded toward the event if a route is known.

In a dynamic network, where node failure is a possibility, it is sometimes possible to get looping routes. This is avoided through TTL in the query packet, but can be further avoided by assigning a random id to each query, and keeping a list of recently seen queries in the node. If a query arrives at a node by which it had already been forwarded, the node should send it to a random neighbor, not the route it has toward the event.

Some queries will not reach their destination, and the application that originates them must detect the failure, and handle it. Since queries have a maximum TTL, the application has a reliable value for a timeout. Failure can be handled in a variety of ways, but the simplest is to flood the query. This is very expensive, but guarantees delivery. Under most circumstances the percent of undelivered queries is very low, and can be reduced further by increasing the queries TTL.

5. Simulation Results

All simulations were performed in LecsSim [14] on a network of N = {3000,4000,5000} nodes scattered randomly on a two-dimensional field of 200x200m^2. A simple radial propagation model was used, where each node could reliably send packets to any node within 5m from it. The impact of realistic propagation models is left for future simulation, and experimental studies. A static event map was generated, randomly scattering E = {10,50,100} events of circular shape with radius of 5m, across the field. A query pattern was then randomly generated, creating 1000 queries, each from a random node to a random event. The nodes were initialized, and began generating agents, as proscribed by the algorithm. When the agents finished setting up their paths, the query pattern was run, and the number of successful routed queries was recorded.

5.1 Comparison to Event Routing and Query Routing

If we adapt a naïve strategy of flooding undelivered queries, and thus guarantee 100% reliability, we will need to perform additional N*(1000-Qf) sends, where Qf is the number of delivered queries. The average energy used for each query, after the paths are created, is (Eq + N*(1000-Qf))/1000, where Eq is the energy spent routing queries.

The average energy per query, along with the setup energy, can be used to find the total energy utilized by the network to route Q queries.

$$\mathbf{Et = Es + Q*(Eq + N*(1000-Qf)/1000)}$$

This value can then be compared to query flooding, where

$$\mathbf{Et = Q*N}$$

as well as event flooding, where

$$\mathbf{Et = E*N}$$

Several simulations were performed, with N set at 3000, 4000, and 5000 nodes, and E at 10, 50, and 100 events. For every pair of E and N, the algorithm parameters were varied to find which parameters lead to the best energy utilization.

The values for the following parameters were tested in each scenario:

Number of Agents (A) could not be varied directly, but the probability of an agent being generated was varied, and the number of resulting agents recorded.
Agent TTL (La) was tested for 100, 500, and 1000 hops.
Query TTL (Lq) was tested for 1000 and 2000 hops.

Agent TTL of 100, along with a small number of agents (around 25) generated poor results. Although the setup cost was minimal, only about 60% of the queries could be delivered successfully. A large number of Agents (around 400) had a high setup cost (above event flooding), but also a very high delivery rate (99.9%), as well as lower average energy per query. Even if undelivered queries were assumed to be flooded, for a wide range of settings and scenarios, the Rumor Routing algorithm performed better than event flooding.

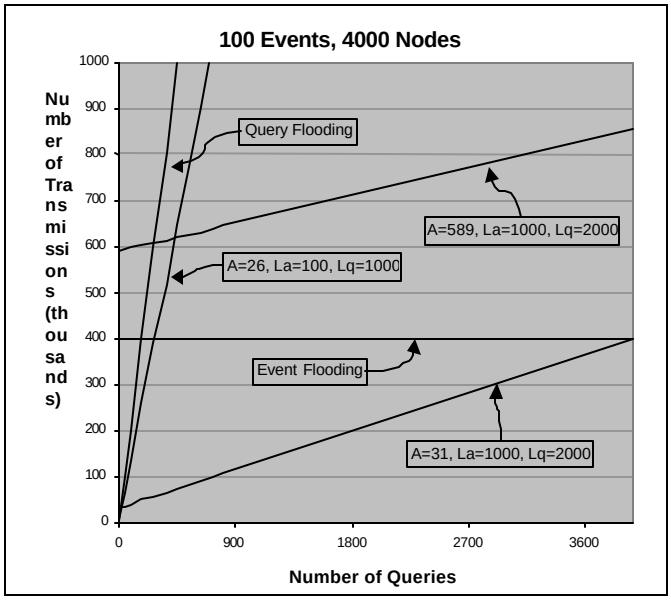

Figure 6: Some possible configurations of Rumor Routing. Although certain parameters result in costs greater than the flooding alternatives, others allow lower total cost for up to 36 queries per event with 98.1% delivery rate.

The best result (Figure 7) requires only a small number of agents (around 31), with a high TTL (1000). It successfully delivers 98.1% of all queries, with an average cost of 92 cumulative hops per query, or about 1/40 of a network flood. This comes with a setup cost of 31031 transmissions, or about 8 floods. This means that if we need to send out less than 3600 queries (36 per event), *Rumor Routing* can achieve significant savings over event flooding.

Most of parameter values caused better performance than event flooding up to a certain event cost threshold (Te). Usually, Te increased with the number of nodes and events, since the cost of event flooding grows linearly with both. In certain cases, usually when the number of events is low (10), there was more than one set of parameters that could be used.

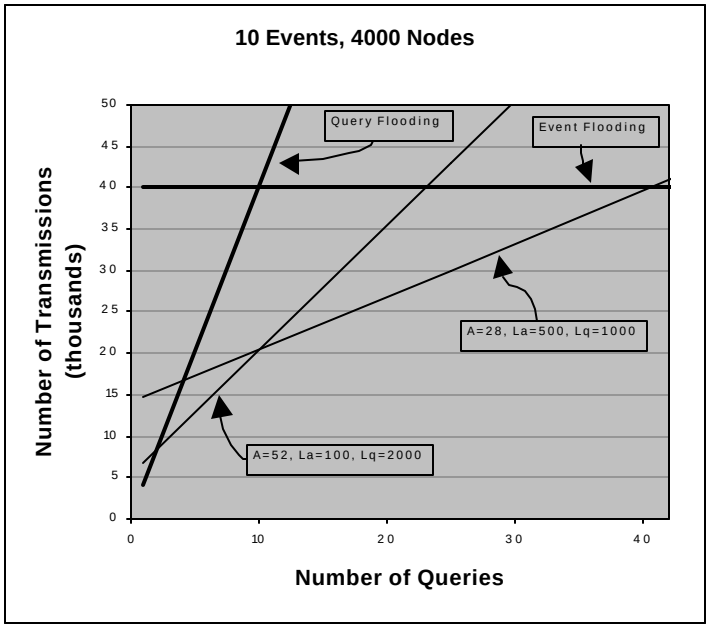

Figure 7: If the number of queries per event is less than ten, a smaller setup cost is better than a smaller per-query delivery cost. If, however, we want to deliver more queries (up to 40), a larger investment in path building yields better results. Delivery is guaranteed, as undelivered queries are flooded.

5.2 Algorithm Stability

Because this algorithm relies on random decisions (when determining which way to send agents and queries, and which nodes generate the agents), it is important to show that its performance does not vary significantly over several runs. To test the stability of the algorithm, we used the same set of parameters, event, node, and query maps to run 50 simulations. Each time we seeded the random number generator with a different seed. For this particular run, the Te was found to be 118 on average, with a standard deviation of 4.6. This means that 99% of the values for Te will be found between 104 and 131, and so the algorithm is stable for at least this particular configuration.

5.3 Effects of Event Distribution.

Although most reasonable values for the algorithm parameters yield better results than event flooding, we would like a method of picking the values to maximize Te for a known network and event density. To be able to accurately predict Te for a set of parameter values, we need to find how much the algorithm is affected by the distribution of the events, as opposed to their density.

To measure the effect of event distribution, the same set of parameter values was used on 100 randomly generated event, node, and query maps. For each simulation, the delivery probability was found, and a CDF graph is presented below.

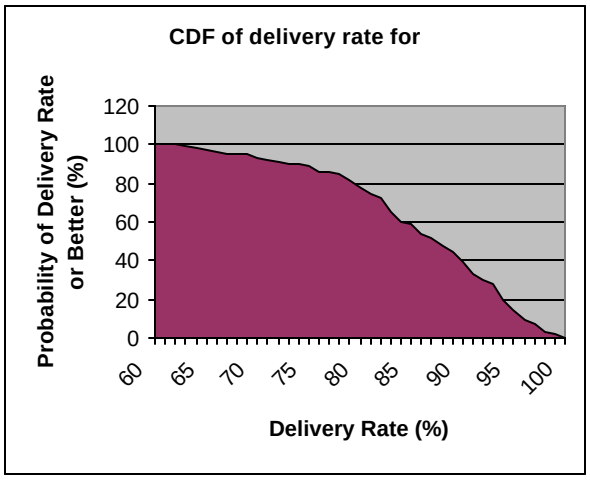

Figure 8: The probability that Rumor Routing successfully delivers at least that number of queries for any event/node/query distribution.

Although this set of parameter values always produces good results, delivery rate varies significantly with the random event/node/query distribution. Most values are centered on 500, but can sometimes go as high as 2200. The mean is 85%, with a very high standard deviation of 8.8%. The number of agents generated in each run varied between 24 and 56, with the mean of 37, since each node had a random chance of generating an agent. This did not affect the delivery probability in any significant way, and there appears to be no trend between the number of agents and the delivery probability in this range.

5.4. Fault Tolerance

To test the algorithm's ability to deal with failed nodes two failure modes were tested, random and clustered. After the routes were established some of the nodes were disabled. This poses an upper bound on the algorithm's failure, since in practice agents from other events will effectively repair damaged routes. The algorithm behaved the same way under both failure conditions. The probability of delivery degraded slowly for 0-20% node failure. For percentages over 20%, the performance degraded more severely, as expected. The delivery probability was strongly correlated to the number of failed nodes, with the correlation coefficient of 0.91. The best linear approximation of the error data had a slope of –1.8, with measurements taken up to 20% of node failure. This means at about 5% node failure we can expect 90% of the queries to be delivered successfully. The data is better approximated with a polynomial, since the delivery rate begins to decrease more dramatically after 5% failure. These results show the inherent redundancy in the routes that are created, and show a very graceful degradation. The effect of the repair mechanism, inherent in the algorithm, will be explored in the future.

No reliable trend has been found in the maximum query to event ratio with increasing node and event densities. This is probably due to the small sample size (one run per set of algorithm values for each density).

6. Future Work

There is further work to do investigating a wider range of scenarios, and investigating some algorithm design alternatives.

6.1 Wider range of simulation scenarios

Network Dynamics - the simulations in this paper were based on simultaneous occurrence of all the events, followed by a fixed setup time for the paths. In reality, the events will occur in time, making the paths to older events more prevalent than to the younger ones. The energy requirements, than, depend on the time between the event and queries.

Consider Collisions - the behavior of this algorithm should perform much better than the flooding alternatives, if collisions are considered. *Rumor routing* performs fewer simultaneous transmissions than there are agents in the system. Since the number of agents is small, compared to the number of nodes, this should not lead to a high collision rate. But reliable hop delivery is

required, and there may be significant overhead involved in dealing with collisions.

Asynchronous Events – currently in the simulations, all events happen simultaneously. For most applications, however, events are distributed in time, as well as space. This algorithm would favor older events, and it would be interesting to study how the rate of events affects the performance of *Rumor Routing*.

Non Localized Events - this paper focused on localized events, of a fixed size, but the algorithm allows for a much broader definition of an event. An event can be distributed through the entire network, but only detected by some of the nodes. It does not have to be bound to a sensor reading, and can simply represent node capabilities. This algorithm, for example, can be used to route queries to nodes that have a camera, and enough energy to use it. Since each node decides which events it has observed, there is a lot of flexibility in using the events for all sorts of data centric queries. Although the simulations showed this algorithm to be successful in localized events, it would make sense for it to be even more so in a distributed one. This will be tested in the future.

Non-random Query Pattern - the traffic pattern used for running queries on the network was randomly generated for the simulations in this paper. This assumes that any node is likely to request data on any event. In reality, it could be that the frequency of queries is not uniformly distributed. In many applications, it is more likely that a node closer to the event will want to query it, since most algorithms will tend to perform local computations where possible. On the other hand, local flooding may be a better approach to use when close-by nodes need to send a query to an event. *Rumor Routing*, than, can become a method for allowing far away nodes to efficiently query events.

6.2 Algorithm design alternatives

Non Random Next Hop Selection - currently, agents randomly pick their next hop, constrained by the straightening algorithm. There may be smarter ways of deciding where the agent should go. If localization information is present, it can attempt to maximize the probability of a crossing by trying to divide the network into equal halves. Agents can leave information about the frequency of trails they have encountered in a remote part of the network, and other agents can try to move toward the less explored regions.

Use of Constrained Flooding - queries are randomly forwarded until they find a path to the target. Doing limited floods may provide for a more efficient path finding method. This creates a problem of finding which queries to prune after the flood. More than one query can also be generated, creating a higher likelihood of delivery, but at a higher energy cost per query.

Parameter Setting Exploration - finding optimal parameters for a particular application is very important. As this paper shows, the event and query pattern has significant effect on the algorithm performance, and thus, optimal parameter values. Discovering whether the parameters can be tuned gradually by individual nodes through local observations, or approximated based on a model of the event and traffic patterns, is another important area that requires more research.

After some additional exploration, we plan to implement rumor routing on our wireless testbed at which time additional issues will undoubtedly arise.

5. Conclusion

There is an obvious need for delivering queries to events in the network, and large costs associated with both flooding the query, or alternatively, establishing a global coordinate system for geographic routing. Simulations show that the *Rumor Routing* algorithm provides a good method for delivering queries to events in large networks under a wide range of conditions, with energy requirements lower than the alternatives. It is designed to be tunable to different application requirements, and be adjusted to support different query to event ratios, successful delivery rates, and route repair. Furthermore, it is able to handle node failure gracefully, degrading its delivery rate linearly with the number of failed nodes. It remains for future work to develop appropriate methods for tuning the algorithm parameters.

There is an obvious need for delivering queries to events in the network, and large costs associated with flooding queries. When a geographic structure exists in the data, geographic coordinates and geo-routing [2][17] can be used effectively to reduce interest and data propagation overhead. However, when interests are expressed in terms of non-geographic attributes (such as searching for high concentrations of a particular chemical, or acoustic events that match a

particular signature), geographic routing does not apply. Simulations show that the Rumor Routing algorithm may provide a powerful and efficient method for delivering queries to events in large networks under a wide range of conditions. It is designed to be tunable to different application requirements, and be adjusted to support different query to event ratios, delivery rates, and route repair. Furthermore, it is able to handle node failure gracefully, degrading its delivery rate linearly with the number of failed nodes. It remains for future work to develop appropriate methods for tuning the algorithm parameters and to verify the power and efficiency of the scheme first with real sensor-data sets, and ultimately in real, not simulated, systems.

7. Acknowledgements

Thanks to Ben Greenstein for suggesting the monte-carlo simulation for the probability of intersecting lines in a bounded region.

8. References

[1] Chalermek Intanagonwiwat, Ramesh Govindan and Deborah Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCOM 2000), August 2000, Boston, Massachusetts.

[2] Yan Yu, Ramesh Govindan and Deborah Estrin. Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks. UCLA Computer Science Department Technical Report UCLA/CSD-TR-01-0023, May 2001.

[3] "GRAdient Broadcast: A Robust, Long-lived Large Sensor Network, http://irl.cs.ucla.edu/papers/grab-tech-report.ps"

[4] Sensors: The Journal of Applied Sensing Technology.

[5] Deborah Estrin, Lewis Girod, Greg Pottie, Mani Srivastava. Instrumenting the world with wireless sensor networks. In Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001), Salt Lake City, Utah, May 2001.

[6] Ya Xu, John Heidemann, Deborah Estrin. Geography-informed Energy Conservation for Ad-hoc Routing. In Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom), Rome, Italy, July 16-21, 2001.

[7] Alberto Cerpa and Deborah Estrin. Ascent: Adaptive Self-Configuring sEnsor Network Topologies. UCLA Computer Science Department Technical Report UCLA/CSD-TR 01-0009, May 2001.

[8] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan, Energy-Efficient Communication Protocols for Wireless Microsensor Networks, Proc. Hawaaian Int'l Conf. on Systems Science, January 2000.

[9] Deborah Estrin, Ramesh Govindan, John Heidemann and Satish Kumar.Next Century Challenges: Scalable Coordination in Sensor Networks. In Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99), August 1999, Seattle, Washington

[10] D. Subramanian, P. Druschel, J. Chen. Ants and Reinforcement Learning: A Case Study in Routing in Dynamic Data Networks. In Proceedings of IJCAI-97, 1997.

[11] M. Lin, K. Marzullo, S. Masini. Gossip versus deterministic flooding: Low message overhead and high reliability for broadcasting on small networks. UCSD Technical Report TR CS99-0637. http://citeseer.nj.nec.com/278404.html

[12] M. Dorigo, V. Maniezzo, A. Colorni. The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol.26, No. 1, 1996, pp.1-13

[13] Deepak Ganesan, Bhaskar Krishanamachari, Alec Woo, David Culler, Deborah Estrin, Stephen Wicker. Large Scale Network Discovery: Design Tradeoffs in Wireless Sensor Systems. Poster in Proceedings of the Symposium on Operating Systems Principles (SOSP 2001). Lake Louise, Banff, Canada. October 2001.

[14] http://lecs.cs.ucla.edu/~daveey/art/code.html

[15] Sylvia Ratnasamy, Deborah Estrin, Ramesh Govindan, Brad Karp, Scott Shenker, Li Yin, Fang Yu. Data-centric storage in Sensornets. http://lecs.cs.ucla.edu/~estrin/papers/dht.pdf

[16] Since symmetric connectivity is not always the case in sensor networks, nodes will only count neighbors where two-way communication is possible.

[17] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 243–254, Boston, Mass., USA, August 2000. ACM.

[18] A.A. Abidi, G.J. Pottie, W.J. Kaiser. Power-Conscious Design of Wireless Circuits and Systems. Proceedings of the IEEE, vol. 88, no. 10, pp. 1528-45, October 2000.

Appendix A: RumorRouting pseudo-code.

```
NODE.EVENTS <- {}
NODE.NEIGHBORS <- {}
broadcast "hello" packet with nodeId
repeat for ever
    if sensors detect event E
        call eventDetected(E)
    if "hello" packet received
        neighbors <- neighbors U neihborID
    if "agent" packet received
        call agentReceived(AGENT, SOURCE)
    if "query" packet receieved
        call queryReceived(QUERY, SOURCE)

eventDetected(E)
    NODE.EVENTS[E].DISTANCE <- 0
    create new agent A
    AGENT.EVENTS = {}
    AGENT.NUMHOPS=0
    // send the agent to your self
    ForwardAgent(NODE.ID)
agentReceived(AGENT,SOURCE)
    AGENT.NUMHOPS <- AGENT.NUMHOPS+1
    // update the node's events table based on the agent's
    foreach event named E in AGENT.EVENTS
        if (NODE.EVENTS does not contain E) OR (NODE.EVENTS[E].NUMHOPS > AGENT.EVENTS[E])
          NODE.EVENTS[E].DISTANCE <- AGENT.NUMHOPS - AGENT.EVENTS[E].VISIT_TIME
          AGENT.EVENTS[E].DIRECTION <- SOURCE
    // update the agent's events table based on the node's
    foreach event named E in NODE.EVENTS
        AGENT.EVENTS[E].VISIT_TIME <- (- NODE.EVENTS[E].DISTANCE)
    if AGENT.NUMHOPS < AGENT_TTL
        DESTINATION <- pick neighbor based on agent forwarding policy
        forwardAgent(AGENT,DESTINATION)

queryReceived(QUERY,SOURCE)
    QUERY.TTL <- QUERY.TTL - 1
    if NODE.EVENTS[QUERY.EVENT_NAME].DISTANCE=0
        // the query reached a valid destination
        handleValidQuery(QUERY)
    else if NODE.EVENTS[QUERY.EVENT_NAME].DISTANCE > 0
        // the node has a path to the event
        forwardQuery(QUERY,NODE.EVENTS[QUERY.EVENT_NAME].DIRECTION)
    else
        DESTINATION <- pick neighbor based on query forwarding policy
        forwardQuery(QUERY,DESTINATION)
```

Forwarding policies can range from simple random schemes, to ones trying to optimize the path intersection probabilities. In the simulations, a simple heuristic was used to "straighten" paths. Each packet kept a small history of the nodes it has visited and avoided them as potential destinations. Usually only a few nodes need to be remembered since a straight path will move the packet out of their territory in a few hops.