BARD: Bayesian-Assisted Resource Discovery In Sensor Networks

USC/Information Sciences Institute Technical Report ISI-TR-2004-593

Fred Stann, John Heidemann

***Abstract*-Data dissemination in sensor networks requires four components: resource discovery, route establishment, packet forwarding, and route maintenance. Resource discovery can be the most costly aspect if meta-data does not exist to guide the search. Geographic routing can minimize search cost when resources are defined by location, and hash-based techniques like data-centric storage can make searching more efficient, subject to increased storage cost. In general, however, flooding is required to locate all resources matching a specification. In this paper, we propose BARD, Bayesian-Assisted Resource Discovery, an approach that optimizes resource discovery in sensor networks by modeling search and routing as a stochastic process. BARD exploits the attribute structure of diffusion and prior routing history to avoid flooding for similar queries. BARD models attributes as random variables and finds routes to arbitrary value sets via Bayesian estimation. Results of occasional flooded queries establish a baseline probability distribution, which is used to focus additional queries. Since this process is probabilistic and approximate, even partial matches from prior searches can still reduce the scope of search. We evaluate the benefits of BARD by extending directed diffusion and examining control overhead with and without our Bayesian filter. These simulations demonstrate a 28% to 73% reduction in control traffic, depending on the number and locations of sources and sinks.**

Index terms - System design, Network measurements, Simulations, Experimentation with real networks/Testbeds

I. INTRODUCTION

Data dissemination in wireless sensor networks requires four components: resource discovery, route establishment, packet forwarding, and route maintenance. Resource discovery consists of finding data that is relevant to the application. The other three components are referred to collectively as *routing*. In IP and ad hoc routing, resource discovery is layered on top of routing. A good deal of work has been done to improve the efficiency of the route establishment component of data dissemination in wireless networks. DSR [12] and AODV [17] utilize cached routing information to limit overhead by deferring route establishment until existing route segments are no longer valid. Reinforcement Learning has been used in ad-hoc networks to forward data on links that are part of the shortest path to a destination address [1].

This work was supported by DARPA under grant DABT63-99-1-0011 as part of the SCAADS project, and was also made possible in part due to support from Intel Corporation and Xerox Corporation. John Heidemann is also partially supported through the NSF Division of Civil and Mechanical Systems, grant number E01-CMS-0112665. Fred Stann and John Heidemann are with USC/Information Sciences Institute, 4676 Admiralty Way, Marina Del Rey, CA, USA E-mail: fstann@usc.edu, johnh@isi.edu.

Data-centric protocols, like directed diffusion [11], combine resource discovery with the route establishment function of routing. For resource discovery, there are numerous schemes to limit cost. We divide them into five classes: data-centric storage, in-network aggregation, geographic-assist, target tracking, and probabilistic. We evaluate related work in Section VIII. Most of these schemes work well in their intended environments. However, each of these approaches is designed for a specialized application; we would instead prefer a general technique that exploits prior routing information to constrain flooding.

A unique characteristic of our approach is that it exploits attribute-based routing present in diffusion. IP-based schemes route only on the IP address. They can easily cache and reuse routes to prior addresses. By contrast, data-centric routing exposes application-level information in the form of attributes, combining routing and resource discovery. Attribute-based routing precludes simple route caching because even minor changes in the attributes mean that prior routes fail to match, rendering simple caching inapplicable. By contrast, some prior schemes have exposed limited application-specific information. For example, DCS/GHT [18] hashes attributes to physical location. Database techniques [16, 22] and diffusion [11] use application-specific in-network caching. Geographic approaches [2, 24] use physical coordinates to limit flooding. Target tracking techniques, like Spatiotemporal Multicast [10], tightly integrate routing and tracking.

Although each of these approaches is appropriate in their own context, none provide a general approach to limit flooding. Our goal is to provide a generic mechanism to exploit application specific information, exposed through attributes to limit routing overhead. Our approach exploits probabilistic approaches and reasoning approaches from artificial intelligence [19]. We model a real world problem in terms of a *belief agent* operating over a set of random variables. The belief agent chooses whatever action has the highest probability of achieving success. We model route discovery in diffusion as a distributed problem in which each node is a belief agent that must select a subset of links on which to forward route discovery messages. The agent must periodically flood in order to maintain a probability distribution and to locate singular real-time

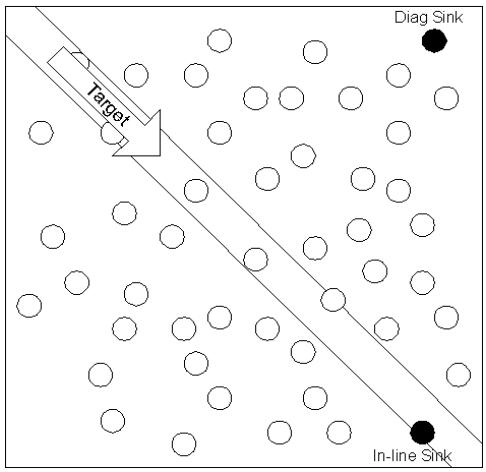

Figure 1
Sample simulation layout to model a complex example consisting of a target moving on a diagonal path through the simulation field.

events, but the flooding interval is much less frequent than in diffusion.

Consider, for example, a sensor network tracking vehicles moving along a road, as in Figure 1. A naïve approach to query for vehicles along the road would periodically query all sensors. Alternatively, if the location of the road were known, queries could be geographically limited. As another alternative, specific applications might track individual vehicles as they moved on the road. Instead, we aim to automatically observe, within diffusion, that a class of queries is looking for vehicles that elicit responses from sensors near the road, and automatically infer the location of the road over time based on query history. This general approach is similar to reinforcement learning techniques where it has been applied to general routing, but without application-specific information [1]. Explicit representations of belief have also been exploited in the context of specific sensor network applications [26]. To our knowledge, we are the first to propose integration of application-influenced learning at a generic routing level.

The contribution of this paper to improved efficiency in diffusion data dissemination protocols is BARD, *Bayesian-Assisted Resource Discovery*. BARD takes the form of a wrapper around diffusion routing using "filters" [9]. The BARD filter observes the control traffic generated by the underlying diffusion routing algorithm, does statistical analysis of that traffic, and routinely "squelches" a large percentage of control traffic that was intended to be flooded. BARD must periodically flood in order to maintain a probability distribution and to locate singular real-time events, but the flooding interval is much less frequent than in diffusion.

II. Bayesian Estimation and Resource Discovery

The resource discovery algorithm presented in this paper relies on the Bayesian method of statistical inference [21]. Bayesian estimation relies on a *prior probability distribution* $f(\theta)$, where θ is a random variable for which we have a prior distribution given a set of prior conditions, which are also random variables. Those same conditions, examined in a current sample, can be combined with $f(\theta)$ to compute a new distribution that predicts the likelihood of various values of θ. In our algorithm, θ is a set of links to neighboring nodes, which will be considered as candidates for the forwarding of resource discovery traffic. The "conditions" are the attributes and corresponding values of the resources that exist in the application. In effect, Bayes' theory provides a mechanism to calculate the likelihood that a particular hypothesis is true, given the current state of events and background evidence. Our hypothesis is simply: "is this node likely to lead us to a path between a resource provider (source) and a resource consumer (sink)?" The prior distribution is a history, collected over a window of time, relating neighbors to resource attributes.

We now demonstrate the derivation of a Bayesian estimate in the context of a sink seeking out sources with seismic and accoustic readings above some threshold. Suppose that sensor network "node X" has four neighbors: N1 through N4. We assume that node X has previously resorted to flooding to locate resources for queries related to seismic and accoustic data, keeping track of the frequency with which each neighbor provided a path to every resource attribute. Node X would like to limit flooding by exploring only those neighbors that are more likely to deliver seismic and acoustic sensor readings above the application-prescribed values.

Because we wish to express the prior probabilities in question as a sample space, we could divide the sample space into several mutually exclusive events N_1, N_2, N_3, … , N_k (one for each neighbor), and two pieces of evidence: seismic and acoustic. The expression of this as a sample space yields a three-dimensional joint probability distribution such as that depicted in Figure 2 (see top of next page). Unfortunately, joint probability distributions are difficult to maintain because their size grows exponentially with each added random variable. The primary advantage of using Bayes' Rule is that it can dynamically calculate conditional probabilities to a great deal of precision without maintaining a complete joint probability distribution when certain conditions hold, as noted below [19].

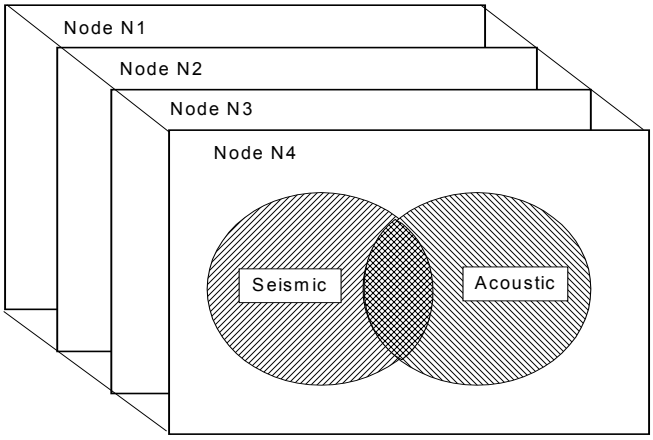

Figure 2
A sample space consisting of mutually exclusive Node events, and conditionally independent events Seismic &Acoustic.

Node X is looking for neighbors that can provide both seismic and acoustic sensor data (i.e. event S∩A). We want to answer the question: what is the probability that a given neighbor is a constituent part of the desired composite event? The contribution of a prior history is that it helps to predict which neighbors will most likely provide certain resources in present time. The following formula expresses the probability that neighbor N_3 has sound and acoustic sensor data available, i.e. S∩A occurred, given a joint distribution:

$$P[N_3 \mid S \cap A] = \frac{P[S \cap A \mid N_3]P[N_3]}{P[S \cap A]} \qquad (1)$$

Bayes' formula allows us to simplify the above equation when events S and A are *conditionally independent*. Evidence variables that are dependent in the joint, can be conditionally independent relative to the hypothesis variable if they are both a direct result of that variable. If we know that N_3 leads to S, the fact that N_3 also leads to M is irrelevant to the conditional calculation of S (i.e. $P[S \mid N_3] = P[S \mid N_3 \cap M]$). The following formula is the Bayes' estimate for neighbor N_3 with Seismic and Acoustic as our combined evidence:

$$P[N_3 \mid S \cap A] = \alpha\, P[N_3]P[S \mid N_3]P[A \mid N_3] \qquad (2)$$

The normalizing constant α is equal to the constrained sample space 1/P[S∩A] (i.e. the denominator of eq. (1)). This constrained space is calculated trivially by exploiting the fact that it can be converted into the following conditional terms:

$$P[N_3]P[S \mid N_3]P[A \mid N_3] + P[!N_3]P[S \mid !N_3]P[A \mid !N_3] \qquad (3)$$

III. BARD AND DIRECTED DIFFUSION

The effort to use Bayesian estimates to limit flooding during resource discovery, requires an understanding of how specific diffusion routing algorithms utilize flooding. The initial diffusion algorithm is now called two-phase pull diffusion [8]. In two-phase pull, interest messages that describe the attributes of desired data are flooded from a sink to all nodes. "Exploratory data" is then reverse-flooded from sources which have data matching the attributes. When the Exploratory data arrives at a sink, high quality (i.e., low latency) paths are "reinforced" by control messages unicast from a sink toward sources. The high-quality routes provide efficient (single path) data transfer for a period of time. Interests are periodically flooded to re-establish reinforced paths, in order to cope with changing network conditions. Therefore, two-phase pull has two flooding stages involved in resource discovery and path establishment.

Two more recent versions of diffusion are *push* and *one-phase pull* [8]. In push diffusion, sinks have interests that are held locally, rather than flooded. Resource discovery in push consists of sources finding paths to interested sinks via exploratory data flooding and reverse-path reinforcement. One-phase pull diffusion only floods interest messages. Data from sources travels along the reverse path of the lowest latency interest arrivals. One phase pull thus eliminates exploratory traffic flooding. The primary difference between push and pull (one or two-phase) is the direction of resource discovery via flooding. One-phase pull employs sink-to-source resource discovery and push uses source-to-sink resource discovery. Although Bayesian methods can be used to limit flooding in either direction, push provides fertile ground for BARD development because it is simpler in implementation than either the original diffusion routing algorithm or one-phase pull. All of the experiments presented in this paper were run over push diffusion. Push diffusion works well for applications, such as tracking, where many sensors are looking for data to publish, but actuations are relatively rare. As future work we plan to modify BARD to limit flooding in one-phase pull and two-phase pull diffusion.

IV. BARD FUNCTIONAL DESCRIPTION

Bayesian-Assisted Resource Discovery limits flooding in push diffusion via the application of Bayesian methods of estimation to predict what nodes to forward route discovery messages to. In push diffusion, route discovery packets are simply data packets, which are periodically marked as exploratory and flooded. When an exploratory data packet reaches a sink, the sink sends

a positive reinforcement message along the reverse path toward the source. Intermediate nodes typically receive multiple copies of the exploratory route discovery packet (once from each neighbor). The intermediate nodes use a heuristic, such as latency, to select a single neighbor to forward the positive reinforcement to. Once a reinforced path from source to sink is established, subsequent data packets are unicast along that path until the next "exploratory interval." When there are multiple sinks, a non-redundant distribution tree is formed between sinks and source. BARD spends some time building a per-attribute reinforcement history based on exploratory data traffic and reinforcement messages. Once a sufficient mass of history is collected, BARD attempts to limit flooding by forwarding exploratory traffic only to neighbors that are likely to yield reinforcements. In diffusion, data is exchanged when sources publish data whose attributes logically match those subscribed to by sinks. BARD employs the same matching rules when comparing resource-discovery packet content to reinforcement history. The goal is to predict what neighboring nodes have the greatest probability of yielding a working connection.

There are two primary functions that the BARD filter must provide in order to achieve its goal of efficient resource discovery: ongoing maintenance of the prior distribution and flooding limitation. BARD must collect statistical information about which neighbors have provided effective routes for data from sources to sinks. In doing so, it must "dissect" the routing attributes in a packet so that each routing attribute's history is maintained separately, because subsequent traffic flow may have a limited degree of intersection with current traffic Periodically BARD refreshes the reinforcement history by permitting flooding, to cope with changing conditions in the network. Therefore BARD maintains a sliding window of history, discarding the oldest entries as the window moves forward. Periodic flooding also guarantees that low probability events will be sensed, albeit at a reduced fidelity. The second primary function of BARD is to suppress flooding based on collected history. This function of BARD employs Bayesian methods of estimation to assign probabilities to outgoing links in order to predict which links will yield a positive result during route discovery. Nodes whose Bayesian estimate falls beneath a calculated threshold will not have resource discovery messages forwarded to them. We call this function *limited flooding*. The ratio of actual flooding to limited flooding dictates the efficiency achieved by BARD as well as the latency of its response to change. Tuning the ratio represents a configurable trade-off between real-time responsiveness and energy savings.

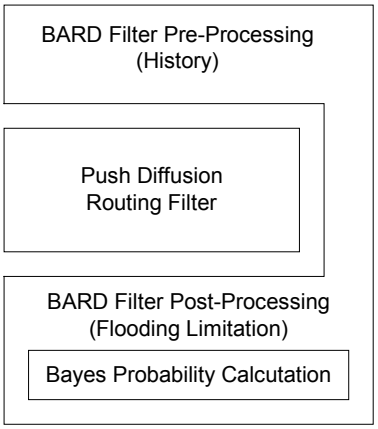

Figure 3
Depiction of Bayesian Filter in relation to Push Diffusion

V. BARD IMPLEMENTATION DETAILS

There are three elemental pieces that make up the BARD filter: a history gathering routine that is called whenever data is flooded, a flooding limitation routine that is called to constrain flooding, and a Bayesian module that calculates probabilities in support of flooding limitation (see Figure 3). A helpful concept in describing how the BARD filter interacts with push diffusion is that of an *exploratory epoch*. An exploratory epoch in diffusion begins when data is tagged as exploratory (to be flooded) by a source. The exploratory epoch is the maximum time interval during which data is unicast over a reinforced path. In general, BARD restricts flooding during four out of five exploratory epochs declared by push. As an optimization, BARD will allow flooding for three exploratory epochs in a row when there is no prior history. BARD limits the flooding of push exploratory messages by converting broadcast addresses to specific unicast addresses. Statistically, the best candidates to forward exploratory data to are those which have some history of providing positive reinforcement for the particular set of attributes contained in the exploratory data packet.

To support the flooding limitation function, BARD adds a unique attribute to every exploratory data packet that emanates from a push diffusion source. This attribute indicates whether the packet is going to be fully flooded or subject to limited flooding. The attribute travels with the packet as it is forwarded through the network. BARD pre-processing at a source node, therefore, controls how often an exploratory epoch is allowed to flood. Although this is currently set to 1/5th of exploratory epochs, it is configurable. Applications that are more entropic will require more frequent refreshing of the history.

The statistics gathering function of the BARD pre-filter maintains the prior distribution used by the post-filter during limited flooding. BARD keeps track of how many positive reinforcements arrive from each neighbor per flooded exploratory data. The reinforcement statistics are kept on both a per-neighbor and per-attribute basis. This allows BARD to deal with a varying degree of intersection in the attribute sets generated by different push sources. In order to enable attribute matching for attribute quantities that are not strictly identical (such as GT or LT), BARD requires that the subscription data from sinks be provided in the positive reinforcement. Push does not currently provide this data. Our prototype implementation therefore uses a simpler matching scheme that considers only attribute equality and not complete matching. We are in the process of extending the reinforcement information to allow complete matching.

The single purpose of the post-processing section of the BARD filter is to limit the flooding of exploratory data packets to a subset of neighbors. Neighbors are chosen that have demonstrated a historical probability of providing reinforced paths from sinks to sources for the attribute set contained in the exploratory packet. Because the reinforcement history is kept on a per-attribute basis, it is possible to construct probabilities incrementally using multiple pieces of "evidence" and Bayesian estimation.

In order to identify the subset of neighbors that will receive resource discovery packets, during limited-flooding, BARD uses *thresholding*. The *threshold* is the minimum Bayesian probability that must be achieved by a given neighbor in order to have a resource discovery packet forwarded to it. If the threshold is set too high, BARD may not find a route from source to sink, and if the threshold is set too low BARD efficiency will unnecessarily deteriorate. The appropriate threshold for forwarding is a function of three variables:

Fan-Out – the number of neighbors that can symmetrically communicate with a node.
Error Rate – the percentage of received packets to packets sent.
Number of Sources – the number of sources that are generating data traffic.

Dense topologies and multiple active sources produce multiple neighbors with moderate probabilities rather than fewer high-probability neighbors. Higher error rates dilute probabilities through reinforcement packet loss.

The following empirical formula, derived by observing how each component affected the number of alternate routes (in simulation), represents thresholds that resulted in delivery rates equal to or better than simple push:

$$\left(\sum_{0}^{S}\frac{1}{(d/4)^{S+2}}\right)-.8(e-.05) \qquad (4)$$

S is number of sources, d is the degree or fan out, and e is the error rate. The right hand side represents the reduction in threshold required to keep pace with an increasing error rate. The left hand side represents the graceful decay when multiple sources activate for a single event. The reciprocal of this formula yields the projected number of alternate routes. For example, with 10 neighbors, a single source, and an error rate of .05, the threshold would be .25 and the number of viable alternate routes within 50 minutes (the history window of the BARD filter) would be 4.

VI. Analysis

Bayesian estimation techniques rely on the existence of a prior distribution. BARD collects information periodically via flooding so as to establish and maintain this prior distribution for reuse during periods of limited routing. If we assume that BARD will successfully identify the "corridor" of alternate routes between source and sink, then we can predict the amount of traffic we expect to be generated during limited routing. We know the amount of control traffic generated by push diffusion during route discovery. Each node broadcasts the exploratory message exactly one time, and then a single route is reinforced. If we have a square grid with nodes evenly spaced and a source and sink at diagonally opposite corners, then the cost of routing in bytes per event is simply:

$$f_t * \frac{\left(n+\sqrt{n}\right)x}{events_t} \qquad (5)$$

where n is the number of nodes, f_t is the number of exploratory intervals per time t, x is the number of bytes in an exploratory packet, and $events_t$ is the number of events occurring in time t. The term $\sqrt{n}$ represents the cost of the positive reinforcement. We can actually simplify Equation (5) because when push diffusion floods exploratory data, the first exploratory packet to arrive at the sink will double as an actual data packet. Thus the $\sqrt{n}$ term represents useful data transfer rather than overhead and can be deleted, yielding:

$$f_t nx / events_t \qquad (6)$$

BARD, operating in steady state (as configured for this paper), floods 1/5 as often as push. We can express this cost per event as:

$$\frac{1/5 f_t nx + 4/5 f_t h(a-1)x}{events_t} \quad (7)$$

where a is the number of alternate routes that the BARD filter identifies during limited routing and h is the average hop count of those routes. We decrement a for the same reason we removed $\sqrt{n}$ in Equation (6). As we add more sources, or increase the error rate, we expect a to grow as the reciprocal of (4). If we wish to express the percentage gain that BARD should have over push, we can simplify our calculation:

$$1 - \frac{f_t\left(1/5n + 4/5h(a-1)\right)}{f_t n} \quad (8)$$

VII. Experimental results

In order to quickly evaluate the efficiency and potential of Bayesian-Assisted Resource Discovery, we designed a series of experiments run primarily in simulation. Simulation allowed us to quickly and systematically explore a multi-dimensional problem space in a manner that is impractical with an actual testbed. The simulation test environment employed was ns-2 [3], version 2.26. Most experiments varied an individual aspect of an n-dimensional problem space. The varied aspects were: number of nodes, density of nodes, number of sources, number of sinks, send rate, and error rate. A simulation experiment was also performed to approximate the advantage that BARD might bring to a complex "real-world" application.

We also conducted an experiment on an actual testbed that incorporated several aspects of our simulations. The testbed that we used consisted of *Stayton* nodes. These are small form factor systems running Linux over a 32-bit Intel embedded processor, with 64M of SDRAM, and 32M of flash memory. They are fitted with a multi-channel radio capable of 38.4 Kbaud. Radio range can be modified to enable multi-hop experiments.

We conducted our ns-2 experiments running multiple simulations over randomly generated node patterns. For each pair of data points in an experiment, we ran simple push diffusion with and without the BARD filter over ten random topologies and averaged the results. Additionally, we calculated and plotted 95% confidence intervals. We expressed results in terms of control byte transmissions required for resource discovery normalized by the number of events that generated traffic in the simulation (i.e. whenever a source decides to send). We collected these numbers by instrumenting the simulation code to log resource discovery related transmissions with unique tags. We then processed the logs via awk scripts, which used the tags and byte counts to summarize transmission activity.

For ns-2 node placement we modified an existing topology-generator program [8] to create random topologies of progammable size. The generator has useful options such as optional corner placements of sources and/or sinks. This allowed for any randomly or strategically placed sources and sinks. The generator tests node patterns for connectedness given radio range. If sources and sinks cannot communicate with each other, the generator scraps the topology and generates another.

Software loaded above the MAC layer in each experiment included the diffusion filter core, the diffusion push routing filter, the BARD filter (when using BARD), and simple source and sink apps. For ns-2 experiments, we used the 802-11 MAC layer provided in ns-2 with a retry count of 4. For all ns-2 experiments the radius of the radio range was 39.6 meters. Our testbed experiment used the SMAC link layer [23] with a retry count of 3. For all experiments, routing was based on three attribute keys: motion, sound, and target type.

A. Performance as node count increases

The first experiment was aimed at quantifying the savings achievable with BARD as the number of nodes increases while density remains the same. The size of the simulation area was increased with node count, such that the average node density was held constant at 10.9 neighbors. The selected density insured a high probability of network connectivity, despite random node placement. A single source and sink were placed at the greatest diagonal distance apart in the simulation field. The node count was varied from 25 to 100. A 128 byte packet was sent every 20 seconds from the source. Flooding was performed once per 60-second epoch by push and once every 5 minutes by BARD, after 3 initial flooding epochs to establish a prior history. Our expectations for this experiment, based on Equations (4) and (8), were a 45% improvement running with 25 nodes and a 53% improvement at 100 nodes. Because the dominant term in our calculation is the node count we expected approximately linear growth for both BARD and push. Because BARD allows flooding 1/5 as often as simple push, we expected less slope for BARD.

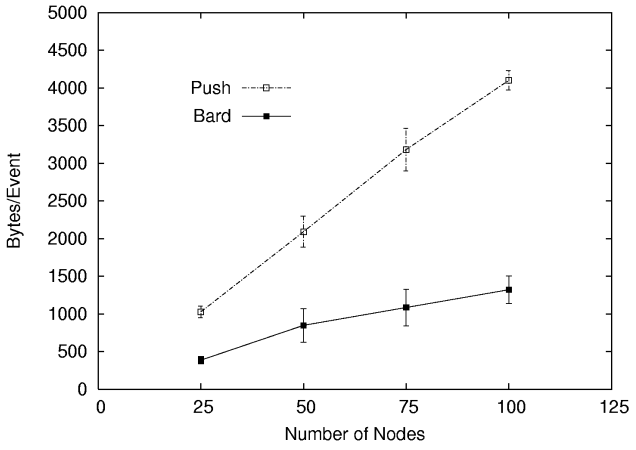

Figure 4

Control bytes per event required to transfer 180 packets of 128 bytes each diagonally from source to sink using BARD Routing vs. simple Push Routing with a variable number of nodes.

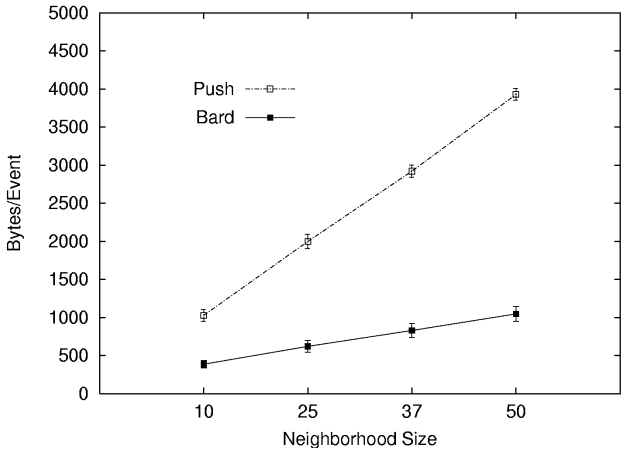

Figure 5

Control bytes per event required to transfer 180 packets of 128 bytes each diagonally from source to sink using BARD Routing vs. simple Push Routing with increasing density

The graph in Figure 4 depicts the results, along with 95% confidence levels. At 25 nodes BARD averaged 386 B/event while simple push averaged 1027 B/event. This represents a 62% improvement by BARD. At 100 nodes BARD averaged 1320 B/event while simple push averaged 4100 B/event, a 68% improvement by BARD. More importantly, the average rate of increase (slope) for BARD from 50 to 100 nodes is 4.25 times less than the growth rate for simple push. The results of this experiment were better than predicted. Log analysis showed that the savings exceeded analysis because of a progressively greater limitation of alternate routes by BARD as the simulation progressed. When diffusion establishes new routes at the beginning of an exploratory epoch, it does not immediately retire previously used routes. Only when routes consistently display greater latency than their alternatives, are they retired. Therefore data is often replicated on multiple paths by diffusion. BARD established fewer alternate routes than simple push as time passed.

B. Performance as node density increases

Changes in density affect BARD in two ways. First, higher density increases the number of alternate routes, and second nodes involved in viable routes are a smaller percentage of total node count. To understand these tradeoffs we varied density from 10 to 50 nodes in the average neighborhood. (although nodes are randomly placed, we computed approximate density analytically by assuming a nominal radio range of 39.6 meters and uniformly distributed sensors, ignoring edge effects). We expected that BARD would achieve greater savings than in the first experiment. The rational was that having a smaller proportion of total node count involved in routing would dominate results. The results, shown in Figure 5, validated these expectations. At 10 nodes per area (25 nodes), BARD required 62% less control traffic than simple push. At 49.3 nodes per area (100 nodes), BARD required 73% less control traffic.

C. Varying the number of sources

This experiment compares the performance of BARD to unmodified push diffusion when multiple randomly positioned sources are generating interesting data. We wished to establish the efficiency with which BARD could locate multiple resources. The number of sources was varied from one to five. A single sink received all of the traffic generated by the sources. Each source generated exploratory packets every 60 seconds, which were filtered with limited routing 4/5ths of the time in the case of the BARD filter. Our expectation for this experiment was that the percentage of savings incurred by BARD would lessen as the number of sources increased. We assumed that as more sources became active, more nodes would necessarily become involved in routing thus decreasing BARD's efficiency.

The graph in Figure 6 (see top of next page) shows that with a single source BARD averaged 703 control B/event, and simple push averaged 1928 bytes. This represents 63% improvement by BARD. With 5 sources BARD averaged 3607 control B/event and simple push averaged 8454 B/event, a 54% improvement by BARD. Therefore the percentage of improvement decreases with source count. Log analysis showed that in the case of multiple sources, BARD must discover more routes per event.

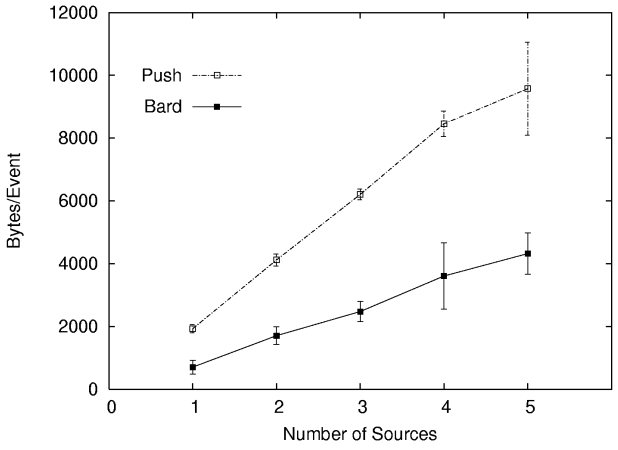

Figure 6

Control byte transmissions per event required to transfer 90 packets of 128 bytes each from a variable number of sources to a single sink using BARD Routing vs. simple Push Routing

D. Varying the number of sinks

This experiment compares the performance of BARD to unmodified push diffusion when multiple sinks are interested in the same events generated by a single source. Because push diffusion does not flood interest messages, like two-phase pull, the expectation was that adding sinks with a constant number of sources (in this case 1) would have a linear growth rate that is relatively flat. For each sink, we expected adding $\sqrt{n}$ * bytes/pkt because each sink sends its own positive reinforcement. The BARD filter does not influence the activity of the underlying push filter when it comes to positive reinforcements. BARD only squelches exploratory traffic. With a single source, the relative improvement provided by BARD should be consistent.

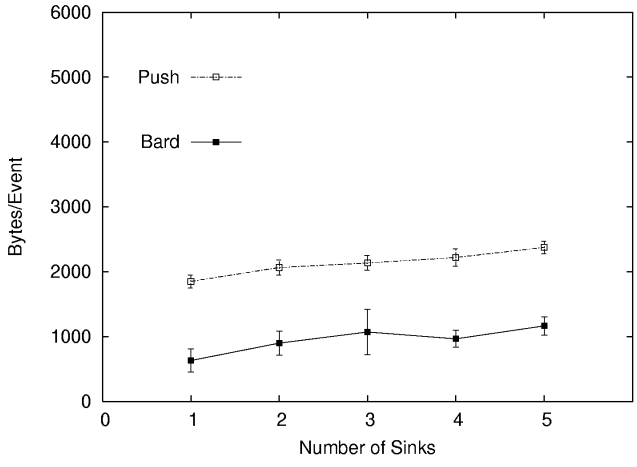

Figure 7

Control byte transmissions per event required to transfer 90 packets of 128 bytes each from a single source to a variable number of sinks using BARD Routing vs. simple Push Routing

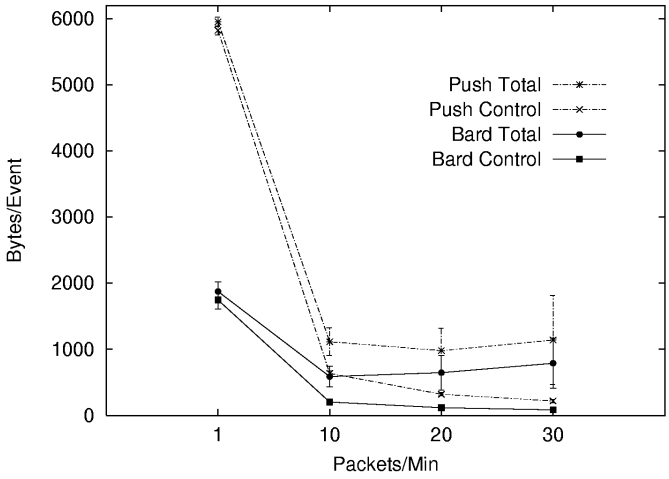

Figure 8

Total and control byte overhead per event for BARD Routing vs. simple Push Routing with an increasing send rate.

The results for this experiment are summarized in Figure 7. As can be seen in the figure, the slopes of both lines are nearly identical, varying by less than 10%. BARD was approximately 60% more efficient than push. If we had chosen to use one-phase pull as the underlying routing protocol, and used BARD to limit the flooding of interests, we would expect this graph to look more like Figure 6, since one-phase pull does sink initiated flooding.

E. Increasing the send frequency

What happens when the cost of moving data along discovered paths becomes the major overhead in a sensor net application? In this experiment we increase the frequency with which data is sent from a single source to a sink in field of 50 nodes, and measure total overhead per event (includes data traffic) as well as the control overhead. The simulation time was 1 hour, and the send rate was varied from 1 to 30 messages/minute. Our expectation was that the control byte overhead for push vs. BARD would begin to converge at higher send rates due to amortization across an increased number of events. The plot is shown in Figure 8. As predicted, the control byte overhead for push approaches that of BARD at 30 packets/minute (84 B/event for BARD and 219 B/event for push). For a single packet/minute, total and control B/event are nearly identical because every packet is exploratory (1875 B/event for BARD and 5952 B/event for push). A surprising result was that total overhead did not demonstrate greater convergence. Although total savings with BARD lessened with send rate, it didn't converge with push very rapidly after 10 packets/min. As observed in the 1st experiment, BARD

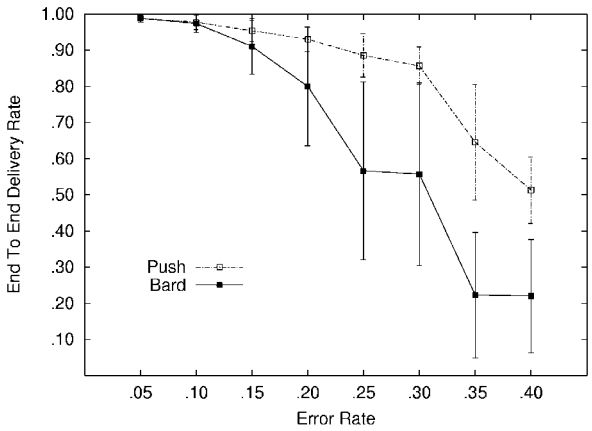

Figure 9
Percentage of packets sent that successfully arrived at a sink given various error rates using BARD Routing vs. simple Push Routing

provides an unexpected benefit over unmodified push because fewer alternate data paths are maintained over time. Diffusion doesn't immediately retire older data paths when new ones are established. BARD reduces the number of alternate paths that carry redundant data.

F. Sensitivity to transmission error

In this experiment we wished to investigate what influence packet loss has on the efficacy of BARD modified routing. We employed a simulation space that includes 50 nodes at a density of 10 nodes per radio area with a single source and sink placed at approximately diagonally opposite corners. We incrementally increased the error rate by modifying the error model in ns-2. Our expectation was that BARD would show lesser savings as the error rate increased. The rationale was that the reduction of flooding would result in the discovery of fewer viable routes. The error rate was varied from5% to 40%. Previous studies [25] have demonstrated that error rates as high as 40% are present between some nodes in a sensor network, albeit rarely at adjacent nodes. A network with a 40% error rate between all adjacent nodes would be fairly useless. Flooding reduces the negative effects of transmission errors. Diffusion nodes only forward data that has not been flooded previously. If the average node in a sensor network has 9 neighbors, then the broadcast from any of these neighbors could be considered the "first" broadcast if earlier ones are lost. This situation is equivalent to having 9 retries, rather than the 3 or 4 provided by the MAC layer. When error rates are high, even unicast transmissions, such as positive reinforcements, can suffer poor end-to-end reliability. The results for this experiment are summarized in Figure 9. As predicted, the decay in the end-to-end delivery rate is worse for BARD than simple push for global error rates in excess of 10%.

There are two curious aspects to the graph in Figure 9. The first is the stepwise decay in the BARD end-to-end delivery rate for rates greater than 25%. Log analysis showed that this was related to entire exploratory epochs being lost when positive reinforcements don't make it from sink to source. This resulted in an effective quantization of the curve that we would expect. The other interesting aspect is the high variance. This is to be expected under high entropy conditions.

G. Complex "real world" scenerio

We selected previous scenarios to systematically explore the effects of BARD. In this section we consider a more complex scenario to approximate how we see BARD being used in practice. Consider a sensor field deployed to track vehicles. Vehicles typically follow a common path - perhaps a road. In our scenario (refer back to Figure 1), a target follows a diagonal road from upper left to lower right. As the target passes by nodes near the diagonal, they are triggered to generate traffic. A lone sink reinforces the traffic from all transmitting nodes (sources). We initially ran the experiment with the sink diagonally opposite of the imaginary path. Sources did not transmit in response to a timer expiration. Instead, a"trigger method" was added to each node; which, when invoked, forced the transmission of source data. The target was modeled as an ns- 2 application that woke up monotonically and calculated its location according to a pre-determined velocity. Nodes within 10 meters of the current location of the target were sent a trigger message so that they would generate traffic. The topology generator employed in the previous experiments was used to generate topologies, so that performance could be averaged over 10 different random node placements. Our expectation for this experiment was that the reduction in control traffic realized by BARD would not be as dramatic as in earlier experiments. The rationale was that BARD would need to maintain a network of paths from the sink to the diagonal that would encompass half of the simulation area used by simple push flooding. Because BARD floods 1/5 as often as push, we estimated a savings in the range of 40% of the overhead incurred by simple push (i.e. 1/2*4/5). Results for this experiment are summarized in the left half of the bar graph in Figure 10 (at the top of the next page). BARD performance was approximately 28% better than simple push, slightly worse than expected. Because the sources along the diagonal were angularly separated relative to the sink and multiple paths existed to each source, some links were used multiple times during exploratory epochs.

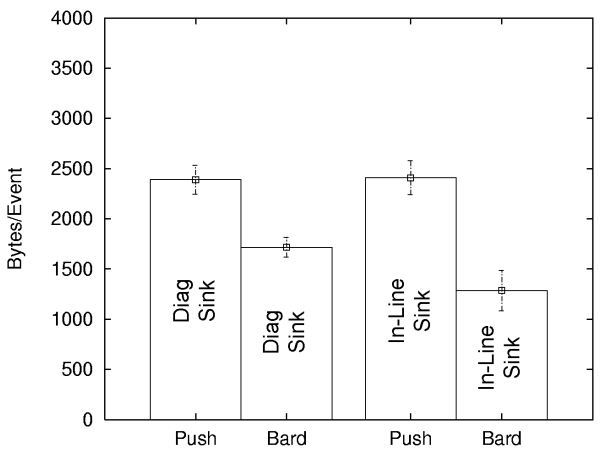

Figure 10
Control byte overhead per event for target tracking application for BARD Routing vs. simple Push Routing when sink is diagonal from target route vs. in-line with target route

We then speculated that we could improve on the relative performance of BARD vs. simple push by placing the sink in the lower right corner of the simulation area. This would put the active sources somewhat in-line with the sink, thus creating fewer alternatives for the limited routing function of BARD. The mean for these simulations is shown on the right half of Figure 10. Notice that the efficiency of straight push is virtually unchanged, whereas the improvement by BARD over push goes from 28% to 47%.

H. Testbed experiment

Although we found it convenient to explore the problem space relevant to diffusion / BARD via ns-2, we also wanted to deploy BARD in an actual testbed. Past experience has taught us that actual testbed experiments often result in unanticipated problems, which require design revisions. In this case we had available a testbed of 10 nodes, described at the start of this section. The deployment of the testbed is shown in Figure 11. The testbed consisted of a "fat" end in which nodes had multiple neighbors and a "thin" end in which nodes had a single neighbor on each side. The sink placement was just to the right of the middle of the testbed.

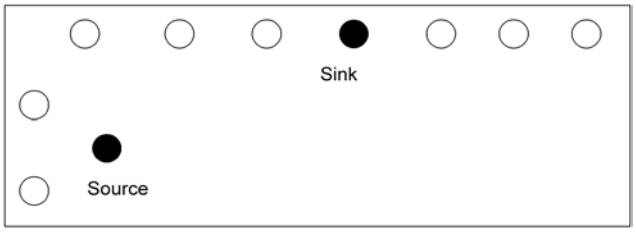

Figure 11
Stayton Testbed Topology

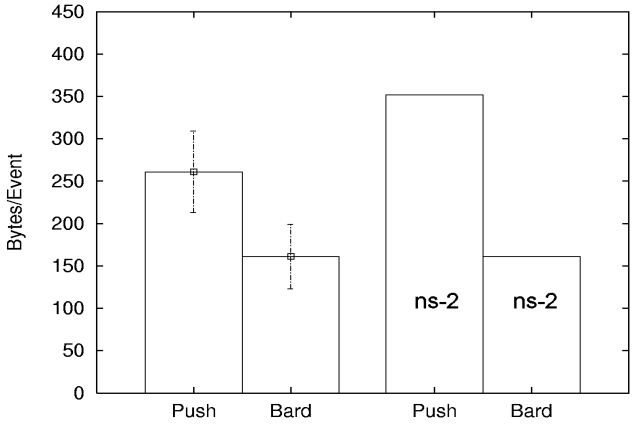

Figure 12
Control byte overhead per event for Stayton testbed experiment - BARD Routing vs. simple Push Routing

This allowed us to ascertain if BARD was capable of eliminating a non-productive segment of the network from attempts to find resources. The node to which we attached the source had the highest neighbor count and the greatest number of possible paths to the sink. Five 30 minute runs were performed with this configuration and the control traffic was counted for each run. The mean control traffic per event and confidence intervals are plotted in Figure 12, along with results from an ns-2 experiment patterned identically. Bayesian assisted routing was 38% more efficient on average than simple push routing. This was in line with expectations. We expected the nodes on the right hand side of the testbed to be contacted by BARD approximately 40% as often as they would be by simple push. This is a reflection of the initial 3 data epochs used by BARD to build a prior distribution, followed by infrequent flooding by BARD to maintain the distribution. We expected BARD to limit the number of paths on the left hand side of the testbed to those which displayed the least latency. Because broadcast packets do not have ARQ in SMAC, the neighbors that heard the initial exploratory packet transmission from the source varied with time, which resulted in a variance of the exact paths maintained by BARD. Typically BARD maintained one or two paths on the left hand side. In every case BARD found a path from source to sink. Although our testbed was of limited size, this experiment nonetheless validated that our simulation results were consistent with results in a real-world test environment

VIII. Related Work

The work surveyed for this paper can be partitioned into two categories: routing-related and resource-discovery-related; and six groups: route caching, geographic-assist, probabilistic forwarding, in-network aggregation, data-centric storage, and target tracking.

Route caching is a proven method for flooding-limitation in ad-hoc networks that employ addressed-based routing. DSR [12] and AODV [17] avoid flooding via route caching and on-demand (non-periodic) route discovery. A probabilistic technique for adaptive routing, which has been used in ad-hoc networks, is reinforcement learning [1], wherein nodes learn the probabilities that their neighbors will provide the shortest path to a given address. Approaches from addressed-based protocols cannot be directly applied to attribute-based routing because the state space is extremely large and grows exponentially in the number of attributes used for routing. Additionally, multiple nodes may be able to satisfy any given resource request. Finally, such techniques are of limited benefit when slight variations in attributes preclude matches. We instead adopt Bayesian-based filtering to allow partial matches.

When resources can be bound to geographical coordinates and sensor network nodes are geographically aware, several algorithms provide efficient scalable solutions to the problem of flooding. Both GPSR [13] and GEAR [24] exploit geography by greedily forwarding route discovery packets to individual neighbors that are closer to a target location. The primary advantage of BARD over purely geographic methods is its ability to exploit non-geographic aspects of a problem domain when accomplishing resource discovery. Additionally, sinks often don't know where sources reside initially (or vice verse).

Gossiping is a resource discovery technique that was developed in the context of networked databases [4], and subsequently applied to ad hoc routing in wireless networks [7]. The basic idea in gossiping is to forward route discovery messages with some pre-configured probability geared to the average degree (fan-out) of the network. BARD can adapt to skewed situations better, wherein some large segment of the network is not generating any interesting traffic. Rumor Routing [2] is a technique to limit flooding that can work with data-based routing and attribute-based queries. Events from sources and queries from sinks are propagated along approximate straight lines that are likely to intersect. The caching of event descriptions (attribute tuples) in rumor is similar to the caching of per-attribute probabilities in BARD.

There are a large number of algorithms that limit flooding in sensor networks via in-network aggregation. This principle is loosely related to earlier work on clustering [15]. Aggregating data from multiple sensor nodes results in a reduced cost of accessing that data when the path to the aggregated data is either known or local updates percolate the aggregated data up an implicit hierarchy towards an active sink. Both SPIN [14] and COUGAR [22] form on-the-fly clustered hierarchies in which upper layers have partially aggregated information collected from lower layers.

DCS/GHT (Data-Centric Storage in Sesnornets with Geographic Hash Tables) [18] requires an underlying geographic routing layer in order to perform in-network aggregation. All data with the same "name" (which could be an attribute tuple) is stored at the same node. The node is selected by hashing the name into geographic coordinates. It is appropriate for situations in which the same limited set of long-standing queries persists in a stable network. The cost of DCS is that preemptively moving data to hash sites can be expensive if the data is never accessed. By contrast, data-centric routing places the burden of search on queries, and BARD helps reduce that cost.

Target tracking is a specialized form of in-network aggregation. GEAR has been successfully combined with push diffusion to use geography to follow a target in a location aware sensor network [26]. Spatio-temporal Multicast in Sensor Networks [10] predicts target path and establishes a "delivery zone" that has direction and velocity along the predicted path. FRESH [5] is intended for mobile ad-hoc networks where flooding is triggered frequently due to node movement, and geographical information is not available. In FRESH, nodes forward route discovery traffic only to nodes that have more recently encountered the target. If BARD were configured to use time as the only attribute with which to calculate its probabilities, it would be similar to FRESH.

Most of the flooding limitation techniques sited in this section focus on a particular aspect of resource discovery related to the applications that they support. For example: GEAR uses geographical location in Euclidean space, and FRESH uses temporal information. Data aggregation methods, like DCS/GHT aggregate information to assigned nodes, assuming that the savings incurred for queries outweighs the on-going cost of aggregation for storage. Effectively, each algorithm is capable of limiting the flooding associated with resource discovery for a particular class of application. The advantage of BARD is its generality. BARD can predict viable routes using any number or type of attributes in an environment where multiple routes alternate in their effectiveness to reach target data. BARD is also capable of efficient resource discovery when the possible set of queries can't be quantized into a set of small cardinality.

IX. FUTURE WORK

Our initial results have motivated us to further explore the potential of *Bayesian-Assisted Resource Discovery*. Foremost we wish to run more experiments over a real sensor network with a greater number of nodes. Experience has taught us that testbeds bring out race conditions, implosions, and correlated error conditions. We also need to explore trade offs related to the frequency with which BARD floods to update its prior distribution (i.e. efficiency vs. real-time response). Additionally, BARD needs to be expanded to work with *Pull* diffusion. We also wish to examine the potential of BARD to cope with "attribute intersection." The initial test results presented in this paper did not explore cases where multiple sinks are interested in different events that have incomplete overlap in terms of attribute matching. We believe that coping with such situations is a distinguishing aspect of the future potential of BARD.

X. CONCLUSIONS

In our investigation into the application of Bayesian estimation techniques to limit flooding during route discovery we demonstrated that significant savings are available in terms of control traffic per event. Savings depend on the amount of traffic correlation in the application and the location of the data consumer. With completely uniform queries, BARD will not help, but when traffic is topologically correlated with some feature (such as a road), BARD can automatically discover and exploit that correlation even if it is not explicitly known to the application or user. We demonstrated savings from 28% to 73% for correlated traffic, depending on placement of the data consumer. Applications requiring the utmost real-time response should not use BARD. BARD uses occasional flooding to be responsive to network change and previously unseen events. The primary benefit of BARD is the pruning of repeated exploratory traffic across links that are not providing routes to interesting data. It is adaptable to a broad range of queries and event types. As a diffusion filter it can be easily added to existing applications that are running (push) diffusion.

REFERENCES

[1] J. Boyan and M. Littman. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In *Advances in Neural Information Processing Systems*, V6, pages 671-678., 1993.

[2] D. Bragansky, and D. Estrin. Rumor Routing for Sensor Networks. First Workshop on Sensor Networks and Applications, pages 22-31, Sept, 2002.

[3] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in Network Simulation. IEEE Computer, V.33 (N. 5), pages 59-67, May 2000.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Proc. ACM Symposium on Principles of DistributedComputing*, pages 1-12, 1987.

[5] H. Dubois-Ferrier, M. Grossglauser, and M. Vetterli. Age Matters: Efficient Route Discovery in Mobile Ad Hoc Networks Using Encounter Ages. In *Proceedings of the 4th ACM International Symposium on Mobile Ad-Hoc Networking*, pages 257-266, Annapolis, Maryland, USA 2003

[6]Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, pages 263-270, Seattle Wash., Aug 1999.

[7] Zygmunt J. Haas, Joseph Y. Halpern, , and Li Li. Gossip-Based Ad Hoc Routing. In IEEE INFOCOM, pages 1707-1716, June, 2002.

[8] John Heidemann, Fabio Silva, and Deborah Estrin. Matching Data Dissemination Algorithms to Application Requirements. In Proceedings of the ACM SenSys Conference, pp. 218-229. Los Angeles, California, USA, ACM. November, 2003.

[9] John Heidemann, Fabio Silva, Yan Yu, Deborah Estrin, and Padma Haldar. Diffusion Filters as a Flexible Architecture for Event Notification in Wireless Sensor Networks. USC/ISI Technical Report 2002-556

[10] Q. Huang, C. Lu, and G. Roman, Spatiotemporal Multicast in Sensor Networks. In Proceedings of the ACM SenSys Conference, pp. 218-229. Los Angeles, California, USA, ACM. November, 2003.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A scalable and Robust Communication Paradigm for Sensor Networks. In Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking, pages 56-67, Boston, MA, USA, August 2000. ACM.

[12] D. Johnson, and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. in Mobile Computing, pages 153-181. Kluwer Academic, 1996.

[13] Brad Karp, and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In Proceedings of the 6th Annual MOBICOM, pages 243-254, Boston, MA, 2002.

[14] J. Kulik, W Rabiner, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, pages 174-185, Seattle, Washington, USA 1999

[15] C.R. Lin and M Gerla. Adaptive Clustering for Mobile Wireless Networks. IEEE Journal on Selected Areas in Communications, 15(7): 1265-1275, 1997.

[16] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: Tiny AGregate Queries in Ad Hoc Sensor Networks. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, Boston, Massachusetts, USA, December 2002.

[17] C.E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing. In Proceedings of the Second IEEE WMCSA '99, pages 90-100 New Orleans, LA, Feb 1999.

[18] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-Centric Storage in Sensornets with GHT, A Geographic Hash Table. In Mobile Networks and Applications (MONET), pages 427-442 Kluwer, 2003.

[19] Stuart Russell and Peter Norvig. Artificial Intelligence A Modern Approach Routing, Prentice Hall Inc., 1995. ISBN 0-13-103805-2.

[20] Ivan Stojmenovic and Xu Lin, Power aware localized routing in wireless networks, IEEE Transaxtions on Parallel and Distributed Systems, 12(11): 1122-1133 , Nov. 2001

[21] R. Walpole, R. Meyers, S. Meyers. Probability and Statistics for Engineers and Scientists, Prentice Hall Inc., 1998. ISBN 0-13-840208-6.

[22] Yong Yao and Johannes Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. In SIGMOD, pages 9-18, 2002.

[23] Wei Ye, John Heidemann, and Deborah Estrin. An Energy-efficient MAC Protocol for Wireless Sensor Networks. In Proceedings of the IEEE Infocom. pages 1567-1576. New York, NY, USA. June 2002.

[24] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and Energy Aware Routing: a recursive data dissemination protocol for wireless sensor networks. Technical Report TR-01-0032, University of California, Los Angeles, Computer Science Department, 2001.

[25] J. Zhao, R. Govindan. Connectivity Study of a CSMA based Wireless Network. Technical Report TR-02-774, USC/ISI, Los Angeles, CA, 2002.

[26] F. Zhao, J. Shin, and J. Reich. Information-Driven Dynamic Sensor Collaboration for Tracking Applications. In IEEE Signal Processing Magazine, 19(2):61-72, March 2002