Synthesis Project
Exploiting Organization and Learned Knowledge to Reduce the Exploration Cost of Routing

Huzaifa Zafar, Victor Lesser and Deepak Ganesan

Computer Science Department
University of Massachusetts, Amherst
UMass Computer Science Technical Report 07-68

December 21, 2007

Abstract

Large scale multi-agent systems (MAS) benefit greatly from an overlay organization design [5, 8] that guides agents in determining when to communicate, how often, with whom, with what priority and so on. However, this same organization knowledge is not utilized by general-purpose wireless network routing algorithms normally used to support agent communication.

Here, we look at modifying the QRouting algorithm, in particular the confidence based extension of QRouting called CQRouting, to take advantage of the following set of information; 1) The layout of the organization in terms of roles and message flow, 2) Application knowledge in-terms of expected bandwidth, response time and message priority and 3) Sleep cycle knowledge.

In this paper, we show an improved application-level bandwidth and response time by applying organization knowledge to network-level routing algorithms. This increased bandwidth and response time is especially important in communication-intensive and power-limited application settings such as agent-based sensor networks where node availability and link dynamics make providing sufficient inter-agent communication especially challenging.

Contents

1 Introduction

There has been a lot of work recently in deploying sensor networks for monitoring and assessment of harsh environmental conditions. The characteristics of such networks include a limited power source and a dynamic network. This requires better management of unstable links and nodes while delivering packets in a timely and energy conserving manner. CNAS (Collaborative Network for Atmospheric Sensing) [4] is one such power-limited, wireless sensor network. It is a collection of power-aware sensor agents, spread sparsely over a topological region. Even small topological distances can have significant differences in atmospheric conditions. CNAS makes an overall assessment of detailed local atmospheric conditions by combining data from sensor agents within predefined cluster areas. A hierarchy of communication is used, where all the sensor agents collect data from their environment and transfer it to a dynamically designated sensor agent performing a second role of a "cluster head" responsible for aggregating and reporting the conditions in its cluster area. Moreover, the agent acting as the cluster head is also responsible for providing direction to the rest of the agents in its cluster. Due to the harsh environmental conditions, the network-level routing algorithm at each agent is working behind the scene to build dynamic paths for each message destination. These network dynamics are due to failures and changes in communication-link characteristics. The network-level routing algorithm at each agent in the network performs regular exploration of links to its neighbors. This determines the availability of the neighbor and the utility of the link between them in order to maintain paths to the cluster head. Also, agents acting as cluster heads communicate with a regional node, but with less frequency. Nevertheless, paths to the regional node must also be maintained. However, this form of communication is of extremely high priority and there is a stronger penalty for delayed delivery or dropped messages.

A major part of the research work on wireless routing algorithms has been in conserving the limited battery power available to the sensor nodes. Since the WiFi component is a major energy consumer [2], by balancing load on the links, and regulating the number of messages being sent and received, routing algorithms can have a strong influence on the longevity of the network. However there is only so much they can do. One practical solution would be to turn off the WiFi at each node periodically in order to increase its lifetime. In sparse networks like CNAS, there do not exist many paths a data message could take to its destination node, so switching off nodes can lead to network disconnections [18]. Furthermore, it does not make sense to have a certain percentage of the network go down periodically, as bottleneck nodes will die out faster (due to lack of power), rendering the network ineffective. An alternative approach is to have the entire network have a synchronous sleep cycle (provided there is no strong real-time demand for data). The task of the each agent in CNAS is therefore to wake up, perform all its communication as quickly as possible, and then go back to sleep again. Moreover an agent cannot go to sleep if another agent needs to use it as one of the hops in communicating with another agent. In this situation, the routing protocol needs to perform quick path discovery as soon as the network wakes up (even if the paths discovered are sub-optimal) as each agent tries to minimize the time spent in the wake cycle, so as to conserve its energy. The agents can then use these paths to quickly determine sleep cycles of the various nodes in the network based on their global critically in sending messages to the cluster head.

Standard routing algorithms perform regular exploration messages through the network. These exploration messages are used to handle the dynamics of the network, and are performed irrespective of the extent of these dynamics. This translates to a delay in sending messages every time the network starts up while the routing algorithm develops routes to the destination node. In cyclical networks like CNAS, this delay has to be borne every wake cycle. Furthermore, routing algorithms do not have a way to transfer their routing tables from one wake cycle to the next. Even though the a table from the past wake cycle can be used as a seed for the next wake cycle, there is no way to determine the validity of the old table and the algorithm has to re-explore. Finally, there has been no work done in embedding organization knowledge into routing algorithms to better direct exploration.

The focus of this research is to obtain better application bandwidth, response time and global utility in power-limited wireless sensor networks by better directing exploration. We extend the CQRouting algorithm (calling our algorithm eCQRouting) by using organization knowledge to guide exploration. We also use Kalman filters to determine the mean and variance on the delay in sending a message from the source to the destination and use this knowledge in determining which paths need to be explored. This learned mean and variance can then be transferred from one sleep cycle to another and used in developing better paths upon waking up.

2 CQRouting

In this paper, we modify the CQRouting algorithm [13] calling our extension eCQRouting. CQRouting is itself an extension to the QRouting [1] protocol based on the distributed QLearning algorithm [17][1]. QRouting defines the network as a MDP where the rewards associated with actions change over time. This requires constant exploration, in order to maintain the policies developed by the reinforcement learning algorithm. Each node in the network represents a state in the MDP. An action is to transmit a message from the current state to one of its neighboring states. A QValue, $Q_x(y,d)$, for node x states the expected time taken to transfer a message to destination node d through neighbor node y. A policy determines which neighbor a message is forwarded to so that it reaches its destination with minimum delay.

QRouting uses the "Full Echo" algorithm to perform exploration [1]. Each node in the network periodically requests each of its neighbor the policy being used by that neighbor in determining paths to various nodes in the network (which is defined by the routing table of that neighbor, and is based on the Bellman-Ford Algorithm for routing, first described in the ARPANET [14]). The querying node then updates its own tables based upon the following QLearning equation:

$$Q_x(y,d) = Q_x(y,d) + \alpha(Q_y(\hat{z},d) + q_y - Q_x(y,d)) \tag{1}$$

where α is the learning rate; $Q_y(\hat{z},d)$ is the utility of neighbor y in sending a message to destination d through $\hat{z}$, where $\hat{z}$ is the next hop on the optimal path from y to d; and q_y is the cost of sending a message to neighbor y. QRouting is different from QLearning in that the objective of the policy is to minimize QValues rather than to maximize it. This is because the QValue represents the expected cost in sending a message to the destination rather than expected reward used in QLearning.

In CQRouting each node also maintains a confidence in its QValues, $C_x(y,d)$. The confidence of node x in sending a message to node d, through node y, represents the expectation of node x that current $Q_x(y,d)$ value accurately represents the utility of using that path in sending future messages. Confidences ranges from [0,1], where a value of 1 implies perfect knowledge. When sharing routing tables, a node updates its confidence based on its neighbor's confidence by using the learning Equation as shown below:

$$C_x(y,d) = C_x(y,d) + \alpha(C_y(\hat{z},d) - C_x(y,d)) \tag{2}$$

The confidence of a node in sending a message itself is always 1. An agent then updates its own QValues by using confidence as follows:

$$Q_x(y,d) = Q_x(y,d) + \alpha C_x(y,d)(Q_y(\hat{z},d) + q_y - Q_x(y,d)) \tag{3}$$

Each node in the network, adapts to the new QValue provided for by its neighbors based on the its own confidence (and correspondingly its neighbors confidence) on the new value being the right value. Moreover,

[1]CQRouting is used over standard wireless network routing algorithms in our research as it learns the utility of network paths rather than using the current value. This provides for more stability when using Kalman filters in Section 4. Also, CQRouting has a pre-defined confidence measure which we use for directing exploration in Section 5.4.

for each time step the agent does not explore, its confidence decays due to the expectation that the network is dynamic and its QValues might not reflect the current state of the network, effecting future updates of its confidence.

3 eCQRouting

In the following subsections, we modify the CQRouting algorithm to allow for a smoother implementation in wireless sensor networks.

3.1 Cycles

The CQRouting algorithm does not guarantee that paths are free from cycles. It tracks and stores at each node a matrix of all possible next hops and their corresponding QValues to the destination. In case of link failures, when nodes pick their second or third best paths, steps must be taken to make sure that no cycles are generated. Currently we use two standard algorithms to prevent cycles, one is poison reverse [7], where the cost of a path from node A to its destination through its neighbor B is infinity if B uses A as its next hop to the same destination. This way, if the best path for A fails, A does not switch to B as its alternative path, and searches for other alternatives. The second algorithm handles the scenario where A has no paths to its destination except through B and is forced to use B. A cycle is prevented because each node on the current path to the destination adds its address to the header of the data message being sent, and does not send a message to a node already listed in that header.

3.2 Damping Factor

Consider the simple network shown in figure 1, where messages are being sent from the source node S to the destination node D.

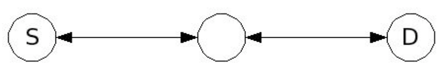


Figure 1: Small network example

At the end of the first exploration cycle, QValues are determined for each node that defines the expected delay in sending a message to the destination node D. Figure 2 shows the link costs and the resultant QValues. For the first cycle, since the QValues have yet to be initialized, we do not apply the learning factor to it, this is for ease of understanding. Applying learning factor does not effect the example.

The second exploration cycle, updates the cost of sending a message from node A to node D to 2.3 units. Figure 3 shows the resultant network, and the effect of applying Equation 1 with $\alpha = 0.80$ to it.

At the end of the second exploration cycle, node A uses node S as its next hop. As additional exploration is performed over that path, the QValue of both S and A increase, till the QValue of A using S as the next hop becomes greater than 1.9, at which point the flaw is corrected.

Since every exploration message consumes potential application-level bandwidth, we fix this flaw by modifying the QRouting algorithm to dynamically adjust the learning factor each time an agent updates its QValues. The new learning factor depends on the current QValue and the QValue provided by the agent's neighbor. For example, if we pick α at node S (and node A when updating its QMatrix for node S) to be

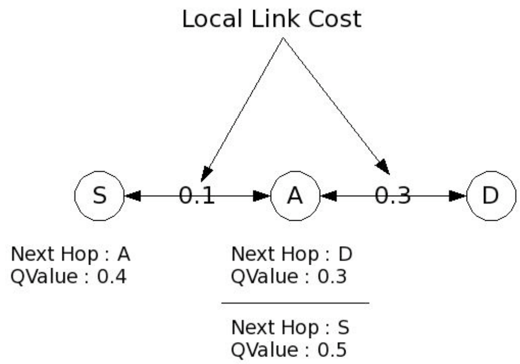


Figure 2: Small network example after exploration cycle 1

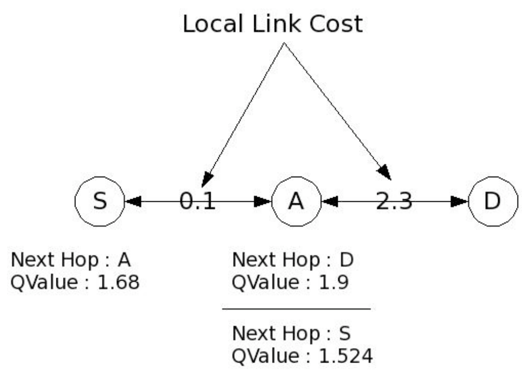


Figure 3: Small network example after exploration cycle 2. $\alpha = 0.8$ was used as the learning factor

greater than 0.9375, the QValues are corrected as shown in the example below (Figure 4). The policy then picks the right path, without having to undergo additional exploration. By continuing to use the learning factor, we retain the benefit of keeping the optimal path from flip-flopping between two close paths, at the same time reducing the number of exploration messages required to develop cycle free optimal paths. α can be calculated as follows:

$$\begin{aligned} Q_x(y,d) + \alpha(Q_y(\hat{z},d) + q_y - Q_x(y,d)) &> Q_y(\hat{z},d) \\ \alpha &> \frac{Q_y(\hat{z},d) - Q_x(y,d)}{Q_y(\hat{z},d) + q_y - Q_x(y,d)} \end{aligned} \tag{4}$$

The above value gives us a lower bound on the value of α, with the upper bound being 1. The new damping factor is

$$\alpha = 1 - \hat{\alpha} * (1 - \frac{Q_y(\hat{z},d) - Q_x(y,d)}{Q_y(\hat{z},d) + q_y - Q_x(y,d)}) \tag{5}$$

Here $\hat{\alpha}$ is the new constant that regulates how close we want our learning rate to be to 1. For CQRouting, we modified the α as follows:

$$\alpha = 1 - \hat{\alpha} * (1 - \frac{Q_y(\hat{z},d) - Q_x(y,d)}{C_x(y,d) * (Q_y(\hat{z},d) + q_y - Q_x(y,d))}) \tag{6}$$

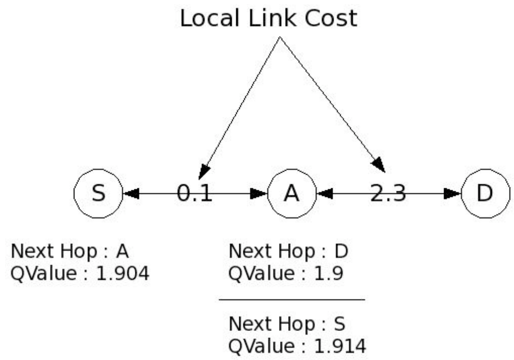


Figure 4: Small network example after exploration cycle 2. $\alpha = 0.8$ at node A when updating the QMatrix for next hop D and 0.94 at node S and node A when updating the QMatrix for next hop S

3.3 Updated Utility Function

There are number of path metrics used for wireless routing algorithms, Expected Transmission Time (ETT) [6] is one such example. ETT is a function of the loss rate and the bandwidth of the link defined by the following formula:

$$ETT \quad = \quad \frac{1}{d_f * d_r} * \frac{timeDelay}{packetSize} \tag{7}$$

Where d_f is the probability of the packet successfully arriving its destination, d_r is the probability of the acknowledgment successfully arriving, timeDelay is the time taken to send the exploration message across and packetSize is the size of the exploration message. The issue with ETT is the time and exploration messages required in gaining a usable approximation of both d_f and d_r, which makes the method very expensive for energy conserving wireless networks. A cheaper approximation is provided by the time delay function used in CQRouting. $Q_x(y, d)$ is defined as the expected time taken in sending a message from the source node x to the destination node y. For any link in the network, the delay is high if the link is overused, has a lot of interference, has a lot of collisions or if the bandwidth of the link is low.

The issue with using time-delay comes from the coupling of all effects into one representation of the utility function. In eCQRouting, we are interested in networks where path usage is likely to exceed its bandwidth limitations. We would like our utility function to account for the minimum bandwidth across a path from the source to the destination, in-effect capturing potential bottleneck nodes. The utility function should be able to balance between paths where the response time is low and paths where the available bandwidth is high. We redefine $Q_x(y, d)$ as follows:

$$Q_x(y, d) \quad = \quad A * \frac{current - response - time}{average - response - time} + B * \frac{average - bandwidth}{current - bandwidth} \tag{8}$$

where average-response-time and average-bandwidth are constants provided by the application, and can be calculated by estimating the response time and bandwidth available on the network before hand. The function of the two constants are to normalize the two values for different units, and they do not have to be precise. A and B are weights on each of the fraction so as to determine which of the two is more important in determining the utility of the path and are constant too.

3.4 Exploration

CQRouting uses the Full Echo' algorithm for exploration as defined in Section 2. We call this kind of exploration, where the source node is responsible for finding its destination, source based exploration. In Section 5.3, we move to destination based exploration, where the destination node is responsible for informing sources of its existence. However the criteria for exploration remains the same across the two methods. In eCQRouting, exploration is triggered once the confidence of the source node in the utility of the path to the destination drops below a certain threshold. In our experiments, the threshold is fixed at 0.5. In sending a message from node y to node x, node y provides the current-response-time and current-bandwidth value of the path used by its policy. Next each node locally calculates their new QValue as follows:

$$
\begin{aligned}
current-response-time_x &= current-response-time_y + q_y && (9)\\
current-bandwidth_x &= min(current-bandwidth_y, current-bandwidth_x) && (10)\\
Q_x(y,d)_{new} &= A * \frac{current-response-time_x}{average-response-time} + B * \frac{average-bandwidth}{current-bandwidth_y} && (11)\\
Q_x(y,d) &= Q_x(y,d) + \alpha C_x(y,d)(Q_x(y,d)_{new} - Q_x(y,d)) && (12)
\end{aligned}
$$

hence replacing Equation 3 with the Equation 12.

The current bandwidth between two paths is calculated by sending two exploration messages consecutively, and monitoring the added time delay in receiving the packet. The bandwidth is calculated by dividing the size of the packet with this added delay.

4 Using Kalman Filters to develop confidences and direct exploration

For cyclical networks, we would like to be able to model the QValues to accomplish two things: 1) We would like to be able to generate QValues at the beginning of every wake cycle. This allows us to develop (possibly suboptimal) paths to the destination without having to perform any exploration messages. 2) Exploration has to be performed if the QValues do not accurately represent the state of the network. In order to be able to determine the accuracy of our QValues, we want our model to tell us which parts of the network is most likely to deviate from the current QMatrix and perform exploration to fix this deviation and find optimal paths.

eCQRouting uses Kalman filters [9] to develop a mean and variance on the utility function defined in equation 8. The mean and variance is then used to measure confidence that the utility value remains unchanged as the network transitions from one time step to another. Indirectly the confidence also measures the relative goodness of the decisions taken in the new time step based on the values from a previous time-step. Exploration can then be performed to update the utility of taking a decision, only if the confidence for that utility drops below a certain pre-defined threshold.

The validity of knowledge transferred from one time step to another depends on the stability of the network. In an unstable network, especially when nodes wake up after having their WiFi radio turned off for an extended period of time, the network could have changed enough from one time period to another to invalidate most of the QValues. However in stable networks, its highly unlikely values and decisions have changed much over time, and we can continue using paths from our previous time period. Given a model of the past, a stable network will be able to better predict the future, resulting in high confidence and low exploration. In unstable networks, confidences would remain low, resulting in more exploration. Moreover if there is some structure in the way the unstable network changes over time, our learning algorithms may be able to catch that being reflected in the model of the QValues and be able to further lower exploration.

Kalman Filter is a mathematical procedure that allows for a least-squares estimation of any given model. Its a two step algorithm. The first step, predicts a short term value using the model developed so far. In eCQRouting, this prediction is used to develop the Q-Matrix and the corresponding Routing table at any given time step. The second step, updates the model based on the next observation. The accuracy of its prediction is modeled using the Gaussian estimation, providing a confidence on the QValue.

Each node uses the following model to develop the local estimation of the state of the network every-time an exploration is performed.

$$
\begin{aligned}
Q_x(y,d)_{k+1} &= A * \frac{expected - response - time}{current - response - time} + B * \frac{current - bandwidth}{expected - bandwidth} && (13)\\
Q_x(y,d)_{k+1} &= Weight * Q_x(y,d)_k && (14)
\end{aligned}
$$

The Kalman Filter algorithm takes in A, B, $Q_x(y,d)_{k+1}$ and the model defined in Equation 14. It learns the two fractions defined in Equation 14 based on its inputs.

Kalman filters also outputs a Gaussian based confidence depending on the variance in $Q_x(y,d)$. If the variance is high, the confidence on a value close to the mean would be lower than if the variance were low. Finally, the further away the current value is as compared to the learned mean, the lower the confidence. Also, if the number of updates provided is low, Kalman filters will be unable to learn an accurate mean, and will output a low confidence value.

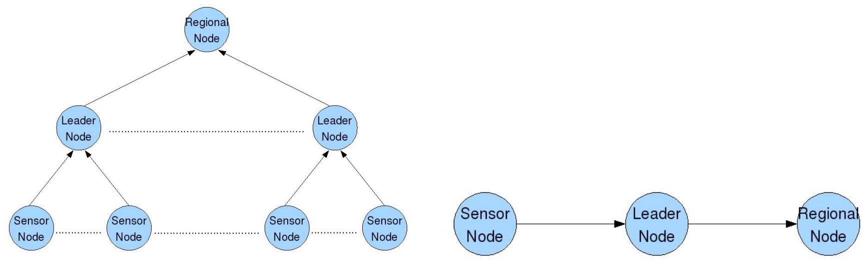
The confidence developed by the kalman filter, provides for a natural way to trigger exploration. Whenever confidence drops below a certain threshold, nodes perform additional exploration so as to improve their local models. As a result, while the model is being learned the frequency of exploration is high. With additional messages, the model is able to represent the QValues better leading to higher confidences and reduced exploration.

To summarize, the QValue defined by Equation 8 (and updated using the QLearning Equation 3) is used by eCQRouting to determine its policy. Confidence values from the Kalman filter is used to direct exploration. The mean value learned by the Kalman filter is used to define an approximate QMatrix when the network restarts after a sleep cycle before sending out exploration messages.

5 Integrating organization knowledge with the routing protocol

Most agent-based sensor networks, including CNAS, have an organizational structure that can be provided to the routing algorithm. For each agent in the MAS, the organization knowledge defines which other agents they will be communicating with, in effect defining the flow of messages in the overall network. For example, consider the three level hierarchical organization used in CNAS (Figure 5a).

Every sensor agent in CNAS sends its observations to a cluster head (depicted as leader agents in Figure 5a). The leader agent then accumulates observations from all the sensor agents assigned to it, and sends it to the regional agent. In the following subsections, we describe incorporating this knowledge in the routing algorithm used at each agent and the corresponding routing-level exploration changes, given this knowledge.



(a) (b)

Figure 5: CNAS — Organizational Design

5.1 Utility Function

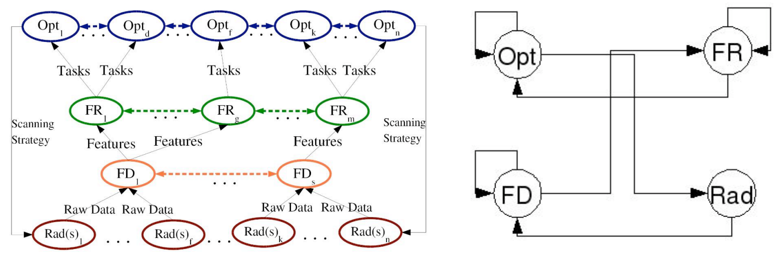
As mentioned in Section 3.3 the application knowledge provides the average bandwidth and average response time along with the weights on both to the routing algorithm. This is done for each source-destination pair and is available to each node in the network. Since bandwidth represents both the frequency with which messages are to be received and the size of the message, they don't have to be represented separately in the application knowledge or in the routing algorithm.

5.2 Knowledge of the organization

We represent the organization as a directed weighted graph (the weights will be described shortly). Each agent role is a vertex in the graph, and the edge represents the direction in which messages are being sent. Each agent has a copy of the graph, and it knows the roles it plays in the organization. This knowledge is incorporated into the routing algorithm used at each agent, and exploration decisions are based on the roles that a specific agent is assigned. Figure 5b shows the graphical representation for Figure 5a.

Figure 6a shows a second, more complicated organization as used in CASA (Collaborative Adaptive Sensing of the Atmosphere) [12] another agent-based atmospheric sensor network with considerably higher bandwidth requirements than CNAS. In CASA, there are four roles an agent can take; Rad(s) (Radar(s)), FD (Feature Detector), FR (Feature Repository) and Opt (Optimizer). The organization is in the form of a graph based hierarchy. Here an agent is responsible in communicating with other neighboring agents that are performing the same role as itself as well as communicating with parent agents, however with a lower priority. Also, agents with roles higher on the hierarchy communicate less frequently, and these messages have a much higher priority than communications among agents that are lower in the hierarchy. The corresponding graph is shown in Figure 6b.

Each agent is also provided with a table that lists all the other agents in the network, the roles they play, and the destination agents they are interested in communicating with. We believe that providing this kind of "global" view of the organization knowledge is reasonable as it is considerably more stable than the routing tables used by the network-layer routing algorithms, and building this global perspective locally has negligible costs when compared to sharing routing tables. As the size of the network increases the scale of both the organization structure and the routing table increase. However the organizational knowledge limits the size of the routing table by limiting destinations nodes each source node communicates with by appropriately guiding exploration messages and consecutively path development.



(a) (b)

Figure 6: CASA - Organizational Design

5.3 Exploration Based on the Organizational Representation

A destination agent in the organization knows (from its organizational knowledge) all the nodes that are interested in communicating with it. Also, in most networks like CNAS the number of destination agents is much smaller than the number of source agents. In eCQRouting, the destination node performs periodic exploration of the network[2]. The extent and direction of the exploration is defined by the placement of agents interested in communicating with it. The initial exploration messages are used by the destination to find the other agents. Future exploration messages are then directed according to the location of agents interested in communicating with the destination. The QValue at the source node is defined by the delay in receiving the exploration messages.

5.4 Exploration based on Confidences in QValues

In eCQRouting, an agent explores when its confidence drops below a certain threshold. Since exploration happens at the destination, while confidence is calculated at the source, confidence is piggybacked on application messages from the source to its destination. The destination agent updates a local copy of the confidences of all agents sending messages to it and uses its local copy to determine when to explore. Furthermore the destination copies decay over time so that the agent can explore even if there are no application messages sent to it.

5.5 Message Priority

In Section 5.2, we defined our organization knowledge as a weighted graph, the weight on an edge of the graph is based on the priority of the messages sent from one agent to another. Figure 21 is an extension of Figure 6b with priorities added. Messages that are critical have a much higher priority as compared to other messages and require additional exploration to maintain the optimal path, even though the number of messages sent might be much lower. For example, the part of the role of the "Optimizer" from the organization design used in CASA is to define the scanning strategy for the "Radar" nodes. Lost scanning strategy messages can be very expensive to the application as it could potentially mean radar nodes are following an out-dated scanning strategy. Since CASA uses focused radars, this could lead to loss of tracking data.

[2] eCQRouting has its destination nodes perform the exploration over source nodes because if there is a change in the path from the source to the destination node, the source has to send a message to the destination and wait for the return message before it can make decisions based on the changed value. If the destination performs the exploration, the source node can take advantage of the changes the moment the exploration message arrives. This cuts down the number of exploration messages to half. Destination based exploration is performed in OLSR, while DSDV and CQRouting does source based exploration

At the application level, an “Optimizer” node can delay sending messages to other “Optimizer” nodes if it has a scanning strategy that needs to be communicated, by using a priority queue which regulates when to send what message based on its importance. A problem with this strategy is priority-based starvation, where low priority messages wait infinitely while high priority messages are added ahead of them in the queue. As a routing algorithm however, we would like lower priority messages to be communicated using suboptimal paths and be eventually delivered, while saving optimal paths for high priority messages.

In eCQRouting we solve this problem by using two techniques. The first technique uses an approximate measure of message priority for each source-destination pair. The priority is then used to regulate the frequency with which exploration is performed by the destination agent. In the second technique, each agent has a local measure of the performance of the MAS based on the probability of dropping a message to the destination. For each message, decisions are then taken based on this message probability.

5.5.1 Exploration based on message priority

Starting with a very simple case, shown in Figure 7. After the first round of exploration is completed, S_1

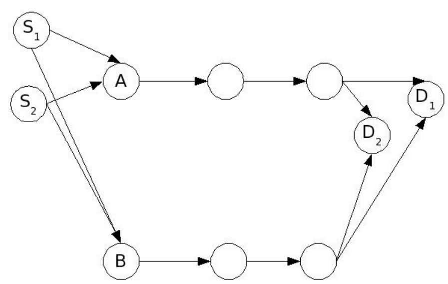


Figure 7: A simple network to illustrate priority behavior

and S_2 both determine A to be the best path from themselves to D_1 and D_2 respectively. However as both of them start using A as their next hop, the network discovers the bandwidth from A to D_1 and D_2 is insufficient to transmit both sets of messages. This increases the time delay in sending messages from S_1 and S_2 to their respective destinations. The next time an exploration happens, both D_1 and D_2 explore, find the path through B to be the better next hop and S_1 and S_2 both choose B causing an overload on B.

In eCQRouting however, we lower the threshold at which a destination node explores based on the priority by using the formula $threshold = priority * sd - threshold$. The sd-threshold is uniform for all source-destination pairs. This tolerates a lower variance on the value of high priority messages before forcing exploration to resolve them, resulting in better mean and variance development for those values. It also means if S_1 has a higher priority over S_2, it will explore more, forcing it to pick a suboptimal path through B, in order to make sure it resolves the bottleneck. Next time S_2 explores, it would keep the optimal path through A. The technique is suboptimal is a static network, but permits a high priority source-destination pair to be more aware of the state of the network in developing paths to the destination.

The disadvantage of this algorithm is the system has to wait for the next exploration message to determine new paths, which can delay decision making past a critical point.

5.5.2 Per-Message decision depending on message loss

The second algorithm takes into account the expected effect of performance with relation to message loss and performs local decisions based on that effect. For example, once S_1 and S_2 start sending messages, they realize 10% of their messages are being dropped before they reach A. If the performance curve for messages from S_1 is such that it cannot handle the 10% loss, it will pick the path through B with the expectation that that path will deliver more than 90% of its messages.

With exploration, the probabilities are shared in the following way. Lets assume the network in Figure 7 has the probabilities shown in Figure 8. Here, E will share 1.0 with node C. Node C will share $0.93 * 1.0 = 0.93$

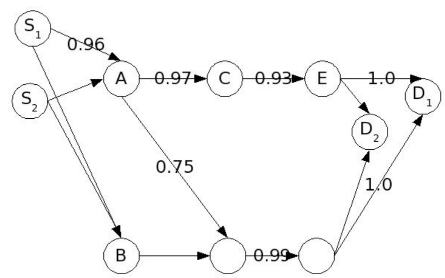


Figure 8: A simple network with probability of sending a message from one node to another

with A. A has two options $0.75 * 0.99 * 1.0 = 0.7425$ and $0.97 * 0.93 = 0.9021$. Since 0.9021 is the greater, it shares that value with S_1. In making a decision S_1 will either pick A knowing the probability of messages to get through to be 0.866, or pick B assuming the probability of getting the message across is 1.

The disadvantage of this algorithm is it takes the next exploration to make a global decision, and a local decision might not always be optimal. The second disadvantage is it takes time to learn the probability of dropping messages on any link, which can be detrimental to the performance of the network.

6 Experimental Analysis

All experiments were conducted using NS2 [11], a standard network simulator. NS2 provides models for standard network link dynamics as well as message interference, both of which strongly govern available bandwidth. We also used the widely used NS2 implementation of OLSR developed at University of Murcia [16] and the Kalman filter implementation from the Bayesian Filtering Library [10]. In our experimental analysis, we explore the implications of learning the variance of the network, as well as adding organization knowledge to the lower level routing algorithm by looking at 1) the effect of increasingly unstable networks 2) the effect on performance with increasing number of exploration messages, 3) the scalability of the algorithm as the number of non-application nodes increase, 4) the effect of bandwidth available on messages with different priority

6.1 Performance in Non-Cycling Networks

In this section, we look at networks where the bandwidth available is insufficient for the demands of the application, and exploration messages have to share space with application specific messages. Since bandwidth available to the network remains relatively constant across sleep cycles, we start by looking at networks with no sleep/wake cycles. We evaluate the performance of our algorithm with standard proactive routing algorithms OLSR and DSDV by measuring the bandwidth available for sending application messages.

6.1.1 The effect on performance with increasing number of exploration messages

In this experiment, we use the CNAS organization structure on a 1-D network shown in Figure 9.

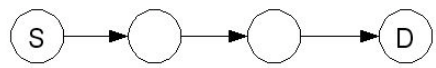


Figure 9: 1D network

Here S represents a sensor agent, and D a cluster head agent. Next, we add neighbors to both the sensor agent and the cluster head that are not used by the high-level application and thus not part of the organization structure. The neighbors are added such that they do not add additional paths from the source to the destination. Figure 10 shows what the network looks like after adding 3 neighbors to both the source and the destination.

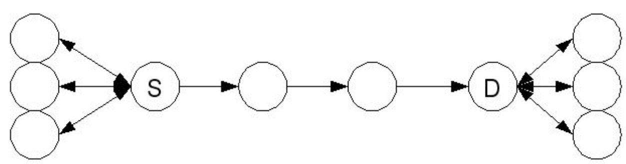


Figure 10: Adding 3 neighbors to both source and destination in Figure 9

In this setting, with no additional neighbors, the maximum bandwidth available if no exploration were to be done was determined to be 180Kbits/sec. Agent S sends 180 1Kbit messages to agent D every second for 150 seconds. The average number of messages that reach agent D every second determines the bandwidth for that run. We performed 10 such runs and display the average bandwidth over the 10 runs. Also, the network is completely stable. No links fail during the course of the experiment. As additional neighbors nodes are added to the network, traditional routing algorithms (OLSR and DSDV in our experiments) performed additional exploration messages to include these nodes in the routing tables of all other nodes in the network. eCQRouting prevents this from happening by taking advantage of the CNAS structure in determining agent D to be the only destination node in the network. The effect on the bandwidth available to agent S in sending its data to agent D is shown in Figure 11

As the number of neighbors increase, the application level bandwidth available when using eCQRouting is significantly better than the next best algorithm (OLSR), with performance improvements of 20.9% with one neighbor to 36.55% with 10 neighbors. The algorithm also provides 30.5% additional bandwith with one neighbor over CQRouting, which increases to 44% improvement with 10 neighbors. The reason for this improvement is the limited number of exploration messages in eCQRouting. eCQRouting uses 15% of the number of exploration messages in OLSR with one neighbor, which drops to 7% with 10 neighbors.

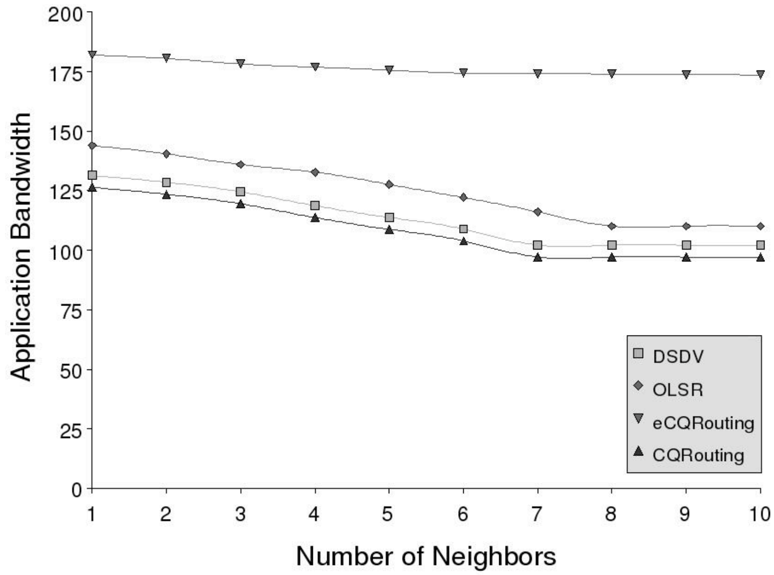


Figure 11: Changes in bandwidth as number of neighbors (and in-effect the number of exploration messages) increase. Bandwidth is calculated in KBits/sec and averaged over 150 seconds

6.1.2 Scalability of the algorithm with increasing number of nodes

In this experiment, we keep the CNAS application from the previous experiment, but move to a 2D grid network. We now have 3 sensor agents, and 1 cluster head. The 4 agents lay at the 4 corners of the grid. We explore grids of size 2x2 through 10x10. Figure 12 shows an example 5x5 grid layout. The sensor agents are marked as "S" and the cluster head is marked as "D". The density of the network is such that each agent has at a maximum 4 neighbors.

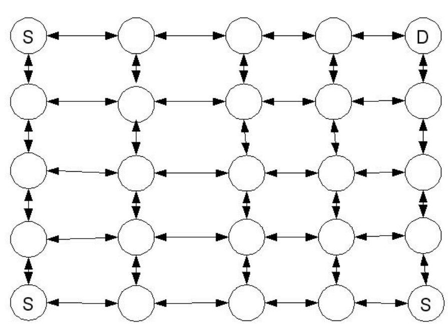


Figure 12: 5x5 grid layout for the CNAS organization

As the size of the network increases, the number of exploration messages also increase. However, because we keep the density of the network the same, the number of hops to the destination also increases. The algorithm needs to be able to generate paths, that are viable over the longer distances. Sensor agent sends 180 1Kbit messages to agent D every second for 150 seconds. The average number of messages received by agent D every second determines the application-level bandwidth for the run. We performed 10 runs and display the average. Again, the network is completely stable. Figure 13 shows the effect of this increasing size on the bandwidth.

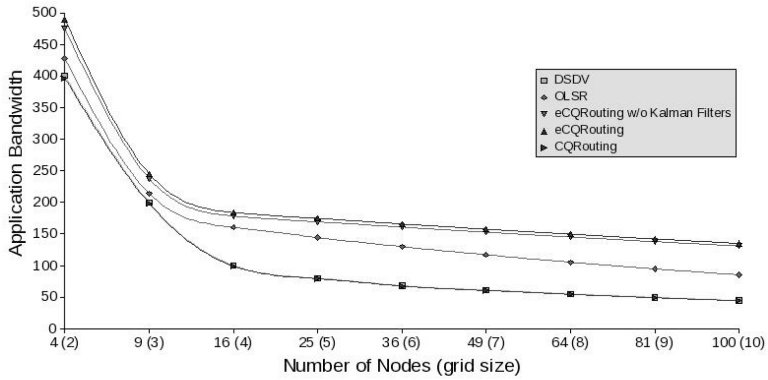


Figure 13: Changes in bandwidth as size of the grid network increases. Bandwidth is the average bandwidth over 150 seconds, measured in KBits/sec

The figure shows an initial drop in the bandwidth available, as the destination node is no longer 1 hop away from the sensor agents. The reduction in application-level bandwidth diminishes as additional hops are added between the source and the destination. The performance improvement in this setting range from 10% for the 2x2 grid network to 35% for the 10x10 grid network over OLSR. The improvement over CQRouting goes from 16.5% for the 2x2 network to 66% for the 10x10 network. Both DSDV and CQRouting have very similar performance due to similar source based exploration algorithms. As the number of hops increase, the time to converge to the optimal path also increases.

6.1.3 Scalability of the algorithm with increasing density

In this experiment, we start with the 5x5 grid layout defined in Figure 12 with the CNAS organization structure. We keep the area constant, and add 25 to 200 nodes in random positions within the network. This increases the density of the network, which inturn increases the number of paths from the source nodes to the destination node. Figure 14 shows the result of running the experiment. We performed 20 runs for each density variation and calculate the average application bandwidth over the 20 runs. Each run is for 150 seconds.

Figure 14 shows, eCQRouting does about 17% better than OLSR in the 5x5 grid network with minimal density. When density is doubled, the performance improvement increases to about 19%. At triple the density, the performance improvement is at 10%. When the density reaches 8 times the minimal density, OLSR starts outperforming eCQRouting. This is because OLSR determines multi-point relay (MPR) nodes, which it uses to curb the increasing rebroadcasts of destination messages with increasing density. MPR nodes are selected such that it is the minimal subset of nodes that rebroadcast an exploration message, while making sure the entire network receives them. It is this optimization that causes OLSR to do much better than all routing algorithms in extremely dense networks like the one used in this experiment.

Secondly, as the density increases, the number of paths to the destination nodes increase, which inturn increases bandwidth. However, after a certain point, the cost of exploring the increasing number of paths, outweighs the bandwidth benefit, and the application level bandwidth drops. OLSR drops the slower than the rest, due to the MPR optimization.

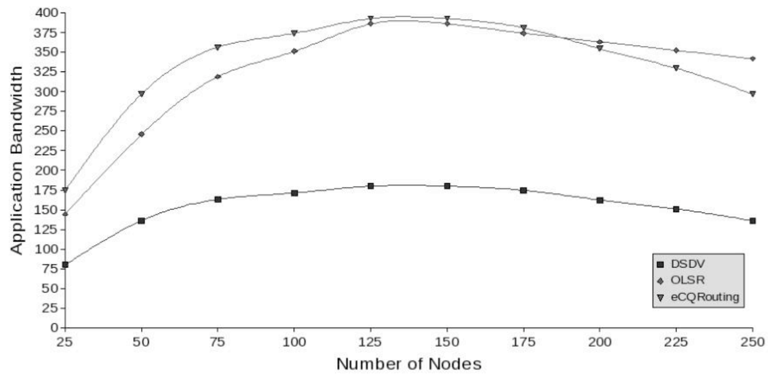


Figure 14: Changes in bandwidth as network density increases

6.1.4 Scalability of the algorithm with increasing source and destination nodes

In this experiment, we take the 10x10 grid network, keeping the CNAS organization. Every node in this network is a source node (therefore we have 100 source nodes). We randomly place destination nodes all across the network. Each source node communicates with the destination nodes nearest to it. We performed 20 runs for each destination node count, and calculate the average bandwidth over the 20 runs. In Figure 15 we plot the total bandwidth as the number of destination nodes in the network increases.

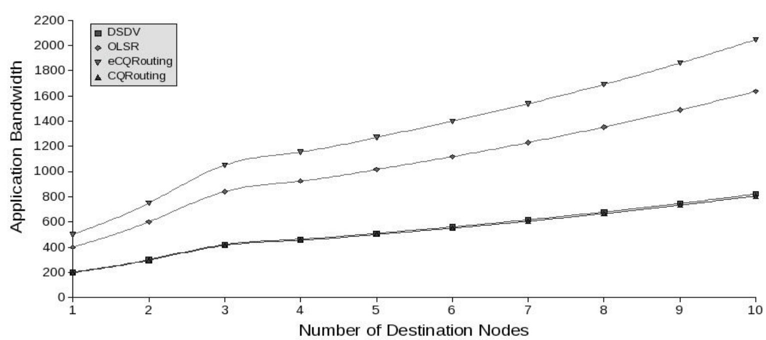


Figure 15: Changes in bandwidth with increasing number of destination nodes in a 10x10 grid network with CNAS organization structure. Bandwidth is the average bandwidth over 150 seconds, measured in KBits/sec

Figure shows, eCQRouting increases its performance improvement from about 20% with 1% destination nodes to 25% with 10% destination nodes. With increasing destination nodes, the number of exploration messages increases in eCQRouting. However the performance improvement over other routing algorithms also increase as exploration messages guided by the organization design are not spread over the entire network.

6.2 Performance in Sleeping Networks

In this section, we introduce sleep/wake cycle in selected networks from the past section, and analyze the effect of our routing algorithm on them. In most of our experiments, the network sleeps for 40 seconds followed by a 20 second wake cycle. There are a total of 10 such sleep cycles. We are interested in monitoring the delay in sending the first message from each source to the destination node.

The delay is denoted as response time and is calculated as follows:

$$ResponseTime(cycle_i) = \frac{\sum_{i=1}^{numberOfNodes} delay(node_i)}{numberOfNodes} \quad (15)$$

$$AverageResponseTime = \frac{\sum_{i=1}^{10} ResponseTime(cycle_i)}{10} \quad (16)$$

Where $ResponseTime(cycle_i)$ is defined as the response time for the i^{th} cycle. $numberOfNodes$ is the number of source nodes in the network. $delay(node_i)$ is the time taken for the first application message from node i to reach its destination. The graphs in the following subsections depict the result of Equation 16.

6.2.1 Effect on performance with increasing number of exploration message

In this experiment, we use the network from Section 6.1.1 (and depicted in Figure 10). Table 1 shows the results.

Routing Algorithm	Response Time
eCQRouting with Kalman Filters	0.7 Seconds
eCQRouting w/o Kalman Filters	3.2 Seconds
OLSR	4.3 Seconds
DSDV	6.2 Seconds
CQRouting	6.5 Seconds

Table 1: Response Time for various routing algorithms

As the number of neighbors increase the response time remains the same. This is because, time delay is usually dependent on the number of hops from the source to the destination and the usage of the path. In our experiment, the number of hops to the destination node remains constant. Usage on the other hand does increase. However, usage effects time delay only if the number of exploration messages is greater than the bandwidth of the link. In sending the first message, the bandwidth of the path is not exceeded, and increasing the number of exploration messages does not make any difference.

eCQRouting with Kalman Filters shows a significant performance improvement over the other algorithms due to the transfer of knowledge from one sleep-cycle to another. In stable networks like the one used in this experiment, there is minimal change between consecutive wake cycles. By modeling and transferring knowledge from one cycle to another, the average response time drops significantly as compared to other routing algorithms. This improvement in response time complements the bandwidth improvement (shown in Figure 11). Low response time implies the routing algorithm is able to send its first message relatively quickly, while the high application-bandwidth allows for future messages to have a much lower response time.

6.2.2 Effect on performance with increasing number of nodes

In this experiment, we use the network from Section 6.1.2 (see Figure 12). However, we increase the wake time from 20 seconds to 1 minute. The sleep time is increased from 40 seconds to 4 minutes. This is to account for the high response times for both DSDV and CQRouting. The rest remains the same from the previous experiment. Figure 16 shows the results.

In small networks, the number of hops from the source node to the destination node is small. Because of this, there is no effect of the exploration messages on the response time. However, as the number of hops

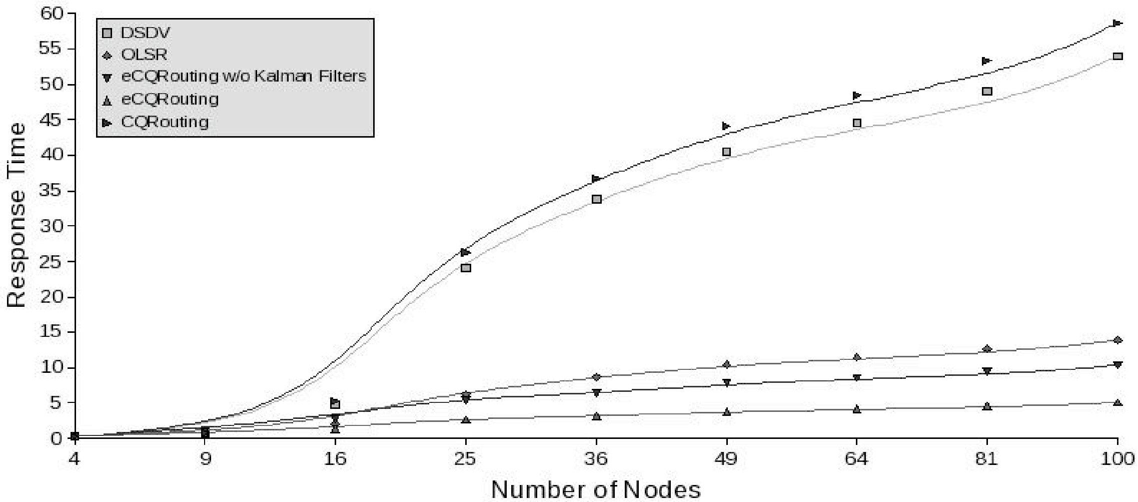


Figure 16: Changes in performance time as the size of the grid network increases

(and in-effect the size of the network increases), there is a corresponding increase in the response time. With 100 nodes, the response time for eCQRouting with Kalman Filters is 37% of OLSR.

6.2.3 Effect of performance on networks with increasing density

This experiment uses the same setting from Section 6.1.3. Figure 17 shows the results.

As the figure shows, with 25 nodes, eCQRouting requires 43% of the time required by OLSR in sending the first message from the source to the destination. At about 150 nodes, this increases to 59%. However, after this point, even though OLSR is able to reduce the number of exploration messages, and perform much better bandwidth wise, it does not have the same effect on the time taken to send the first message and even at 250 nodes, eCQRouting takes 59% of the time. This is mainly because of being able to transfer knowledge learnt in one sleep cycle to another, and being able to direct exploration better.

6.3 Performance in unstable networks

For the rest of the experiments, we move from the CNAS organization structure to the CASA organization structure. This allows for both increasing the percentage of destination nodes in the network and exploring the implication of priority on different messages. In the following experiment, we also evaluate the effectiveness of the routing algorithm on unstable networks, and its performance when compared to other pro-active routing protocols. We use the CASA organization on a 5x5 grid network. The density of the network is such, that each node is at the edge of the WiFi transmission capacity of its neighboring node and hence has at most 4 neighbors. We have 4 source nodes and 4 destination nodes as shown in Figure 18. The figure is to be interpreted as follows: Each node in the network plays two roles. Messages are sent with respect to the organizational structure shown in Figure 6. However, in-order to better regulate the follow of the network, each node is interested in communicating with the node diagonally across it on the network. The role that comes in play when sending messages is labeled as the source role, and the other role is the destination role. By having nodes send messages diagonally across the network, each source-destination pair is required to explore the entire network or potential paths.

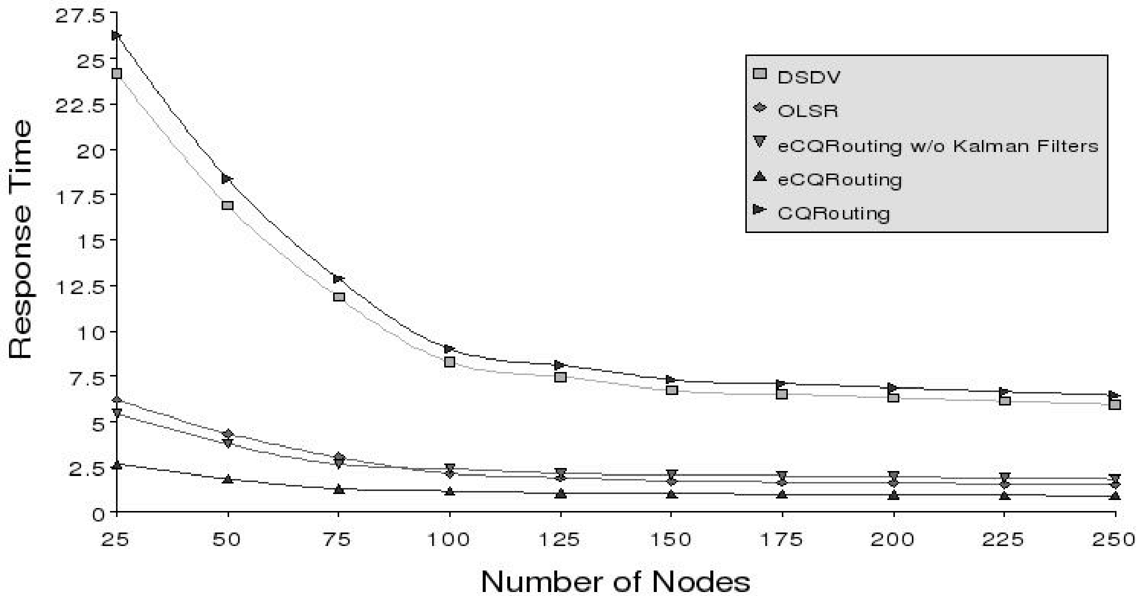


Figure 17: Changes in performance time as the density of the network increases

Each link in the network fails at each time step with a certain probability. In this setting, the maximum total bandwidth available if no exploration were to be done was set at 150Kbits/sec. With the CASA network, the majority of the messages are between the Radars and the Feature Detectors (FD in Figure 18). In our experiment the Rad node sends 100 1Kbit messages to the FD node every second. Next, FD nodes send 25 1Kbit messages to the FR nodes per second. FR nodes send 20 1Kbit messages to the Opt node and finally Opt nodes send 5 1Kbit message per second. The average total number of messages that reach agent destination agent every second determines the bandwidth for that run. We performed 10 such runs. Figure 19 displays the average bandwidth over the 10 runs.

As the probability of link failure increases, the effective bandwidth drops. eCQRouting is able to maintain a higher application-level bandwidth throughout. As the failure rates increase, additional exploration messages are required to find new paths, because of which the percentage improvement eCQRouting without Kalman Filter has over OLSR [3] and DSDV [15] drops. However eCQRouting with Kalman filter, measures and uses the mean and variance of the link for the time period that it is available. Because of which it is able to maintain a better bandwidth when there is high link failure rates. However once the probability increases to 0.7, there is a sharp drop as links are un-available for a majority of the time period and the routing algorithm is unable to use its learned values.

For the second part of this experiment, we have the network sleep for 40 seconds and wake for the remaining 20 seconds in a minute. Figure 20 shows the results of this experiment. We do not display the results from DSDV and CQRouting as their response time is greater than the 20 seconds the network is awake for, because of which no messages are able to get through.

As we can see, the response time for eCQRouting with Kalman Filters remains low when compared to OLSR. eCQRouting takes into account the sleeping cycles, and transfers knowledge from one sleep cycle to another. Because of this the average time for sending the first message is quite low. As the probability of link nodes failing increases, eCQRouting is unable to learn much about the network in each cycle, causing its response

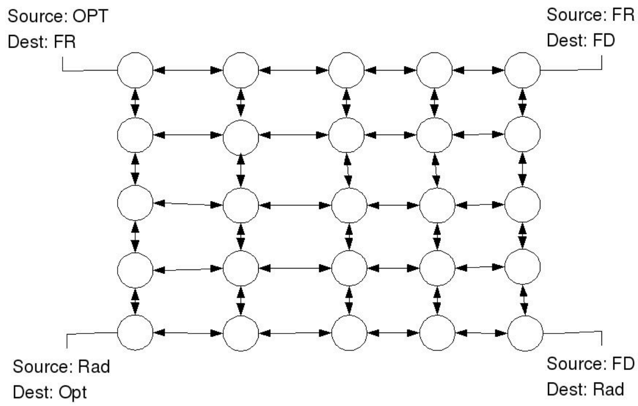


Figure 18: 5x5 grid network using the CASA organizational structure, showing the 4 source and 4 destination nodes

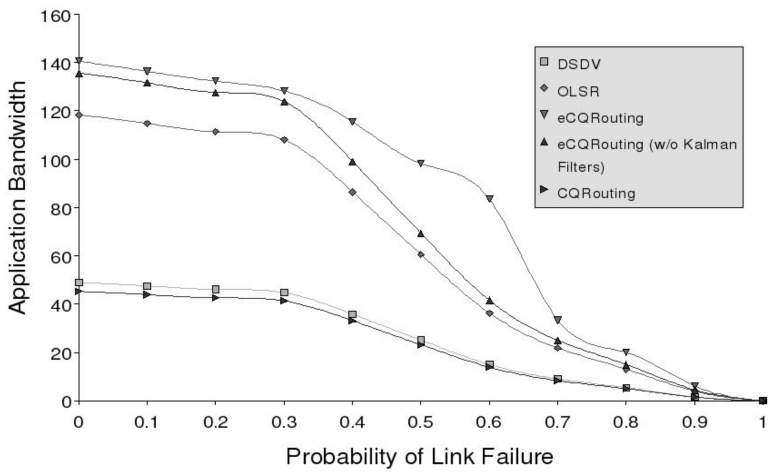


Figure 19: Changes in bandwidth as the probability of link failure increases

time to increase. However the response time for OLSR increases at a much faster rate with the lack of this information.

6.4 Effect of the bandwidth available on messages with different priority

In this section, we explore the effect of a significant drop in the percentage of messages that get across from the source to the destination using the CASA organization structure. We show how using priority helps maintain global utility.

6.4.1 Effect where performance is related to priority

To explore the effect of priority, we used the CASA structure in a 5x5 grid network similar to the one used in the previous experiment (Figure 18). We added priorities on the messages between agents as shown in Figure 21

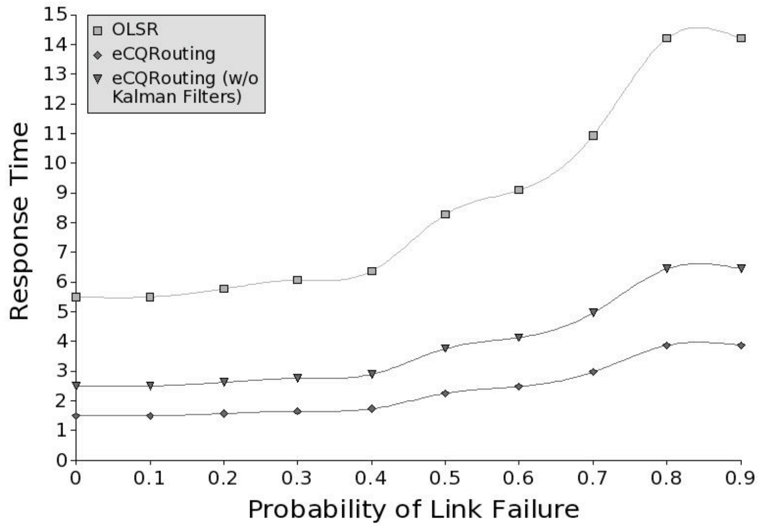


Figure 20: Changes in response time as probability of link failures increases

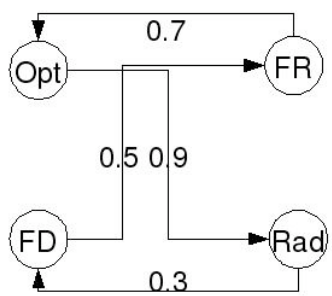


Figure 21: Role Priorities

In this experiment, we keep the probability of link failure at 0.2%. We know from the previous experiment, that the cumulative application-level bandwidth obtainable by eCQRouting in this network (with this setting) is about 135 Kbits/sec. In this experiment, each source agent sends 32 1Kbit messages to the destination agent per second for 150 seconds. At 135 Kbits/sec, each agent is able to send all its messages to the destination node. Figure 22 shows the effect of reducing the effective bandwidth on the global utility. The global utility is calculated by assigning a score equal to the priority of the message for every 32 messages received at the destination node. For example, if the Radar agent receives 4000 messages from the optimizer agent during the course of the run, the utility contributed towards the global utility would be $4000/32*0.9 = 112.5$.

As the bandwidth available to the source nodes drop, the utility of the application drops, since not all messages are delivered. However, the global utility of eCQRouting with priority drops at a much slower rate, as it is able to generate better paths for those agents where the priority is higher by performing more frequent exploration. The majority of the available bandwidth is then used by the agents with higher priority, leading to more of their messages being delivered and a higher global utility. However once the available bandwidth drops to 42% of the required bandwidth, the routing algorithm is unable to find better paths for higher priority messages and the benifit of priority is lost.

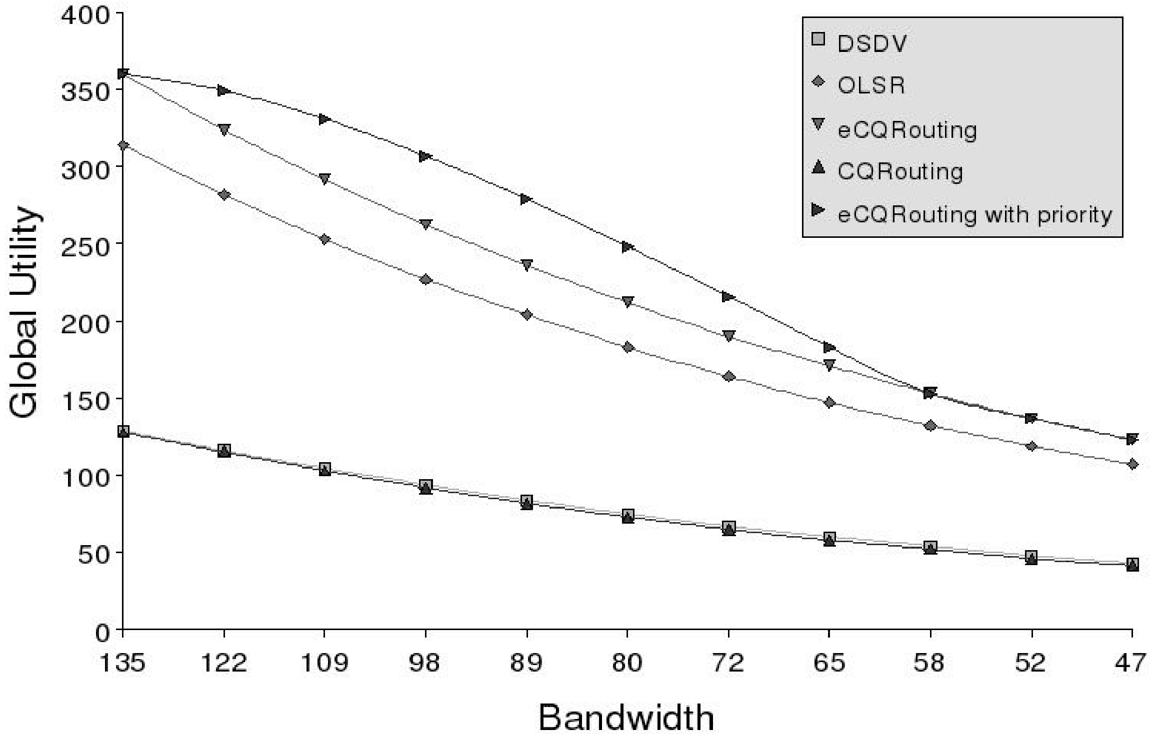


Figure 22: Change in global utility as the available bandwidth drops

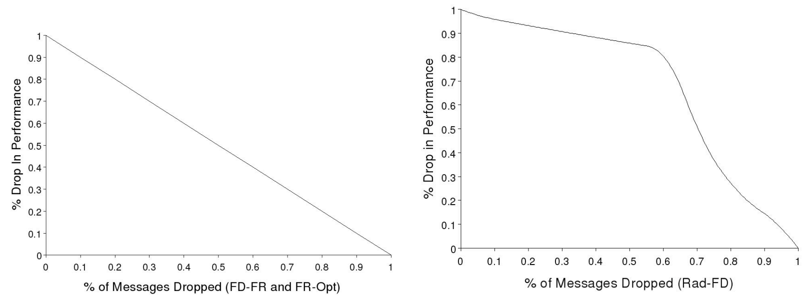
6.4.2 Effect where performance is related to message loss

Here we look at networks where the MAS can handle the loss of a certain percentage of messages without significant loss in performance. We use the CASA organization structure from previous experiments, while changing the global utility computation. In this experiment, the performance degradation with message lost from the Feature Detector (FD) node to the Feature Repository (FR) and messages lost from FR nodes to Optimizer (Opt) node is linear as shown in Figure 23a

Messages from Radars (Rad) to FR nodes have some slack, which allows for a greater percentage of messages to be dropped before there is a significant effect on performance. This is shown in Figure 23b. Messages from Optimizers to Radars have almost no slack, and there is a significant drop in performance with drop in messages. This is shown in Figure 24a

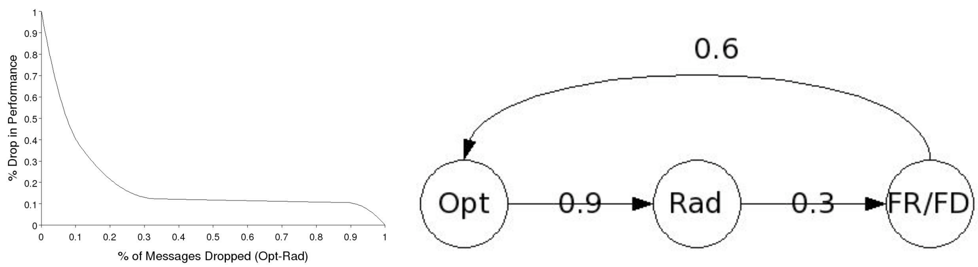
For this experiment, we modify the priority of the messages to reflect the importance of messages as per the above structure. The priority of most of the messages remain the same as shown in Figure 21. Since messages from FD to FR and Rad to FD have the same performance degradation, we gain no additional benefit by analyzing the two separately, and can perform better analysis by merging the role of FR and FD into a single role (calling it the FR/FD role). The priority is averaged and the modification is shown in Figure 24b

We use the sparse 4x4 network. The network is divided into 4 clusters. Next we increase the density of the network by keeping the area the nodes are spread over, but moving from 16 nodes to 160 nodes. These nodes are added randomly across the network. For each cluster, we randomly assign the role of Opt to 1 node. The role of FR/FD to another node. The rest of the nodes are all Radar nodes. An example 25 node network looks as shown in Figure 25. The figure shows a random placement of 25 nodes in the network, with random assignment of FR/FD role, and the Opt role. Nodes on the border between two clusters interact



(a) (b)

Figure 23: (a) Performance degradation as messages get dropped for messages from FD nodes to FR nodes and FR nodes to Opt nodes (b) Performance degradation as messages get dropped for messages from Rad nodes to FD nodes



(a) (b)

Figure 24: (a) Performance degradation as messages get dropped for messages from Optimizers to Radars. (b) Corresponding priority of messages

with clusters where their center lies. All un-marked nodes are Radar nodes. The Rad node sends 100 1Kbit messages to the FD/FR node every second. FR/FD nodes send 25 1Kbit messages to the Opt node and finally Opt nodes send 5 1Kbit message per second to the Rad nodes.

In this experiment, we start with enough bandwidth to make sure all messages get through. Next, the bandwidth is reduced till a certain amount of messages are dropped for each routing algorithm. The threshold before 10% of messages are dropped for example is different for different routing algorithms. We calculate the effect of dropping messages, and the values are calculated at different bandwidths. Figure 26 shows the performance degradation with messages loss. Note, the global performance across the three message types is calculated as the product of individual performances.

As we can see from Figure 26, the global performance is much higher for the eCQRouting algorithm because the low priority messages are dropped before high priority messages. OLSR, DSDV and CQRouting all performed the same, when forced to drop a certain percent of messages, since the message loss is equivalent across all messages types for each of the three algorithms. Interestingly, the experiment shows no improvement in using either one of the two priority techniques described in Section 5.5 (represented as eCQRouting with priority for the simple priority based measure and eCQRouting with performance measure for the message-loss based approach). This is primarily because both algorithms attempt to do the same optimization on the network. The differences arise in when the evaluation occurs and the degree of evaluation. With the priority based representation, the status of network and the goodness of the path are evaluated at the time of exploration. Exploration happens more frequently for source-destination pairs where the priority of messages is much higher, hence allowing them to find sub-optimal paths that provide better probabilities of message delivery. However, if the state of the network changes between exploration messages, the technique

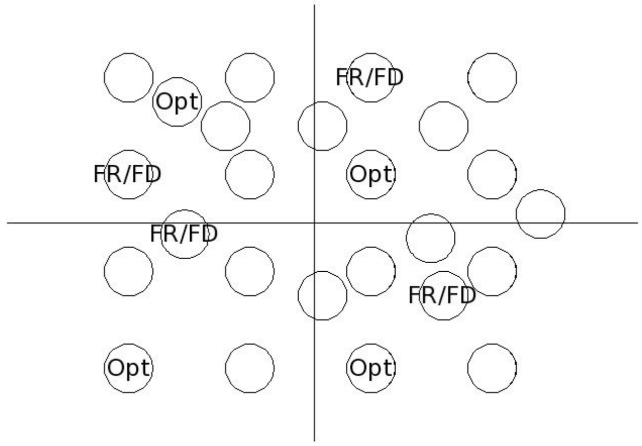


Figure 25: Example 25 node network

will have to suffer a delayed reaction to it. On the other hand, the per-message decision is based on local knowledge of the next hop, and an expectation of the rest of the network. However the benefit of having global information is lost. In this experiment, it seems the global and local benefit balance each other, resulting in similar results when used individually and about 2% improvement when combined.

7 Conclusion and Future Work

In this paper, we showed that using application-level knowledge of the agent organization in network-level routing significantly improves application-level bandwidth and response time relative to standard network-level routing. In a 10x10 grid simulation of the CNAS sensor network, we obtain an increase in available application bandwidth of 35% as compared to OLSR, one of the more widely used routing algorithm for wireless ad hoc networks. Moreover, we show a 37% imporvement in response time over OLSR in the same setting. Additionally we show an increase in global utility by focusing path exploration and, in turn, generating better paths for those source-destination pairs where communication has higher priority.

In this work, information flowed from the application-level organization to the lower-level routing algorithms. We assumed that the organizational structure was appropriate given the network structure. If we relax this assumption, an obvious extension of this work would be to allow the organizational structure to be modified based on the changing characteristics of the network. The network-level routing algorithm produce network capability and status information that can be used as input for possible adaptive or redesign of the organizational structure. Furthermore, as the organization structure changes, the updated information is passed to the routing algorithm, increasing application bandwidth. For example, cluster heads are picked dynamically in CNAS. However, we made the assumption that the cluster head was already chosen in the organizational knowledge provided to the routing algorithm. In practice, the routing layer could provide the sensor agent with a list of all other agents that it can currently communicate with that are within its cluster area. A sensor agent can then decide which of these nodes is the cluster head. This selection is then made available to the routing algorithm. Moreover, with the increasing research in emergent organizational design, a MAS application can start with no organizational structure, and use the routing information to develop an initial organization design, which provides exploration direction to the routing algorithm for better paths and bandwidth.

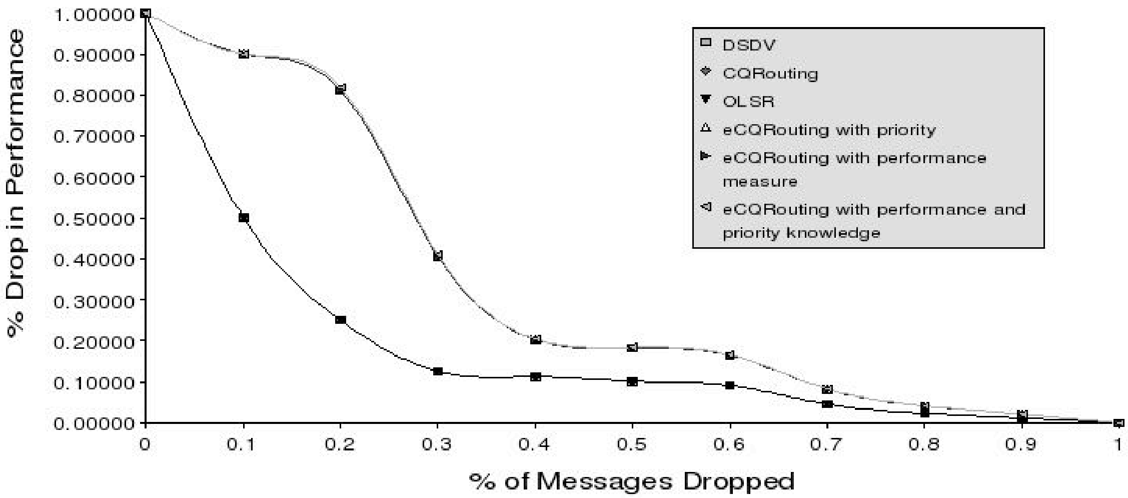


Figure 26: Performance degradation with loss of messages

References

[1] J. Boyan and M. Littman. Packet Routing in dynamically changing networks: A reinforcement learning approach. *Cowan, J.D.;Tesauro, G.;and Alspector, J., eds., Advances in Neural Information Processing Systems*, 1994.

[2] J. Chang and L. Tassiulas. Energy conservation routing in wireless ad hoc networks. *IEEE Infocom*, pages 22–31, 2000.

[3] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol. *RFC 3626, Internet Engineering Task Force (IETF)*, October 2003.

[4] D. Corkill, D. Holzhauer, and W. Koziarz. Turn Off Your Radios! Environmental Monitoring Using Power-Constrained Sensor Agents. *First International Workshop on Agent Technology for Sensor Networks (ATSN-07)*, 2007.

[5] D. Corkill and V. Lesser. The use of meta-level control for coordination in a distributed problem solving network. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, August 1983.

[6] D. D. Couto, D. Aguayo, J. Bicket, and R. Morris. High-throughput path metric for multi-hop wireless routing. *MOBICOM*, 2003.

[7] C. Hedrick. Routing Information Protocol. *RFC 1058, Internet Engineering Task Force (IETF)*, 1988.

[8] B. Horling and V. Lesser. A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2005.

[9] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transaction of the ASME Journal of Basic Engineering*, pages 35–45, March 1960.

[10] Kalman Filter Implementation. http://www.orocos.org/bfl.

[11] S. Keshav. REAL: A Network Simulator. *tech. report 88/472, University of California, Berkeley*, 1998.

[12] M. Krainin, B. An, and V. Lesser. An Application of Automated Negotiation to Distributed Task Allocation. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007)*, Fremont, California, November 2007.

[13] S. Kumar. Confidence based Dual Reinforcement Q-Routing: An On-line Adaptive Network Routing Algorithm. *Master's thesis, Department of Computer Sciences, The University of Texas at Austin, TX-78712, USA Tech. Report*, A:198–267, 1998.

[14] J. McQuillan and D. Walden. The ARPANET Design Decisions. *Computer Networks*, 1, August 1992.

[15] C. Perkins and P. Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. *ACM SIGCOMM Computer Communication Review*, 24(4):234–244, October 1994.

[16] UM-OLSR Implementation for NS2. http://masimum.dif.um.es/?software:um-olsr.

[17] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1989.

[18] Y. Xu, J. S. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Mobile Computing and Networking*, pages 70–84, 2001.