

Self-Adaptive Dissemination of Data in Dynamic Sensor Networks

David Dorsey
Bjorn Jay Carandang
Moshe Kam
Drexel University
Data Fusion Laboratory
3141 Chestnut Street, Philadelphia, PA, U.S.A
{dave, jay, kam}@minerva.ece.drexel.edu

Chris Gaughan
US Army RDECOM
Edgewood Chemical Biological Center
APG-Edgewood Area, Maryland 21010
chris.gaughan@us.army.mil

Abstract

The distribution of data in large dynamic wireless sensor networks presents a difficult problem due to node mobility, link failures, and traffic congestion. In this paper, we propose a framework for adaptive flooding protocols suitable for disseminating data in large-scale dynamic networks without a central controlling entity. The framework consists of cooperating mobile agents and a reinforcement learning component with function approximation and state generalization. A component for agent coordination is provided, as well as rules for agent replication, mutation, and annihilation. We examine the adaptability of this framework to a data dissemination problem in a simulation experiment.

1

1. Introduction

The large-scale, dynamic, and time-varying nature of operational environments for ad hoc wireless mobile and sensor networks present formidable challenges to network designers. Approaches based on centralized control over the network are often infeasible because they do not scale well and assume, unrealistically, a static structure in the form of routes [17] or routing tree structures [23]. Decentralized approaches that do not assume any structure in the network [22] often rely upon a gradient that emerges as data flow toward the sink node. The problem with these structure-less approaches is that they assume the gradient is relatively static; all broadcasts originate from a single sink node and all sensor data flow to this same node

¹This project was supported in part by a US Army Competitive ILIR (In House Laboratory Independent Research) Project approved by the Assistant Secretary of the Army for Acquisition, Logistics, and Technology (ASA(ALT)).

(“convergecasting”[14]). However, if the sensor network is deployed for the purpose of communicating with mobile or static nodes working *within* the area of deployment, these traditional protocols are no longer suitable because any node may be the source of a broadcast to all other nodes.

Consider a sensor network that is deployed for use by first-responders battling a forest fire or in a military engagement where there is a threat of chemical or biological agents. In these scenarios, mobile networks are likely to be deployed with little or no existing infrastructure; the individual sensor nodes communicate through their wireless interface with other nearby nodes to form links. The effect of these local interactions among nodes is the emergence of a connected, ad-hoc wireless sensor network over the area of deployment. The resulting network does not possess any global perspective; nodes are only aware of neighbors in communication range. The mobile nodes accompany the first-responders, warfighters, or unmanned ground vehicles (UGV) that move through the area of deployment. Mobile nodes may act as both sources and sinks of information. The goal is for the sensor network to provide the mobile nodes with updated information about the entire area of deployment while also acting as a medium for sharing information between mobile nodes. When a sensor is triggered or a mobile node sends a packet to a sensor node, the event is recorded and then broadcast to neighboring nodes. Ideally, the event is broadcast to the entire network so that when a mobile node comes in communication range of a sensor node, it may obtain updated event information from across the network.

An obvious challenge that arises is the problem of broadcasting messages in these heterogeneous networks. Since nodes do not need to communicate with a specific destination, traditional routing protocols are inappropriate. Traditional flooding protocols are designed specifically for network initialization and route discovery within traditional routing algorithms. These protocols assume that there is a

single broadcaster (a sink node or gateway) delivering messages to many or all other nodes. All responses are returned back to the same sink. Since all messages from the network are assumed to converge onto the static sink node, flooding protocols assume a structure on the network in the form of a tree or a gradient.

In order to deal with scenarios as those described above, where any node may be a potential broadcaster to all other nodes, a broadcast protocol should be adaptive. It should not rely on established routes that would require maintenance and should be able to modify its operational mode when the network environment changes.

We propose an adaptive framework for disseminating information in dynamic, large-scale networks including both mobile nodes and stationary sensors. Our technique involves the use of mobile software agents that pick up and/or drop off data to nodes while recording information about the environment onto the nodes at each migration. Agents make decisions about which data to pick up and which link to select for migration according to a state-action value function, which is formed by extracting and generalizing features from the local information recorded onto the nodes by other agents. The payoff to the agent is related to the number of timely deliveries of event data and the energy expended (agents consume energy at each node for computation, transmission, and reception). The global objective is to minimize the average delay between the time a message is sent and when it is received by all other nodes while minimizing the average energy expended across all nodes in response to the dissemination task. The framework provides rules for agent replication, mutation, and annihilation based on local rewards. We also incorporate reinforcement learning with function approximation to provide agents with the ability to make generalizations and learn about their environment through a value function and one-step feedback.

2 Dissemination in Ad-Hoc Networks

Most of the current methods for disseminating data on wireless *ad hoc* sensor networks require a structure to be enforced on the network. By structure we mean a set of pre-established reliable links or routes; this includes hierarchical structures such as clusters, or flat structures such as trees. Gradient-based algorithms such as Gradient Broadcast (GRAB)[22] work well in situations where there is a static sink node to which all data are delivered. However, they are not designed to deal with situations where there is no pre-defined sink node or where the network topology is highly dynamic because they still require maintenance of the gradient. For these situations, an adaptive framework is necessary.

Research in Swarm Intelligence[3] led to many algorithms for adaptive routing in these dynamic networks using

the approach from AntNet[6]. AntNet is an adaptive, distributed, protocol using mobile software agents to search for an optimal path between a source and a sink. The method is based on the ant colony metaphor, where "forward ants" are sent from the source node and, once they have located the sink node, they trigger the release of "reverse ants" that update the routing tables on the hosts between the source and the sink.

Other methods employ a randomized approach where a node, upon receiving a message, will rebroadcast to a random subset of its neighbors: these include work in gossip protocols[12], rumor routing[4], and epidemic routing[10].

3 Multiagent Reinforcement Learning

An agent is defined as anything that can perceive its environment through sensors and act upon the environment through actuators [19]. In reinforcement learning, an agent must learn to act through direct interaction with an environment. A reinforcement signal provides feedback to the agent so that it may evaluate its past actions and improve future actions accordingly. This model is attractive for dynamic, decentralized systems because it does not rely on prior information and pre-planned strategies. Instead, agents are left to decide how to act during system operation through reward and punishment signals. At each time step, an agent observes its environment and selects the next action based on its own evaluation of the observation. In the following time step, the agent observes the effect of its previous action and receives a payoff indicating the quality of that action. This cycle is repeated while the agent attempts to learn a policy π which maps features (or states) associated with its observations to actions in order to maximize its expected discounted reward.

This interaction between the agent and the environment is usually formalized as a discrete time, finite state Markov decision process (MDP). An MDP is a tuple (S, A, P, R) , where S is the state space, A is the action space, $P_a(s, s')$ is the probability that action a in state s will lead to state s' in the next time step, and $R(s)$ is the (immediate) reward received after transition from s to s' . The total reward the agent receives from time t onward is $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$, where $\gamma \in [0, 1]$ is the discount rate. In cases where a model of the system dynamics is not available (as is usually the case), the agent will interact with the environment in order to iteratively produce an estimate of $P_a(s, s')$.

In Q-Learning [21], the action-value function of a given policy π associates all state-action pairs (s, a) with an expected reward for performing action a in state s and following π thereafter. This function (the Q-function) is the expected value of R_t given (s, a) :

$$Q(s, a) = \max_{\pi} E[R_t | s_0 = s, a_0 = a, a_{t>0} = \pi(s_t)] \quad (1)$$

The update rule for Q is

$$Q_{t+1}(s, a) = (1 - \alpha)Q_t(s, a) + \alpha[r + \gamma \max_{a'} Q_t(s', a')] \quad (2)$$

where γ is the discount, r is the immediate reward and $\alpha \in [0, 1]$ is the learning rate. These Q-values are used by the agent to decide which action to take in the next time step. To allow agents to explore unvisited states, these algorithms include an exploration policy such as the ϵ -greedy policy, where agents select a random action with probability $\epsilon < 1$, and action $a = \arg \max_{a'} Q(s, a')$ with probability $1 - \epsilon$.

Although well-understood algorithms with good convergence properties are known for solving single-agent tasks, particularly in static (stationary) environments, the use of multiple agents in the same learning environment provides new challenges. In a multiagent system, the presence of other agents makes the environment nonstationary from the point of view of each agent, so it is unlikely that an agent will be able to learn the state transition probabilities from the MDP model. This nonstationarity also invalidates any of the convergence properties of single agent reinforcement learning algorithms. However, in the context of dissemination of data with multiple agents in a dynamic network, the emphasis is not on convergence, but on adaptability to other agents and the changing environment. The focus of a multiagent reinforcement learning algorithm in this class of problems is to allow an agent to coordinate learning by incorporating the rewards and value functions of other agents. For a comprehensive survey of multiagent reinforcement learning techniques, we refer to [5].

3.1 Function Approximation

Most reinforcement learning algorithms assume the use of tables to store the action-values or state-values. In the case of agents traversing a wireless network, however, the energy and latency costs to carry such a table from host to host make this impractical. Since each agent must carry its own action and state values, we require a compressed representation of the value function. Recent literature in multiagent reinforcement learning has considered the use of a function approximation to estimate the value function [1, 8]. One way to represent the value function is as a weighted linear function of a set of features (basis functions):

$$\begin{aligned} \tilde{Q}(s, a) &= w_0 + w_1\theta_1 + w_2\theta_2 + \dots + w_m\theta_m \\ &= w_0 + \mathbf{w}^T \boldsymbol{\theta}. \end{aligned} \quad (3)$$

These basis functions are formed by extracting relevant features from the environment. As an example from [19], the value of a particular state of a chess game could be computed using θ_i to represent the number of each kind of piece on the board, and the w_i could be the value of the pieces (1 for a pawn, 3 for a bishop, etc.). This method provides an enormous compression of the number of stored values an agent needs to keep while allowing the agent to make generalizations from states it has visited about states it has not visited.

Each time the agent receives a reward, the weights \mathbf{w} are updated using a version of the delta rule for online least-squares:

$$w_i \leftarrow w_i + \alpha \left[r + \gamma \hat{Q}(s', a') - Q(s, a) \right] \frac{\partial Q(s, a)}{\partial w_i}, \quad (4)$$

where s' and a' are the next states and next actions. The modification of the feature weights is proportional to the difference between the received reward and the estimated value of the state-action pair. Since (3) is defined to be linear in $\boldsymbol{\theta}$, (4) reduces to

$$w_i \leftarrow w_i + \alpha \left[r + \gamma \hat{Q}(s', a') - Q(s, a) \right] \theta_i(s, a), \quad (5)$$

which is easier to compute. These weights represent what the agent has learnt through interaction with the environment and serves as the agent's "memory", since any changes in \mathbf{w} will affect the evaluation of all other states. If an agent receives a large reward for taking an action a and moving to a state s , then the value of w_i will be modified in proportion to the extent that the value θ_i participated in this decision.

4 Proposed Method

4.1 Components

Before presenting the outline of the algorithm, we first introduce the components of the framework.

4.1.1 Events

We refer to any data that are produced by sensors in the network as a result of a detection as an event. However, it is not necessary that events originate exclusively from sensor detections; an event may also be an annotation on a collaborative whiteboard application used by first responders (for an example of such a collaborative application for first responders, see [2, 15]). Each event has a time-dependent numerical value determined by its age (how much time has passed since the event was detected), its priority class (higher priority events have greater initial values), and its expiration date

(the amount of time for which an event is relevant). We denote the value of event i created at time t_0 with priority p and expiration x as $v_i(t - t_0, p, x)$ or simply $v_i(t)$. We denote the set of events on a host as E_h and the events that an agent is carrying as E_a .

4.1.2 Swarm Agents

There have been several investigations into the use of mobile software agents for exploring and providing decentralized services in wireless *ad hoc* networks [16, 7, 18]. A mobile agent is a composition of software and data which is able to autonomously move from one host to another while continuing its execution at each destination. Mobile agents are well suited to adaptive communication protocols because they are goal-oriented and can continue to operate even after the host from which they originate is removed from the network. The mobile agents used in the dissemination framework are composed of a link-action value estimator, $\hat{Q}(\ell, a)$, a decision policy, π , a parameter set w , an event payload, and a fixed memory size for storing important statistics. We refer to these as *swarm agents*. The link-action estimator computes the value of the agent choosing a link, ℓ , and an action, a . The choice of ℓ is selected from the set of neighbors of the current host as well as the current host (agents may elect not to migrate). The action is selected from the action space, \mathcal{A} , which is comprised of the events the agent will carry to the next host, the number of clones to create, the decision to live or die, and the decision to mutate the agent's parameters, w , with another agent's parameters. Swarm agents collaborate through messages left on the hosts to evaluate links and actions.

4.1.3 Swarm Agent Collaboration

The messages left by swarm agents for collaboration are contained in a *visit entry* at each host they visit. These entries contain information about the agent's payload as well as information about the local environment as perceived by the agent. For example, the visit entries might contain:

- The quality of the link (LQ) upon which the agent migrated; this is a function of the received signal strength at this host and the number of migration failures the agent experienced on this link.
- The amount of energy the last host has consumed (*energy*).
- The amount of time the agent spent at this host and the last host it visited (*delay*)
- The events that the agent carried in its payload to this host and from this host (*events*)

- Event meta-data (*MD*) used to represent any data elements that the agent encountered at the last host but did not pick up
- The value of the events at the last host (*value*)
- The agent's parameter set, w
- The agent's average reward \hat{R}
- The number of times the agent has replicated (*clones*)
- The amount of time the agent has been alive (*age*)

The data in these visit entries are sorted into link advertisements and agent advertisements on the hosts, as shown in Tables 1 and 2. When an agent arrives at a new host after a migration, it will use the information in the link advertisements to determine the value of migrating across a link. If a link does not have an advertisement, then the agent will categorize the link as *unexplored*. Because agents always take the set of events that will maximize their expected reward, agents may elect to leave some events on hosts it has visited and take only the meta-data describing the event to the next host. The use of meta-data for exchanging information between nodes in a broadcast protocol was first introduced in [13].

The agent will also evaluate its own performance by viewing agent advertisements and comparing its own average reward with that of other agents. In [11], the authors use a genetic crossover method between swarms of randomly wandering agents on a network. The objective was to continuously mutate the agent populations in order to find an optimal memory size for agents that would result in the reduced latency of event delivery. In swarm dissemination, the agents mutate their parameter set w ; when an agent that has recently visited is performing significantly better (according to the advertisement), the agent may perform a crossover mutation between its own parameter set and that of the successful agent (Section 4.1.7). Visit entries and advertisements are deleted from the host after a set period of time to avoid crossovers and decisions based on stagnant information.

4.1.4 Reward

At each time step, agents receive a reward R for delivering an event to a host that has not already received the event. The reward is proportional to the number of events that are delivered at each migration and the value of the events (which decreases with time). The agent is also penalized for consuming energy at each migration; the amount of energy that is consumed when an agent migrates is related to the number of events the agent is carrying. If an agent migrates with no events, it is still penalized a constant amount c to

Table 1. Table of state values for each link to a neighbor of the current host. The values are updated periodically between hosts or through agent visit entries.

Link	LQ	$events/MD$	$value$	$energy$	$delay$
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
...	-	-	-	-	-

Table 2. Table of state values for each agent to recently visit the current host. The values are updated through agent visit entries.

Agent	\hat{R}	w	$clones$	age
1	-	-	-	-
2	-	-	-	-
3	-	-	-	-
...	-	-	-	-

carry its executable code and its memory. If the agent is carrying N events, the reward is computed as

$$R = a_1 \sum_{i=1}^N I(v_i) - a_2 N + c \quad (6)$$

where a_1 , a_2 , and c are constants, and $I(\cdot)$ is the indicator function

$$I(v_i) = \begin{cases} v_i & \text{if host did not have the event} \\ 0 & \text{otherwise} \end{cases}.$$

The agent keeps a running weighted average of these rewards \hat{R} which is updated at each time step according to

$$\hat{R}(t) \leftarrow R(t) + (1 - \alpha)\hat{R}(t - 1) \quad (7)$$

4.1.5 Link-Action Value Approximation

As described in Section 3.1, function approximation can significantly reduce the amount of information that swarm agents carry while also allowing agents to generalize unvisited states from ones that they have visited. Swarm agents form the function approximation using the data contained in the link advertisements from Table 1. We refer to the data in the table as the *features*; each row contains features for a single link from the current host.

The first feature is the link quality, LQ , which represents the probability that an agent will be able to successfully migrate across this link. If an agent attempts to migrate, but

the packet is dropped due to interference, then the agent will have to retransmit (and consume more energy at this host). The value of LQ is a function of the received signal strength, which indicates the strength of the signal received from the neighbor host, as well as the number of failed migration attempts reported by agents on this link. This function is computed so that more recent reports are given greater weight (this is true for all values in the advertisement table). The basis function θ_1 that computes this feature is a simple sigmoid function mapping the LQ value on the interval $[-1, 1]$.

The second feature is the $events/MD$. This feature is the set of events that the neighbor on this link is known to possess as of the most recent visit entries; if the event is represented only by meta-data, then the event is possessed by the neighbor but not this host. With this information, two additional basis functions, θ_2 and θ_3 are formed. The first computes the maximum value of the events that the agent expects to "pick up" at this neighbor; the second is the maximum value of the events the agent expects to "drop off" upon migration. If hosts periodically update each other with the value of the events they currently possess, then the third feature, $value$, can be incorporated into θ_2 and θ_3 to verify the validity of the visit entry information. In the case where there is no information about this link from visit entries, an additional basis with a boolean value is used to indicate that this link is unexplored.

The $energy$ feature represents the amount of energy known to have been consumed by the neighbor on a link. We used a sigmoid function to represent the value of this feature. The $delay$ feature represents the amount of time agents spend at this neighbor. We assume in our experiments that if multiple agents reside on the same host that they are placed into a queue to wait until the CPU is available. The value also represents the congestion on this link. The basis function for this feature is the same as the one used for the $energy$ feature.

The value of a link is computed with these basis functions using (3), replacing s with ℓ . For each link, the action (set of events to carry to the neighbor on a link) that maximizes the value of $\hat{Q}(\ell, a)$ is denoted \hat{Q}_ℓ^* .

After an agent has reached the next host, having selected action a and link ℓ and received reward R , the weights for the linear function are updated using:

$$w_i \leftarrow w_i + \alpha \left[r + \gamma \hat{Q}(\ell', a') - Q(\ell, a) \right] \theta_i(\ell, a), \quad (8)$$

Note that agents are also evaluating their link selection ℓ' and next action, a' with respect to the current state and the last action taken when updating w . That is, agents update w after they have selected what link and action to take next and incorporate the estimated value of this decision into the learning update, as shown in Figure 1.

4.1.6 Migration Policy

The values for $\hat{Q}_{\ell_i}^*$ for each link ℓ_i are used to form a random distribution for the agent migration decision at each node. Two probabilities are calculated: the first is the probability of migrating this time step and the second is, given that the agent has chosen to migrate, the probability of selecting a particular link. In [9], the authors provide a model for examined patterns in army ant raids, where the ants leave the nest to seek out food sources. At each step, each ant decides whether to advance or stay at its current site. According to the model, if ρ_l and ρ_r are the amount of pheromone to the left and to the right of the ant in the forward direction, then the probability that the ant will advance is given by:

$$p_m = \frac{1}{2} \left[1 + \tanh \left(\frac{\rho_l + \rho_r}{100} - 1 \right) \right] \quad (9)$$

We adopt this model to determine the probability that an agent will advance in a given time step, where here the pheromone amounts are replaced with values for $\hat{Q}_{\ell_i}^*$. Let N be the number of neighbors at node i . Then,:

$$p_m = \frac{1}{2} \left[1 + \tanh \left(\sum_{k=0}^N \hat{Q}_{\ell_k}^* - 1 \right) \right] \quad (10)$$

A model from the army ant raids [3] is also used to determine the probability of visiting a neighbor node. The probability of choosing link i is:

$$p_i = \frac{(\mu + \hat{Q}_{\ell_i}^*)^\beta}{\sum_{k=0}^N (\mu + \hat{Q}_{\ell_k}^*)^\beta} \quad (11)$$

where μ quantifies the degree of randomness involved in the decision and β determines the nonlinearity of the decision function. The values of p_i form a distribution \mathbf{P} over which the next link is selected. A higher value of β implies that small differences in \hat{Q}^* will result in large probability variations, so agents will be discouraged from choosing a link with even a slightly lower \hat{Q}^* . These parameters are used to adjust the agents' propensity for exploring new states or exploiting known information to maximize reward.

4.1.7 Evolution

Swarm agents are given a provision for reproduction, annihilation, and mutation. These operations are performed based on the value of the agents average reward \hat{R} and the value of \hat{R} reported by other agents in the agent advertisements (see Table 2). If an agent's reward is above a threshold r_{max} , then the agent will create a clone of itself (with identical parameters and events) with probability

p . The number of possible clones increases linearly as \hat{R} increases above r_{max} until reaching the number of links from this host. If the agent's reward is below r_{min} , then the agent will die with probability p . For reward values anywhere between r_{min} and r_{max} , the agent will compare \hat{R} to the maximum average reward posted by an agent on the advertisement. If an agent advertised a reward that is higher than \hat{R} at time t_a , then a mutation will occur with this agent with probability $p = e^{\alpha(t-t_a)}$. The mutation operation involves replacing random w_i from the agent's parameter set with the corresponding w_j from the advertised agent's parameter set. Specifically, if the successful advertised agent's reward is \hat{R}_m and its parameter set is \mathbf{v} , each element $w_i \in \mathbf{w}$ is replaced with $v_i \in \mathbf{v}$ with probability $\frac{\hat{R}_m - \hat{R}}{\hat{R}_m}$.

The purpose of crossover is to allow agents who have already learnt the environment to share this information with other agents. When agents are receiving increasing rewards for dropping valuable data to hosts and minimizing the number of unnecessary events in their payload, the agents will reproduce and share learnt behavior with other agents. This results in an increase in the agent population. When there are few opportunities for reward (either there are no data to be disseminated or most nodes already have the events), then agents will receive negative rewards and the agent population will diminish. The result is an agent population that grows and diminishes in proportion to the requirements of the network.

4.2 Algorithm

Figure 1 shows an overview of the swarm agent behavior cycle; a more formal listing is shown in Algorithm 1. Each time an agent migrates to a new host, it creates a visit entry and records the data described in Section 4.1.3. These data are added to the link and agent advertisement tables on the host. Then the agent merges its events with the events of the host and computes the received reward \mathbf{R} and the average reward \hat{R} . Next, the agent checks if \hat{R} is within $[r_{min}, r_{max}]$; if so, then it will look into the agent advertisement table and find \hat{R}_{max} to evaluate whether or not to perform a crossover operation. If the reward is above the interval, the agent will perform a replicate operation. It will enter the die routine if the average reward is below the interval. Next, the agent will evaluate possible next links and actions using the link advertisement table and the value function $\hat{Q}(\ell, a)$ described in Section 4.1.5. This loop will return a value of $\hat{Q}_{\ell_i}^*$ for each link, which are used in the event/link decision, which is determined using the migration policy described in Section 4.1.6. Before migrating to the next host, the agent updates its parameters using (8).

Algorithm 1 Swarm Dissemination Agent

inputs: Host, E_h , link and agent advertisements

static: π , a link decision policy

θ , the basis function vector

w , the $m \times 1$ parameter vector

$R, \hat{R}, \hat{Q}, clones, age$, initially zero

E_a , the events the agent is given at creation

$delay, clones, age$, initially zero; $value, energy$

while Agent is alive **do**

 Create visit entry

log: $E_a, delay, value, MD, energy, \hat{R}, w, clones, age$

$E_a \leftarrow E_h \leftarrow (E_a \cup E_h)$

$R \leftarrow a_1 \sum_{i=1}^N I(v_i) - a_2 N + c$

$\hat{R}(t) \leftarrow R(t) + (1 - \alpha)\hat{R}(t - 1)$

if $\hat{R} < r_{min}$ **then**

 die()

end if

from agent advertisements: get entries

if $\hat{R} < \hat{R}_{max}$ **then**

 crossover()

else

if $\hat{R} > r_{max}$ **then**

 replicate()

end if

end if

from link advertisements: get entries

for all $i \in L', j \in A'$ **do**

$\hat{Q}_{\ell_i}^* \leftarrow \max_{a_j} \{w_0 + w^T \theta(\ell'_i, a'_j)\}$

end for

if $p_m > \epsilon$ **then**

 select ℓ' from P

else

$\ell' \leftarrow null, a' \leftarrow null$

end if

for $i = 0..m$ **do**

$w_i \leftarrow w_i + \alpha [R + \gamma \hat{Q}(\ell', a') - Q(\ell, a)] \theta_i$

end for

 migrate on link ℓ' and perform action a'

end while

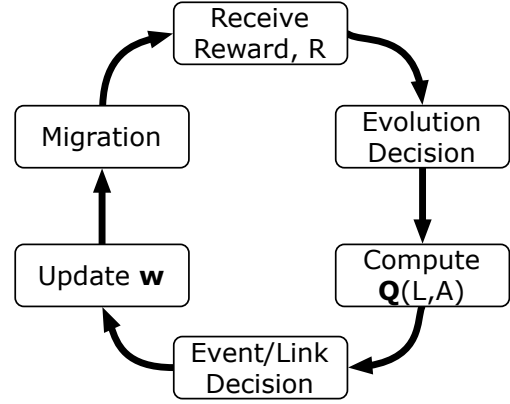


Figure 1. At each migration, an agent selects an action based on the current perceived state, receives a reward, and modifies the parameter set to minimize the difference between the perceived value of a state-action pair and the actual reward obtained for that action.

5 Experiment and Results

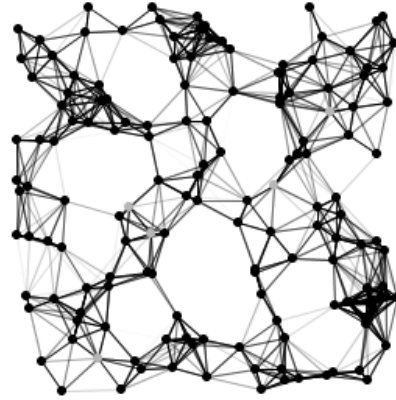


Figure 2. One sample simulation topology

We simulated the swarm agent dissemination algorithm using the Macro Agent Transport Event-based Simulator (MATES) [20] on a 150 node network with a randomly generated topology for each experiment and 10 mobile nodes in each network. Figure 2 shows an example topology in the simulator. The Euclidean distance between two hosts was used to determine the link quality; the lines connecting nodes in the simulator denote neighbors (thinner lines represent lower link quality). Initially, 5 agents were randomly placed in the network and given a single event to be disseminated to all nodes. After running the simulator for 15 time

steps, we added 5 more events onto the hosts. During each time step, we tracked the number of agents that were alive, the number of agents that were migrating, and the event coverage. Denoting the number of events on all hosts as S , the number of hosts as H , and the number of event types as N , the event coverage was calculated as $\frac{S}{HN}$. This experiment was run 10 times and the average was taken over the runs. Figure 3 shows the average event coverage over time.

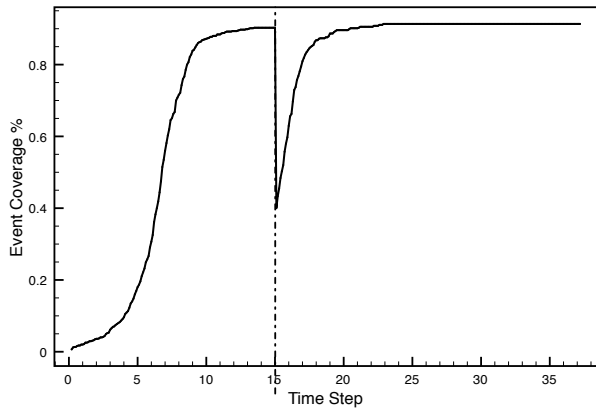


Figure 3. Event coverage over time. At 15 time steps, 5 additional events were added to the network, causing the coverage percentage to drop below 50%.

for the 10 runs. Notice that at 15 time steps, the coverage drops below 50%, since the number of event types doubled. The coverage then rapidly increases back to near 92% as new agents are produced to handle the dissemination requirements. Figure 5 shows how the number of agents in the network increases rapidly in response to low event coverage. Once the coverage percentage approaches the maximum, the number of agents quickly diminishes, since agents are unable to find opportunities for reward. At time step 15, when 5 additional events are introduced into the network, the agent population increases again until the event coverage reaches about 90%. Also note in this figure that at any given time step, many agents are not active. For example, near time step 7, the total agent population reaches 130, while only 85 are active. This is due to the fact that when the agent population grows quickly, some agents are either queued at a node waiting to gain access to the CPU or they are deciding not to migrate because there is little opportunity to gain reward in their location.

Figure 4 shows the average energy consumed at each host versus the percentage of coverage over the network. These results were obtained using both static and mobile networks. For the static network, we ran 5 simulations with 5 events to be disseminated. The remaining two

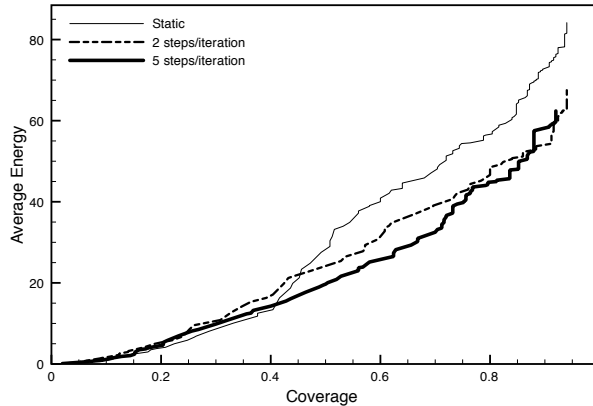


Figure 4. Average energy consumed at each host to achieve percent coverage. The curves show averages over 5 simulations for a static network and networks with mobile nodes making random walks.

curves show energy versus coverage when all hosts are moving each iteration in a random direction. The "velocity" of the hosts was simulated by increasing the distance the host traveled in each iteration. We ran 5 simulations for each of two different velocities, 2 steps/iteration and 5 steps/iteration, where a step is 1/20 of the length of the simulation area. We note that higher mobility seems to reduce the amount of energy required to disseminate data; however, at 5 steps/iteration, the network became disconnected before full coverage could be obtained.

6 Conclusion

We have described a framework for self-adaptive dissemination of data in large dynamic networks using mobile software agents, inter-agent collaboration, and multi-agent reinforcement learning. The learning component of the agent is a compact representation of the values of visited states which is used to provide generalizations about unvisited states. Agents share their experience by crossing their parameters with successful agents and sharing information about the environment in visit entries. We show that the agent population adapts to the needs of the network by reducing the number of agents in response to increasing event coverage and increasing the agent population to meet the dissemination requirements of the network. The agents who learn quickly and increase their reward reproduce and influence other agents, while agents who do not perform well are eliminated in order to reduce their impact on the network. The result is an autonomic dissemination support protocol that can be used to provide a decentralized

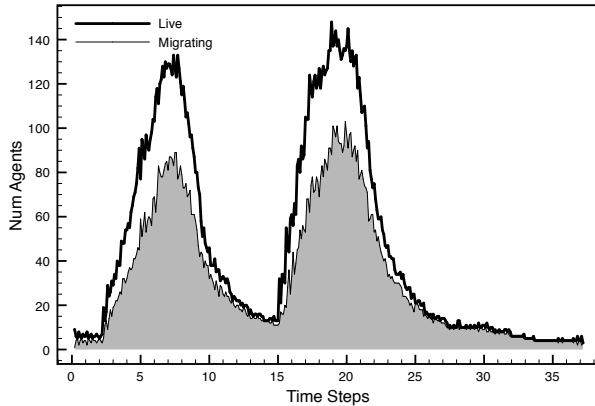


Figure 5. Number of agents on the network over time. The shaded area shows the number of agents that are migrating at any time; the bold line above shows the total number of agents in the network.

dissemination service in the *ad hoc* networks described in Section 1.

The size of the agent population reacts to the number of hosts that still require delivery. We have experimented with some of the parameters in the the swarm algorithm, and we have noticed that the system is sensitive to various parameters. For example, increasing the probability for replication, p , will decrease the average delay experienced at hosts, while also increasing the energy consumption on the hosts. In future work, we will examine the set of tunable parameters in order to study performance trade-offs and patterns of stability.

References

- [1] O. Abul, F. Polat, and R. Alhajj. Multiagent reinforcement learning using function approximation. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 30(4):485–497, Nov 2000.
- [2] G. Anderson, L. Urbano, G. Naik, D. Dorsey, A. Mroczkowski, D. Artz, N. Morizio, A. Burnheimer, K. Malfetone, D. Lapadat, E. Sultanik, S. Garcia, M. Peysakhov, W. Regli, and M. Kam. A secure wireless agent-based testbed. *Information Assurance Workshop, 2004. Proceedings. Second IEEE International*, pages 19–32, 8-9 April 2004.
- [3] E. Bonabeau, M. Dorigo, and G. Theraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, 1999.
- [4] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31, New York, NY, USA, 2002. ACM.
- [5] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, March 2008.
- [6] G. D. Caro and M. Dorigo. Mobile agents for adaptive routing. In *HICSS '98: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 7*, page 74, Washington, DC, USA, 1998. IEEE Computer Society.
- [7] V. Ciciello, A. Mroczkowski, and W. Regli. Designing decentralized software for a wireless network environment: evaluating patterns of mobility for a mobile agent swarm. *Multi-Agent Security and Survivability, 2005 IEEE 2nd Symposium on*, pages 49–57, 30-31 Aug. 2005.
- [8] A. da Motta Salles Barreto and C. Anderson. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, Jan 2008.
- [9] J. Deneubourg, S. Goss, N. Franks, and J. Pasteels. The blind leading the blind: Modeling chemically mediated army ant raid patterns. *Journal of Insect Behavior*, Jan 1989.
- [10] P. Eugster, R. Guerraoui, A. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *Computer*, Jan 2004.
- [11] J. Haas, M. Peysakhov, and S. Mancoridis. Ga-based parameter tuning for multi-agent systems. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1085–1086, New York, NY, USA, 2005. ACM.
- [12] S. Hedetniemi, S. Hedetniemi, and A. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks(New York)*, Jan 1988.
- [13] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185, New York, NY, USA, 1999. ACM.
- [14] Q. Huang and Y. Zhang. Radial coordination for converge-cast in wireless sensor networks. *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 542 – 549, Oct 2004.
- [15] J. Kopena, E. Sultanik, R. Lass, D. Nguyen, C. Dugan, P. Modi, and W. Regli. Distributed coordination of first responders. *Internet Computing, IEEE*, 12(1):45–47, Jan.-Feb. 2008.
- [16] D. Massaguer, C.-L. Fok, N. Venkatasubramanian, G.-C. Roman, and C. Lu. Exploring sensor networks using mobile agents. In *AAMAS '06: Proceedings of the fifth international conference on Autonomous agents and multiagent systems*, pages 323–325, New York, NY, USA, 2006. ACM.
- [17] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90–100, 25-26 Feb 1999.
- [18] M. Peysakhov, D. Artz, E. Sultanik, and W. Regli. Network awareness for mobile agents on ad hoc networks. *Autonomous Agents and Multiagent Systems*, Jan 2004.

- [19] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.
- [20] E. Sultanik, M. Peysakhov, and W. Regli. Agent transport simulation for dynamic peer-to-peer networks. *Multi-Agent-Based Simulation VI*, pages 162–173, 2006.
- [21] C. J. C. H. Watkins and P. Dayan. Technical note: q-learning. *Mach. Learn.*, 8(3-4):279–292, 1992.
- [22] F. Ye, G. Zhong, S. Lu, and L. Zhang. Gradient broadcast: A robust data delivery protocol for large scale sensor networks. *Wireless Networks*, Jan 2005.
- [23] Y. Zhang, M. Fromherz, and L. Kuhn. Smart routing with learning-based qos-aware meta-strategies. In *Quality Of Service In The Emerging Networking Panorama*, Jan 2004.