# Mobilized ad-hoc networks: A reinforcement learning approach

Yu-Han Chang
MIT CSAIL
Cambridge, MA 02139
ychang@csail.mit.edu

Tracey Ho
LIDS, MIT
Cambridge, MA 02139
trace@mit.edu

Leslie Pack Kaelbling
MIT CSAIL
Cambridge, MA 02139
lpk@csail.mit.edu

## Abstract

*With the cost of wireless networking and computational power rapidly dropping, mobile ad-hoc networks will soon become an important part of our society's computing structures. While there is a great deal of research from the networking community regarding the routing of information over such networks, most of these techniques lack automatic adaptivity. The size and complexity of these networks demand that we apply the principles of autonomic computing to this problem. Reinforcement learning methods can be used to control both packet routing decisions and node mobility, dramatically improving the connectivity of the network. We present two applications of reinforcement learning methods to the mobilized ad-hoc networking domain and demonstrate some promising empirical results under a variety of different scenarios in which the mobile nodes in our ad-hoc network are embedded with these adaptive routing policies and learned movement policies.*

## 1. Introduction

Mobile ad-hoc networking is emerging as an important research field with a number of increasingly relevant real-world applications, ranging from sensor networks to peer-to-peer wireless computing. Thus far research has focused on creating specialized algorithms and protocols for specific situations. They lack the adaptivity and automatic configuration that methods based on machine learning could potentially offer. This paper begins to develop ideas at the confluence of AI and networking. We cast mobilized ad-hoc networks as a multi-agent learning domain and discuss some motivations for this study. We apply reinforcement learning techniques to two distinct problems within this domain: packet routing and node movement. Using relatively straightforward adaptations of these methods, we are able to demonstrate good empirical results.

Mobile ad-hoc networks have not traditionally been considered a multi-agent learning domain partly because most research in this area has assumed that we have no control over the node movements, limiting research to the design of routing algorithms. Each node is assumed to be attached to some user or object that is moving with its own purpose, and routing algorithms are thus designed to work well under a variety of different assumptions about node mobility patterns.

However, there are many applications in which we can imagine that some of the nodes would have control over their own movements. For example, mobile robots might be deployed in search-and-rescue or military reconnaissance operations requiring ad-hoc communication. Sensor networks with thousands of nodes, each with some built-in mobility, might be used to track wide-spread phenomena or environmental conditions. In such cases it may be necessary for some nodes to adjust their physical positions in order to maintain network connectivity. In these situations, what we will term *mobilized* ad-hoc networks becomes an extremely relevant multi-agent learning domain. It is interesting both in the variety of learning issues involved and in its practical relevance to real-world systems and technology.

There are several advantages gained by allowing nodes to control their own movement. Stationary or randomly moving nodes may not form an optimally connected network or may not be connected at all. By allowing nodes to control their own movements, we will show that we can achieve better performance for the ad-hoc network. One might view these controllable nodes as "support nodes" whose role is to maintain certain network connectivity properties. As the number of support nodes increases, the network performance also increases. Given better movement and routing algorithms, we can achieve significant additional performance gains.

It is important to note that there are two levels at which learning can be applied: (1) packet routing and (2) node movement. We will discuss these topics in separate sections in this paper. Packet routing concerns the forwarding decisions each node must make when it receives packets destined for some other node. Node movement concerns the actual movement decisions each node can make in order to optimize the connectivity of the ad-hoc network. Even though we will use reinforcement learning techniques to tackle both these problems, they must be approached with different mindsets. For the routing problem, we focus on the issue of online adaptivity. Learning is advantageous because it allows the nodes to quickly react to changes in network configuration and conditions. Adaptive distributed routing algorithms are particularly important in ad-hoc networks, since there is no centrally administered addressing and routing system. Moreover, network configuration and conditions are by definition expected to change frequently.

On the other hand, the node movement problem is best handled off-line. Learning a good movement policy requires a long training phase, which would be undesirable if done on-line. At execution time, we should simply be running our pre-learned optimal policy. Moreover, this movement policy should encode optimal action selections given different observations about the network state; the overall policy does not change due to changing network configuration or conditions and thus does not need to adapt online. We treat the problem as a large partially-observable Markov decision process (POMDP) where the agent nodes only have access to local observations about the network state. This partial observability is inherent to both the routing and movement portions of the ad-hoc networking problem, since there is no central network administrator. Nodes can only observe the local state around them; they do not have access to the global network topology or communication patterns. Even with this limited knowledge, learning is useful because it would otherwise be difficult for a human designer to create an optimized movement policy for each network scenario.

## 2. Related Work

We draw inspiration for this work from two different fields: networking and reinforcement learning. In the networking literature, some work on the effect of node mobility in ad-hoc networks has been done for applications in which movement and topology changes are on the time-scale of packet delivery. Nodes are then able to act as mobile relays physically transporting packets from one location to another. Grossglauser and Tse [1] analyze a strategy in which source nodes send packets to as many different nodes as possible, which store the packets and hand them off whenever they get close to the intended destination nodes. Li and Rus [2] consider a scenario in which mobile hosts make deviations from predetermined trajectories to transmit messages in disconnected networks. Chatzigiannakis et al [3] consider the case where a subset of mobile nodes are constrained to move to support the needs of the protocol, and act as a mobile pool for message delivery.

Our work is in a different setting, in which topology changes are on a much longer time scale than packet delivery delay constraints. Nodes move in order to form and maintain connected routes, rather than to physically deliver packets in a disconnected network. Routing is thus an important aspect of our algorithms that influences and is informed by movement decisions. Perkins and Royer [4] and Johnson and Maltz [5] examine routing in mobile ad-hoc networks where no control over node movements is assumed, while Chen et al. 6 deal with the issue of preserving a connected topology with only some portion of nodes awake at any one time.

From the reinforcement learning community, there has been some interest in applying learning techniques to improve network performance. Such adaptive algorithms may be better able to perform well under widely varying conditions. Boyan and Littman [7] applied reinforcement learning techniques to the problem of routing in static networks. They showed that a simple adaptive algorithm based on the Q-learning algorithm [8] can out-perform a shortest-paths algorithm under changing load and connectivity conditions. Peshkin [9] used policy search rather than Q-learning on the same problem, which allowed the system to search a richer policy space. By using stochastic routing policies, the system is able to manage high loads by finding multiple possible source-destination paths. We apply Q-learning to the case of mobile networks, where node connectivity is constantly changing.

Moreover, since we assume control over the nodes' movements, we can also influence these connectivity changes by learning a good control mechanism for the node movements. Several papers mentioned above propose various methods for controlling node and packet movement under specific assumptions. In the general setting, we wish to select optimal actions at each time step to maximize the long-term system performance. This type of problem lends itself to reinforcement learning techniques [10], where the goal of the learner is to maximize long-term reward by learning an optimal behavior policy through simulation. Stone and Sutton [11] and Bowling and Veloso [12] studied methods for scaling up reinforcement learning techniques to complex domains such as robotic soccer.

However, a major problem in the networking domain is the high degree of partial observability. From the perspective of any one node in the network, most of the rest of the network state cannot be discerned. The node can only rely on messages passed to it from its neighbors to glean information about the rest of the network, and to compound the issue, we would like to minimize such informational messages since they only contribute to network congestion.

Thus, we will have to deal with partial observability in creative ways.

## 3. Problem overview

Our mobilized ad-hoc network consists of one or more source nodes, one or more receiver nodes, and a number of other wireless nodes which may act as message relays within some constrained area. In our model, all the nodes' locations are independently initialized according to a uniform distribution over this area. The sources generate packets at a constant rate and move according to a random waypoint model. That is, they choose a location within the allowed area using a uniform distribution and then move to that location, repeating this process once they reach the location. The aim of the relay nodes is to transmit the maximum possible number of packets from the sources to the receivers. Some of these nodes can move so as to aid transmission, while the rest are stationary. All of the nodes are limited by a fixed transmission range.

Performance is measured by the proportion of packets successfully transmitted to the receivers when the sources are all generating packets at their maximum rate. When inter-node link capacities are limited and buffer sizes are finite, packet drops may occur due to buffer overflow, thereby decreasing network performance. When inter-node link capacities are sufficiently large compared to the source packet generation rates, an equivalent performance metric is the average proportion of sources connected to the receivers over time.

The packet transmission success probability achievable in an ad-hoc network depends heavily on various parameters including the transmission range $r$, the number of network nodes $n$, and the proportion of mobile nodes $m$. Alternatively, we may consider what values of these parameters are sufficient to achieve a desired success probability. For example, the following simple result illustrates these relationships and gives some sense of the potential benefits of having controllable mobile nodes in an ad-hoc network. It is a simple extension of a theorem by Gupta and Kumar [13] that sets a bound for the minimum (or critical) range necessary for $n$ stationary nodes in a given area to be fully connected. They state that for the case of a disk of area $A$ and $n$ approaching infinity, the critical range for the network is $r_n = \sqrt{\frac{A(\log n + \gamma_n)}{\pi n}}$, where $\gamma_n$ is a function of $n$ that grows arbitrarily slowly to infinity as $n \to \infty$. We extend this result to the case of partially mobile networks.

**Theorem 1** *Let $r_n$ be the minimum, or critical, range needed for $n$ nodes independently and uniformly distributed on a given area to form a fully connected network. If a proportion $m = \frac{k-1}{k}$, $k$ an integer, of the nodes are mobile, then a transmission range $r = \frac{r_n}{\sqrt{k}}$ is sufficient to make it possible for the mobile nodes to move to form a fully connected network. If the range $r$ is fixed at $r_n$, with $m = \frac{k-1}{k}$, $k$ an integer, then a total number $\frac{n}{k}$ of nodes suffices for full connectivity.*

This is an illustration of the type of results we can demonstrate for the desirability of using some fraction of mobile nodes in the network. They provide insights about the bounds on the potential improvement that would be possible using mobile nodes instead of stationary ones. The proof and further theoretical results will be presented in a separate paper; in this article, we will focus on the applications of learning that can begin to reap some of these benefits. We will show empirical evidence for the advantage of using learning techniques for both routing and movement in mobilized ad-hoc networks.

We note that in the type of analysis given above, we assume global knowledge of the positions of all of the nodes in the network. Some omniscient, centralized controller is able to see this information and compute optimal policies for each node in the network to follow. However, in most realistic settings, the nodes only possess local knowledge of their immediate surroundings, and furthermore, their movement speed is limited. Thus it becomes harder to reach the potential optimal network performance. Nevertheless, we can develop learning algorithms that perform well given these constraints. In the remainder of the paper, we develop such learning algorithms for movement and routing, and compare their performance against both non-learning and centralized algorithms, where performance is measured by the proportion of packets successfully transmitted to the receivers under various network scenarios.

## 4. Q-Routing

To optimize the performance of the ad-hoc network, we need to design good control algorithms for routing packets at the support nodes. The nodes will need to adapt to changing conditions and communication patterns using intelligent routing and movement algorithms. We focus on the routing problem here and give a brief overview of the movement problem, for which results will be presented in greater detail in a forthcoming paper.

Q-routing [7] is an adaptive packet routing protocol for static networks based on the Q-learning algorithm, which we adapt for use in mobile ad-hoc networks. It is very similar to a distributed Bellman-Ford algorithm. The algorithm allows a network to continuously adapt to congestion or link failure by choosing routes that require the least delivery time. When a route becomes congested or fails, Q-routing learns to avoid that route and uses an alternate path. Due to its adaptive nature, we might expect that Q-routing would also work well in the mobile ad-hoc setting.

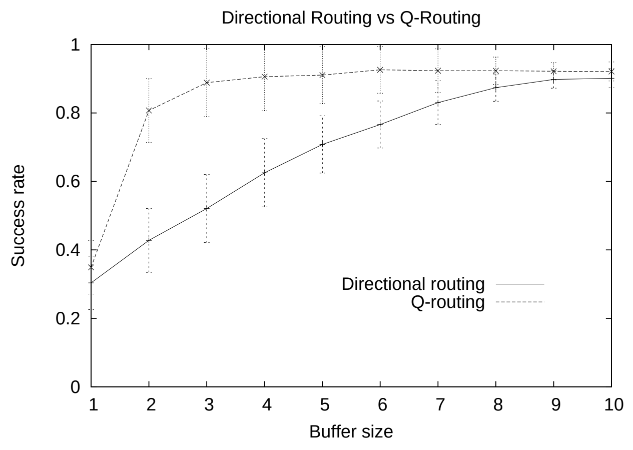

**Figure 1. A comparison of directional routing vs Q-routing in a network with 10 sources, 15 mobile agents, and one receiver. Simulations were run over 20 initialization positions and 5 source movement scenarios for each different initialization. For each buffer size, averages over all of these trials are depicted, with error bars denoting one standard deviation.**

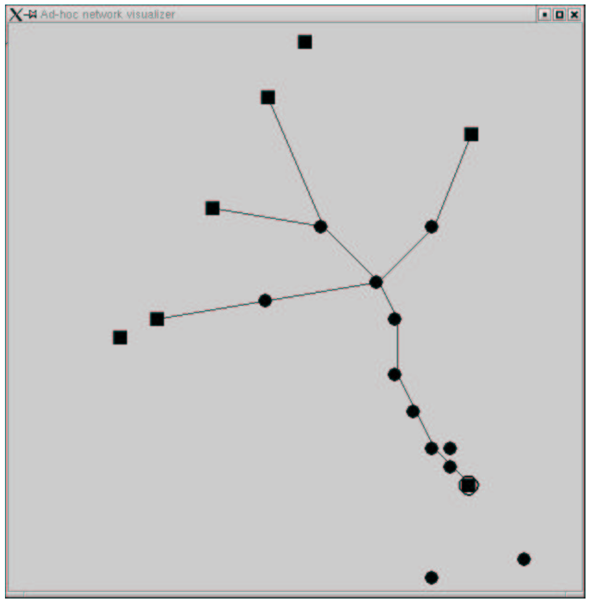

**Figure 2. Using the directional routing policy, packets often become congested on the trunk of the network tree. Sources are shown as squares, mobile nodes are circles, and the receiver is an encircled square. This shows the simulator after 10000 periods.**

Q-routing is a direct application of Watkins' Q-learning [8] to the packet routing problem. Q-routing is a distributed routing scheme where each node in the network runs its own copy of the Q-routing algorithm. A node $x$ faces the task of choosing the next hop for a packet destined for some receiver node $d$. Using Q-routing, it learns the expected delivery times to $d$ for each possible next hop $y$, where each possible next hop $y$ is a neighbor node connected to $x$ by a network link. Formally, Q-routing keeps Q-tables $Q^x$ for each node $x$ and updates these tables at each time period $t$ as follows:

$$Q^x_t(d,y) = (1-\alpha)Q^x_{t-1}(d,y) + \alpha(b^x_t + \min_z Q^y_{t-1}(d,z)),$$

where $0 < \alpha < 1$ is parameter that controls the learning rate, and $b_t$ is the time the current packet spent on the buffer or queue at node $x$ before being sent off at time period $t$.

Q-learning estimates the $value$ or $cost$, $V$, associated with each state $d$, with $V = \min_z Q^x(d,z)$. In our case, the value of a state is the estimated time for delivery of a packet from the current node $x$ to destination $d$ via node $z$. Once the nodes have learned the values associated with each state-action pair, they simply execute a greedy policy to behave optimally. When a node receives a packet for destination $d$, it sends the packet to the neighbor $y$ with the lowest estimated delivery time $Q^x(d,y)$. In experimental trials, Boyan and Littman found that fast learning rates (around 0.5) worked well since it is beneficial for the network to adapt quickly.

Adapting Q-routing to the mobile ad-hoc network routing domain is fairly straightforward. Neighbor nodes are defined as the nodes within transmission range. The main difference is that the neighbor nodes $y$ may appear and disappear quite frequently due to node mobility. When a node $y$ moves out of range, we set the estimated delivery time to $d$ via $y$ to $\infty$, i.e. $Q^x(d,y) = \infty$. When a node $y$ moves into range, we optimistically set the estimated time to 0, i.e. $Q^x(d,y) = 0$. This optimistic bias encourages exploration. That is, node $x$ will always try sending packets via a node $y$ that has just come into range. If this action results in a high estimated delivery time, then node $x$ will quickly revert to its original behavior since $Q^x(d,y)$ will quickly be updated to its true value. On the other hand, if this action results in a good delivery time, then node $x$ will continue to send packets via node $y$.

## 5. Empirical Results

Our empirical results give an indication of the power of using adaptive learning techniques in designing movement

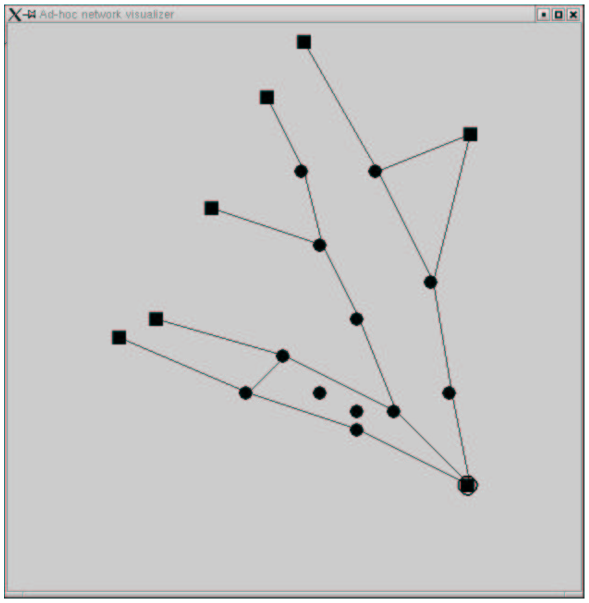

**Figure 3. Using Q-routing on the same experimental setup as Figure 2 (note that the source and receiver nodes are in the same position in both figures), the mobile nodes in the ad-hoc network spread out to distribute packet load. This shows the simulator after 10000 periods, using the same initialization and movement files as in Figure 2.**

and routing algorithms for mobilized ad-hoc networks. The setup is described in Section 3. There are source, receiver, and support nodes in a square grid, usually 30x30 in size. Each node has a transmission range of 6. The support nodes may either be fixed stationary nodes or mobilized agent nodes. There are two time scales: one for transmission and one for movement. During each movement time step, the node can choose one of its movement actions and perform 10 packet transmissions. Each packet transmission is the result of a Q-routing decision and update. Sources generate packets every two transmission time steps, and the number of packets received by a node in any time period is only limited by the node's buffer size.

## 5.1. Q-routing vs directional routing

To evaluate the performance of Q-routing, we implement a global-knowledge routing policy that is given information about the receiver location. This is done for comparison purposes only; in reality nodes generally do not have access to such information. With this information, nodes can route each packet towards the correct destination by forwarding the packet to a neighboring node that is closest to being in the direction of the receiver. We call this our *directional routing* policy. Specifically, our implementation forwards a packet to the neighbor that is closest to being in the direction of the receiver, up to a maximum deviation of 90 degrees. If no such neighbor exists, then the packet is dropped.

We compared Q-routing with directional routing under a variety of different scenarios. In almost all cases, Q-routing performs better than directional routing. Especially when the nodes are limited by small buffer sizes, Q-routing performs significantly better. Results for a typical set of network scenarios are shown in Figure 1. This is due to the fact that Q-routing will create alternate paths to the receiver as soon as a path becomes congested. Thus, packets will be less likely to be dropped due to buffer overflow caused by path congestion or limited buffer sizes since alternate paths will be constructed. In directional routing, on the other hand, often certain paths will become overloaded with traffic, causing significant packet drop.

Q-routing outperforms directional routing even in cases where buffer size is not a direct constraint, such as the case shown in Figure 1 where the buffer size is 10. This illustrates the underlying problem of network capacity. Since the total source packet generation rate exceeds the transmission rate of any one inter-node link, directional routing may still run into trouble with bottleneck links regardless of buffer size. The network capacity achieved using Q-routing is larger since more alternate paths are created around such bottlenecks. Furthermore, directional routing is unable to find circuitous paths from sources to the receiver. Since it only routes packets to neighbors that are in the direction of the receiver, any path that requires a packet to be forwarded away from the receiver for one or more hops will never be found.

These comparisons are done using a fixed movement policy we will call the *centroidal movement* policy. Under this policy, a node that is holding a connection will attempt to move to the centroid of its connected neighbors, which increases the likelihood of preserving these connections over time. If it is not holding a connection, then it simply moves about randomly searching for a new connection. Thus, the next hops determined by the routing policy strongly influence the direction of movement, since the next hops determine the node's connections to its neighbors.

When a random movement policy is used instead of the centroidal policy, Q-routing exhibits inferior performance relative to directional routing. One example is given in Figure 4, which shows the evolution of average system performance over time in a typical scenario. The table below gives averages over 100 different scenarios:

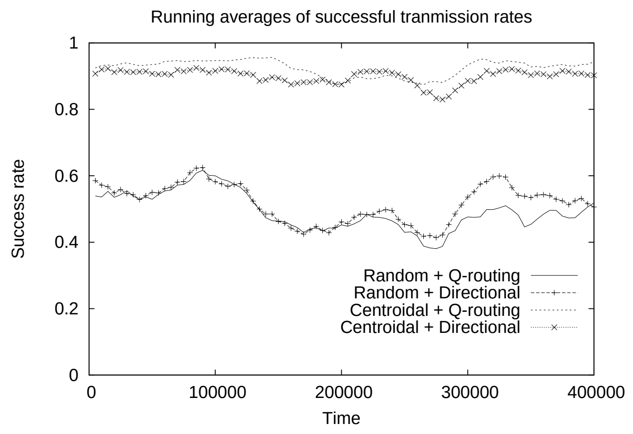

**Figure 4. This graph shows a running average of successful transmission rates for a sample network scenario under four cases: Q-routing with centroidal movement, directional routing with centroidal movement, directional routing with random movement, and Q-routing with random movement.**

| Movement policy | Routing policy | Average performance |
|---|---|---|
| Centroidal | Q-routing | .924 |
| Centroidal | Directional | .896 |
| Random | Q-routing | .498 |
| Random | Directional | .519 |

This phenomenon is due to the fact that Q-routing influences the centroidal movement policy in a positive manner, whereas it is unable to influence a random movement policy. In some sense, Q-routing with centroidal movement is able to find circuitous source-destination paths and rope them in using the centroidal movement.

Due to this type of influence, Q-routing and directional routing result in very different configurations for the network when coupled with centroidal movement. Directional routing tends to form a network backbone, which usually comprises the most direct route to the receiver for a large portion of the source nodes. Other sources send packets towards this backbone, resulting in a tree-like network configuration, as shown in Figure 2. Q-routing, on the other hand, always seeks the shortest path towards the receiver, even when buffer sizes are not a constraint. This results in a fan-shaped network configuration, also shown in Figure 2, where each source has its own shortest path to the receiver as long as there are a sufficient number of mobile nodes to create these paths. From this observation, we can begin to see that there is an interplay between the choice of routing protocol and the movement policy of the mobile nodes.

This leads to a subtle but telling explanation for the improved performance of Q-routing over directional routing. In Q-routing, the mobile agent nodes tend to become more dispersed, since no network backbone is created. Thus, as source nodes move about, the Q-routing ad-hoc network is more likely to be able to remain connected without drastic reconfigurations. This interplay between routing and movement forces us to carefully consider the movement policy we choose to pair with our selected routing policy.

## 6. Learning to move

The problem of learning a good movement policy is much more difficult. We wish to again apply reinforcement learning techniques to this problem. First of all, the problem of partial observability is much more pronounced than in the packet routing problem. For example, when the network is in a disconnected state, information about the global network topology is impossible to collect but would be important for determining movements that would optimize network connectivity. Moreover, the local observation space could still be quite large, depending on the observed variables we choose to encode. Secondly, the choice of action space is unclear. At the most basic level, the agents could move in any direction at any specified speed. Or, we could constrain them to moving at a constant speed North, South, East, or West, or simply staying put. These action choices are unsatisfactory: the latter is too constrained, and the former allows too many degrees of freedom. We will limit the action choices by designing more complex actions that incorporate domain knowledge. This allows the agents to learn complex behaviors while preventing our action space from growing too large. This section briefly outlines our application of Q-learning to learn a reasonable movement policy despite the fact that Q-learning generally fails in POMDPs.

We proceed by using the observation space as our assumed “state space”. Usually, state space refers to the true underlying state of the world. Here we will imagine that our observations give us a complete picture of our current state of the world. This can potentially lead to problems of aliasing, since the world is actually only partially observable and different underlying states may appear the same in our observations. We choose our observation variables carefully in order to attempt to avoid this pitfall. Treating our observation space as the state space reduces the complexity of the problem, since we will not try to infer knowledge about the underlying true state space using our history of observations, as is done in many partially observable learning domains. This domain is simply too large to try to infer global node locations using observation histories.

Instead, we use our observation variables directly, which gives us some limited knowledge beyond our immediate surroundings. For example, since the nodes communicate using a shared wireless medium, a node can "sniff" packets sent by neighbors to destinations other than itself. Thus, a node can detect the presence of neighbor nodes and their connections, even if it is not involved in any of these connections itself. Moreover, since the receiver nodes send back acknowledgement packets along these connections, our agent node can also collect statistics about these source-destination paths by sniffing these acknowledgement packets. Each agent node's observation space can thus includes the number of neighbors currently surrounding it, the number of connections it is currently holding, the number of nearby connections, and the maximum and minimum average hop lengths of these source-destination paths.

We also incorporate some domain knowledge into our design of an appropriate action space. This allows the agents to learn rich and interesting policies without the need for exponentially large state and action spaces. For example, there is little need to train the nodes to learn a policy for avoiding obstacles along their desired path of movement. We can pre-program the nodes with the necessary algorithm to do this. Learning is focused on higher-level action selection that is difficult to pre-program effectively.

A subset of our action space is given in the table below. Many of these actions could take multiple time periods to complete. We currently get around this problem by allowing the agents (nodes) to choose to either continue or change an action during each time period. A more disciplined approach using the framework of macro actions is also being developed.

| Action | Description |
|---|---|
| stay | Stay put; don't change position. |
| direction | Head in a particular direction. |
| plug | Searches for the sparsest path and attempts to fill in the largest gap along that path. |
| leave | Leaves a path and becomes available for other actions. |
| circle | Circles around a node in search of more connections. Attempts to retain connection to the source around which it is circling. |
| lead | Leads other mobile agents in search of more connections. Attempts to stay within range of its followers. |
| follow | Identifies and follows a leader node, while maintaining connection to previous hop. |
| center | Moves to the centroid of the neighbors to which it is connected. |
| explore | Random exploration. |

Using this state and action space, learning can thus be focused on higher-level action selection that is difficult to pre-program optimally for different environments which the agents might encounter. One of the main advantages of using a learning method to generate movement policies is that we can quickly create new agents optimized for new environments simply by training them under a different simulation. By experiencing a simulation of the environment in which they will be expected to operate, the agents can learn to piece together pre-programmed lower-level behaviors to create a good high-level movement policy tailored for those particular environmental factors.

Finally, the reward given to the nodes during each time period corresponds to the percentage of successful transmissions during that time period, which is available since we are conducting this training off-line. During online execution of the ad-hoc network, this network performance measurement would not be available since there would be no trainer that has access to the global state of the network. With these assumptions, we can construct a standard Q-learning algorithm that tries to learn good movement policies as the agents interact in simulation.

## 6.1. Movement policy comparisons

We evaluate the performance of our learning algorithm against the centroidal movement policy given in Section 5, a hand-coded policy that uses the same observation space as the learning algorithm, and a global-knowledge central controller. Under simulation, we give the central controller access to all the node, source, and receiver positions, which would usually be unavailable to the agent nodes. Since our learning algorithm only has access to local knowledge, the central controller's performance should approximate an upper bound for the learning algorithm's performance. Moreover, this performance bound may fluctuate over time as network conditions change depending on source movement scenarios.

The central controller is designed to approximate an optimal movement policy given global knowledge about network conditions. It begins by identifying connected components among the stationary nodes. If a packet is received by any node of a connected component, then all nodes of that component can also receive the packet. However, if a packet's destination is in a different component, then the controller finds the shortest path to the destination component and recruits mobile nodes to help construct the necessary links needed to deliver the packet.

Figure 5 gives the average performance of a sample network scenario over time using each of these movement policies. As we can see, the learning algorithm does eventually learn a policy that behaves fairly well, but it never achieves the performance of the global knowledge controller. This is expected since the learner does not have access to the global

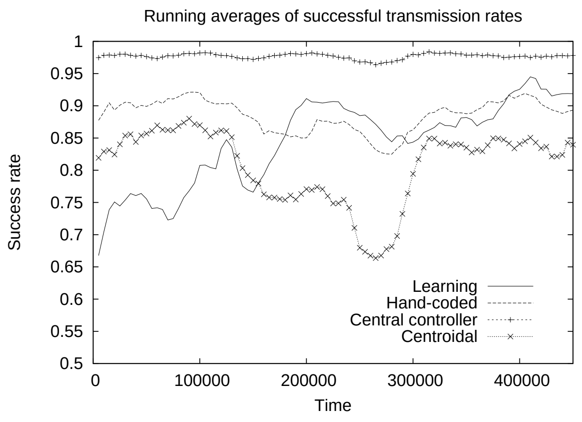

**Figure 5. Graph showing the average performance of various movement policies over time in a typical scenario. The learning policy is shown during its training phase.**

network topology. On average, the learned policies perform slightly better than the hand-coded policy over large sets of network scenarios. However, in certain scenarios, it never learns to perform as well, possibly due to aliasing problems.

## 7. A simplifying assumption

These results for node movement, while promising, are still inconclusive and more work needs to be done to create better learning algorithms for optimizing node movement policies. One problem is that the learning agent cannot distinguish between states where high network performance is due to its own good actions, versus states where the high network performance is due only to actions taken by other agents in the network. Chang et al. [14] describe one potential mechanism for solving this credit assignment problem by using a simplified model of the world to estimate an agent's deserved reward given only observations of a global reward. We could apply this technique to the mobilized ad-hoc network simulations that we consider here.

## 8. Conclusion and future work

This paper presents a new domain for multiagent reinforcement learning and establishes several first results in this area. There is much more work to be done to create a more robust movement learning algorithm that deals directly with the partial observability of the domain. Policy search and Monte Carlo methods are two techniques that are currently being examined. A well designed algorithm might also be able to exploit the interplay between routing and movement.

## References

[1] M. Grossglauser and D. Tse. Mobility increases the capacity of ad-hoc wireless networks. In *INFOCOM*, 2001.

[2] Q. Li and D. Rus. Sending messages to mobile users in disconnected ad-hoc networks. In *MOBICOM*, 2000.

[3] I. Chatzigiannakis, S. Nikoletseas, N. Paspallis, P. Spirakis, and C. Zaroliagis. An experimental study of basic communication protocols in ad-hoc mobile networks. In *5th Workshop on Algorithmic Engineering*, 2001.

[4] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *MILCOM Panel*, 1997.

[5] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, volume 353. 1996.

[6] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance. In *SIGMOBILE*, 2001.

[7] J. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in NIPS*, 1994.

[8] C. J. Watkins. Learning with delayed rewards. *Ph.D. Thesis, University of Cambridge*, 1989.

[9] L. Peshkin. Reinforcement learning by policy search. *Ph.D. Thesis, MIT*, 2002.

[10] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[11] P. Stone and R. S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *ICML*, 2001.

[12] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[13] P. Gupta and P. R. Kumar. Capacity of wireless networks. In *Stochastic Analysis, Control, Optimization, and Applications*. Birkhauser, 1998.

[14] Y. Chang, T. Ho, and L. Kaelbling. All learning is local: Multi-agent learning in global reward games. In *Proceedings of NIPS '03*, 2003.