

# Reinforcement Learning based approach to Routing in Sensor Networks CS252 Project

Ambuj Tewari

May 12, 2005

## Abstract

Routing in sensor networks is a challenging problem for a variety of reasons and several approaches have been proposed [1]. In this report, we discuss existing reinforcement learning based work on network routing. We then present a simple routing algorithm for sensor networks which is inspired by the Q-learning algorithm from the reinforcement learning literature. Its implementation in TinyOS is described and simulation results are obtained using TOSSIM.

## 1 Introduction

Sensor networks consist of resource constrained nodes with limited computation, memory and communication capabilities. Further, the connectivities of nodes are usually asymmetric and time varying. Thus the most popular model of a communication network as a graph with edge weights representing link costs breaks down for two reasons. First, the edge weights are not symmetric and constant. Second, algorithms solving the routing problem with such a model may not be implementable using the restricted set of resources available on sensor network nodes. There are theoretical reasons to believe that changing edge weights cause the problem of network routing to become significantly harder. While the theory of shortest path problems has well known elegant polynomial time solutions (e.g. Dijkstra, Bellman-Ford, Floyd-Warshall algorithms), there are some results that show that if edge weights can change probabilistically, the routing problem can become NP-hard [4].

There are other characteristic features of sensor networks that make the routing problem for sensor network different from the same problem in other kinds of networks. The nodes are typically deployed in an ad-hoc fashion and so the network has to self-organize as there is little or no manual intervention. Once deployed, the nodes do not move as much. However, we do expect other kind of changes such as changes in link quality, node failures or change in traffic pattern. The dominant pattern of traffic flow is from the nodes to a distinguished node called the base

station. This happens, for example, when the sensor network is used for some periodic data collection task.

Reinforcement learning (or adaptive control) provides us with a set of tools to attack such hard problems where we are pressed to make decisions in the face of uncertainty. Each node in the network can be viewed as an agent that not only locally originates packets but also receives packets from other nodes. The decision that the node has to take is which immediate neighbor should the packet be sent to. In this report, we model the sensor network routing problem as a reinforcement learning problem and present and evaluate a variant of the well-known Q-learning algorithm.

This report is organized as follows. In Section 2, we describe some related work on modeling routing as a reinforcement learning problem. Section 3 presents the algorithm we used. The TinyOS implementation of the algorithm is described in Section 4. Some simulation results are presented in Section 5. We conclude in Section 6.

## 2 Related Work

The idea of applying reinforcement learning to the problem of network routing is certainly not new. The first paper, to the best of our knowledge, with this idea was by Boyan and Littman [2]. They included the time delay due to local queue size in their cost function but did not penalize cycles in any way. They tried approximating the cost function using a neural network but did not get any conclusive results. For us, using any sophisticated approximation method is out of question because of resource constraints. One would not want to run a neural network learning algorithm on a sensor network node.

The problem of call admission and routing in telecommunications networks was tackled using reinforcement learning by Marbach et. al. [3]. Instead of using Q-learning like Boyan and Littman, they chose to use Sutton's  $TD(\lambda)$  algorithm to learn a good policy. Their paper nicely describes how to decompose the problem in way that allows for the possibility of a decentralized approach. They worked with a network with a very large number (about  $10^{45}$ ) of different possible states and showed that their approach did better than commonly used heuristics.

There also have been some recent papers on the topic of applying reinforcement learning techniques to network routing [5, 6, 9]. The Tao et. al. paper [5] defines reward in terms of the total trip time of the packet which then has to be communicated by the destination node back to the sending node. Nodes make decisions independently of each other based on parameters that are stored and updated locally. To be precise, suppose node  $n$  has  $k$  immediate neighbors  $m_1, \dots, m_k$ . Then a (stochastic) policy for  $n$  can be represented by  $k$  real parameters  $\theta_1, \dots, \theta_k$  where the probability that node  $n$  will route a packet through  $m_i$  is

$$\frac{e^{\theta_i}}{e^{\theta_1} + \dots + e^{\theta_k}} \quad (*)$$

They update these parameters based on the rewards obtained over time. Cycles were detected by storing a last few nodes a packet has been through

in the packet header. Interestingly, they decided to give a large negative reward to *all* the nodes if cycles were detected during some time period. However, even such a simple strategy was able to improve performance. We will have something to say on a somewhat similar effect we observed in our simulations. The Peshkin and Savova [6] paper uses a similar representation of (stochastic) policies with the difference that there is an extra “temperature” parameter  $T$  and the expression (\*) becomes

$$\frac{e^{\theta_i/T}}{e^{\theta_1/T} + \dots + e^{\theta_k/T}}$$

Note that as  $T \rightarrow 0$ , the above expression becomes 1 if  $\theta_i$  is the maximum of  $\{\theta_1, \dots, \theta_k\}$  and 0 otherwise. So, it finally converges to some deterministic policy. Finally, Zhang and Fromherz applied Q-learning to the problem of routing in sensor networks. They proved the convergence of their algorithm when the network is static. Their algorithm has to be supplied an initial heuristic estimate of the Q-values and their simulations (done using Prowler) show that the performance depends quite a bit on the initial heuristic.

### 3 Q-Routing

As mentioned before, our objective is to route all data packets to a distinguished node called the base station. We assume that each node keeps information about its neighbors in a neighbor table and also keeps a forwarding queue for messages that have been received from immediate neighbors and are waiting to be routed onwards to the base station. The state at each of the nodes thus consists of:

- the forwarding queue
- the neighbor table

Events that can occur at a particular node are:

- a data message originates locally
- a data message arrives from a neighbor
- a route message arrives from the neighbor

The actions that a node can take are:

- drop a packet if it has been in the network for too long
- choose a neighbor to route the packet to

Given the state of all the nodes and the actions they take, the only other thing that determines the subsequent state is the environment. So, if the environment does not base its actions based on observations of the network state, the dynamics of the network are Markovian, i.e. given the current state and current action, the probability distribution over the next states we end up in is independent of the previous history of the network. This allows us to treat the problem as a reinforcement learning problem. We note that this assumption breaks down when we have a malicious adversary instead of a randomly behaving environment.

We want to learn a rule mapping states to actions so that we achieve good routing performance. Such a rule is called a policy in reinforcement learning literature. A popular way to learn a policy keeps a running estimate of the cost of taking action  $a$  in state  $s$  and then choosing an action for which this estimate is the minimum. If, as a result of taking action  $a_t$  in state  $s_t$  at time  $t$ , we incur cost  $c_t$  and transition to state  $s_{t+1}$ , then the update performed is:

$$Q(s_t, a_t) \leftarrow \alpha Q(s_t, a_t) + (1 - \alpha)(c_t + \min_a Q(s_{t+1}, a))$$

where  $\alpha \in (0, 1)$  is the learning rate. This is the well-known Q-learning algorithm of Watkins [8].

In our case, each node  $n$  keep a running estimate  $Q(n)$  of the cost of reaching the base station. For a path  $n_1, \dots, n_k$ , we chose a cost function:

$$c_1 \sum_{i=1}^{k-1} \frac{1}{\text{link\_prob}(n_i, n_{i+1})} + c_2 \sum_{i=2}^k \text{fwd\_queue\_len}(n_i)$$

where  $c_1, c_2 > 0$  are constants.  $\text{link\_prob}(n_i, n_{i+1}) \in (0, 1)$  is an estimate of the quality of the link between the two nodes. We show how we maintain that in the implementation section.  $\text{fwd\_queue\_len}(n_i)$  is just the length of the forwarding queue at node  $n_i$ .

Once we maintain the Q-values, decision making is simple. We simply choose the neighbor  $m$  in the neighbor table which minimizes:

$$\frac{c_1}{\text{link\_prob}(n, m)} + Q(m).$$

After making the decision, we update our own Q-value to:

$$Q(n) \leftarrow \alpha Q(n) + (1 - \alpha)(c_2 \text{fwd\_queue\_len}(n) + \frac{c_1}{\text{link\_prob}(n, m)} + Q(m))$$

## 4 Implementation

The Q-routing algorithm was implemented in nesC for the TinyOS operating system. The `QRout` configuration encapsulates the functionality of the Q-routing algorithm and exports the `Send` and `Receive` interfaces. Its relationship to other TinyOS components is shown in Figure 1.

Since nodes need access to the Q-values of their neighbors, nodes have to periodically broadcast a route message containing their Q-value. The estimate `link_prob` is simply maintained by counting the fraction of route message broadcasts of our neighbors that we actually hear. Moreover, if we miss more than a certain number of route broadcasts of a neighbor, that neighbor is evicted from the neighbor table. Each entry of the neighbor table has the following structure:

```
typedef struct QNbr {
    uint16_t addr;           // address of neighbor
    uint8_t times_heard;    // no. of times this nbr's
                           // route msg was heard in
                           // the last interval
```

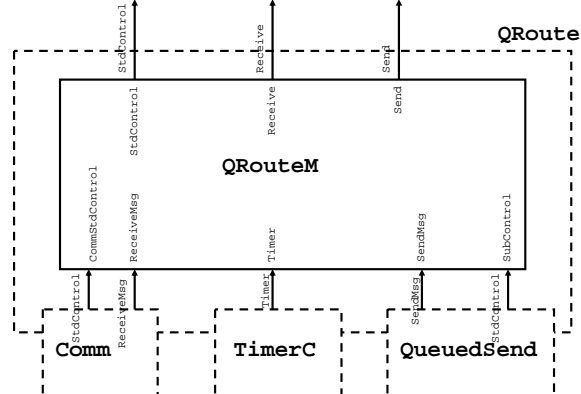


Figure 1: Interface provider→user relationships for the QRoute configuration

```

uint8_t times_heard_current; // no. of times this
                             // nbr's route msg was heard
                             // in the current interval
int16_t last_seqno;         // last sequence number
                             // received
uint16_t Qval;              // Q value
} QNbr;

```

The packet format for the Q-routing algorithm is:

```

typedef struct QMsg {
    uint16_t source; // the node where this packet has just
                    // been forwarded from
    uint16_t origin; // the node where this packet originated
    uint16_t dest;   // destination; 0 always for now
    int16_t seqno;  // sequence number to check for
                    // duplicates
    uint8_t type;   // whether routing message or data
    uint8_t hop_count; // how many nodes has this packet been
                    // routed through
    uint8_t data[(TOSH_DATA_LENGTH - 10)];
} QMsg;

```

We drop a packet instead of forwarding it if the hop count exceeds some threshold. The node which drops the packet incurs a cost and updates its  $Q$  value:

$$Q(n) \leftarrow Q(n) + \text{DROP\_PENALTY}$$

This may be contrasted with the approach of Tao et. al. [5] who penalize the entire network equally. We are at the other extreme and only penalize

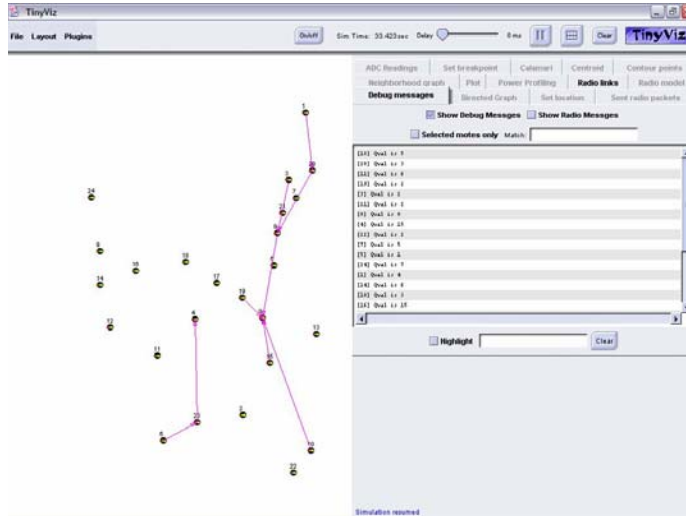


Figure 2: A 25 node topology

the node which drops the packet. However, we too observe that a non-zero drop penalty improves performance (see next section).

## 5 Simulations

TOSSIM, Tython and TinyViz were used to perform the simulations. We generated a random topology for 25 nodes and saved it (Figure 2). The same topology was used for all simulations. The nodes were programmed to send a packet to the base station (node number 0 in all simulations) every 2 seconds. The route message broadcast was done every second. The learning rate  $\alpha$  was set to 0.5. The size of the neighborhood table was 20.

Figure 3 shows the contour plot for Q-values after the network has stabilized. The Q-value difference between successive lines is 5. One node (node 1) was unable to get its Q-value initialized. Figure 4 shows how the Q-values are learned over time for 4 nodes chosen from different parts of the network.

The Q-values correlate well with the success rate (fraction of packets successfully delivered to the base station). This is illustrated in Figure 5. There is significant negative correlation (-0.59) between the 2 quantities.

To see the impact of the drop penalty, we plot the overall node to base station success rate in Figure 6. We can see that using a low drop probability is a good choice. The performance is worse without drop penalty and also with large penalty.

Figure 7 shows the overall node to base station success rate for the 7 combinations of  $c_1, c_2$  values obtained as  $c_1$  varies from 1 through 7 and  $c_2$  is set to  $8 - c_1$ . We see that it is better to set  $c_1 < c_2$ . Putting too

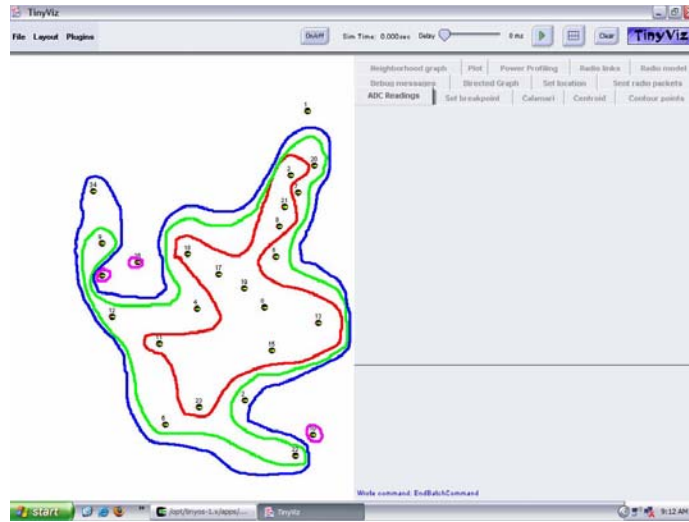


Figure 3: Contours of Q-value

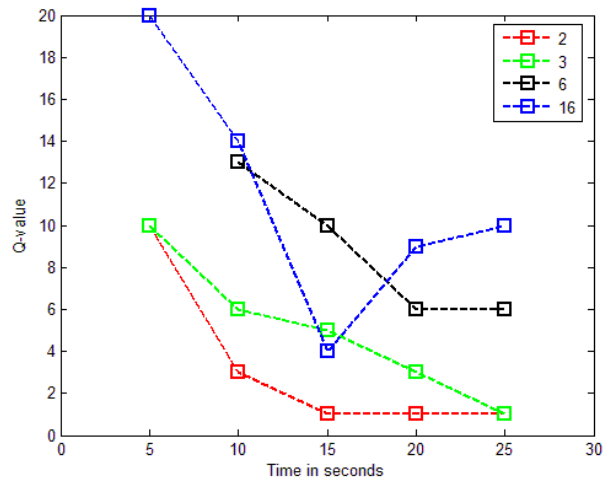


Figure 4: Plots of Q-values over time

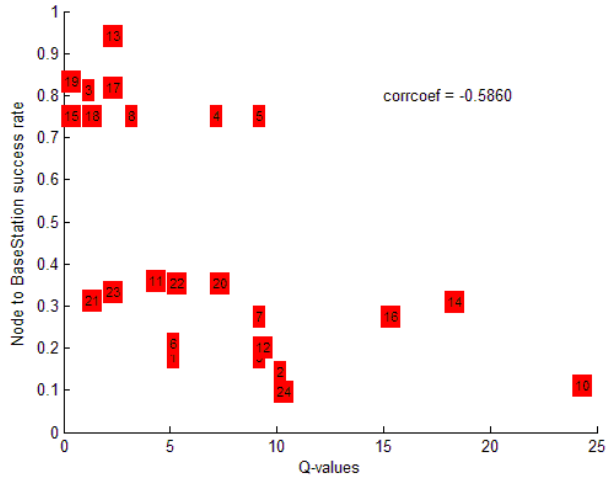


Figure 5: Correlation between success rate and Q-values

much weight on the `link_prob` term hurts us, probably because they are not well estimated by the simple counting mechanism we follow.

## 6 Conclusion

In this report we presented and analyzed some aspects of a particular reinforcement learning algorithm applied to the problem of routing in sensor networks. The algorithm is quite simple and all steps in the algorithm involve only a few simple operations. It is interesting that the algorithm is able to create an implicit tree without explicitly representing parents of nodes. The performance of the algorithm is not very good and the overall success rate remains below 0.5 even for the best settings of the parameters. We saw that penalizing a node when an old packet is dropped helps improve the performance. So, the algorithm can obviously benefit from a better cycle detection and penalization strategy.

We were discouraged from applying other reinforcement learning techniques because they usually involve more complex computations. However, one can think of adapting these algorithms to the task at hand by approximating their exact computations with simple ones.

Even if we restrict ourselves to Q-learning style algorithms, there are still some issues to explore. For example, following Tao et. al. [5], we can have reward packets flowing back from the base station serving as a delayed feedback on the performance of past actions. More generally, one can have feedback packets flowing within subtrees rather than the whole tree. This strategy bears similarity to routing algorithms based on ant colonies [7].

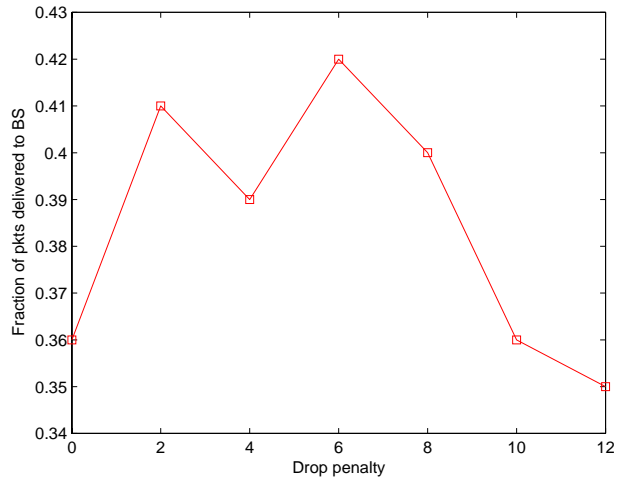


Figure 6: Overall success rate vs. drop penalty

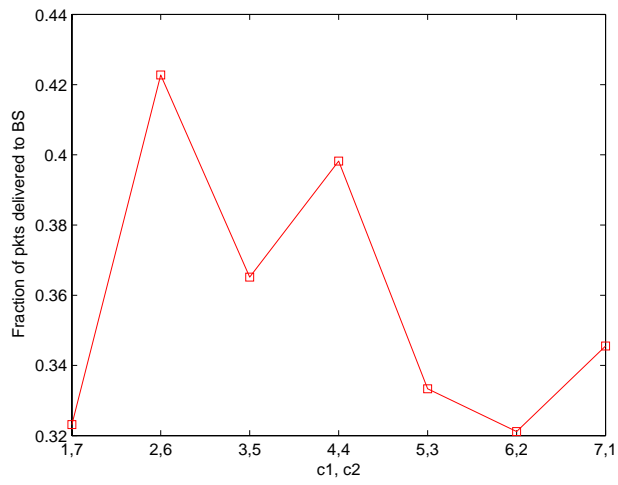


Figure 7: Overall success rate vs.  $c_1, c_2$

## References

- [1] Al-Karaki, J.N. and Kamal, A.E.: Routing Techniques in Wireless Sensor Networks: A Survey. *IEEE Wireless Communications* **11**:6, Dec 2004, pp. 6–28.
- [2] Boyan, J. A. and Littman, M. L.: Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems 6*, 1994, pp. 671–678.
- [3] Marbach, P., Mihatsch, O., Schulte, M. and J. N. Tsitsiklis: Reinforcement learning for call admission control and routing in integrated service networks. In *Advances in Neural Information Processing Systems 11*, 1999.
- [4] Orda, A., Rom, R., and Sidi, M.: Minimum delay routing in stochastic networks. *IEEE/ACM Transactions on Networking (TON)* **1**:2, April 1993, pp. 187–198.
- [5] Tao, N., Baxter, J., and Weaver, L.: A multi-agent, policy gradient approach to network routing. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.
- [6] Peshkin, L. and Savova, V.: Reinforcement learning for adaptive routing. In *Proceedings of the International Joint Conference on Neural Networks*, 2002.
- [7] Subramanian, D., Druschel, P. and Chen, J.: Ants and reinforcement learning: A case study in routing in dynamic networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997, pp. 832–839.
- [8] Watkins, C. J. C. H.: Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England, 1989.
- [9] Zhang, Y. and Fromherz, M. P. J.: Search-based adaptive routing strategies for sensor networks. In *AAAI-04 Workshop on Sensor Networks*, 2004.