# Towards Autonomous Data Ferry Route Design through Reinforcement Learning

Daniel Henkel and Timothy X Brown
University of Colorado at Boulder
Boulder, CO 80309, USA
{henk,timxb}@colorado.edu

## Abstract

*Communication in delay tolerant networks can be facilitated by the use of dedicated mobile "ferries" which physically transport data packets between network nodes. The goal is for the ferry to autonomously find routes which minimize the average packet delay in the network. We prove that paths which visit all nodes in a round-trip fashion, i.e., solutions to the Traveling Salesman Problem, do not yield the lowest average packet delay. We propose two novel ferry path planning algorithms based on stochastic modeling and machine learning. We model the path planning task as a Markov Decision Process with the ferry acting as an independent agent. We apply Reinforcement Learning to enable the ferry to make optimal decisions. Simulation experiments show the resulting routes have lower average packet delay than solutions known to date.*

## 1 Introduction

Future mobile ad-hoc networks (MANETs) are expected to reliably connect fixed and mobile nodes to support a range of applications. In a sensor network, for instance, sensor task nodes can help collect scientific data or provide situational awareness for natural and man-made disaster response. In many of these situations networks might only be sparsely and intermittently connected. Under these circumstances only delay-tolerant communication is feasible. Within the Delay Tolerant Networking (DTN) framework [3] message passing and flooding algorithms for data delivery have been developed which rely on the natural, intrinsic and uncontrolled movement of network nodes [11, 7]. The mobility of the task nodes themselves is exploited to improve the network performance. Even when there is never a contemporaneous end-to-end connection between sender and receiver, these methods deliver packets if such a path is formed over time. Nodes physically store and carry packets until forwarding to a suitable next node is possible. If movement of task nodes can be controlled, routes for mobile nodes can be created to connect network partitions. However, this requires cooperation of the deployed nodes and burdens nodes with extra processing.

We consider networks where the mobility of certain nodes can be controlled. Such nodes can change location to specifically aid communication. In sparse DTN so-called ferry nodes can go further and physically carry data packets between otherwise disconnected task nodes [14, 8]. The main advantages of the ferrying approach are its ability to overcome network partitioning and reduce network congestion since data is physically carried rather than retransmitted multiple times. However, a motion is slow relative to transmission and propagation delays. It is not clear how ferries should move to minimize overall average packet delay.

Recent work puts an emphasis on designing optimal paths for the ferry's movement. The majority of route planning work relies on solutions to the well-studied Traveling Salesman Problem (TSP) to maximize throughput or minimize delay [14, 13] or provide differentiated services in a DTN [12]. Other work focuses on buffer constraints in sensor networks and schedules ferries according to the Earliest Deadline First algorithm [9].

Our work is motivated by the ongoing development of an Ad-hoc Unmanned aircraft to Ground Network (AUGNet) testbed comprised of mobile and fixed ground nodes as well as airborne network nodes with controlled trajectories [2]. The aircraft can span large distances in a relatively short time thus simplifying data collection from far-flung sensors and facilitating communication among ground nodes. In this specific scenario we found prior work to perform sub-optimally, and specifically TSP-based ferry route designs seem not well suited for communication between multiple sensors and one common data sink. The question now is how autonomous machine learning approaches can be used to generate optimal ferry routes and what issues arise when trying to apply these algorithms to real-world networks.

We will introduce the path planning problem in detail in Section II and talk about the deficiencies of current TSP-based solutions in Section III. Section IV describes a planning algorithm based on a stochastic model and Section V outlines the Markov Decision Process (MDP) formulation and solution using Reinforcement Learning (RL). We compare the performance of the algorithms to previous solutions and point out implementation issues in Section VI, and we conclude the paper in Section VII.

## 2 The Path Planning Problem

We address the design of movement paths for data ferries that physically carry delay-tolerant traffic. Our network consists of sparsely distributed stationary nodes, called *task nodes*, which need to forward data to a sensor monitoring station (SMS). Mobile helper nodes, called *ferries*, aid in delivering traffic.

There are $n$ task nodes distributed on a *survey area*.Each task node has a large buffer and generates a data flow at rate $f_i$. A network is considered stable if these flows are met over time. It is assumed that the separation distance between nodes is much greater than the communication range of the radios. Thus there is no connection between nodes and no possibility for routing traffic among them without the help of some mobile node. Data ferries are capable of moving around the survey area to any location requested. We assume constant ferry velocity $v$, but ferries can also loiter above a task node for data exchange. The ferry has a large buffer of size $b$.

The goal of the ferry is to choose routes that minimize the average packet delay throughout the network. An individual packet's delay is comprised of the wait time at the node until arrival of the ferry, the transmission time of the node's buffered traffic to the ferry, the ferry's flight time to the SMS, and the transmission time of the ferry's buffer to the SMS. We are assuming negligible packet transmission times compared to ferry travel times, since wireless data rates are high and buffered traffic is small.

Moving a ferry requires significant time relative to communication rates. The choice of when and where to move a ferry has long term consequences that require planning. Previous work assumes this planning is done by an external entity, while in Sec. 5 we introduce the ferry as an intelligent agent that makes independent decisions. The planning consists of two tasks: assessing communication needs and making helper node assignments. Assessing communication needs can be modeled with varying degrees of realism, ranging from a god-like knowledge of all task node's communication needs to an incomplete prediction based on previously observed flows. We assume that the task nodes send out periodic low-rate and thus long-range packets that contain communication requests. The next section considers one method for making helper node assignments.

## 3 Shortcomings of the TSP Solution

The path planning problem has been investigated in the research community, and most of the results are based

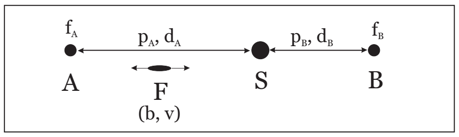

**Figure 1. The hub, $S$, communicates with two stations $A$ and $B$ at distance $d_A$ and $d_B$.**

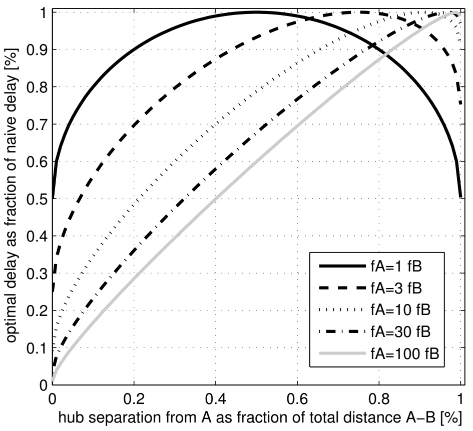

**Figure 2. Comparison of naïve TSP vs. optimal ferry assignment algorithms.**

on solutions to the common Traveling Salesman Problem (TSP) [14, 13]. However, any TSP solution which aims at visiting all nodes in a given cycle is sub-optimal. Consider the network in Fig. 1. Two task nodes $A$ and $B$ are communicating data to a central hub $S$ with respective traffic flow rates $f_A$ and $f_B$. The hub can be located anywhere on the straight line between A and B, and its location relative to node A can be expressed as a fraction of the total distance $|AB|$. One ferry serves both nodes transporting data from the nodes to the hub. The goal is now to schedule visits to $A$ and $B$ such that the average packet delay is minimized. The allocation will be expressed here as the fraction of a visits, $p_A$ respective $p_B$, that the ferry visits nodes $A$ and $B$. Based on queueing analysis an equation for delay can be derived. This can be minimized to find the optimal allocation of visits between the two nodes. This analysis will be derived in Section 4 for $n \leq 2$.

Since there are only two nodes the only solution for TSP is to alternately visit each node. This corresponds to a naïve allocation which would visit each node equally often, i.e., $p_A = p_B = 0.5$. The optimal allocation visits each node with varying frequency per cycle, i.e., $p_A \neq p_B$, and can have significantly lower delay. Fig. 2 shows the optimal average delay as a fraction of the delay using the naïve algorithm. The optimal allocation's delay can be a few percent of the naïve allocation's delay, up to a factor of 100 smaller in the $f_A = 100 f_B$ case. This simple example is still analytically tractable and it shows the shortcomings of using the TSP approach. In Section 4 we will give a detailed analysis of this algorithm and generalize to more than 2 nodes.

## 4 Stochastic Algorithm Design

Now we evaluate an algorithm which assigns ferry time to nodes in the case of $k$ task nodes transmitting to one data sink (aka hub). We assume full knowledge of the network parameters, including traffic flow rate $f_i$ for each node $i$, distances $d_i$ from node to hub, buffer size $b$ and constant speed $v$ of the ferry. The hub communicates with each station one at a time. Communication takes place from the task nodes to the central hub with the help of one ferry. Figure 1 shows the network setup with $k = 2$ nodes. The first goal is to see if the set of flows can be carried, that is, if $\sum_i \frac{f_i}{R(d_i)} < 1$ where $R$ is the transmission rate. As a lower bound we assume that the ferry travels the complete distance between hub and stations for each transfer, i.e. $R(d) = \frac{vb}{2d}$. Using these assumptions, the capacity constraint is:

$$\frac{2}{vb}\sum_i f_i d_i < 1$$

We can observe that as we scale to larger systems that span larger regions (i.e., the $d_i$ increase), the load with ferrying increases linearly. The ferry can always meet any required throughput load if $b$ or $v$ are made large enough. The time to visit a station in the ferry case varies from station to station depending on the distance. A simple round robin visit to each station in turn may not be appropriate to minimize delay. A station that is closer can be visited more often lowering this station's delay with less of an impact on overall delay. Visiting a node with a larger flow rate more often will also tend to lower the overall delay. A visit from the ferry can carry one or more packets up to the limit of the ferry buffer. If the buffer is sufficiently large the effect of visiting a station is to clear all backlogged packets. The following sections discuss these in more detail. We consider a probabilistic model of choosing the next station to visit. In this model, when the current station is finished being served, the station $i$ is chosen to be the next station with a probability $p_i$ where $p_i \geq 0$ and $\sum_i p_i = 1$. The expected time for any packet to be served is:

$$T = \sum_i p_i \tau_i, \tag{1}$$

where $\tau_i$ is the time to serve one packet from node $i$ (we will return to this later). If we assume the visits are chosen stochastically, then the time between visits to a given node $i$ is approximately exponentially distributed with mean $T/p_i$ for node $i$. Thus, regardless of whether other packets are already waiting or not, when a packet arrives to be serviced at node $i$ the expected delay before the next packet is served at node $i$ is $T/p_i$. If we assume that the ferry buffer is large, then when the node is serviced, all packets waiting will be served. As a result the average service time for a packet is $T/p_i$. The goal is to minimize this delay for packets across nodes weighted by the probability a packet will come from each node. The weighted delay is:

$$\tau = \sum_i \frac{T f_i}{p_i F} = \sum_i \sum_j \frac{p_j \tau_j f_i}{p_i F},$$

where $F = \sum_i f_i$ and $f_i/F$ is the fraction of the total traffic from node $i$. This equation can be minimized with respect to the set of $p_i$. The result is:

$$p_i = \frac{\sqrt{f_i/\tau_i}}{\sum_j \sqrt{f_j/\tau_j}} \tag{2}$$

which can be seen by substituting the above solution into the derivative of $\tau$ with respect to $p_k$ and showing the result is zero for any $k$.

For the ferry communication, we assume the ferry travels all the way between the node and the hub, $\tau_i = \frac{2d_i}{v}$. Substituting $\tau_i$ into $p_i$ and $p_i$ into the delay equation:

$$p_i = \frac{\sqrt{f_i/d_i}}{\sum_j \sqrt{f_j/d_j}}, \tag{3}$$

and

$$\tau = \frac{1}{F}\left(\sum_i \sqrt{f_i \tau_i}\right)^2 = \frac{2}{vF}\left(\sum_i \sqrt{f_i d_i}\right)^2.$$

Equation (3) shows that a node $i$ will be visited more often as $f_i$ increases and $d_i$ decreases.

A simple stochastic algorithm would choose the next node to visit according to the probabilities $\{p_i\}$. A deterministic algorithm would use a fractional visit counter $c_i$ for each node to track visits. This specifies a fair algorithm in the sense that if all $p_i$ are equal then each node will be visited once per cycle.

**Algorithm 1** Stochastic Ferry Scheduling

```
1: $\forall i : c_i \leftarrow 0$
2: **while** $\max_i\{c_i\} < 1$ **do**
3:     $\forall i : c_i \leftarrow c_i + p_i$
4: **end while**
5: $k \leftarrow \mathrm{argmax}_i\{c_i\}$
6: Visit node $k$ and $c_k \leftarrow c_k - 1$
7: When ferry visit is finished, go to 2:
```

This solution is a zero-state-information solution since the decision of which node to visit is based only on prior information and does not depend on the current volume of data waiting at each node. An alternative formulation includes a signaling mechanism or a traffic accumulation estimator so that when the ferry is at the hub making its next visit decision it can consider the traffic waiting. Alternatively, if traffic is in the opposite direction from the central hub to the task nodes, then the ferry would have full information about how much traffic is waiting for each node. The next section considers these scenarios when the ferry has more state information when making its next visit decision.

## 5 Markov Decision Process Approach

Previous TSP-based ferry route planning approaches and the above stochastic model are at opposite ends of the spectrum of possible cycles through the network. While the former aims to visit all nodes in one cycle, the latter visits one node only, then returns to the hub. Our key idea in the following section is to exploit cycles which visit a subset of nodes before returning to the hub for data delivery.

We propose to view the planning as a multi-period decision problem. The decision maker, i.e., the ferry, can learn the dynamics of the system from interaction with its environment. Initially, the ferry learns by observing a model of the system. Thereafter, it updates its flight path while using this initial knowledge in the real environment.

### 5.1 Problem Formulation

A Markov Decision Process (MDP) is a mathematical framework for analyzing decision problems where outcomes are partly random and partly influenced by the decision maker [6]. It consists of an *agent* taking *actions* in an *environment*. Every action changes the environment's state and leads to a *reward* for the agent, which it uses for learning. The agent's goal is to maximize the sum of all rewards over time, called the *return*. The sequence of actions that maximizes the return is called an *optimal policy* for the agent to follow.

In this paper we are concerned with a single ferry serving multiple sensor nodes that wish to send sensor readings to a central monitoring station. The agent is the ferry, which aims at minimizing the overall average packet delay (i.e., latency) by visiting task nodes in an optimal sequence, i.e., the policy. It relies on observing the nodes' buffer states, i.e., the environment, and learns through the reward received when returning to the monitor station to deliver traffic. The ferry makes a new decision about which nodes to visit next every time it delivers traffic to the monitoring station. We are assuming packet generation at each node to continue indefinitely, which leads to an infinite-horizon MDP.

For small MDPs with full knowledge of the environment, the optimal policy can be found analytically, or by using dynamic programming techniques. If we consider larger state spaces it is more computationally feasible to think of the MDP as a problem of *learning from interaction to achieve a goal*. In that case Reinforcement Learning techniques can be used that can learn approximate solutions in reasonable time [1, 10]. Here, the agent does not have to know the dynamics of the process prior to taking actions. It learns about the process and the optimal policy while interacting directly with its environment.

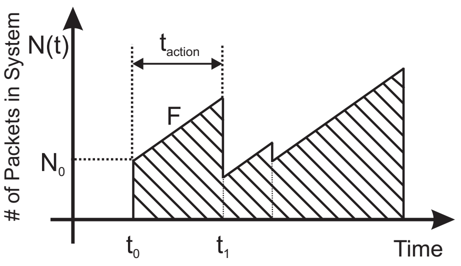

**Figure 3. Reward criterion: the shaded area under the curve represents packet delay.**

### 5.2 System Analysis and Agent Reward

Our goal is to minimize the average packet delay in the system. This can be achieved by minimizing the time each packet spends in the network. Fig. 3 shows the total number of packets in the network over time with one ferry as the transport medium. We assume constant traffic flow rates $f_i$ at each node, and the sum of all flow rates is denoted as $F$. At time $t_0$ there are $N_0$ packets in the network. Now the ferry chooses some action in which it starts at the hub, visits a specified sequence of nodes picking up their traffic, and returns to the hub. At the end of its tour at time $t_1$, the ferry delivers all the traffic to the monitor station, reducing the total number of packets in the network. Three ferry tours are shown. The area under the curve can be interpreted as the total time packets have spent in the network. If we reward the agent with the negative integral of N(t) over the action duration, it will try to minimize the number of packets in the system to maximize the return. In our infinite-horizon decision problem we discount the reward by a discount factor determined by $\beta$:

$$r(s,a) = -\int_{t_0}^{t_1} (Ft + N_0)e^{-\beta t}dt$$

Thus the agent has an incentive to visit nodes which have accumulated a large buffer of undelivered traffic. However, it also penalizes flying to a far-away node more than visiting a near-by node.

The set of possible ferry actions contains the ferry tours through all subsets of the nodes starting and ending at the hub, $\mathbb{A} = \{$A, B, C, AB, BA, AC, CA, ...$\}$. E.g., we denote a tour which visits nodes A, C, and D in this order then returns to the hub as action ACD. The number of actions increases quickly with the number of nodes in the network: $|\mathbb{A}| = \sum_{i=0}^{n-1} \frac{n!}{i!} < n!e$.

The state space comprises all tuples of traffic accumulated in each of the nodes, e.g., $\mathbb{S} = \{(b_A, b_B, b_C)\}$ for a

3-node network. The state space is well defined, since traffic generation and flight times are considered to be known in the model-based learning environment. The state transition function is given by

$$b_i(t+1) = \begin{cases} (t_a - t_i) f_i & \text{if } i \in action \\ b_i(t) + t_a f_i & \text{else,} \end{cases} \tag{4}$$

where $b_i$ is the traffic accumulated in node $i$, $t_a$ is the time to complete an action, and $t_i$ denotes the time it takes to reach node $i$ on the action path.

Optimal MDP policies are deterministic. So, we use deterministic policies that choose the next action with probability one given the current state.

### 5.3 Solving MDP with Reinforcement Learning

For solving the decision problem we have implemented reinforcement learning with the Temporal Difference (TD) method to approximate a state value function as described in [10].

The idea here is that each state has a value which reflects the benefit of being in that state. This value is defined as the expectation of the sum of the discounted future rewards obtained when starting in that state and following a fixed strategy thereafter:

$$V^\pi(s) = E_\pi \left\{ \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \middle| s_t = s \right\} \tag{5}$$

where $\gamma < 1$ is a discounting factor. An optimal strategy would therefore maximize the sum of rewards when starting in an arbitrary state and following that policy thereafter. Thus, the value of a state is dependent upon that policy. The above value function maps states to state values, and once the optimal value function is found, the optimal policy can be easily extracted using

$$\pi(s) = \underset{a}{\operatorname{argmax}} \left( r(s,a) + e^{-\beta t_a} V(s_{t+1}) \right) . \tag{6}$$

An approximation to (5) can be learned iteratively by observing the reward for taking actions in the environment [10]:

$$V_{t+1}(s) = V_t(s) + \alpha (r(s,a) + e^{-\beta t_a} V_t(s')) \tag{7}$$

where $\alpha$ denotes the learning rate and $t_a$ is the action duration. In the limit of $t \to \infty$ the state value function converges to the optimal values $V^*(s)$.

The agent only learns when permitted to explore different actions over time instead of always taking the greedy action using (6). Therefore, we use $\epsilon$-action selection during learning runs, i.e., we take a random action with probability $\epsilon$. For efficient learning, we let $\alpha, \epsilon \to 0$ over time, starting at a value of $0.3$ for both.

We consider a continuous state space as per (4). For learning purposes we use state aggregation to discretize the state space [10].

## 6 Experimental Algorithm Evaluation

### 6.1 Simulation Environment

We have implemented ferrying in the OPNET discrete event simulator to evaluate average packet delays in typical AUGNet data collection scenarios. We assume 3 to 5 stationary sensor nodes distributed on an area of 1.5km x 1.5km with the monitoring station (hub) located in the middle of the lower part of the area (see Fig. 4, top). Each node generates $f_i$ packets per second, ranging from 1 to 8 pkt/sec, and each packet is destined to the monitoring station where it is consumed. All node and ferry buffers are assumed large, so packets are never dropped. The ferry travels with a constant speed of $v$=20m/s and it visits the sequence of nodes determined by the path planning algorithm under evaluation. Radio communication range is assumed 100m in a disc around the transmitter with a data rate of 10Mbps. A simple ARQ protocol governs the data exchange between nodes and ferry. The ferry flies all the way to the node's location, giving it $T_c = 10\text{sec}$ of contact time. Thus a total of $B = 12.5$Mb of data can be transmitted. We chose a small packet size of $L$=100 bytes to represent a typical sensor measurement packet. Each evaluation is run for 2hrs of simulation time, and the end-to-end delays of all packets are measured, then averaged to yield a final test result.

### 6.2 Scenarios and Results

The scenarios in Fig. 4 represent networks of high and low traffic rate sensors. We assume a high rate sensor generates 8 times the traffic load as a low rate sensor. We apply four distinct algorithms to schedule ferry visits in these scenarios:

- RR: naïve round robin scheduling. The ferry visits each node in turn returning to the hub between visits.
- TSP: the shortest path through the network visiting all nodes then returning to the monitoring station.
- STO: the route determined by our stochastic model according to Algorithm 1.
- RL: the route learned by the ferry applying TD-V Reinforcement Learning according to (5) and (7).

The optimal routes for TSP and RL are displayed in Fig. 4. TSP routes have a simple pattern, e.g., a tour through nodes A, C, B from the collection station in scenario I, while RL can result in intricate cycles through the network which can considerably lower average packet delay. To better compare the performance of the algorithms, we define the *relative gain* as the percent reduction in average delay compared to the worst performing algorithm. Fig. 4 shows that average delay can be reduced by 24% to 46% by deploying learning agents. TSP still works well in fairly homogenous networks (IV), but RL clearly dominates in scenarios with high variation in flow rates or distances (I–III).

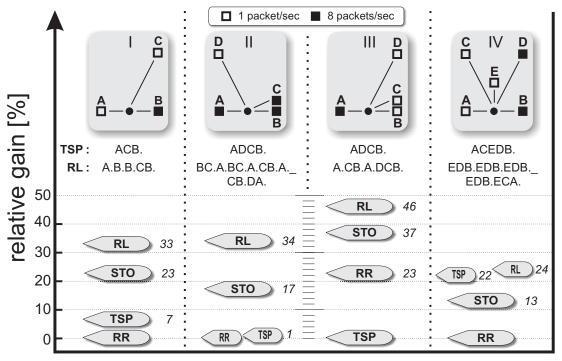

**Figure 4. Relative gains of four ferry tour scheduling algorithms for four scenarios.**

## 6.3 Issues with Models and Algorithm

We are using the standard temporal difference learning approach for finding a solution for the ferrying MDP. In our current problem formulation the computational complexity rises exponentially with the number of nodes in the network. Finding a route takes about 1 hour for 3 nodes, 6 hours for 4 nodes and already around 20 hours for 5 nodes. Applying RL methods that exploit structure in the state and action spaces will allow for a reduction of these computation times.

Throughout the paper we made strong assumptions about the system to make the problem tractable. In reality there might not be steady flow rates at the sensors and the wireless transmission rates and ferry buffers are finite. In these cases the model and solutions change but fit within the framework.

In practice there needs to be a signaling mechanism between nodes and ferry to notify buffer status and flow rates. A low-rate, long-distance channel could serve this purpose or the ferry could empirically learn flow rates while visiting nodes in case of slowly changing rates. Changing flow rates and node locations demand a change in the optimal ferry route over time. A ferry could evaluate the situation every time it returns to the hub and change its route. An extreme version with decision points at each nodes can make many-to-many communication possible.

An extension of the stochastic model to include multi-node actions will improve its performance, but a closed form solution would have to be derived. The improvement is bounded by the RL solution.

## 7 Conclusion and Future Work

Current solutions to the data ferry route planning problem are provably sub-optimal, as shown for solutions based on the TSP approach. By framing the problem as a Markov Decision Process, considering the ferry as a learning agent, and applying Reinforcement Learning as a solution strategy, we obtain solutions which result in lower average packet delay, do not rely on prior knowledge of the system dynamics, and which adapt to changing communication needs. Thus autonomic optimization of a ferry's path through the network is a viable alternative to conventional algorithmic approaches.

Our current framework only considers unidirectional flows from multiple task nodes to one sink and utilizes simple Reinforcement Learning techniques. Future work will extend the planning problem to general flows in the network, which can be solved by more sophisticated RL approaches like sparse sampling, $E^3$ [4], or PEGASUS for partially observable MDP (POMDP) [5].

## References

[1] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[2] T. X. Brown, S. Doshi, S. Jadhav, D. Henkel, and R.-G. Thekkekunnel. A full scale wireless ad hoc network test bed. In *Proc. of ISART'05*, NTIA Special Publications SP-05-418, pages 51–60, Mar 2005.

[3] K. Fall. A delay tolerant network architecture for challenged networks. In *Proceedings of ACM SIGCOMM*, pages 27–31. ACM, 2003.

[4] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning Journal*, 49(2), 2002.

[5] A. Y. Ng and M. Jordan. PEGASUS:A policy search method for large MDPs and POMDPs. In *Proc. 16th Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[6] S. M. Ross. *Introduction to stochastic dynamic programming*. New York : Academic Press, 1983.

[7] K. Sanzgiri and E. M. Belding-Royer. Leveraging mobility to improve quality of service in mobile networks. In *Proc. of MOBIQUITOUS'04*, pages 128–137, August 22-26 2004.

[8] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modelling and analysis of a three-tier architecture for sparse sensor networks. *Elsevier Ad Hoc Networks Journal*, (1):215–233, Sept. 2003.

[9] A. A. Somasundara, A. Ramamoorthy, and M. B. Srivastava. Mobile element scheduling with dynamic deadlines. *IEEE Transactions on Mobile Computing*, 6(4):395–410, 2007.

[10] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, March 1, 1998.

[11] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, Duke University, 2000.

[12] R. Viswanathan, J. T. Li, and M. C. Chuah. Message ferrying for constrained scenarios. In *Proc. of IEEE WoWMoM*, pages 487–489, Taormina, Italy, 13–16 June 2005.

[13] Z. Zhang and Z. Fei. Route design for multiple ferries in delay tolerant networks. In *Proc. of IEEE Wireless Communications and Networking Conference, WCNC'2007.*, pages 3460–3465, 11-15 March, 2007.

[14] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proc. of MobiHoc'04*, May 24-26 2004.