Deficit Round-Robin Based Message Ferry Routing

Ahmed Mansy, Mostafa Ammar and Ellen Zegura
School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332
{amansy, ammar, ewz}@cc.gatech.edu

***Abstract*—Message Ferrying is a mobility assisted scheme in which a special node, called a message ferry, is tasked with delivering data among a set of disconnected wireless nodes. One key challenge for such scheme is to design the ferry route in a way that improves certain network characteristics such as average data delivery delay or data loss ratio. Previous work has proposed solutions to the ferry route design problem in cases where a single or multiple message ferries were used, and in cases where nodes were mobile or stationary. That work optimized ferry travel time, rather than directly optimizing message delivery performance metrics. In this paper, we revisit the basic ferry route design problem for stationary nodes with the goal of providing a framework for optimizing delay performance. We start with a Markovian Decision Problem (MDP) formulation which produces the optimal ferry route that minimizes the average data delivery delay. While this formulation, in principle, enables optimal ferry route design, it is numerically intractable for moderate to large size problems. Solutions to small problems, however, yield insight into the properties of optimal ferry routes. These insights, in turn, lead us to propose a ferry route design algorithm that takes advantage of the similarity between our problem and the link scheduling problem in traditional networks. Our algorithm is inspired by the Deficit Round Robin (DRR) algorithm which has provable properties for delay optimization when applied to link scheduling. Using simulations we show that our DRR-based algorithm produces ferry routes that are close to optimal when compared to the MDP-derived solutions for small problems. Our results also show that our algorithm produces ferry routes for moderate to large problems that significantly out perform existing solutions.**

I. INTRODUCTION

Message ferrying is a networking paradigm that was developed to perform data routing in intermittently connected or completely disconnected wireless networks [1]–[3]. In this scheme, a set of mobile nodes called *message ferries* take responsibility for carrying messages among disconnected nodes (see figure 1). This approach is very attractive especially in cases when nodes are sparsely deployed with limited contact opportunities. It is also the only feasible connectivity approach when nodes are stationary and sparsely deployed.

A key challenge when using the Message Ferrying scheme is the design of the route of the *message ferries* in a manner that achieves desirable performance characteristics such as minimizing the average end-to-end delay or the packet drop rate. Previous work proposed solutions to this problem in cases where single or multiple ferries were used, and also in cases where the nodes were mobile or stationary. A special case of the ferry route design problem was studied in [4]. In this case all the nodes were assumed to be stationary and only one *message ferry* was used. The authors in [4] proved that the ferry route design for this case is NP-hard by reducing

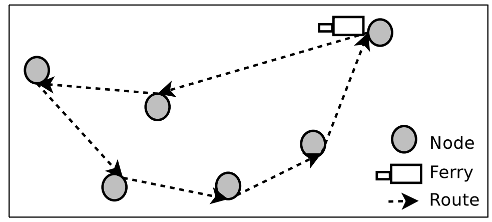


Fig. 1. Message ferry routing scheme

the Euclidean Traveling Salesman Problem (TSP) to the ferry route design problem. They also proposed a two-phase solution to the problem. In the first phase they generated an initial route by using an algorithm from the well studied TSP literature. In order to meet certain bandwidth requirements, they applied a second phase where they extended the amount of time the ferry spends at each node by solving a linear optimization problem. This solution and other solutions like [5], [6] are commonly based on graph traversal algorithms and mainly approach the problem as a variation of the TSP. Although this approach is appealing because of the richness of the TSP literature, it does not actually allow for optimizing specific network characteristics such as the average data delivery delay. This is mainly because this approach does not take into consideration the non-uniformity of distances and traffic rates between different nodes in the network. For this reason, we see the need for developing a new ferry route design algorithm that has the capability of optimizing given performance metrics.

To this end, we revisit in this paper the basic problem of ferry route design for stationary sparsely deployed wireless nodes which was originally studied in [4]. Our goal is to develop a scheduling algorithm geared towards minimizing end-to-end message delays. First, we formulate the problem as a Markovian Decision Problem (MDP), and then solve the MDP to get a route for the ferry that minimizes the average end-to-end delay. One major problem we encountered with using MDPs was the exponential growth of the state space with the size of the network. Because of this we were able to get optimal ferry route solution only for small size networks. However, studying these solutions provided us with insights regarding the properties of the structure of optimal ferry routes. Guided by these insights, we next propose a ferry route design algorithm that, takes advantage of the similarity between our problem and the link scheduling problem in traditional networks. As a preview, the ferry acts as the shared resource needed for delivery, just as the link is the shared resource in link scheduling. Our algorithm is based on the Deficit

Round Robin (DRR) algorithm which has provable desirable throughput and delay properties in traditional networks.

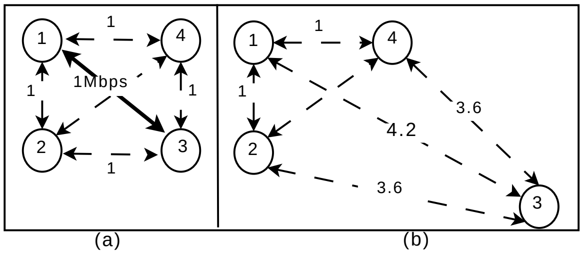


Fig. 2. Example of two different networks, distance in kilometer

To illustrate the difference between our approach and previous solutions we present the following example. Figure 2 shows two different geographical configurations for a four-node network. In network Figure 2(a), the nodes form a square whose side is is 1KM long. Each node sends data to each other node at a rate of 10Kbps, except for traffic going between nodes 1 and 3 which is 1Mbps in both directions. In this case, intuitively we think that in order to minimize the average data delivery delay, the optimal route for the ferry should visit nodes one and three more often than nodes two and four. However, a TSP based solution such as [2] will produce a cyclic route of the form $\{1, 2, 3, 4\}$. Since the new technique developed here generally does not produce cyclic routes, we use relative frequencies of node visits to describe the ferry route. In this example, our technique will produce a route that will have the following relative node visit frequencies $\{2 : 1 : 2 : 1\}$, which means that on average node 1 gets visited twice as much as node 2. This route gives about 55% improvement in the mean data delivery time over the TSP-based route.

Another case is illustrated in Figure 2(b), where three of the four nodes have the same placement as the previous example, while the fourth node (node 3) is placed significantly farther away, see distances in figure. The traffic among all nodes is equal to 100Kbps. Again one should expect that the optimal route in this case should visit node 3 less often than the other three nodes because as the ferry travels to node 3 it increases the delay for other traffic. A TSP based solution will again produce a cyclic route of the form $\{1, 2, 3, 4\}$. Our technique, however, produces a route with the following relative visit frequencies $\{3 : 2 : 1 : 2\}$ which gives about 76% improvement in average data delivery delay over the TSP solution when tested through simulations.[1]

Message ferrying was first introduced in [1], [4] as a network paradigm for data delivery in sparse MANETs. The same authors proposed an algorithm for controlling the mobility of multiple message ferries in [2]. An algorithm for ferry route design with mobile nodes in sparse DTNs was proposed in [3]. Different than previous ferry algorithms, this algorithm assumed no coordination between the nodes and the ferry. As mentioned previously, the most relevant work to ours is [2], in which the authors presented a solution to the same problem of ferry route design with stationary nodes and a single ferry.

Similar to message ferries, Data MULEs were introduced in [7] as wireless mobile agents that move randomly on a grid to facilitate connectivity with sparsely deployed stationary sensor nodes. A load balancing algorithm for distributing sensor nodes over multiple data mules was introduced in [5]. Sugihara et al. introduced an algorithm in [6] for finding an optimal path for a data mule to minimize data delivery latency. Although this problem is similar to our problem, the proposed algorithm was designed specifically for sensor nodes environments where power efficiency is significantly important, and nodes are more dense unlike the sparse environment we are considering in this paper.

The rest of the paper is organized as follows. The MDP formulation for the ferry route design problem is presented in section II. In section III we present our DRR-based route design framework. We present some simulation results in section IV and then we conclude the paper in section V.

II. MDP FORMULATION

In this section we formulate a simplified instance of the ferry route design problem as a Markovian Decision Process and discuss our experience in producing optimal ferry schedules with this formulation. The instance is simplified in order to render the state space manageable. Furthermore, an MDP formulation requires us to make Markovian and exponential assumptions that may be unrealistic. Recall, however, that our goal here is to learn features of optimal ferry routes from solutions we obtain from the MDP formulation. We then apply these insights to the construction of a heuristic which is applicable to systems without constraining Markovian assumptions.

The system we model consists of a set of N stationary wireless nodes. Each node has a wireless device with a certain connectivity range R. These nodes are assumed to be sparsely distributed over a certain area and they do not fall in the communication range of each other. It is also assumed that any of these nodes can be a data source or destination. In addition, there is a single message ferry that can be moved on demand by a controlling party. Our objective is to develop a strategy that can control the movement of the ferry in such a way that we can achieve low end-to-end delay for traffic among the sparsely deployed nodes.

A. Ferry movement

The message ferry is assumed to move among nodes pausing at each node for a period of time to allow data exchange. Hence, the position of the ferry can be described as either i when the ferry is visiting node i, or $i \to j$ when then ferry is in transition from node i to node j, where $i, j \in (1, \ldots, N)$. When a message ferry visits node i, the time the ferry pauses there before moving to a new position is assumed to follow an

[1]It may be possible to reformulate the TSP problem so that it can capture non-uniform traffic considerations or non-uniform node locations, for example by artificially making nodes appear farther away when they have lower traffic loads. We have not explored this option, preferring to deal with variability by our more direct method.

exponential distribution with mean $1/\mu_i$. In addition, the time taken by the ferry to travel from node i to node j is assumed to be exponentially distributed with mean $1/\eta_{ij}$.

B. Data delivery

We assume that data arrival follows a Poisson process with rate λ. The probability that a message arrives at node i is p_i such that $\sum_{i=1}^{N} p_i = 1$. The probability that a message is destined to node j given that it arrived at node i is p_{ij}, $i, j \in (1, 2, \ldots, N)$, where $\sum_j p_{ij} = 1$. We also assume that each node has a buffer of a limited size K (messages). New messages arriving at a full buffer will be dropped. The message ferry buffer is assumed to be of a finite capacity. In our example we assume the ferry can hold up to KN^2 messages.

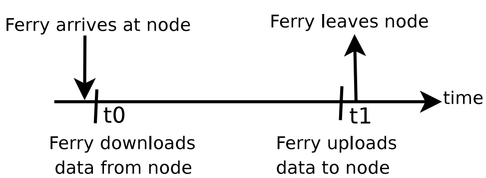


Fig. 3. Data delivery between ferry/node

When a ferry comes in contact with node i, we assume that data exchange occurs according to the order in figure 3. At the beginning of the contact duration, the ferry starts downloading all messages in the node buffer that have been generated prior to time t_0. We follow this model to guarantee that the ferry will stay only a finite period of time at each node. Then, towards the end of the contact period, the ferry starts uploading data in its buffers that are destined to node i.

C. System state

In order to model the above described system as a Markovian process we define the following system state that describes the system at any point in time, $[(B_1, \ldots, B_N), F, L]$ where:

- $B_i = (b_{i1}, \ldots, b_{ij}, \ldots, b_{iN}), i \neq j$, where b_{ij} represents the number of messages in the buffer of node i destined to nodes j.
- $F = (f_1, \ldots, f_N)$ where f_j is the number of messages in the ferry buffer that are destined to node j.
- L is the position of the ferry, and it can take any value of the set $\{w, w \rightarrow g : w, g \in (1, \ldots, N)$ and $w \neq g\}$.

State transitions in this system occur due to one of the following events:

- New message is generated at node i destined to node j. If the system is in state: $[(B_1, \ldots, (b_{i1}, \ldots, b_{ij}, \ldots, b_{iN}), \ldots B_N), F, L]$ then the system will enter state $[(B_1, \ldots, (b_{i1}, \ldots, b_{ij} + 1, \ldots, b_{iN}), \ldots B_N), F, L]$ with transition rate $p_i p_{ij} \lambda$.
- Ferry in transition from node i to node j (in state $i \rightarrow j$) reaches to its destination j. This event will change the system state from:
  $[(\ldots, (b_{j1}, \ldots, b_{jN}), \ldots), (\ldots, f_j, \ldots), i \rightarrow j]$ to
  $$[(\quad \ldots, (b_{j1} = 0, \ldots, b_{jN} = 0), \ldots), \\ (f_1 + b_{j1}, \ldots, f_j, \ldots, f_N + b_{jN}), j]$$
  and the transition rate will be η_{ij}.
- Ferry visiting node i starts moving towards node j. This event will change the system state from:
  $[(B_1, \ldots, B_N), (f_1, \ldots, f_i, \ldots, f_N), i]$ to
  $[(B_1, \ldots, B_N), (f_1, \ldots, f_i = 0, \ldots, f_N), i \rightarrow j]$ with a transition rate of μ_i.

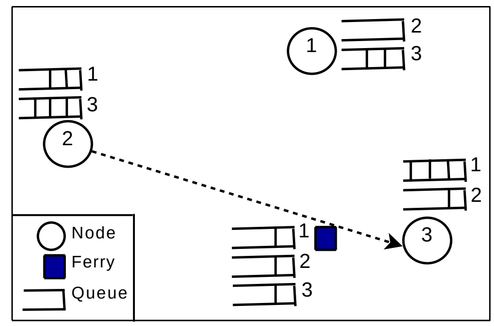


Fig. 4. Example of system state

Figure 4 shows a simple network of three nodes and a single ferry. Each node has two queues for outgoing traffic; a queue for each destination. Likewise, the ferry has a separate queue for each destination. Queues are labeled with their destinations. As can be seen in the figure, the ferry is currently in transition from node 2 to node 3. The current system state can be written as

$$\left[\big((0,2),(2,3),(3,1)\big),(1,1,1),2 \rightarrow 3\right]$$

When the ferry reaches node 3, the ferry downloads all messages from queues of node 3, and the new system state becomes

$$\left[\big((0,2),(2,3),(0,0)\big),(4,2,1),3\right]$$

We now define the system decision process that specifies for each feasible system state what actions could be taken. In our case, the action simply means "where should the ferry go next?". The decision could be that the ferry should stay in its current position, or it should move from its current position i to a different position j. The set of decisions for all feasible system states forms a scheduling policy. The decision process can be written in the following form

$$d[(B_1, .., B_N), F, L] = \begin{cases} L^F = i \\ L^F = i \rightarrow j,\ j \in (1, .., N) \ , \ i \neq j. \end{cases}$$

D. MDP definition

We formulate the above described problem as a Markov decision process and represent the cost function in terms of the delivery time. Generally, MDPs can have two types of costs; state transition cost and state occupancy cost. In our model, the transition cost is equal to zero. The cost of occupying a state has a rate of n per time unit, where n is the total number of messages waiting in all the buffers in the system, or $n = \sum_{i,j} b_{ij} + \sum_w f_w$. If $C(T)$ is the total cost incurred by the system up to time T and $n(t)$ is cost rate at time instant t, then for a given scheduling policy we have $C(T) = \int_0^T n(t)dt$ and the average cost per unit time of using a certain policy becomes

$$C = \lim_{T \to \infty} \frac{C(T)}{T}$$

Since $n(t)$ represents the number of messages in the system at time t, it is easy to see that C is the mean number of messages in the system. We also know that the number of messages in the system is related to the mean message delay from Little's theorem. More specifically, minimizing the system average cost per unit time will minimize the mean delay. We use the *policy iteration* algorithm developed by Howard [8] to solve the resulting MDP problem, which results in an optimum policy that minimizes the mean system delay.

E. Optimal policy properties

We were able to solve the MDP and get the optimal ferry route for small size networks (3 or 4 nodes) with limited size buffers. However, even with such small networks we still had a state space explosion problem. Hence, we decided to use an alternative state space description that aggregates some of the details in the original system state. We tried multiple alternative state descriptions and decided to use *aggregated node buffers*. In this case, the buffer of each node is represented by a single number which is the total number of messages in that node's buffer. This state description does not hide much of the details of the original state and reduces the number of system states significantly in the same time.

An optimal route can be thought of as the set of pairs ($state$, $decision$), where $state$ represents the state of the system as described in II-C and $decision$ is the next node to visit. Basically, this pair means that if the system is in state $state$, the ferry should move to node $decision$. When applying optimal routes to ferry movement, it does not necessarily result in cyclic routes. Hence, we choose to characterize optimal ferry routes with relative frequencies of node visits.

We obtained the optimal ferry routes for many small networks that have different distance and traffic configurations (including networks in Figure 2). Studying relative frequencies of node visits for these routes, we came to the following observations:

- Nodes that have higher traffic rates (in, out, or both) tend to have higher relative frequency than nodes with lower traffic rates. This observation means that a node that sends or receives more data gets to be visited more frequently than other nodes.
- Optimal routes tend to favor travelling shorter distances, meaning that nodes that are located relatively far from other nodes usually have lower relative frequencies. On the other hand, nodes that are closer to other nodes have higher relative frequencies.

These observations about the relationship between relative frequencies of node visits and both distances and traffic rates were reminiscent of the fair link scheduling problem. In this problem, it is required that packets from different flows that share the same link be scheduled in a way that achieves fairness among flows. Deficit Round-Robin (DRR) [9] is an algorithm that produces fair link schedules. With DRR scheduling, flows with large packets tend to be serviced less frequently than flows with small packets. We explore this analogy a little further in the next section where we use it to to develop the DRR-based ferry scheduling heuristic.

III. DRR-Based Ferry Route Scheduling

One key observation in this paper is that the ferry route design problem is analogous with the link scheduling problem in traditional networks. This, in principle, allows us to use any link scheduling algorithm to solve our problem [10]–[12]. We choose to use the Deficit Round-Robin [9] algorithm because of its simplicity and desirable provable throughput and delay properties. In this section we describe the ferry route design process. We start by briefly describing the DRR operation in traditional networks, then we show the analogy between the link scheduling problem and the ferry route problem, and finally we describe our algorithm in detail.

A. Deficit Round-Robin Scheduling and Message Ferry Routing

Deficit Round Robin (DRR) is a scheme that allows fair scheduling of multiple flows over a single shared link. It is designed to deal specifically with the case of flows with unequal and unpredictable packet lengths and does not require a-priori knowledge of average packet sizes for flows.

The details of DRR can be found in [9]. A router that implements DRR has a separate queue for each incoming flow. For each queue there is a deficit counter that keeps track of the amount of service granted to that flow. Before the DRR algorithm starts, all deficit counters are initialized to zero, and in each round a flow is allocated a quantum Q_i worth of bits. In the simplest case $Q_i = Q_j$ for all flows i, j, which will result in equal shares of the link bandwidth for all flows. However, each flow can be assigned a different relative bandwidth by having different values of Q_i. Queue i's allocation is accumulated in a deficit counter DC_i. When a queue is visited one or more packets are serviced with the condition that their total size is less than the accumulated deficit counter. If B_i bytes are serviced then the deficit counter is decremented by B_i bytes and the scheduler moves on to the next queue.

We consider using a DRR-based scheduling heuristic as the basis of a message ferry route design that embodies the insights we learned from solutions to our MDP formulation. It is straightforward to see an analogy between link scheduling using DRR and message ferry route design. Observe that the ferry itself represents a resource (analogous to a link) shared among a set of stationary nodes (analogous to flows). The distance between the current location of the ferry and a particular node is analogous to the size of the packet waiting in a flow's queue – the distance should be traversed only if the node has accumulated enough credit. A ferry visit to a node can be considered in this analogy equivalent to serving a packet from a certain flow.

To illustrate this analogy, we apply it to the network in figure 2(a). In this case we have four "flows" labeled 1 through 4, one representing each node. If we assume the ferry to be currently at node 1, it means that "flow" 1 has just been served and the DRR algorithm will have to decide which "flow" to serve next. That also means that packet lengths coming from flows (nodes) $2, 3, 4$ are represented by their distances from node 1 which are $1, \sqrt{2}, 1$ respectively.

We emphasize that this is only an analogy that inspires our heuristic and is not a formal mapping between the link scheduling problem and the message ferry route problem. It is, nevertheless, useful as it provides guidance for the construction of an efficient heuristic. Furthermore, there are enough degrees of freedom in the DRR scheduling framework that allows us to incorporate ferry route design features that were learned from our MDP formulation.

B. Ferry Route Scheduling Heuristic

Figure 5 shows the high level pseudo-code for the DRR-based route scheduling process. The details of the subroutines are explained in text. The function `getNextLocation` is called by the ferry each time it concludes a visit to a node and needs to find its next destination.The operation of our algorithm can be explained as follows. As in traditional DRR, we maintain a service quantum counter for each stationary node i, we call it $credit[i]$. Initially, $credit[i]$ is set to zero for all i. Each time the ferry needs to make a decision it computes a set of nodes that can potentially be the next destination, we call this set the $candidate\_list$. Function `getDRRCandidates` in Figure 6 computes this list. It simply returns all nodes i that satisfy the condition $credit[i] \geq d(curr\_loc, i)$, where the latter is the distance between node i and the current position of the ferry. The ferry then applies a tie breaking algorithm to select a destination from the $candidate\_list$. After that, the credit value of the selected node i, $credit[i]$, is reset to zero. The tie breaking algorithm is implemented in function `tieBreak` which will be explained in detail later.

When the function `getDRRCandidates` returns an empty candidate set, it means we reached the end of a round and the beginning of a new round in the algorithm operation. At this point, the credit counters of all nodes, $credit[i]$, have to be updated. This can be seen in Figure 5 in the function call `updateCredits.` The details of the approach we follow to update credit counters will be explained shortly.

Looking at the algorithm pseudo code, it is clear that its operation can be controlled by the two functions **updateCredits** and **tieBreak**. Using different approaches in these two functions can change the performance of the algorithm significantly. We have considered many possible implementations for these two functions. Below we present the functions that

```
getNextLocation (credits):
if getDRRCandidates(curr_loc) ≠ ∅ then
   /* Continue with the current round */
   candidate_list = getDRRCandidates(curr_loc);
   next_location = tieBreak(candidate_list);
   credit[next_location] = 0;
   return next_location;
else
   /* Start a new round */
   updateCredits(credits);
   getNextLocation(credits);
end if
```

Fig. 5. getNextLocation function, it gets called each time the ferry makes a decision to select a new destination

```
getDRRCandidates (curr_loc):
candidate_list = {}
for all node i ≠ curr_loc do
   if credit[i] ≥ distance(curr_loc, i) then
      candidate_list += i;
   end if
end for
```

Fig. 6. getDRRCandidates function, it computes the $candidate\_list$

we finally settled on for our heuristic, and the rationale behind choosing them.

updateCredits. This function is called at the beginning of each new round k. It generates the quantum service values Q_i^k for each node i and updates each node's credit counter $credit[i]$ by adding Q_i^k to the counter's old value. We compute the values Q_i^k as follows. Let r_{ij} be the traffic rate going from node i to node j, and s_i be the total aggregated traffic rate originating from and destined to node i, then s_i can be written as

$$s_i = \sum_{j=1}^{N} \left(r_{ij} + r_{ji}\right)$$

We also define $s = \min_i(s_i)$ as the minimum aggregated data rate for all the nodes. Now, let d_{ij} be the distance between nodes i and j, and let $d_i = \sum_{i=1}^{N} d_{ij}/(N-1)$ be the average distance between node i and all other nodes. If we assume that the ferry was at node i at the beginning of round k, then we compute the quantum value Q_j^k for node j using the following formula

$$Q_j^k = d_i \times \sqrt{\frac{s_j}{s}}$$

The reason behind making the credit quantum directly proportional to the average distance is to ensure a candidate set that is big enough to give the ferry flexibility in choosing its next destination. This makes the credit increase consistent whether a node is close or far while favoring nodes whose distance from the current node is below average.

We also used the square root in the above equation because of the resemblance we found between the ferry scheduling problem and other problems that proved to have optimal solutions when using square roots of similar ratios. For example, in [13] it was shown that in a teletext broadcast system, the optimum mean response time can be obtained if the ratio between the probabilities of selecting any two different pages for transmission was equal to the square root of the ratio of the probability of these pages getting requested by users. In a different context, a similar result was found in optimal replication strategies in unstructured peer-to-peer networks. In [14], it was proved that for any two data items, the optimal ratio of allocation is the square root of the ratio of the query rates, where allocation of an item means the number of replicas of the item in the peer-to-peer network.

tieBreak. This function is called by the ferry to select one of the candidate nodes as the next destination. To make this decision, we go through multiple levels of filtering, reducing the size of the candidates set after each level. In the first filter, we select the w least recently visited nodes by the ferry. The ferry can easily do that by keeping track of the last time it visited each node. In our implementation we chose $w = N_{candidates}/2$ where $N_{candidates}$ is the size of the candidates set. After applying the first filter, if the candidates set still has more than one node then we apply the second filter where we select half of the nodes that are closest to the current position of the ferry. In the third stage, if needed, we select the node with the maximum out traffic rate r_i where $r_i = \sum_{j=1}^{N} r_{ij}$. If multiple nodes have the same maximum value R_i we select the one that has more messages in the ferry buffer, and finally if we still have a tie we randomly choose one of the nodes.

Using these levels of filtering gives us significant flexibility in choosing the node from the candidate set that satisfies multiple properties at the same time. For example, insight gained from our MDP solution indicates preference for the selection of least recently visited nodes, and this is represented by the first filter. The same applies for the next filters where we chose the nodes that are closer to the ferry then the nodes with higher traffic rates, and so on. We have tried to change the order of these filters and even tried other filters, but the above described set of filters gave us the best results when tested through simulations.

IV. Simulation results

In this section we evaluate the performance of our DRR-based ferry route design algorithm and compare it to the performance of the TSP-based solution in [2], [4]. We use the mean and standard deviation of data delivery delay as our performance metrics. We perform the evaluation under different node placement and traffic patterns. For node placement, we use both random uniform and random clustered node allocations. We also evaluate the performance under uniform and non-uniform traffic distributions.

A. Simulation setup

We used the ONE simulator [15] as our simulation environment. We use a network of 20 stationary nodes distributed in a $4km \times 4km$ area. There is one additional mobile node which acts as a message ferry. Every node has a wireless interface with a radio range of 250 meters and a bandwidth of $10Mbps$. The ferry is assumed to move in straight lines and at constant speed of 15 m/sec. All stationary nodes are strictly configured to communicate only with the ferry, which means if two stationary nodes fall in radio range of each other they do not communicate or exchange data. Each point in the plots in this section was averaged over 36 simulation runs. In these 36 runs we use 9 different node placements generated with different random seeds, and we use each of these placements with 4 different traffic traces again generated with different random number seeds.

As mentioned earlier, nodes can be either uniformly placed in the area or clustered. Random uniform allocation is simply done by selecting both x and y coordinates of any node as a uniform random variable in the interval $(0, 4000)$. Clusters are formed in the following manner. Assume we have M clusters, then we first select M random cluster centers uniformly distributed over the area. We do that in the same way we did with the uniform random nodes. Then, for each of the N nodes, we first assign a node randomly to any of the clusters and then compute its x and y coordinates using the formula $x = X_c + Z * 4000/\sqrt{M * N}$ and $y = Y_c + Z * 4000/\sqrt{M * N}$ where $Z \sim N(0, 1)$, and (X_c, Y_c) are the coordinates of the cluster center.

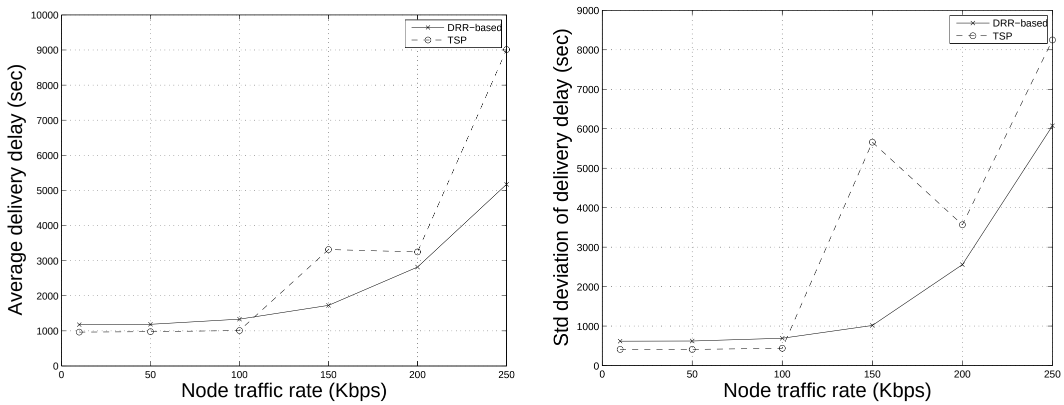
Message arrivals follow a Poisson process in our simulations. If a flow sends traffic at a rate of w bps then the corresponding Poisson arrivals flow will have a rate of w/msg_size, where msg_size is the message size in bits. Since we are interested in measuring the mean and the standard deviation of the data delivery delay, we assume that the ferry and all nodes have infinite buffers and that messages have an infinite TTL (Time-To-Live).

We use both uniform and non-uniform traffic distributions. In uniform traffic, every node sends traffic to every other node with the same data rate. If node i is sending data at a total rate of r Kbps, then the traffic rate from node i to any other node j will be $r/(N-1)$ Kbps. In non-uniform traffic distribution, certain flows have higher data rates than others. We randomly select 10% of the flows as high data rate flows and keep the rest of the flows sending at rates similar to the uniform traffic case. In all the experiments below, high data rate flows have rate of 500Kbps. Note that a flow is defined here by a $(source, destination)$ pair.

B. Uniform node allocation

We start our evaluation by the uniform node allocation case. We compare the performance of the DRR-based algorithm with the TSP-based solution in [2]. We perform this study using uniform and non-uniform traffic distributions.

Uniform traffic. We vary the total traffic rate going out from each node, from 10 kbps to 250 kbps. Figures 7(a) and 7(b) respectively show both the average and standard deviation of data delivery delay versus node traffic rate. Our first observa-



(a) Mean delay (b) Standard deviation of delay

Fig. 7. Uniformly placed nodes, uniform Poisson traffic

tion is that our solution and the TSP-based solution have very similar performance at low data rates (upto 100 kbps), with the TSP-based solution being slightly better. When node traffic increases to 150 Kbps, we observe a significant improvement in our scheme over the TSP solution. The improvement is approximately 80% in the mean delivery delay and 480% in the delay standard deviation. Our explanation for that behavior is as following. In the case of uniform traffic, queues at nodes fill up at the same rate with messages waiting to be sent. The TSP-based solution serves these queues at the same rate because the ferry visits a node only once in a cycle. Our solution, on the other hand, takes advantage of the non-uniformity in distances between node pairs and favors nodes that are relatively closer to other nodes.

At node traffic rate of 200 Kbps, the TSP-based solution performance improves significantly. In order to explain the reason for that, recall that the solution in [2] is a 2-phase solution. In the first one a TSP route is obtained, and in the second phase the periods of time the ferry spends at each node can get extended to meet certain bandwidth requirements. Before 200 Kbps, phase-2 of the TSP solution gives zero extended periods, however, at 200 Kbps or above it gives positive extended periods which improves the performance of the TSP solution. However, at 250 Kbps despite having positive extended periods, our solution performs about 80% better than the TSP solution.

Non-uniform traffic. Figures 8(a) and 8(b) show the average and standard deviation of the delivery delay plotted against node traffic rate for this case. Note that the x-axis represents the traffic rate of a node after excluding the high traffic rate flows. Looking at Figure 8(a), we can see that our scheme always performs about 30% better than the TSP-based scheme. The only exception is at 100 Kbps where the two schemes perform very close to each other. The reason is that at 100 Kbps the TSP-based scheme starts getting positive extended periods of time for the ferry which improves both the mean and standard deviation of the delay (see Figure 8(b)).

C. Clustered node allocation

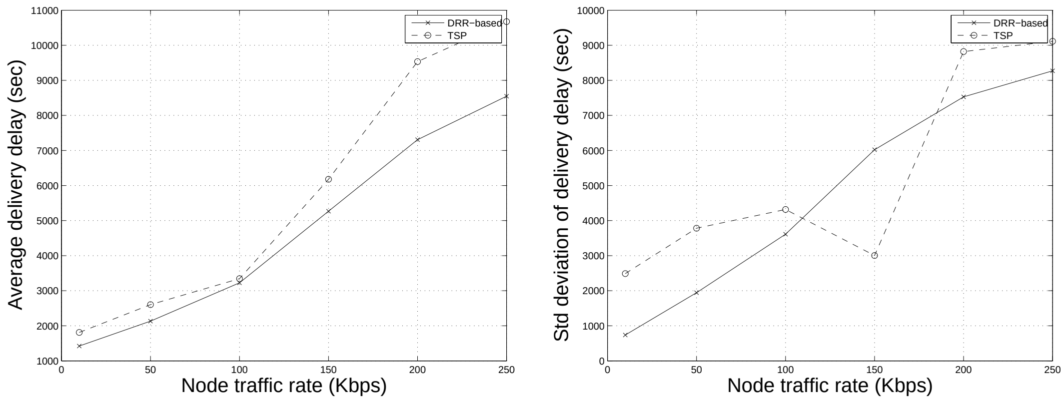
In this section we evaluate our algorithm with the nodes being clustered into 4 clusters. Clusters are randomly created as explained earlier in the section. Evaluation is performed for both uniform and non-uniform traffic.

Uniform traffic. Figure 9 shows the results for the 4-clusters case with uniform traffic. We can observe that at lower data rates ($10 - 50$ Kbps), similar to the uniform node allocation case, our algorithm and the TSP-based solution have similar performance. After that, our algorithm improves significantly over the TSP solution. For example, at 150Kbps and 200Kbps we get about 150% and 200% better delay respectively. Improvement in the standard deviation is even larger, we can see in Figure 9(b) that in our case the standard deviation is a low constant until we reach traffic rate of 150 Kbps then it slightly increases after that. Looking closer at the ferry routes with clustered nodes, we observe the following.

- The ferry goes more frequently to clusters with more nodes. In such clusters, the ferry will collect and deliver more data specially with uniform traffic. This helps reduce the average data delivery delay for the whole network.
- The ferry goes less frequently to clusters that are relatively far from the other clusters. This behavior is similar to our observation with single nodes in the uniform node allocation case.

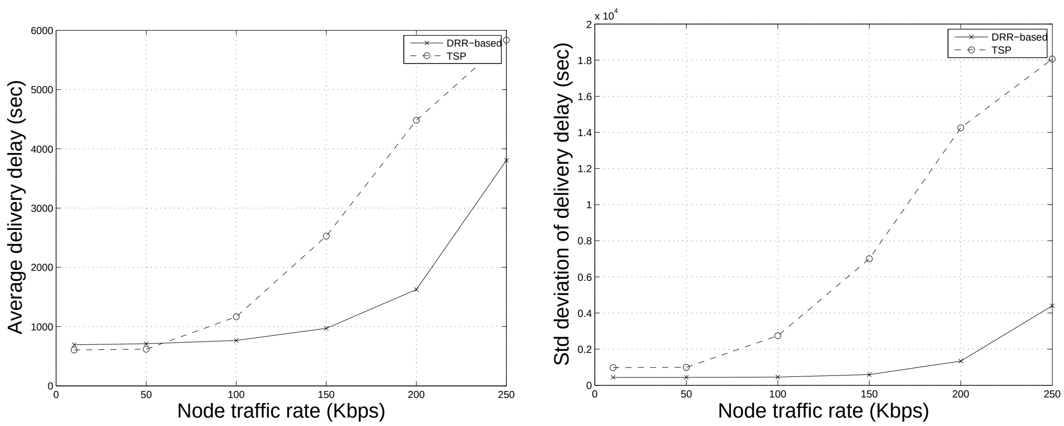
Non-uniform traffic. In this case, we do not differentiate between intra-cluster and inter-cluster flows when choosing the high rate ones, instead they are selected randomly from all flows in the network.

Figure 10 shows the results for the 4-clusters case with non-uniform traffic. Our algorithm always outperforms the TSP-based solution. For low to moderate traffic rates ($10 - 150$Kbps), our algorithm achieves about $80\% - 150\%$ lower average delay than the TSP-based approach. At higher rates ($200 - 250$Kbps) the improvement in delay becomes lower, however, standard deviation in our case is still about 160%



(a) Mean delay (b) Standard deviation of delay

Fig. 8. Uniformly placed nodes, non-uniform Poisson traffic



(a) Mean delay (b) Standard deviation of delay

Fig. 9. Clustered nodes (4 clusters), uniform Poisson traffic

lower than the TSP-based case.

D. Weighted DRR-based ferry routing

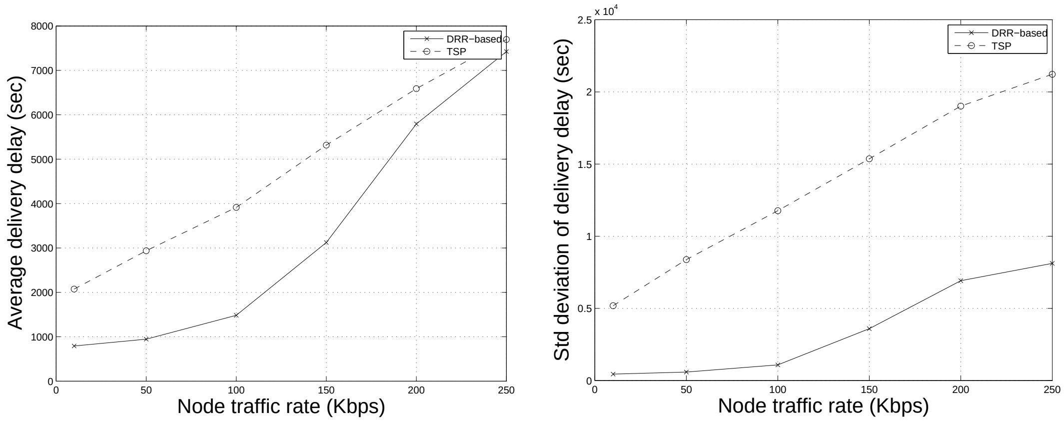
In this section we explore the possibility of using our DRR-based routing algorithm to impose specific user preferences on certain flows in the network. Basically, we ask the following question: can we use our algorithm to give preference to reducing the mean delivery delay of certain flows in the network more than others? This can be very beneficial if certain flows contain important information that needs to be delivered faster than data coming from other flows of lower priorities.

In order to give preference to reducing the mean delay of any flow over others, the end nodes of that flow should be visited more often than with the normal algorithm application. As mentioned earlier in section III-B, the operation of our algorithm is governed by the functions `updateCredits` and `tieBreak`. In order to increase the probability of choosing either of the two ends of the flow of preference, we may need to modify both functions. Increasing the amount of service quantum granted to nodes that are members (senders or receivers) of preferred flows will increase the likelihood they will be selected as potential candidates when the ferry is deciding about its next destination. However, without changing the tie breaking logic, this is not enough to guarantee a higher chance of getting actually selected as the next destination. To achieve that, the `tieBreak` function should be more sensitive to these preferred nodes such that if the $candidate\_list$ contains any nodes that belong to preferred flows, the tie breaking logic should be biased towards selecting one of these nodes.

TABLE I
DRR VS WEIGHTED-DRR RESULTS

Scheme	Mean delay	mean $\{1-3\}$ delay	$1:2:3:4$
DRR	556	556	$1:1:1:1$
Weighted-DRR	722	473	$2:1:2:1$

We give a simple example to illustrate the effect of using these changes on both the resulting ferry route and the mean delivery delay. Consider the node configuration in Figure 2(a), but assume all traffic flows are equal to 100 Kbps. We select the flow between nodes $\{1,3\}$ as the preferred one. In this example, we modified the functions `updateCredits` and `tieBreak` as follows. In `updateCredits`, at the



(a) Mean delay (b) Standard deviation of delay

Fig. 10. Clustered nodes (4 clusters), non-uniform Poisson traffic

beginning of a new round, we double the service quantum granted to nodes $1, 3$. In `tieBreak`, after applying the first tie breaking filter (see section III-B), if we find either of nodes $1, 3$ or both in the $candidate\_list$, we randomly select one of them as the next destination. Table I summarizes the results with and without using the weighted DRR approach. Column 4 represents the relative frequencies of visiting nodes $1, 2, 3, 4$ in the resulting ferry route. We observe that using the weighted DRR scheme improved the mean delivery delay for the traffic between nodes $1, 3$ by about 17.5%. However, in the same time, the mean delay time for other flows increased about 30%.

We believe that improving preferred flows mean delay can be done by the weighted DRR scheme. However, this problem needs a more careful and detailed study to develop more sophisticated methods that can be generally applied with more confidence in their results. We leave this as future work.

V. Concluding Remarks

In this paper we revisit the problem of message ferry route design in the case of stationary nodes and a single ferry. Our goal is to develop message ferry scheduling techniques that are more specifically targeted at optimizing message delay and that are able to deal with non-uniformity of traffic and distance among nodes. We start by formulating the problem as a Markovian Decision Problem (MDP) that minimizes the average data delivery delay.

In principle, this MDP formulation should be capable of providing our desired solution. However, due to the exponential growth of the MDP state space, we were able to solve the MDP and get the optimal routes only for small problems.

This led us to adjust our approach and seek a more heuristic solution. Consequently, and driven by both the insights we learned from optimal routes of small problems and the similarity between our problem and the link scheduling problem in traditional networks, we propose a new ferry route design algorithm that is inspired by the Deficit Round-Robin link scheduling algorithm. We evaluate our algorithm in different network conditions through simulations. We use the average and standard deviation of data delivery delay as our evaluation metrics. Simulations results show that our algorithm performs better than other existing approaches for both metrics.

Future work includes developing the weighted version of our DRR-based ferry route design algorithm. We will also consider performing more simulations in different network conditions to improve the algorithm performance.

References

[1] W. Zhao, M. Ammar, and E. Zegura, “A message ferrying approach for data delivery in sparse mobile ad hoc networks,” in *MobiHoc ’04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, 2004, pp. 187–198.

[2] ——, “Controlling the mobility of multiple data transport ferries in a delay tolerant network,” in *IEEE INFOCOM*, 2005, pp. 1407–1418.

[3] M. M. mBin Tariq, M. Ammar, and E. Zegura, “Message ferry route design for sparse ad hoc networks with mobile nodes,” in *MobiHoc ’06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, 2006, pp. 37–48.

[4] W. Zhao and M. Ammar, “Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks,” in *IEEE Workshop on Future Trends in Distributed Computing Systems*, 2003.

[5] D. Jea, A. Somasundara, and M. Srivastava, “Multiple controlled mobile elements (data mules) for data collection in sensor networks,” in *DCOSS*, 2005.

[6] R. Sugihara and R. Gupta, “Improving the data delivery latency in sensor networks with controlled mobility,” in *DCOSS*, 2008.

[7] R. Shah, S. Roy, S. Jain, and W. Brunette, “Data mules: Modeling a three-tier architecture for sparse sensor networks,” in *IEEE SNPA Workshop*, 2003.

[8] R. Howard, *Dynamic Programming and Markov Processes*. MIT press, Cambridge, 1960.

[9] M. Shreedhar and G. Varghese, “Efficient fair queueing using deficit round-robin,” in *SIGCOMM*, 1995, pp. 231–242.

[10] S. K. A. Demers and S. Shenker, “Analysis and simulation of a fair queueing algorithm,” in *Sigcomm*, 1989.

[11] J. Nagle, “On packet switches with infinite storage,” *IEEE Transactions on Communication*, vol. COM-35, 1987.

[12] L. Zhang, “Virtual clock: A new traffic control algorithm for packet switched networks,” *ACM Transactions on Computing*, vol. 9, 1991.

[13] M. Ammar and J. Wong, “The design of teletext broadcast cycles,” *Performance Evaluation*, vol. 5, pp. 235–242, 1985.

[14] E. Cohen and S. Shenker, “Replication strategies in unstructured peer-to-peer networks,” in *Sigcomm*, 2002.

[15] A. Keränen, J. Ott, and T. Kärkkäinen, “The one simulator for dtn protocol evaluation,” in *Simutools ’09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 2009, pp. 1–10.