Utility-Based Routing in Communication Networks with Unstable Links

Mingming Lu and Jie Wu Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

***Abstract*—Traditional Dijkstra and Bellman-Ford routing algorithms can only provide the best route to each destination based on a fixed link cost model. We propose a utility-based routing model that can provide different optimal routes for different routing requirements captured by the benefit value of the successful delivery of a data packet in a network with unstable links. The challenges lie in the identification of the relationship between the benefit values and the optimal routes without exhausting all possible routes. This can be exponential to the number of nodes in the worst case. We analyze the relationship between the benefit values and the optimal routes, and the relationship between different optimal routes. Based on this analysis, we propose an efficient algorithm that can compute all optimal routes by exploiting a unique property of the model. We also implement the algorithm in a distributed and parallel way, and conduct intensive simulations to verify our results.**

Keywords: Link stability, network, routing, utility.

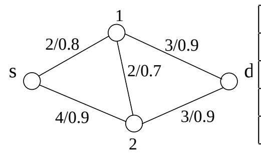
I. Introduction

Traditional routing algorithms, such as the Dijkstra and Bellman-Ford algorithms, characterize network links by a single metric, such as cost or error rate, and limit each source to use a single-best route for each destination. However, the single-best route usually fails to satisfy the diverse requirements from the routing source. For example, a routing source may prefer a lower error rate to a lower cost, or vice versa.

We find that the best way to reflect the routing source's requirement is through the market value. In a market, an item or a service with higher market value usually corresponds to a higher quality, and thus, incurs higher price. In analogy, different data have different market values to the routing source. The source would rather deliver the data with higher market value through the route with higher quality (lower error rate) even at the expense of higher price (cost). To discriminate the market value of the data to be transmitted from the general market value in the real market, we adopt the *benefit value* to denote the former.

Based on the above observation, our objective is to design a routing algorithm that can adaptively select the best route according to the benefit value of the data to be transmitted. In such a routing model, a source may have multiple best routes to the same destination because the source has different types of data with different benefit values. In Fig. 1, we give a simple but concrete example to illustrate the multiple best routes.

This work was supported in part by NSF grants ANI 0073736, EIA 0130806, CCR 0329741, CNS 0422762, CNS 0434533, CNS 0531410, and CNS 0626240. Email: {mlu2@, jie@cse.}fau.edu



	u_d	u_1	u_2	U
r_1	20/30	15/24		10*/17.2
r_2	20/30		15/24	9.5/17.6*
r_3	20/30	8.5/14.8	15/24	4.8/9.8
r_4	20/30	15/24	8.5/14.8	3.65/9.3

Fig. 1. The left figure is the topology of a simple network where the attributes of each link are labeled in the form of 'cost/stability'. The REUs of nodes on each route ($r_1 :< s, 1, d >, r_2 :< s, 2, d >, r_3 :< s, 1, 2, d >$, and $r_4 :< s, 2, 1, d >$) are listed in the right table, where in each cell, two values separated by '/' represent the REUs under benefit of 20 and 30, respectively.

There are four routes from the source s to the destination d. The best routes corresponding to the benefit values 20 and 30 are $< s, 1, d >$ and $< s, 2, d >$, respectively. The detailed calculation of these best routes will be presented in Section II.

The challenges of this problem are three-fold. First, the identification of the relationship between the benefit values and the corresponding optimal routes is needed; second, the avoidance of the exploration of all possible paths, which is exponential to the number of nodes in the worst case, is required; third, the distributed implementation of the proposed solution is needed. We confine our attention to a pair of nodes in the design of the routing algorithm, and extend it to all pairs of nodes in the distributed and paralleled implementation.

The main contributions of this paper can be enumerated as follows: 1) We propose a utility-based routing model where the benefit value reflects the routing requirement, and different benefit values may have different optimal routes. 2) Through the analysis of the properties of the routing model, we identify the relationship between the set of benefit values and the corresponding optimal routes. 3) We design an efficient algorithm to compute the mapping between each benefit value and each optimal route. 4) We design a distributed and paralleled implementation, which can be gracefully integrated into the Bellman-Ford routing algorithm.

II. Preliminaries

In [7], the following unicast routing problem is considered: given a benefit value and a pair of nodes, find the optimal path that connects them and maximizes the expected utility. For a (source, destination) pair (i, j) that can be connected through a link, the expected utility through link (i, j) is $p_{i,j} \times v - c_{i,j}$, where $c_{i,j}/p_{i,j}$ is the link cost/stability. Similarly, the expected utility through a path R is also in the same form, $p_R \times v - c_R$. The stability of a route is simply the product of the stability of each link on the route. On average, the expected cost consumed on a link should be the product of the link cost and the probability that data is transmitted successfully along all upstream links. The *route cost* is the sum of the expected link costs. For example, the expected cost of link $(i, i+1)$ on a multi-hop route $R =< 1, 2, \cdots, r >$, where nodes 1 and r are the source and destination, respectively, is $c_{i,i+1} \prod_{j=1}^{i-1} p_{j,j+1}$, where $p_{j,j+1}$ $(j = 1, \cdots, i-1)$ is the stability of upstream link $(j, j+1)$. Therefore, the expected utility obtained through path R can then be expressed as

$$U = (\prod_{j=1}^{r-1} p_{j,j+1}) \cdot v - \sum_{i=1}^{r-1} c_{i,i+1} \prod_{j=1}^{i-1} p_{j,j+1} \quad (1)$$

where the product $\prod_{j=1}^{r-1} p_{j,j+1}$ is route stability p_R, and $\sum_{i=1}^{r-1} c_{i,i+1} \prod_{j=1}^{i-1} p_{j,j+1}$ is the route cost c_R.

Formula (1) can be derived through the following recursive formula:

$$u_i = p_{i,i+1} \times u_{i+1} - c_{i,i+1}. \quad (2)$$

where $i = 1, \cdots, r-1$. Starting at node $r-1$, the residual expected utility (REU) is $u_{r-1} = p_{r-1,r} \times v - c_{r-1,r}$ by applying Formula (2). Similarly, the REU at node $r-2$ is $u_{r-2} = u_{r-1} \times p_{r-2,r-1} - c_{r-2,r-1}$. The REU at node 1 is $U = u_1 = p_{1,2} \times u_2 - c_{1,2}$, which is equal to the expected utility in Formula (1).

In [7], an efficient algorithm, MaxUtility, is proposed to compute the optimal route for a given benefit value. The basic idea is to calculate the REU backwards, starting from the destination with initial REU being v and repeatedly applying Formula (2) recursively over each link backwardly. The algorithm will construct a tree with initially only the destination, and add one link to the tree at a time. In the tree construction process, the algorithm maintains a priority queue with initial elements being all nodes, and their priorities being their REUs, which are $-\infty$ initially besides that of the destination.

In each iteration, the algorithm selects the node with the highest priority from the queue, then relaxes its neighbors remaining in the queue, and finally removes the node from the queue to the tree (through the link connecting the node and its parent). The relaxation consists of two steps: first, the chosen node calculates the REU of each neighbor through Formula (2); second, the node compares each neighbor's calculated REU with its original REU, saves the larger one, and updates the relaxed neighbor's parent if the neighbor's REU is updated. The tree construction repeats until the source is added. The route from the source to the destination in the tree is, hence, the optimal route.

We illustrate the MaxUtility algorithm by an example shown in Fig. 1. Assume the benefit value $v = 20$. By applying Formula (2) over link $(1, d)$, node 1's REU is updated: $u_1 = 20 \times 0.9 - 3 = 15$. Similarly, $u_2 = 15$. Since $u_1 = u_2$, the tie-breaking rule (the smallest ID first) is applied to select node 1, which then tries to update its neighbor node 2 and source s. But only the REU of s is updated (from $-\infty$ to 10) because node 2's new REU, calculated by node 1, is less than its current one. After node 1's update, node 2 is selected because of its maximum REU among the remaining nodes. Node 2 executes the same update operations as node 1 does, and updates no neighbor. In the end, s is selected and the optimal route $< s, 1, d >$ with the expected utility 10 is found.

III. The Problem

A. Problem definition

The problem in our prior work [7] can be summarized as: given a benefit value v and a pair of nodes, finding the optimal route $R(v)$, which depends on the value of v. If the benefit value v changes, the optimal route $R(v)$ needs to be re-computed. It inspires us to consider a more general problem: finding the optimal routes for all possible benefit values. Formally, the generalized problem can be defined as: finding a set $\mathcal{R}$ of optimal routes for a given range of benefit values (such as $[v_1, v_2]$) so that the optimal route for any benefit value within the given range belongs to the optimal-route set $\mathcal{R}$, i.e., $\forall v \in [v_1, v_2]$, $R(v) \in \mathcal{R}$.

A possible solution to this generalized problem is to reactively compute the optimal route by applying the MaxUtility algorithm each time a routing demand with a new benefit value arrives. The drawback of this solution is its routing discovery latency. An alternative solution is to proactively collect all routes' information and then build the mapping between each benefit value and its corresponding optimal route based on the collected route information. This solution is not efficient because the number of available routes can be exponential to the number of nodes in the networks, and the number of benefit values can be infinite.

In practice, the benefit value is discrete and the number of benefit values within a given interval is finite. Therefore, we assume that there is a granularity δ with a given interval $[v_1, v_2]$. Without loss of generality, we assume that both v_1 and v_2 are feasible benefit values and hence there are exactly $\frac{v_2 - v_1}{\delta} + 1$ feasible benefit values, which are $v_1, v_1 + \delta, v_1 + 2\delta, \cdots, v_2 - \delta$, and v_2, respectively. Thusly, the number of the optimal routes is at most $\frac{v_2 - v_1}{\delta} + 1$.

B. Model and Analysis

We observe that different benefit values may share the same optimal route. For example, for two different benefit values, $v_a \neq v_b$, it is possible that their corresponding optimal routes are $R(v_a) = R(v_b)$. This observation provides a possibility to reduce the complexity of the potential solution. Let's assume that an optimal route is shared by many benefit values. If the mapping between the optimal route and those benefit values can be constructed through the computation of the optimal routes for several (not all) selected benefit values, the complexity can be greatly reduced. To analyze this possibility, we need to explore the relationship between the benefit values and their corresponding optimal routes.

From Formula (1), we already know that the expected utility obtained through a route R can be expressed as $p_R \times v - c_R$, where p_R and c_R are the stability and expected cost of route R, respectively. Since p_R and c_R are constants for a given route R, and v is changeable, the expected utility obtained through route R can be regarded as an affine function:

$$U_R(v) = p_R \times v - c_R, \quad 0 \leq p_R \leq 1, c_R \geq 0.$$

with v as the variable. We name U_R as a *utility function* of route R because the utility obtained through route R can be characterized by function U_R alone. A utility function is determined by its two parameters: p_R and c_R, which in turn determine the utility for any given benefit value.

Usually, there is more than one route between a pair of nodes. If two routes have different stabilities, the straight lines representing them intersect on the coordinate axis. For example, consider two routes R_1 and R_2 with different stabilities and costs. Their utility functions are $U_{R_1}(v) = p_1 \times v - c_1$ and $U_{R_2}(v) = p_2 \times v - c_2$, respectively. The intersection of the two straight lines representing U_{R_1} and U_{R_2} is $v_c = \frac{c_1 - c_2}{p_1 - p_2}$. For a given benefit interval $[v_1, v_2]$, the intersection can be either within the interval or outside of the interval. If $v_c \notin [v_1, v_2]$, one route is always better (in terms of the expected utility) than the other route. Otherwise, one route is better than the other route in $[v_1, v_c)$, but is worse in $(v_c, v_2]$.

In any of these three cases, the relationship between the maximum expected utility and the benefit value can be expressed as the function

$$U(v) = \max_{R \in \mathcal{R}} U_R(v) \tag{3}$$

Similarly, our objective function $R(v)$, the relationship between the benefit values and the corresponding optimal routes, can be described as the function

$$R(v) = \arg\max_{R \in \mathcal{R}} U_R(v) \tag{4}$$

i.e., an optimal route for any benefit value v is the route that maximizes the expected utility $U(v)$ among all routes. Note that it is possible that there is more than one optimal route for a given benefit value. Therefore, $R(v)$ is actually a set of optimal routes. From definitions of functions $U(v)$ and $R(v)$, we can see that these two functions are highly correlated, and thus, $U(v)$ can help us obtain the objective function $R(v)$.

We observe that if two different benefit values share the same optimal routes R, the optimal route for any benefit value between the two benefit values should be the same as R. Therefore, if there are more than two benefit values in an interval (corresponding to an optimal route) and we can identify the minimum and maximum benefit values in the interval, then the optimal route for the other benefit values in the interval is identified. The observation can be characterized by the following theorem.

Theorem 1: For any route R with utility function $U_R(v)$ from source s to destination d, if R is the optimal route for benefit values v_1 and v_2, satisfying $v_1 < v_2$, then R is the optimal route for any benefit value $v \in [v_1, v_2]$.

IV. The Solution

A. Centralized Solution

The basic idea of our algorithm is to recursively partition the set of benefit values into non-overlapping subsets so that the benefit values within each subset correspond to the same optimal route. Our algorithm uses the MaxUtility algorithm

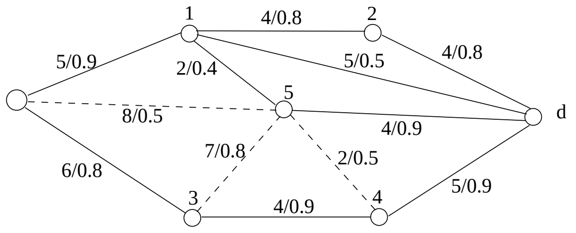


Fig. 2. Illustration of the algorithm.

to identify the optimal routes for a set of selected benefit values. Compared with the naïve solution, which has to run the MaxUtility algorithm for each individual benefit value, our algorithm greatly reduces the number of executions of the MaxUtility algorithm. Compared with the brute-force solution to identify the boundaries, our algorithm does not have to calculate all possible paths from source to destination, and hence, reduces the time complexity.

Now we are able to present our algorithm. The input of this algorithm is a benefit interval $[v_1, v_2]$ and the granularity δ, satisfying that $v_2 - v_1$ can be divided by δ. The algorithm first computes the optimal routes $R(v_1)$ and $R(v_2)$ by applying the MaxUtility algorithm twice. Then it calls a recursive function Partition, the parameters of which include two benefit values, v_a and v_b, and their corresponding optimal routes, $R(v_a)$ and $R(v_b)$. The Partition function is used to determine whether the benefit values within a given benefit interval $[v_a, v_b]$ share the same optimal route. If so, the optimal route for those benefit values will be determined; otherwise, the interval will be partitioned for further operations.

Initially, $v_a = v_1$ and $v_b = v_2$. Within interval $[v_a, v_b]$, the Partition function first checks whether the optimal routes are the same, i.e., $R(v_a) = R(v_b)$. If $R(v_a) = R(v_b)$, the optimal route for any benefit value $v \in [v_a, v_b]$ is $R(v_a)$, according to Theorem 1. Therefore, the interval $[v_a, v_b]$ does not need further partitioning, and the Partition function can determine that the optimal route for interval $[v_a, v_b]$ is $R(v_a)$.

If $R(v_a) \neq R(v_b)$, the Partition function needs to determine the partition point v_c to divide interval $[v_a, v_b]$ into two sub-intervals, $[v_a, v_c]$ and $[v_c, v_b]$, and recursively call function Partition to determine the sub-intervals for intervals $[v_a, v_c]$ and $[v_c, v_b]$, respectively. The selection of the partition point affects the performance of the algorithm. Basically, there are two different ways to determine the partition point.

The first way determines the partition point by considering the utility functions of routes $R(v_a)$ and $R(v_b)$. If routes $R(v_a)$ and $R(v_b)$ are different, there exists an intersection between the straight lines representing routes $R(v_a)$ and $R(v_b)$. The intersection point is adopted as the partition point, which is

$$v_c = \frac{c_{R(v_a)} - c_{R(v_b)}}{p_{R(v_a)} - p_{R(v_b)}}. \tag{5}$$

The advantage of this method is that if no other route is better (in terms of the expected utility) than $R(v_a)$ and $R(v_b)$ within $[v_a, v_b]$, the benefit intervals for $R(v_a)$ and $R(v_b)$ can be determined as $[v_a, v_c]$ and $[v_c, v_b]$ through the intersection point v_c. Although the intersection point cannot always partition the interval $[v_a, v_b]$ into two determined sub-intervals, this method still provides a heuristic to decide the partition point and it is suitable when the number of possible sub-intervals is relatively small.

The second way selects the partition point without considering the utility function. Here, we adopt binary partition, i.e., v_c is the median benefit value. That is,

$$v_c = \frac{v_a + v_b}{2}. \tag{6}$$

The second method is suitable when the number of possible sub-intervals is relatively large because it is costly to identify all of those intersections. Therefore, the advantage of the second method lies in its reducing the number of partitions in the event that there is a large number of sub-intervals. The disadvantage of the second method is that it cannot quickly determine the boundary between two sub-intervals, and the number of partitions to determine the boundary is proportional to the granularity of the benefit value. The smaller the granularity, the more the partitions. The formal description of the algorithm is as follows.

```
Require: Input: a given benefit range $[v_1, v_2]$;
1: Compute optimal routes $R(v_1)$ and $R(v_2)$
2: Partition($(v_1, R(v_1)), (v_2, R(v_2))$);

Partition($(v_a, R(v_a)), (v_b, R(v_b))$)
1: Initialize($(v_a, R(v_a)), (v_b, R(v_b))$)
2: if $R(v_a) = R(v_b)$ then
3:     The optimal route for $[v_a, v_b]$ is identified as $R(v_a)$;
4: else
5:     Compute partition point $v_c$ and optimal route $R(v_c)$;
6:     Partition($(v_a, R(v_a)), (v_c, R(v_c))$);
7:     Partition($(v_c, R(v_c)), (v_b, R(v_b))$);
```

We use the example in Fig. 2 to illustrate the algorithm, and assume that the benefit interval is $[1, 20]$ with $\delta = 1$, i.e., the set of benefit values is $\{1, 2, \cdots, 20\}$ and $k = 19$. First, the algorithm calls procedure MaxUtility twice at the two boundaries $v = 1$ and $v = 20$ and obtains two optimal routes $R(1) =< s, 1, 5, d >$ and $R(20) =< s, 3, 4, d >$. Since $R(1) \neq R(20)$, the initial interval should be partitioned into two sub-intervals at the point $v = 1 + \lfloor \frac{19}{2} \rfloor \cdot 1 = 10$. Then the algorithm calls MaxUtility again to compute optimal route $R(10)$ for further partition in sub-intervals $[1, 10]$ and $[10, 20]$. It turns out that there are two optimal routes at $v = 10$. They are $R(1)$ and $< s, 1, d >$. Since one of $R(10)$ equals to $R(1)$, no further partition in $[1, 10]$ is needed and the optimal route for the set of benefit values $\{1, \cdots, 10\}$ is therefore determined.

However, further partitioning is still needed in $[10, 20]$ since $R(10) \neq R(20)$. Because the partition point is $v = 15$ and $R(15)$ is the same as one route of $R(10)$, $< s, 1, d >$, the optimal route for the benefit set $\{10, \cdots, 15\}$ is also determined. There are a total of 5 levels of partitions, and many partitions can be used to determine the optimal routes for half of the benefit values. The mapping between the optimal routes and the benefit values is as follows:

	n_s	n_1	n_2/n_5	n_4	n_3
0	$([1,20],\cdot)$	$([1,20],\cdot)$	$([1,20],\cdot)$	$([1,20],\cdot)$	$([1,20],\cdot)$
1	$([1,20],\cdot)$	$([1,20],d)$	$([1,20],d)$	$([1,20],d)$	$([1,20],\cdot)$
2	$([1,20],1)$	$([1,10],5)$ $([10,15],d)$ $([16,20],2)$	$([1,20],d)$	$([1,2],5)$ $([3,20],d)$	$([1,20],4)$
3	$([1,18],1)$ $([19,20],3)$	$([1,10],5)$ $([10,15],d)$ $([16,20],2)$	$([1,20],d)$	$([1,2],5)$ $([3,20],d)$	$([1,20],4)$

TABLE I
THE DISTRIBUTED IMPLEMENTATION APPLIED TO THE NETWORK OF FIG. 2. $n_i (i = 1, \cdots, 5)$ DENOTES THE NEXT HOP OF NODE i.

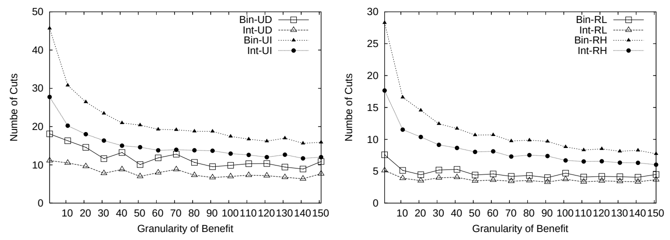
$$\overbrace{1, 2, \cdots, 10}^{R(1)} \overbrace{10, 11, \cdots, 15}^{R(15)} \overbrace{16, 17, 18}^{R(16)} \overbrace{19, 20}^{R(19)}.$$

B. Distributed Implementation

Although we confine our attention to a pair of nodes in the partition-based routing algorithm, we can extend it to all pairs of nodes in the distributed implementation. The basic idea of our distributed implementation is to adopt a modified distributed Bellman-Ford algorithm to compute the optimal route for each selected benefit value and utilize the intrinsic parallelism among the computation of the optimal routes within the same partition level to speed up the routing discovery process. In the traditional distributed Bellman-Ford algorithm [8], each node exchanges cost and routing information with its neighbors on an interactive basis until routing tables at the nodes converge to the appropriate shortest path entries. In our modified distributed Bellman-Ford algorithm, nodes also exchange stability information, and each node computes the maximum ENU path to every other node based on exchanged routing information. Each entry of our routing tables is ((destination,benefit), next hop) instead of (destination, next hop). For example, in Fig. 2, one entry of the routing table of node 1 is $((d, 18), 2)$, which denotes that the next hop along the optimal route of node 1 to d is node 2 if the benefit is 18.

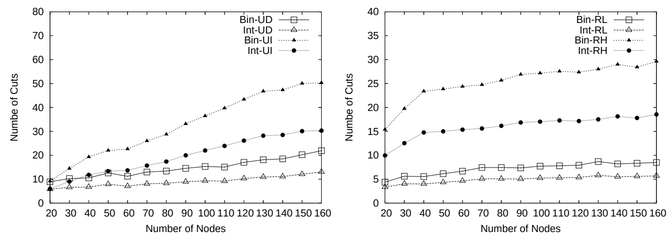
Due to the parallelism within the computation of the optimal routes, we can calculate the optimal routes for different benefit values simultaneously. In the following, we assume that the benefit interval and the benefit granularity are known to every node through broadcast from the destination. The distributed implementation can be illustrated by the example shown in Fig. 2 (with initial interval $[1, 20]$) through the change of each node's the routing tables, as shown in Table I. Each cell of Table I corresponds to ((destination,benefit), next hop), an entry of a node's new routing table. Because we consider one destination for simplicity, we remove the destination from each cell of Table I and keep only the next hop and the benefit. In Table I, $n_i (i = s, 1, \cdots, 5)$ represents the next hop of node i along the optimal route to d.

Initially, the entry to the destination d is empty, as shown in the second row of Table I (labeled as 0). After node d exchanges link cost/stability with its neighbors, nodes $1, 2, 4$, and 5 know that they can directly connect to d, and can calculate the expected utilities associated with benefit 1 and



(a) Simulation in unit disk graph. (b) Simulation in random graph.

Fig. 3. The effect of granularity.



(a) Simulation in unit disk graph. (b) Simulation in random graph.

Fig. 4. The effect of node density.

20. For example, node 1's expected utilities for benefit 1 and 20 are -4.5 and 5, respectively. Based on this calculation, the four nodes update the entries to d as shown in the third row. Then, the four nodes can further exchange routing information with their neighbors. After that, node 1 updates its routing table because it finds that it is better to forward packets to node 2 (5) if the benefit is 20 (1), as the expected utility through node 2 (5) is $5.6 > 5$ ($-3.24 > -4.5$) in the event that the benefit value is 20 (1). Since the optimal routes for benefit 1 and 20 are not the same for node 1, node 1 needs to determine the boundary between the intervals associated with these two optimal routes. At that time, node 1 can collect the route cost/stability for routes $< 1, 2, d >$, $< 1, d >$, and $< 1, 5, d >$, which are 7.2/0.64, 5/0.5, and 3.6/0.36, respectively. Through Equation (5), node 1 identifies the intersection at $v = 13$, where the optimal route is $< 1, d >$. Through further identification of the intersection between routes $< 1, 5, d >$ and $< 1, d >$ (between routes $< 1, d >$ and $< 1, 2, d >$), node 1 can identify its boundary as shown in the fourth row of Table I. Through another round of information exchange, node s can also find its optimal route set as shown in the fifth row of the table.

V. Simulation

We consider two simulation scenarios: (1) random graph model; (2) unit disk graph model. The random graph and the unit disk graph are used to model the general network and the wireless ad hoc network, respectively. To maintain the connectivity of the network, we adopt the Erdös-Renyi random graph model [1], where every pair of nodes is connected with a given probability. The link cost in our random graph model is randomly generated. In the unit disk graph, the connection between any two nodes depends on their geometric positions and their transmission ranges. The link cost is proportional to the link length. The unit disk graph model helps us restrict the effects of various simulation parameters. To further restrict the simulation environment, we can generate the link stability based on the link cost.

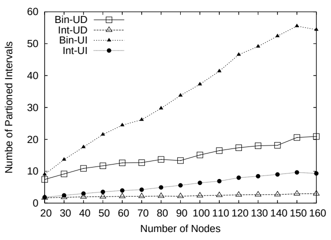
The following parameters are used in the experiments: the benefit granularity δ, the network density n (i.e., node population), and the network topology such as node degree. In the unit disk graph scenario, the simulation is set up in a $100m \times 100m$ area, where all nodes are homogeneous and can be deployed in this area arbitrarily.

In each of the following experiments, we consider four cases: two for unit disk graph: one with link stability dependent on link cost (UD), and one with link stability independent of link cost (UI), and the other two for random graph: one with lower connectivity (RL) in terms of average node degree, and one with higher connectivity (RH). In the UD case, we set the link stability proportional to the link cost. In the UI case, we allow the link stability to vary uniformly within stability range $[0, 1]$. In RL and RH, we control the connectivity by adjusting the coefficient of the probability that connects a pair of nodes.

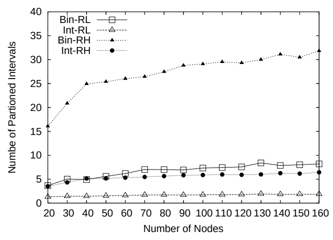
In the first experiment, we compare the number of *cuts* (the number of calls on the MaxUtility procedure) by the two partition methods, which is the intersection-partition (denoted as *Int*) and the binary-partition (denoted as *Bin*). The number of nodes is set to 100. The granularity varies from 1 to 151 in increments of 10. The simulation results (illustrated in Fig. 3) show that the number of cuts used by the two partition methods is sensitive to the graph model, the link stability dependence, and the connectivity. In any case, the intersection-partition method outperforms the binary-partition method in terms of the number of cuts. The reason is that the intersection partition method considers the characteristics of the optimal routes, which help the algorithm quickly identify the boundaries between different optimal routes. We also observe that the benefit granularity affects both of the partition methods. The reason is that the increment of the granularity can greatly reduce the partition level for both methods.

To assess the effect of network density (reflected by node number), we compare the number of cuts produced by the two partition methods through various network densities. From the first experiment, we can see that the effect of the benefit granularity is prominent within the range $[1, 51]$. To reduce the effect of the granularity, we set it to 60. The node number is adjusted from 20 to 160 in increments of 10. The simulation results are illustrated in Fig. 4, where we can conclude that network density can increase the number of cuts because the increment of density introduces additional available routes, which in turn introduce more intervals. From Fig. 4 (a), we can see that more cuts are produced in UI than UD. That is because the randomness of the link stability introduces more available routes. Similarly, the number of cuts in RH is more than that in RL because the increment of average node degree introduces more available routes.

Fig. 5 compares the number of intervals generated by the two partition methods. In our solution, the benefit interval

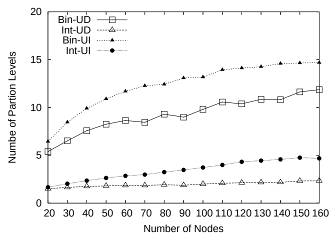


(a) Simulation in unit disk graph.

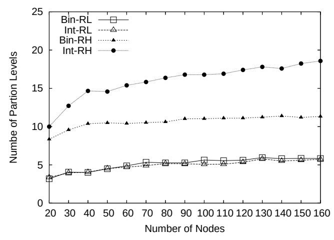


(b) Simulation in random graph.

Fig. 5. Comparison of the partitioned intervals.



(a) Simulation in unit disk graph.



(b) Simulation in random graph.

Fig. 6. Comparison of the partition levels.

for an optimal route may be partitioned into several intervals, which correspond to the same optimal route. It is better that there are as few partitioned intervals as possible. Fig. 5 shows that the intersection partition method has better performance in terms of the number of partitioned intervals.

We also compare the two partition methods from the parallelism point of view. The fewer the partition levels, the fewer rounds the distributed implementation requires in order to generate the route tables for each benefit value. The simulation results are shown in Fig. 6. In the simulation result of the unit disk graph (Fig. 6 (a)), the intersection-partition method has fewer partition levels than the binary-partition method although the former has more available routes, because it takes more partitions for the binary-partition method to identify the boundaries between two optimal routes. Similar observation can be found in the simulation result for the random graph (Fig. 6 (b)). From Fig. 6, we can conclude that the intersection-partition method has higher parallelism.

The simulation result can be summarized as follows: 1) the partition-based solution can greatly reduce the complexity in identifying the optimal routes and their corresponding benefit intervals; 2) the intersection-partition method is better than the binary-partition method from various angles; 3) the benefit granularity has a great effect on both partition methods in terms of reducing the number of cuts, but it cannot change the relative performance between the two partition methods.

VI. Related Work

Numerous existing routing protocols [4] restrict each router to use a single best route to each destination, and hence do not meet the diverse routing requirements. Recent works [9], [10] provided solutions to offer alternative routes for the single optimal route. However, the alternative route might not be the optimal route for a specific routing requirement. To meet the specific routing requirements, the QoS routing problems [2], [5] modeled the routing problem as the minimization of routing cost under various routing constrains such as delay, reliability, or bandwidth. Our model focuses on two QoS requirements: cost and reliability, and adopts the benefit to reflect the QoS requirement. There are many other works adopting similar values to reflect the QoS requirement, such as the utility-based network models [3], [6].

VII. Conclusion

In this paper, we study a utility-based routing model, which can provide different optimal routes to different routing requirements, by using the benefit to characterize the routing requirements (cost and stability) and using the expected utility as the single routing metric. We design an efficient algorithm that can compute all optimal routes for a given range of benefit values, and implement the algorithm in a distributed and paralleled way. We also conduct intensive simulations to verify our results.

References

[1] Béla Bollobás. *Random Graphs, 2nd Edition*. Cambridge University Press, 2001.

[2] A. Chakrabarti and G. Manimaran. Reliability constrained routing in QoS networks. *IEEE/ACM Transactions on Networking*, 13(3):662–675, 2005.

[3] W. Chen and L. Sha. An energy-aware data-centric generic utility based approach in wireless sensor networks. In *Proceedings of IPSN '04*, pages 215–224, 2004.

[4] C. Huitema. *Routing in the Internet, second ed.* Prentice Hall PTR, 2000.

[5] A. Jttner, B. Szviatovszki, I. Mcs, and Z. Rajk. Lagrange relaxation based method for the QoS routing problem. In *Proceedings of IEEE INFOCOM'01*, pages 859–868, 2001.

[6] J.W. Lee, M. Chiang, and A. R. Calderbank. Price-based distributed algorithms for rate-reliability trade-off in network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(5):962– 976, 2006.

[7] M. Lu and J. Wu. Social welfare based routing in ad hoc networks. In *Proceedings of ICPP'06*, pages 211–218, 2006.

[8] J. Wu. *Distributed System Design*. CRC Press, Inc., 1998.

[9] W. Xu and J. Rexford. MIRO: multi-path interdomain routing. In *Proceedings of ACM SIGCOMM'06*, pages 171–182, 2006.

[10] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *Proceedings of ACM SIGCOMM '06*, pages 159–170, 2006.