Distributed Quality-of-Service Routing in Ad Hoc Networks

Shigang Chen and Klara Nahrstedt, *Member, IEEE*

Abstract— In an ad hoc network, all communication is done over wireless media, typically by radio through the air, without the help of wired base stations. Since direct communication is allowed only between adjacent nodes, distant nodes communicate over multiple hops. The quality-of-service (QoS) routing in an ad hoc network is difficult because the network topology may change constantly, and the available state information for routing is inherently imprecise. In this paper, we propose a distributed QoS routing scheme that selects a network path with sufficient resources to satisfy a certain delay (or bandwidth) requirement in a dynamic multihop mobile environment. The proposed algorithms work with imprecise state information. Multiple paths are searched in parallel to find the most qualified one. Fault-tolerance techniques are brought in for the maintenance of the routing paths when the nodes move, join, or leave the network. Our algorithms consider not only the QoS requirement, but also the cost optimality of the routing path to improve the overall network performance. Extensive simulations show that high call-admission ratio and low-cost paths are achieved with modest routing overhead. The algorithms can tolerate a high degree of information imprecision.

Index Terms—Ad hoc quality-of-service (QoS) routing, imprecise state information, ticket-based probing.

I. INTRODUCTION

A N ad hoc network consists of a set of mobile nodes (hosts) that are equipped with wireless transmitters and receivers which allow them to communicate without the help of wired base stations. Since each transmitter has a limited effective range, distant nodes communicate through multihop paths with other nodes in the middle as routers. There are numerous applications for this type of network. A group of moving soldiers in a battlefield communicates and coordinates with each other. A group of islands and ships communicates with the help of floating balloons and passing airplanes. A group of people with portable computers share their data in a conference room without laying cables between them.

 Much work has been done on routing in ad hoc networks: the destination-sequenced distance vector (DSDV) protocol [1], the wireless routing protocol [2], Gafni–Bertsekas algorithms [3], the lightweight mobile routing protocol [4], the temporally-ordered routing algorithms [5], the dynamic source routing protocol [6], the associativity-based routing protocol

[7], the spine-based routing algorithms [8], and the zone-routing protocol [9], etc. Emphasis has been on providing the shortest-path routing and achieving a high degree of availability in a dynamic environment where the network topology changes quickly. However, all the previous routing solutions only deal with the best-effort data traffic. Connections with quality-of-service (QoS) requirements, such as voice channels with delay and bandwidth constraints, are not supported.

The provision of QoS relies on resource reservation. Hence, the data packets of a QoS connection[1] are likely to flow along the same network path on which the required resources are reserved. The goal of QoS routing is twofold: a) selecting network paths that have sufficient resources to meet the QoS requirements of all admitted connections and b) achieving global efficiency in resource utilization.

QoS routing has been receiving increasingly intensive attention in the wireline network domain [10]. The recent work can be divided into three broad categories: source routing, distributed routing, and hierarchical routing. In source routing [11]–[13], each node maintains an image of the global network state, which is based on a routing path that is centrally computed at the source node. The global network state is typically updated periodically by a link-state algorithm [14]. In distributed routing [15]–[18], the path is computed by a distributed computation during which control messages are exchanged among the nodes, and the state information kept at each node is collectively used in order to find a path. In hierarchical routing [19], nodes are clustered into groups, which are recursively clustered into higher-level groups, creating a multilevel hierarchy. In every level of the hierarchy, source or distributed routing algorithms are used.

The QoS routing algorithms for wireline networks cannot be applied directly to ad hoc networks. First, the performance of most wireline routing algorithms relies on the availability of precise state information. However, the dynamic nature of an ad hoc network makes the available state information inherently imprecise. Though some recent algorithms [13], [20] were proposed to work with imprecise information (e.g., the probability distribution of link delay), they require the precise information about the network topology, which is not available in an ad hoc network. Second, nodes may join, leave, and rejoin an ad hoc network at any time and any location;

The authors are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: s-chen5@cs.uiuc.edu; klara@cs.uiuc.edu).

[1] A connection (call) is a transport-layer concept. It means: 1) the logical association between the end users and 2) the correct, ordered delivery of data [5]. A QoS connection is a connection that has an end-to-end performance requirement such as a delay or bandwidth constraint. A connection is implemented at the network layer by a network path (routing channel) through which data packets are delivered.

existing links may disappear, and new links may be formed as the nodes move. Hence, the established paths can be broken at any time, which raises new problems of maintaining and dynamically reestablishing the routing paths in the course of data transmission.

It is difficult to provide QoS in an ad hoc network due to its dynamic nature. The overhead of QoS routing in an ad hoc network is likely to be higher than that in a wireline network because the available state information is less precise, and the topology changes in an unpredicted way. If the topology of an ad hoc network changes too fast, the provision of the QoS can be even impossible. However, QoS is feasible in many other cases where the network topology changes less frequently. For example, in a conference room, most portable computers may be stationary most of the time, while some computers move, join, or leave the room. In this paper, we shall only study the type of ad hoc networks whose topologies are not changing that fast to make the QoS routing meaningless. Since we do not make any specific assumptions ensuring reliable routing paths in ad hoc networks, we want to emphasize that this paper only supports soft QoS without hard guarantees. The soft QoS means that there may exist transient time periods when the required QoS is not guaranteed due to path breaking or network partition. However, the required QoS should be ensured when the established routing path(s) remain unbroken. Many multimedia applications accept soft QoS and use adaptation techniques to reduce the level of QoS disruption [21]–[23]. For instance, the QoS disruption caused by rerouting in an ad hoc network can be mitigated by using the rate-adaptive, layer-based encoded voice/video compression schemes [24].

We propose a distributed QoS routing scheme for ad hoc networks. Two routing problems are studied. They are delay-constrained least-cost routing and bandwidth-constrained least-cost routing. The first one is NP-complete [25]. The second one is solvable in polynomial time, given precise state information. A path that satisfies the delay (or bandwidth) constraint is called a feasible path. We designed routing algorithms for these problems. The algorithms have the following distinct properties.

1) The algorithms can tolerate the imprecision of the available state information. Good routing performance in terms of success ratio, message overhead, and average path cost is achieved even when the degree of information imprecision is high. Note that the problem of information imprecision exists only for QoS routing; all best-effort routing algorithms, such as DSR [6] and ABR [7], do not consider this problem because they do not need QoS state in the first place.

2) Multipath parallel routing is used to increase the probability of finding a feasible path. In contrast to the flooding-based path discovery algorithms used in [6] and [7], we search only a small number of paths, which limits the routing overhead. In order to maximize the chance of finding a feasible path, the state information at the intermediate nodes is collectively utilized to make intelligent hop-by-hop path selection.

3) The algorithms consider not only the QoS requirements, but also the optimality of the routing path. Low-cost paths are given preference in order to improve the overall network performance.

4) In order to reduce the level of QoS disruption, fault-tolerance techniques are brought in for the maintenance of the established paths. Different levels of redundancy provide tradeoff between the reliability and the overhead. The dynamic path repairing algorithm repairs the path at the breaking point, shifts the traffic to a neighbor node, and reconfigures the path around the breaking point without rerouting the connection along a completely new path. Rerouting is needed in two cases. One case is when the primary path and all secondary paths are broken. The other case is when the cost of the path grows large and hence it becomes beneficial to route the traffic to another path with a lower cost.

The rest of the paper is organized as follows. The system models are given in Section II. The idea of ticket-based probing is briefly described in Section III. Delay-constrained routing and bandwidth-constrained routing are studied in Sections IV and V, respectively. Dynamic path maintenance is discussed in Section VI. The simulation results are presented in Section VII. The related work is studied in Section VIII. Finally, Section IX concludes the paper.

II. SYSTEM MODELS

A. Ad Hoc Network Model

A network is modeled as a set V of nodes that are interconnected by a set E of full-duplex directed communication links. V and E are changing over time when nodes move, join, and leave. Each node has a unique identifier and has at least one transmitter and one receiver. Assume that the effective transmission distance of every node is equal. Two nodes are neighbors and have a link between them if they are in the transmission range of each other. We assume there exists a neighbor discovering protocol. Each node periodically transmits a BEACON packet identifying itself [7], [26], so that any node i knows the set V_i of its neighbors. Neighboring nodes share the same wireless media, and each message is transmitted by a local broadcast. We assume the existence of a MAC protocol, which resolves the media contention, supports resource reservation, and ensures that, among the neighbors in the local broadcast range, only the intended receiver keeps the message, and the other neighbors discard the message. An example of such a MAC protocol, one which supports bandwidth reservation, was proposed by Gerand and Tsai [27].

B. Stationary and Transient Links

The links between the stationary or slowly moving nodes are likely to exist continuously. Such links are called stationary links. The links between the fast moving nodes are likely to exist only for a short period of time. Such links are called transient links. A routing path should use stationary links whenever possible in order to reduce the probability of a path breaking when the network topology changes [7].

Given the unpredictable nature of an ad hoc network, it is impractical to determine exactly which links are stationary

and which are transient. However, various approximation approaches exist. One simple approach is based on an empirical observation that moving nodes are more likely to move at the next moment, and stationary nodes are more likely to be stationary at the next moment. An immediate implication is that the links that are just formed are more likely to be broken than the links that have already existed for some time. Therefore, whenever a new link is formed, it is set as a transient link. After the link remains unbroken for a time period, it is changed to be a stationary link. This approach is similar to the one proposed in [7]. Let V_i^s be $\{j|(i, j)$ is a stationary link, $j \in V_i\}$. A node in V_i^s is called a stationary neighbor[2] of i, and a node in $V_i - V_i^s$ is called a transient neighbor.

C. QoS State Metrics

A node i is assumed to keep the up-to-date local state about all outgoing links.[3] The state information of link (i, j) includes: 1) delay (i, j), the channel delay of the link, including the radio propagation delay, the queuing delay, and the protocol-processing time; 2) bandwidth (i, j), the residual (unused) bandwidth of the link; and 3) cost (i, j), which can be simply one as a hop count or a function of the link utilization. In order to make a preference of stationary links over transient links, the cost of a transient link should be set much higher than that of a stationary link. The delay, bandwidth, and cost of a path $P = i \rightarrow j \rightarrow \cdots k \rightarrow l$ are defined as follows:

$$\text{delay}(P) = \text{delay}\,(i, j) + \cdots + \text{delay}\,(k, l)$$
$$\text{bandwidth}(P) = \min\{\text{bandwidth}\,(i, j), \ldots, \text{bandwidth}\,(k, l)\}$$
$$\text{cost}(P) = \text{cost}\,(i, j) + \cdots + \text{cost}\,(k, l).$$

D. Routing Problems

Given a source node s, a destination node t, and a delay requirement D, the problem of delay-constrained routing is to find a feasible path P from s to t such that delay $(P) \leq D$.

When there are multiple feasible paths, we want to select the one with the least cost. Finding the delay-constrained least-cost path is an NP-complete problem [25].

Another problem is bandwidth-constrained routing, i.e., finding a path P such that bandwidth $(P) \geq B$, where B is the bandwidth requirement. When there are multiple such paths, the one with the least cost is selected.

Finding a feasible path is actually the first part of the problem. The second part is to maintain the path when the network topology changes [28].

E. Imprecise State Model

The following end-to-end state information is required to be maintained at every node i for every possible destination t. The information is updated periodically by a distance-vector

protocol for mobile computers. Readers are referred to [1] for a detailed description of such a protocol.[4]

1) Delay: $D_i(t)$ keeps the minimum end-to-end delay from i to t, i.e., the delay of the least-delay path.
2) Bandwidth: $B_i(t)$ keeps the maximum end-to-end bandwidth from i to t, i.e., the bandwidth of the largest-bandwidth path.
3) Cost: $C_i(t)$ keeps the least end-to-end cost from i to t, i.e., the cost of the least-cost path.

The previous information is inherently imprecise in an ad hoc network because the network state and topology may change at any time.

The imprecision model proposed by Guerin and Orda [13] for wireline networks is based on probability distribution functions. For instance, every node maintains, for every link l, the probability $p_l(d)$ of link l having a delay of d units. This model is not suitable for an ad hoc network where links may be short-lived [7] and do not give enough time for collecting the probability distribution. In contrast, we propose a simple imprecision model that does not rely on the topology and can be easily implemented. Two additional state variables are required.

a) Delay variation: $\Delta D_i(t)$ keeps the estimated maximum change of $D_i(t)$ before the next update. That is, based on the recent state history, the actual minimum end-to-end delay from i to t is expected to be between $D_i(t) - \Delta D_i(t)$ and $D_i(t) + \Delta D_i(t)$ in the next update period.
b) Bandwidth variation: $\Delta B_i(t)$ keeps the estimated maximum change of $B_i(t)$ before the next update. The actual maximum bandwidth from i to t is expected to be between $B_i(t) - \Delta B_i(t)$ and $B_i(t) + \Delta B_i(t)$ in the next period.

For the purpose of simplicity, we do not apply the imprecise model to $C_i(t)$. The cost metric ($C_i(t)$) is used for optimization, in contrast to the delay and bandwidth metrics used in QoS constraints. Since a strict cost bound requirement does not exist, a certain degree of imprecision for $C_i(t)$ is tolerable.

In the following, we describe a possible way to calculate $\Delta D_i(t)$. $\Delta B_i(t)$ can be computed similarly. $\Delta D_i(t)$ is updated periodically together with $D_i(t)$. Consider an arbitrary update of $\Delta D_i(t)$ and $D_i(t)$. Let $\Delta D_i^{\text{old}}(t)$ and $\Delta D_i^{\text{new}}(t)$ be the values of $\Delta D_i(t)$ before and after the update, respectively. Similarly, let $D_i^{\text{old}}(t)$ and $D_i^{\text{new}}(t)$ be the values of $D_i(t)$ before and after the update, respectively. $D_i^{\text{new}}(t)$ is provided by a distance-vector protocol. $\Delta D_i^{\text{new}}(t)$ is calculated as follows:

$$\Delta D_i^{\text{new}}(t) = \alpha \times \Delta D_i^{\text{old}}(t) + (1 - \alpha) \times |D_i^{\text{new}}(t) - D_i^{\text{old}}(t)|.$$

The previous formula is similar to the one used by transmission control protocol (TCP) to estimate the round-trip delay. The factor α (<1) determines how fast the history information ($\Delta D_i^{\text{old}}(t)$) is forgotten, and ($1 - \alpha$) determines how fast $\Delta D_i^{\text{new}}(t)$ converges to $|D_i^{\text{new}}(t) - D_i^{\text{old}}(t)|$.

---

[2] It should be noted that stationary neighbors (links) are only relatively stationary by definition.

[3] We assume every node has the precise information about its local state. The problem of information imprecision only exists for the global state.

[4] Because our algorithms can tolerate high degree of information imprecision, a relatively low updating frequency is allowed, which leads to better scalability.

By the previous formula, it is still possible for the actual delay to be out of the range $[D_i(t)-\Delta D_i(t),\ D_i(t)+\Delta D_i(t)]$. One way to make such probability sufficiently small is to enlarge $\Delta D_i(t)$. Hence, we shall modify the formula and introduce another factor $\beta\ (>1)$

$$\Delta D_i^{\mathrm{new}}(t)$$
$$= \alpha \times \Delta D_i^{\mathrm{old}}(t) + (1-\alpha) \times \beta \times |D_i^{\mathrm{new}}(t) - D_i^{\mathrm{old}}(t)|.$$

$\Delta D_i^{\mathrm{new}}(t)$ converges to $\beta \times |D_i^{\mathrm{new}}(t) - D_i^{\mathrm{old}}(t)|$ at a speed determined by $(1-\alpha)$.

It should also be noted that our imprecision model and routing algorithms do not intend to cover every possible situation, which is impractical in an ad hoc network. Our objective is to improve the average performance, and the proposed algorithms based on the previous model may fail in finding a feasible path in the extreme cases where the state and the topology change very rapidly.

III. AN OVERVIEW

We propose a multipath[5] distributed routing scheme, called *ticket-based probing*.[6] Our design is based on the following observations.

- The QoS routing is done on a per-connection basis. Hence, the routing overhead is one of the major concerns. We shall not use any flooding path-discovery approaches, which may send routing messages to the entire network. Instead, we want to localize the routing activity in a portion of the network between the source s and the destination t. More specifically, we want to search only a small number of paths from s to t, instead of making an expensive exhaustive search.
- There are numerous paths from s to t. We shall not randomly pick several paths to search. Instead, we want to make an intelligent hop-by-hop path selection to guide the search along the best candidate paths.

The basic idea of ticket-based probing is outlined below.

A ticket is the permission to search one path. The source node issues a number of tickets based on the available state information. One guideline is that more tickets are issued for the connections with tighter requirements. Probes (routing messages) are sent from the source toward the destination to search for a low-cost path that satisfies the QoS requirement. Each probe is required to carry at least one ticket. At an intermediate node, a probe with more than one ticket is allowed to be split into multiple ones, each searching a different downstream subpath. The maximum number of probes at any time is bounded by the total number of tickets. Since each probe searches a path, the maximum number of paths searched is also bounded by the number of tickets. See Fig. 1 for an example.

Upon receipt of a probe, an intermediate node decides, based on its state: 1) whether the received probe should be split and
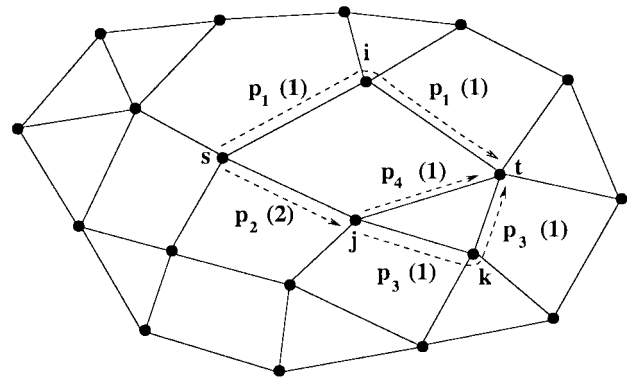
[5] Search multiple paths for a feasible one.
[6] A preliminary version of the ticket-based probing algorithm for wireline networks was published in *Proc. IEEE 7th Int. Conf. on Computer, Communications and Networks (IC3N'98)*.

Fig. 1. Two probes, p_1 and p_2, are sent from s. The number in the parentheses following a probe is the number of tickets carried in the probe. At node j, p_2 is split into p_3 and p_4, each of which has one ticket. There are at most three probes at any time. Three paths are searched, and they are $s \to i \to t$, $s \to j \to t$, and $s \to j \to k \to t$.

2) to which neighbor nodes the probe(s) should be forwarded. The goal is to collectively utilize the state information at the intermediate nodes to guide the limited tickets (the probes carrying them) along the best paths to the destination, so that the probability of finding a low-cost feasible path is maximized. A number of advantageous properties of the ticket-based probing are outlined below.

1) The routing overhead is controlled by the number of tickets, which allows the dynamic tradeoff between the overhead and the routing performance. Issuing more tickets means searching more paths, which results in a better chance of finding a feasible path at the cost of higher overhead.
2) The proposed scheme is designed to work with imprecise state information. The level of imprecision (information uncertainty) has a direct impact on the number of tickets issued. Multipath parallel search increases the chance of finding a feasible path and thus helps to tolerate information imprecision.
3) A distributed routing process is used to avoid any centralized path computation that could be very expensive for QoS routing in large networks. It is not necessary for any node to maintain the topology information. The most current topology is used during the hop-by-hop path selection process.
4) The information at the intermediate nodes, both local and end-to-end states, are collectively used to direct the probes along the low-cost feasible paths toward the destination. This approach not only increases the chance of success but also improves the ability to tolerate the information imprecision because the intermediate nodes may gradually correct a wrong decision made by the source. Stationary links are used whenever possible, which makes the routing path more stable.

Our ticket-based probing approach is proposed as a general QoS routing scheme, which can handle different QoS constraints. Delay-constrained routing and the bandwidth-constrained routing are the most studied QoS routing problems [12], [13], [29], [30]. In the sequel, we shall use them as examples to explain the operation details.

IV. DELAY-CONSTRAINED ROUTING

Based on the idea of ticket-based probing, we propose a heuristic algorithm for the NP-complete, delay-constrained, least-cost routing problem. When a connection request arrives at the source node, a certain number N_0 of tickets are generated, and probes are sent toward the destination t. Each probe carries one or more tickets. Since no new tickets are allowed to be created by the intermediate nodes, the total number of tickets is always N_0, and the number of probes is at most N_0 at any time. When a node receives a probe p with $N(p)$ tickets, it makes at most $N(p)$ copies of p, distributes the received tickets among the new probes, and then forwards them along to selected outgoing links toward t. Each probe accumulates the delay of the path it has traversed so far. A probe can proceed only when the accumulated delay does not violate the delay requirement. Hence, any probe arriving at the destination detects a feasible path, which is the one it has traversed.

There are two problems: 1) how to determine N_0 and 2) how to distribute the tickets in a received probe among the new probes. We will solve these problems based on the following two basic guidelines.

1) We want to assign different numbers of tickets to different connections based on their "needs." For a connection whose delay requirement is large and can be easily satisfied, one ticket is issued to search a single path; for a connection whose delay requirement is smaller, more tickets are issued to increase the chance of finding a feasible path; for a connection whose delay requirement is too small to be satisfied, no tickets are issued, and the connection is immediately rejected.

2) When a node i forwards the received tickets to its neighbors, the tickets are distributed unevenly among the neighbors, depending on their chances of leading to reliable low-cost feasible paths. A neighbor having a smaller end-to-end delay to the destination should receive more tickets than a neighbor having a larger delay; a neighbor having a smaller end-to-end cost to the destination should receive more tickets than a neighbor having a larger cost; a neighbor having a stationary link to i should be given preference over a neighbor having a transient link to i. Note that some neighbors may not receive any tickets because i may have only a few or just one ticket to forward.

A. Determining the Number of Tickets

1) Yellow Tickets and Green Tickets: The N_0 tickets are colored either yellow or green. The two types of tickets have different purposes.

1) The purpose of yellow tickets is to maximize the probability of finding a feasible path. Hence, yellow tickets (or more precisely, the probes carrying them) prefer paths with smaller delays, so that the chance of satisfying a given delay requirement is higher.

2) The purpose of green tickets is to maximize the probability of finding a low-cost path. Green tickets prefer the paths with smaller costs, which may, however, have
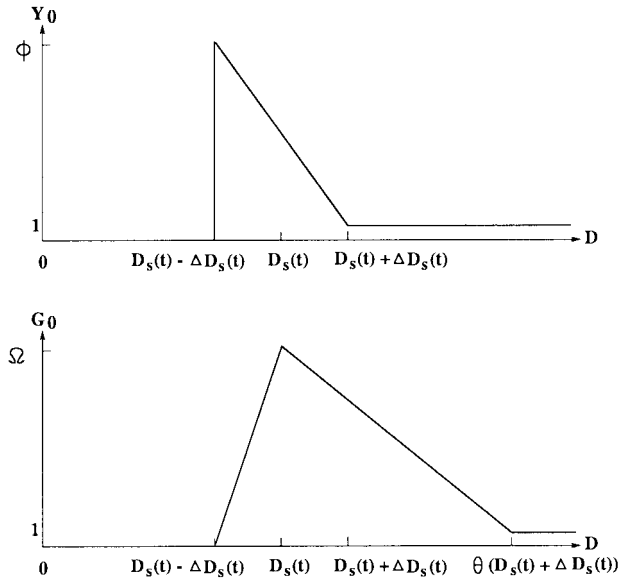

Fig. 2.   Curves of Y_0 and G_0 with respect to D.

larger delays and hence have less chance to satisfy the delay requirement D.

The overall strategy is to use the more aggressive green tickets to find a low-cost feasible path with relatively low success probability and to use the yellow tickets as a backup to guarantee a high success probability of finding a feasible path. $N_0 = Y_0 + G_0$, where Y_0 is the number of yellow tickets, and G_0 is the number of green tickets. We show how to determine Y_0 and G_0 in the following section.

2) Number of Yellow Tickets: Y_0 is determined based on the delay requirement D. If D is very large and can be surely satisfied, a single yellow ticket will be sufficient to find a feasible path. If D is too small to be possibly satisfied, no yellow ticket is necessary, and the connection is rejected. Otherwise, more than one yellow ticket is issued to search multiple paths for a feasible one. Based on the previous guideline, we choose a linear ticket curve in Fig. 2 (upper curve) for simplicity and efficient computation. The curve is explained in the following and the three system parameters are defined in Table I.

Let s and t be the source and the destination, respectively. Y_0 is a function of D, $D_s(t)$, and $\Delta D_s(t)$.

1) If $D \geq D_s(t) + \Delta D_s(t)$, then $Y_0 = 1$. Because D is equal to or greater than the largest possible end-to-end delay $(D_s(t) + \Delta D_s(t))$,[7] a single yellow ticket will be sufficient to find a feasible path.

2) If $D_s(t) - \Delta D_s(t) \leq D < D_s(t) + \Delta D_s(t)$, then $Y_0 = \lceil ((D_s(t) + \Delta D_s(t) - D)/(2 \times \Delta D_s(t))) \times \Phi \rceil$, where Φ is a system parameter specifying the maximum allowable number of yellow tickets. It shows that more yellow tickets are assigned for smaller D.

3) If $D < D_s(t) - \Delta D_s(t)$, then $Y_0 = 0$. Because D is even less than the best expected end-to-end delay $(D_s(t) - \Delta D_s(t))$, such a tight delay requirement will

---

[7] By our imprecise state model, the actual end-to-end delay is expected to be in $[D_s(t) - \Delta D_s(t), D_s(t) + \Delta D_s(t)]$. The probability for the delay to be out of the range is assumed to be negligibly small.

TABLE I
SYSTEM PARAMETERS

Φ	the maximum number of yellow tickets
θ	a threshold value specifying the *sufficiently-large* range for the delay requirement
Ω	the maximum number of green tickets

not be satisfied. The connection request is rejected. Two actions may be taken after a connection request is rejected: 1) the QoS negotiation process is activated for a relaxed delay bound and 2) the QoS routing is repeated after certain time.

3) Number of Green Tickets: G_0 is also determined based on the delay requirement D. The linear curve of G_0 is given in Fig. 2 (lower curve), which is explained in the following list.

1) If $D \geq \theta \times (D_s(t) + \Delta D_s(t))$, then $G_0 = 1$, where $\theta(>1)$ is a system parameter. θ specifies a threshold value, $\theta \times (D_s(t) + \Delta D_s(t))$, beyond which D is considered to be sufficiently large. When D is sufficiently large, only one green ticket is assigned. The reason for the threshold to be $\theta \times (D_s(t) + \Delta D_s(t))$ instead of $D_s(t) + \Delta D_s(t)$ is that green tickets prefer the least-cost paths whose delay may be larger than $D_s(t) + \Delta D_s(t)$, which is the largest delay of the least-delay path. It is generally an engineering issue to determine the most appropriate value for θ. $\theta \in [1.5, 2.0]$ worked very well in our simulation.

2) If $D_s(t) \leq D < \theta \times (D_s(t) + \Delta D_s(t))$, then $G_0 = \lceil ((\theta \times (D_s(t) + \Delta D_s(t)) - D)/(\theta \times (D_s(t) + \Delta D_s(t)) - D_s(t))) \times \Omega \rceil$, where Ω is the maximum allowable number of green tickets. It shows that as D decreases, G_0 increases.

3) If $D_s(t) - \Delta D_s(t) \leq D < D_s(t)$, then $G_0 = \lceil ((D - D_s(t) + \Delta D_s(t))/\Delta D_s(t)) \times \Omega \rceil$. As D approaches to $D_s(t) - \Delta D_s(t)$, the delay requirement becomes increasingly harder to satisfy. Hence, the emphasis of routing shifts from minimizing the cost to maximizing the probability of finding a feasible path. G_0 is decreased in order to reduce the overhead and allow Y_0 to be larger.

4) If $D < D_s(t) - \Delta D_s(t)$, then $G_0 = 0$. The connection request is rejected.

Theoretically, Φ (or Ω) can be $+\infty$, which makes ticket-based probing a flooding scheme. In practice, a small value (≤ 5) for Φ (or Ω) should be sufficient according to our simulation.

B. Forwarding the Received Tickets

1) Candidate Neighbors: If $Y_0 + G_0 = 0$, the connection request is rejected. Otherwise, probes carrying the tickets are sent from s to t. A probe proceeds only when the path has a delay of no more than D. Hence, once a probe reaches t, it detects a delay-constrained path.

Each probe accumulates the delay of the path it has traversed so far. More specifically, a data field, denoted as delay(p), is defined in a probe p. Initially, delay(p) := 0; whenever p proceeds for another link (i, j), delay(p) := delay(p) + delay(i, j).

Suppose a node i receives a probe p with $Y(p)$ yellow tickets and $G(p)$ green tickets. Suppose k is the sender of the probe p. The set $R_i^p(t)$ of candidate neighbors, to which i will forward the received tickets, is determined as follows.

We first consider only the stationary neighbors (V_i^s) of i. Let $R_i^p(t) = \{j | \text{delay}(p) + \text{delay}(i, j) + D_j(t) - \Delta D_j(t) \leq D, j \in V_i^s - \{k\}\}$. There is no need to send any ticket to $j \in V_i^s - R_i^p(t)$ because the best expected delay from j to t, which is $D_j(t) - \Delta D_j(t)$, plus delay(p) and delay(i, j) violate the delay requirement.

$R_i^p(t)$ is the set of neighbors to which the tickets should be forwarded. If $R_i^p(t) = \emptyset$, we take the transient neighbors into consideration and redefine $R_i^p(t)$ to be $\{j | \text{delay}(p) + \text{delay}(i, j) + D_j(t) - \Delta D_j(t) \leq D, j \in V_i - \{k\}\}$. If $R_i^p(t)$ is still empty, we invalidate all received tickets and discard them.

If $R_i^p(t) \neq \emptyset$, then for every $j \in R_i^p(t)$, i makes a copy of p, denoted as p_j. Let p_j have $Y(p_j)$ yellow tickets and $G(p_j)$ green tickets, such that $\sum_{j \in R_i^p(t)} Y(p_j) = Y(p)$ and $\sum_{j \in R_i^p(t)} G(p_j) = G(p)$. We will show how to calculate $Y(p_j)$ and $G(p_j)$ shortly.

In order to compute $R_i^p(t)$, node i must maintain the values of $D_j(t)$ and $\Delta D_j(t)$ for every neighbor node j. That can be easily realized in a distance-vector protocol by keeping the neighbors' distance vectors, which include $D_j(t)$ and $\Delta D_j(t)$. Recall that node i receives its neighbors' distance vectors periodically in order to calculate its own distance vector. This approach works well for medium size networks. For instance, when the number of nodes is at most 100, and the maximum degree of a node is at most ten, it takes no more than 8 kB to store all necessary information if $D_j(t)$ and $\Delta D_j(t)$ take 4 bytes each. A typical ad hoc network, such as one in a conference room, has a small or medium size. For large networks, the memory consumption can be very large. A possible solution for large networks is to inquire $D_j(t)$ and $\Delta D_j(t)$ from j on demand and cache the values locally. In this paper, we would like to consider only medium size networks because the QoS provision is increasingly difficult in large ad hoc networks since long routing paths tend to break more often. Hence, we shall rely on the distance-vector protocol to provide $D_j(t)$ and $\Delta D_j(t)$ for every neighbor node j. The same thing is true for $B_j(t)$, $\Delta B_j(t)$, and $C_j(t)$.

2) Distributing Yellow Tickets: $Y(p_j)$, $\forall j \in R_i^p(t)$ is determined based on an intuitive observation: a probe sent toward the direction with a smaller delay should have more yellow tickets

$$Y(p_j) = \frac{(\text{delay}(i, j) + D_j(t))^{-1}}{\sum_{j' \in R_i^p(t)} (\text{delay}(i, j') + D_{j'}(t))^{-1}} \times Y(p).$$

$Y(p_j)$ calculated by the previous formula may not be an integer. Larger $Y(p_j)$'s have the priority to be rounded to $\lceil Y(p_j) \rceil$, and smaller $Y(p_j)$'s will be rounded to $\lfloor Y(p_j) \rfloor$ so that $\sum_{j \in R_i^p(t)} Y(p_j) = Y(p)$.

3) Distributing Green Tickets: Recall that the purpose of green tickets is to find a low-cost feasible path. A probe sent toward the direction with a smaller cost should have more

green tickets. Hence, $\forall j \in R_i^p(t)$

$$G(p_j) = \frac{(\text{cost}(i,j) + C_j(t))^{-1}}{\displaystyle\sum_{j' \in R_i^p(t)} (\text{cost}(i,j') + C_{j'}(t))^{-1}} \times G(p).$$

Larger $G(p_j)$'s have the priority to be rounded to $\lceil G(p_j) \rceil$.

Finally, if $Y(p_j) + G(p_j) > 0$, then p_j is sent to j, carrying $Y(p_j)$ yellow tickets and $G(p_j)$ green tickets. If $Y(p_j) + G(p_j) = 0$, then p_j is dropped.

In the previous scheme, tickets may cycle around loops. Three possible approaches to avoid cycling infinitely are: 1) at most one probe is allowed to be sent to every outgoing link[8]; 2) the number of hops a probe can traverse is bounded, i.e., over-aged probes will be discarded; or 3) probes record their paths to detect possible loops, and the probes with loops in their paths will be discarded. Each of the previous three approaches prevents infinite cycling. We choose the first one in this paper, particularly in the simulation.

We have discussed two types of tickets. The distribution of yellow tickets is solely based on delay, and the distribution of green tickets is solely based on cost. Another type of tickets may be introduced, whose distribution is based on the combination of delay and cost. We omit the discussion of this type of tickets in this paper.

Our ticket-based probing algorithm reduces to the traditional shortest-path routing algorithm when the imprecise model is not used. Suppose $\Delta D_i(t) = 0, \forall i, t \in V$. Our algorithm reduces to the following approach: if $D \geq D_s(t)$, then the source node issues a single yellow ticket, which traverses the least-delay path to the destination, and in addition, one or more of the green tickets is issued, trying to find a feasible path with a lower cost.

C. Termination and Path Selection

The routing process is terminated when all probes have either reached the destination or been dropped by the intermediate nodes. In order to detect the termination, we require the intermediate nodes to send the invalidated tickets[9] to the destination t instead of discarding them. Therefore, all tickets will arrive at t eventually. The routing process is terminated after t receives all $(Y_0 + G_0)$ tickets.[10] Time out is used to handle the problem of message (ticket) losses that may result from network partition, buffer overflow, or channel errors. If only invalidated tickets are received, t sends a message to s to inform the rejection of the request; otherwise, at least one feasible path is found. Because probes are transmitted in parallel in the network, it takes a message a round trip to find a feasible path or reject a connection.

Whenever t receives a probe with a valid ticket, a feasible path is found, which is the one the probe has traversed. There are two ways to record the path: one is to record the

TABLE II
DATA STRUCTURE

id	system-wide unique identification for the connection request
s	source node
t	destination node
D	delay requirement
$Y_0 + G_0$	total number of tickets
k	sender of p
$path$	path p has traversed so far
$Y(p)$	number of yellow tickets carried by p
$G(p)$	number of green tickets carried by p
$delay(p)$	accumulated delay of the path traversed so far
$cost(p)$	accumulated cost of the path traversed so far

path in the probe itself, and the other is to record the path at the intermediate nodes on a hop-by-hop basis. The first approach requires larger size probes and thus consumes more communication bandwidth and more memory space to store the probes when they are waiting in the queues. The second approach, however, requires memory space at the intermediate nodes to store the path. In this paper, we choose the first approach for its simplicity.

A probe accumulates the cost of the path it traverses. If multiple probes with valid tickets arrive at the destination, the path with the least cost is selected as the primary path, and the other paths are the secondary paths, which will be used when the primary path is broken due to the mobility of intermediate nodes.

After the primary path is selected, a confirmation message is sent back along the path to the source and reserves resources along the way. However, since the topology of the network and the resource availability of every node are constantly changing, the path may be broken or an intermediate node may not have sufficient resources at the time when the confirmation message is received. In either case, the message is turned back to the destination, and a secondary path is chosen as the primary path. Both the source and the destination use time out to prevent themselves from waiting indefinitely in case of message losses.

D. Data Structure

The data structure of a probe p is shown in Table II. The last six fields, k, path, $Y(p)$, $G(p)$, delay(p), and cost(p), are modified as the probe traverses. Tickets are logical tokens, and only the number of tickets is important: there can be at most $Y(p) + G(p)$ new probes descending from p, among which probes with yellow tickets choose paths based on delay, probes with green tickets choose paths based on cost, and probes with both yellow and green tickets choose paths based on both delay and cost.

E. Rerouting

In an ad hoc network, there are a number of situations where rerouting is desired. First, the network topology may change as new nodes join in the network and existing nodes move or leave the network. Rerouting helps to tolerate the network dynamics by adapting the routing paths periodically according to the changing topology. More importantly, when

---

[8] An intermediate node i must record to which outgoing links probes have been sent. When a new probe is received, R_i^p is calculated as $\{j | \text{delay}(p) + \text{delay}(i,j) + D_j(t) - \Delta D_j(t) \leq D, j \in V_i^s - \{k\}$. A probe has not been sent from i to j before.$\}$.

[9] The tickets in a received probe are invalidated if $R_i^p(t) = \emptyset$ (see Section IV-B1).

[10] The number $(Y_0 + G_0)$ is included in the probes sent to t.

a routing path is broken, rerouting can be used to reestablish the connection along a new path. Second, the routes of the connections are typically selected based on the network resource availability at the times when the requests arrive. Long paths are often assigned when resource contention occurs. However, as the network topology changes and connections are established or torn down upon completion, the network state changes locally and globally, which makes the routes of the remaining connections less optimal [31]. Routes with light (heavy) traffic at the beginning may become congested (lightly loaded) later. Shorter paths for some connections may become available. Rerouting helps to balance the network traffic on the fly and improves the resource efficiency, which is especially important in an ad hoc network where resources are scarce.

Rerouting can be done periodically and/or upon triggering when a broken path is detected. It should not be done too frequently in order to avoid excessive overhead and the oscillation of shifting the traffic from one part of the network to another. However, it should also be noted that, compared to the contiguous traffic of a typical voice connection, the rerouting overhead is relatively small as long as it is not done too frequently.

F. Soft States

Routing and rerouting can be used in conjunction with RSVP [32], which is a resource reservation protocol. RSVP is based on soft states, i.e., the resource reservation must be refreshed periodically. Soft states are deleted if not refreshed within a time-out period. This is important in an ad hoc network due to the following reason. As the network topology changes dynamically, routing paths may be broken into pieces, and the network may even be partitioned, which causes the information kept at nodes to become obsolete. Using soft states helps to delete the obsolete information and release the unused resources automatically.

Every node in the network maintains a connection table, which has an entry for every connection passing the node, containing the incoming link and the outgoing link used by the connection. Hence, it takes a lookup of the connection table to decide to which outgoing link that an incoming packet of a connection should be forwarded. In addition, a connection table entry stores the source node, the destination node, the resource reservation, and other information about the connection. Every entry is a soft state that will be deleted if not refreshed.

A refreshing message is sent from the destination back along the routing path to the source periodically [32]. When an intermediate node receives the message, it resets the timer of the corresponding soft state and forwards the message to the upstream node. Note that refreshing messages always follow the most recent routing path established by the last rerouting. Therefore, the soft states at the nodes of the previous paths will eventually be deleted.

G. Local Multicast

In an ad hoc network, each message is transmitted by a local broadcast over the shared media (air). Such a transmis-
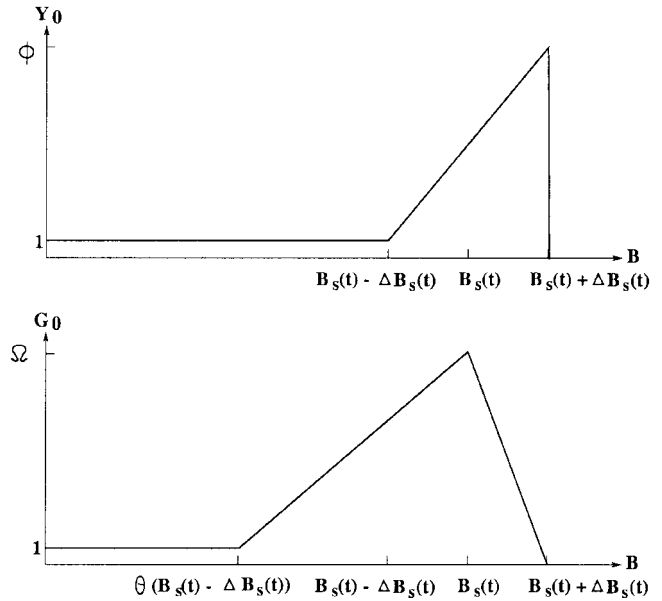

Fig. 3. Curves of Y_0 and G_0 with respect to B.

sion mechanism makes the local multicast possible. A local multicast message allows more than one neighbor to be the receiver. This property can be used to reduce the number of probes sent by a node i. Instead of sending a probe p_j to every neighbor j satisfying $Y(p_j) + G(p_j) > 0$, a local multicast message is constructed, which combines the information in all p_j's. Hence, only a single message is needed, from which the intended receivers extract the tickets belonging to them and other information. Although the number of messages is reduced, the message length increases,[11] and more CPU time is required to process the messages.

V. BANDWIDTH-CONSTRAINED ROUTING

The bandwidth-constrained routing algorithm is briefly described in this section. It shares the same computational structure with the delay-constrained routing algorithm. The differences are the metric-dependent ticket curves and ticket distribution formulas.

A. Determining the Number of Tickets

Consider a connection request whose source, destination, and bandwidth requirement are s, t, and B, respectively. Let the number of yellow tickets be Y_0 and the number of green tickets be G_0. Fig. 3 shows the linear curves of Y_0 and G_0.

The number of yellow tickets, Y_0, is determined as follows.
1) If $B \leq B_s(t) - \Delta B_s(t)$, then $Y_0 = 1$.
2) If $B_s(t) - \Delta B_s(t) < B \leq B_s(t) + \Delta B_s(t)$, then $Y_0 = \lceil ((B - B_s(t) + \Delta B_s(t))/(2 \times \Delta B_s(t))) \times \Phi \rceil$.
3) If $B > B_s(t) + \Delta B_s(t)$, then $Y_0 = 0$.

The number of green tickets, G_0, is determined as follows.
1) If $B \leq \theta \times (B_s(t) - \Delta B_s(t))$, then $G_0 = 1$, where $0 < \theta < 1$.

---

[11] When TDMA is used and each message takes a time slot regardless of the message length, as long as it fits in a time slot, local multicast will still save bandwidth.

2) If $\theta \times (B_s(t) - \Delta B_s(t)) < B \leq B_s(t)$, then $G_0 = \lceil ((B - \theta \times (B_s(t) - \Delta B_s(t)))/(B_s(t) - \theta \times (B_s(t) - \Delta B_s(t)))) \times \Omega \rceil$.

3) If $B_s(t) < B \leq B_s(t) + \Delta B_s(t)$, then $G_0 = \lceil ((B_s(t) + \Delta B_s(t) - B)/(\Delta B_s(t))) \times \Omega \rceil$.

4) If $B > B_s(t) + \Delta B_s(t)$, then $G_0 = 0$.

If $Y_0 + G_0 > 0$, then s sends itself a probe with Y_0 yellow tickets and G_0 green tickets to activate the routing process.

B. Forwarding the Received Tickets

Suppose a node i receives a probe p with $Y(p)$ yellow tickets and $G(p)$ green tickets. Let k be the sender of p. Define $R_i^p(t) = \{j | \text{bandwidth}(i, j) \geq B \wedge B_j(t) + \Delta B_j(t) \geq B, j \in V_i^s - \{k\}\}$.

If $R_i^p(t) = \emptyset$, then redefine $R_i^p(t) = \{j | \text{bandwidth}(i, j) \geq B \wedge B_j(t) + \Delta B_j(t) \geq B, j \in V_i - \{k\}\}$. If $R_i^p(t)$ is still empty, then invalidate all tickets and send them to t for the purpose of termination detection.

If $R_i^p(t) \neq \emptyset$, then for every $j \in R_i^p(t)$, i makes a copy of p, denoted as p_j. Let p_j have $Y(p_j)$ yellow tickets and $G(p_j)$ green tickets. $Y(p_j)$ is determined based on the observation that a probe sent toward the direction with a larger residual bandwidth should have more yellow tickets. $\forall j \in R_i^p(t)$

$$Y(p_j) = \frac{\min\{\text{bandwidth}(i, j), B_j(t)\}}{\sum_{j' \in R_i^p(t)} \min\{\text{bandwidth}(i, j'), B_{j'}(t)\}} \times Y(p).$$

Larger $Y(p_j)$'s have the priority to be rounded to $\lceil Y(p_j) \rceil$.

$G(p_j)$ is determined based on the observation that a probe sent toward the direction with a smaller cost should have more green tickets. $\forall j \in R_i^p(t)$

$$G(p_j) = \frac{(\text{cost}(i, j) + C_j(t))^{-1}}{\sum_{j' \in R_i^p(t)} (\text{cost}(i, j') + C_{j'}(t))^{-1}} \times G(p).$$

Larger $G(p_j)$'s have the priority to be rounded to $\lceil G(p_j) \rceil$.

If $Y(p_j) + G(p_j) > 0$, then p_j is sent to j, carrying $Y(p_j)$ yellow tickets and $G(p_j)$ green tickets. If $Y(p_j) + G(p_j) = 0$, then p_j is dropped.

VI. DYNAMIC PATH MAINTENANCE

The required QoS is ensured during the time when an established path remains unbroken. The QoS provision, however, is disrupted during the rerouting time. We want to be restrictive on the type of networks studied in this paper. Our routing algorithm works well when the average life time of an established path is much longer (such as an order of magnitude longer) than the average rerouting time. In such a case, the required QoS is ensured in most of the connection's lifetime. There are numerous examples of such mobile networks. For instance, a group of soldiers stays in their bunkers and communicates with each other; a group of warships moves into a formation in which the relative positions

of the ships are maintained. In this paper, we shall only consider the type of networks whose topologies are relatively stable because our routing-rerouting architecture does not support ad hoc networks with violently changing topologies.

In this section, we propose solutions for the problems of detecting and reconstructing the broken paths. In addition, we use path redundancy to tolerate the topology dynamics and use path repairing to repair the broken path at the breaking point.

A. Detection of Broken Paths

Let s, t, and P be the source, the destination, and the established routing path, respectively. Each node i on P except s has a preceding node, denoted as pre_i. Similarly, each node i on P except t has a successive node, denoted as suc_i.

In an ad hoc network, P can be broken in any of the following cases:

a) the source s moves too far from suc_s such that link (s, suc_s) is broken;

b) an intermediate node i moves too far from pre_i or suc_i such that either link (pre_i, i) or link (i, suc_i) is broken;

c) the destination t moves too far from pre_t such that link (pre_t, t) is broken; and

d) any node on P leaves the network.

A single approach is proposed for all the above cases: if a node i ($i \neq t$) using the neighbor discovering protocol finds that suc_i is no longer its neighbor, i detects that P is broken at link (i, suc_i). One possible implementation is for each node i to maintain a link table for every outgoing link, storing the set of connections using that link. When i finds that a neighbor node j no longer exists, i.e., link (i, j) has been broken, it concludes that every connection in the corresponding link table has a broken path.

B. Rerouting

Rerouting is commonly used to deal with the problem of path breaking in ad hoc networks [6]. When a node i detects a broken path, it looks up in the connection table to find the source node s of the connection and then sends s a path-breaking message. When s receives the message, it activates the ticket-based routing algorithm and reroutes the connection to another feasible path. Meanwhile, a resource-releasing message is sent along the original path to release the reserved resources. Since the path has been broken, some intermediate nodes will not receive the resource-releasing message. However, that will not cause the resources to be held indefinitely because resource reservation is part of the soft state that will be deleted automatically if not refreshed (see Section IV-F).

In the process of rerouting, data packets are transmitted as best-effort traffic, which means the required QoS is not guaranteed during this period of time. Although the rerouting takes only a message round trip time to reestablish the connection along a new feasible path, we want to reduce the jitter in the QoS provision as much as possible. The common approach to tolerate the fault condition (broken paths) is to introduce redundancy [5].

C. Path Redundancy

We consider a broken path as a fault and use the *path redundancy* for fault-tolerance. There is a tradeoff between the overhead of redundancy and the performance of QoS provision. We propose a multilevel redundancy scheme to meet the diversity of user requirements.

1) First-Level Redundancy: For the most critical connections, the highest level of redundancy is used. The idea is to establish multiple routing paths for the same connection. Every data packet is sent along each path independently. If the destination receives the same packet from more than one path, it keeps the first copy and discards the rest. The required QoS is guaranteed as long as any of the paths remain unbroken.

Recall that our ticket-based routing algorithms may detect multiple feasible paths in a single run. Increasing the number of tickets will improve the chance of finding multiple feasible paths. The number of routing paths actually established depends on the redundancy requirement and the overhead concern. Disjointed paths are preferred because multiple paths will be broken when a shared link is broken. Multiple consecutive runs of the routing algorithm help to establish the disjointed routing paths; the successive runs use only the links that are not on the paths established by the previous runs. Three additional numbers are maintained at the source node for a connection c:

$\min_c$ minimum required number of unbroken paths for providing the nondisrupted QoS[12];

$\max_c$ maximum allowed number of unbroken paths, depending on the overhead concern;

act_c the actual number of current unbroken paths.

Each time a broken path is detected, act_c is decreased by one. As long as $act_c \geq \min_c$, no further action is taken. If $act_c < \min_c$, the routing algorithm is activated to find more feasible paths in order to bring act_c back to $\max_c$ or as close as possible if not enough feasible paths are found.

2) Second-Level Redundancy: For the ordinary connections that allow certain degree of QoS disruption, the second level redundancy is used. It is similar to the first level redundancy, except that, among the multiple established paths, one is selected as the primary path, and the others are secondary paths. Data packets are sent only along the primary path. Although resources are reserved on the secondary paths as well, they are not used. The resources held on the secondary path will not be wasted because they will be used by the best-effort traffic in a work-conserving system [33]. When the primary path is detected broken, the source node selects one from the secondary paths to be the new primary path. When act_c becomes less than $\min_c$, the routing algorithm is activated to find more secondary paths.

3) Third-Level Redundancy: The ordinary connections may also use the third level redundancy, which has the lowest overhead. It is similar to the second level redundancy except that no resource is reserved on the secondary paths. When the primary path is broken, control messages are sent along the secondary paths to check the current resource availability. If some of the secondary paths remain feasible, one is selected
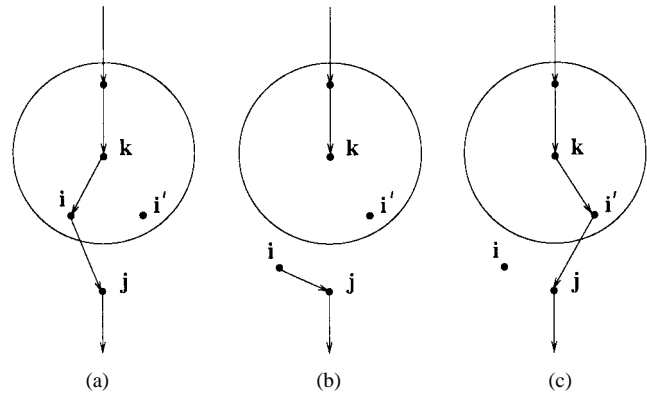
Fig. 4.   The transmission range of node k is shown by the circle. (a) A segment of the routing path is $k \rightarrow i \rightarrow j$. (b) Node i moves out of the transmission range of k. (c) The traffic is dynamically shifted to another path segment $k \rightarrow i' \rightarrow j$ without rerouting the connections to a completely new path.

as the new primary path, and the resources are reserved. If none of them is feasible, rerouting is activated.

D. Path Repairing

Dynamic path-repairing repairs the routing path at the breaking point, shifts the data traffic to a neighbor node, and reconfigures the path around the breaking point without rerouting the connection along a completely new path [7]. Fig. 4 gives an example. The routing path is broken after i moves out of the transmission range of k. Instead of sending a path-breaking message to the source, k tries to repair the path by broadcasting a repair-requesting message to the current neighbors asking if any of them are able to take over the job of i. Upon receipt of the message, the neighbors that have links to j reply their resource availabilities to k. Based on the received information, k finds that i' has sufficient resources for that role. It adds link (k, i') to the routing path and then sends a path-repairing message to i'. Upon receipt of the path-repairing message, i' reserves the required resources and adds link (i', j) to the routing path.

In order to do path repairing, the connection table at k must be extended to store the successive node's successive node, denoted as suc_k^2, which is j in Fig. 4. The repair-requesting message sent by k to its neighbors contains suc_k^2. Every neighbor i' sends back a reply message, including at least a Boolean field telling whether it has a link to suc_k^2. If it does, the message also contains bandwidth(i', suc_k^2) (or delay(i', suc_k^2)), in addition to other information such as cost(i', suc_k^2).

In the case of bandwidth-constrained routing, when k receives a reply message from a neighbor node i', it checks whether both bandwidth$(k, i') \geq B$ and bandwidth$(i', \text{suc}_k^2) \geq B$ are satisfied, where B is the bandwidth requirement. If so, k sends a path-repairing message to i' in order to reestablish the broken routing path through i'. If multiple neighbors satisfy the requirement, the one which minimizes cost$(k, i') + \text{cost}(i', \text{suc}_k^2)$ is selected. If none of the neighbors satisfies the requirement, a path-breaking message is sent to the source for rerouting.

In the case of delay-constrained routing, k selects the neighbor i' that minimizes $\text{delay}(k, i') + \text{delay}(i', \text{suc}_k^2)$. After the broken path is reestablished through i', a validation process is initiated to check whether the new path violates the delay requirement. A path-validation message is sent to the destination. After receiving the message, the destination turns it around. Starting from there, the message travels back along the routing path to the source node and accumulates the end-to-end delay along the way. After the source node receives the message, it checks whether the end-to-end delay violates the requirement. If so, either rerouting or QoS negotiation with the user application is activated.

VII. SIMULATION AND RESULTS

Extensive simulations were done to evaluate the proposed ticket-based probing algorithms. The results about the delay-constrained routing algorithm is presented in this section. Three performance metrics, success ratio, average message overhead, and average path cost, are defined as follows:

$$\text{success ratio} = \frac{\text{number of connections accepted}}{\text{total number of connection requests}}$$

$$\text{avg. msg. overhead} = \frac{\text{total number of messages sent}}{\text{total number of connection requests}}$$

$$\text{avg. path cost} = \frac{\text{total cost of all established paths}}{\text{number of established paths}}.$$

Sending a probe over a link is counted as one message. Hence, for a probe that has traversed a path of l hops, l messages are counted.

The network topology used in our simulation is randomly generated. Forty nodes are placed randomly within a 15×15 m^2 area. The transmission range of a node is bounded by a circle with a radius of 3 m. A link is added between two nodes that are in the transmission range of each other. The average degree of a node is 3.4. The source node, the destination node, and the delay requirement (D) of each connection request are randomly generated. D is uniformly distributed in the range of [30, 160 ms]. The cost of each link is uniformly distributed in [0, 200]. Each link (j, k) is associated with two delay values: delay-old(j, k) and delay-new(j, k). Delay-old(j, k) is the last delay value advertised by the link to the network. Note that $D_i(t), \forall i, t \in V$, is calculated based on the delay-old values of all links. Delay-new(j, k) is the actual delay of the link at the time of routing. Delay-old(j, k) is uniformly distributed in [0, 50 ms], while delay-new(j, k) is uniformly distributed in $[(1 - \xi) \times \text{delay-old}(j, k), (1 + \xi) \times \text{delay-old}(j, k)]$, where ξ is a simulation parameter, called the imprecision rate, specifying the largest percentage difference of delay-new(j, k) from delay-old(j, k)

$$\xi = \sup \left\{ \frac{|\text{delay-new}(j, k) - \text{delay-old}(j, k)|}{\text{delay-old}(j, k)} \right\}.$$

Three algorithms are simulated: the flooding algorithm, the ticket-based probing algorithm (TBP), and the shortest-path algorithm (SP).

TABLE III
SYSTEM PARAMETERS FOR TBP

Φ	4
θ	1.5
Ω	3

The flooding algorithm is equivalent to TBP with infinite yellow tickets and zero green tickets. It floods routing messages from the source to the destination. Each routing message accumulates the delay of the path it has traversed, and the message proceeds only if the accumulated delay does not exceed the delay bound. As shown by Shin and Chou [18], when certain scheduling policies are used and the routing messages are set to the appropriate priority, the routing messages travel at speeds according to the link delays. Hence, the message traveling along the least-delay path arrives first. With this assumption, an intermediate node needs only to propagate the first received message and discard all successively received ones. There will be at most one message sent along every link. The algorithm finds a feasible path whenever one exists and hence is the optimal algorithm in terms of success ratio. The flooding algorithm does not have an efficient mechanism for the termination detection. It selects the routing path when the destination receives the first routing message. The advantage of the flooding algorithm is that it does not need to maintain any global state. The disadvantage is that too many routing messages are sent.

The system parameters of the TBP algorithm are shown in Table III. The values in the table are obtained by extensive simulation runs. See Section IV for an explanation about each parameter.

The SP algorithm maintains a state vector at each node i by a distance-vector protocol. The vector has an entry for every possible destination t, containing two elements, $D_i(t)$ and $\pi_i(t)$. $D_i(t)$ is the delay of the least-delay path from i to t, and $\pi_i(t)$ is the next hop on the least-delay path. $D_i(t)$ and $\pi_i(t)$ may be imprecise since they are calculated based on the last advertised delay values (delay-old) of all links. When a request arrives at s with $D \leq D_s(t)$, the algorithm sends out one routing message along the least-delay path to check the current resource availability for possible connection establishment.

In Sections VII-A–C, we do not consider the node mobility and the path maintenance. The focus is on evaluating the ticket-based probing algorithm itself. In Section VII-D, the node mobility is taken into account, and the algorithm is evaluated together with the path maintenance techniques proposed in Section VI.

A. Success Ratio

Figs. 5–8 compare the success ratios of the three algorithms. Each point in the figures is taken by running 5000 independently generated random connection requests. The success ratio is a function of both the average delay requirement D and the imprecision rate ξ. The former is represented by the x axis, and the latter is shown by different figures. In each figure, as D becomes larger, it is easier to satisfy, and thus the
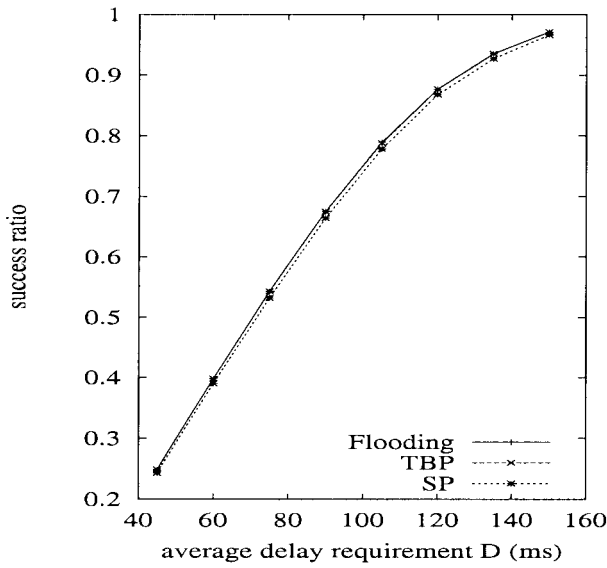
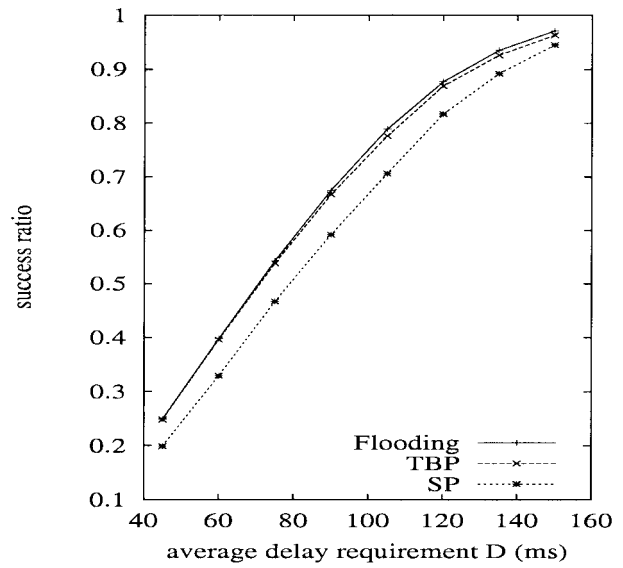Fig. 5. Success ratio (imprecision rate: 5%).
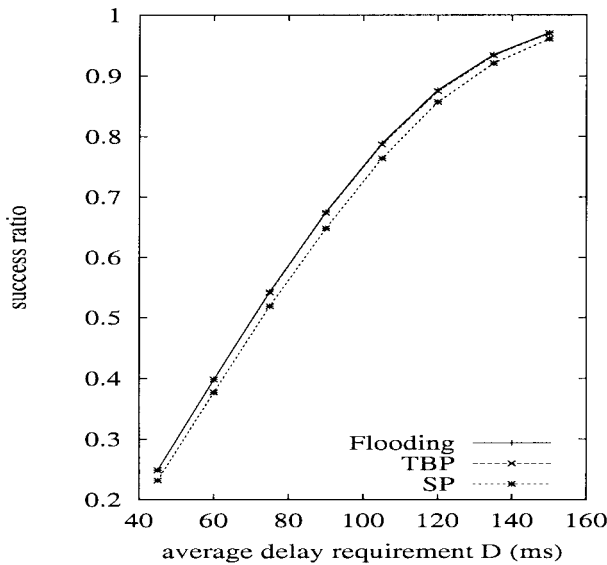

Fig. 7. Success ratio (imprecision rate: 25%).

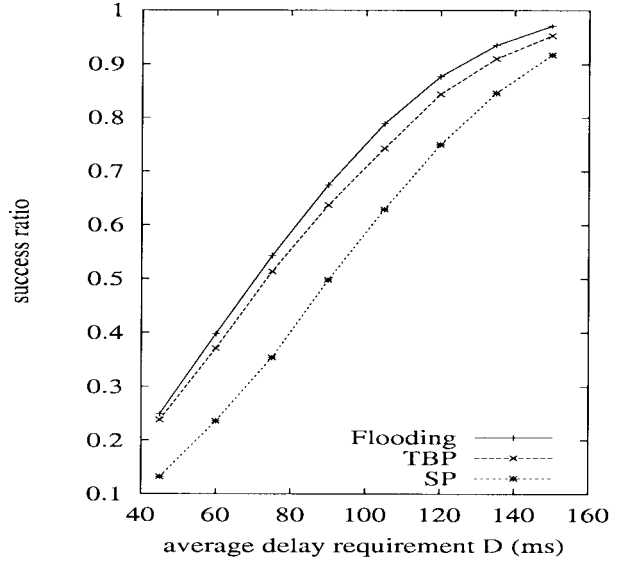Fig. 6. Success ratio (imprecision rate: 10%).

Fig. 8. Success ratio (imprecision rate: 50%).

success ratio is higher. As expected, the flooding algorithm has the best success ratio. The success ratio of TBP is close to that of the flooding algorithm, even when the imprecision rate is as high as 50%. This is because TBP searches multiple paths, and the number of paths searched is adjusted according to how difficult it will be to find a feasible path. In addition, the state information of intermediate nodes is collectively used to direct the probes along the most appropriate paths toward the destination. In contrast, the SP algorithm performs much worse when the imprecision rate is high.

Figs. 9 and 10 show the success ratios of TBP and SP relative to that of the flooding algorithm. For an imprecision rate of 10%, TBP performs as well as the flooding algorithm, while SP is up to 7% worse than the flooding algorithm. For an imprecision rate of 50%, TBP is up to 9% worse than the flooding algorithm, while SP is up to 51% worse than the flooding algorithm.

B. Message Overhead

Figs. 11–14 compare the average message overhead of the three algorithms. The flooding algorithm has a prohibitively high message overhead. SP has the lowest overhead. TBP has a modest overhead that is higher than that of SP but much lower than that of the flooding algorithm. The message overhead of TBP increases as the imprecision rate increases.

C. Average Path Cost

Figs. 15–18 compare the average path cost of the three algorithms. TBP has a lower average path cost than the flooding algorithm and SP. This is because TBP uses both the delay metric and the cost metric to make the routing decision, while the other two algorithms use only the delay metric. Recall that the green tickets are designed to find the low-cost feasible paths.
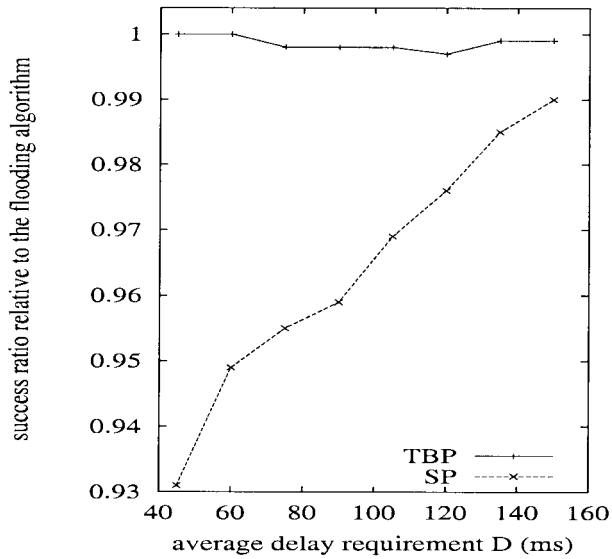
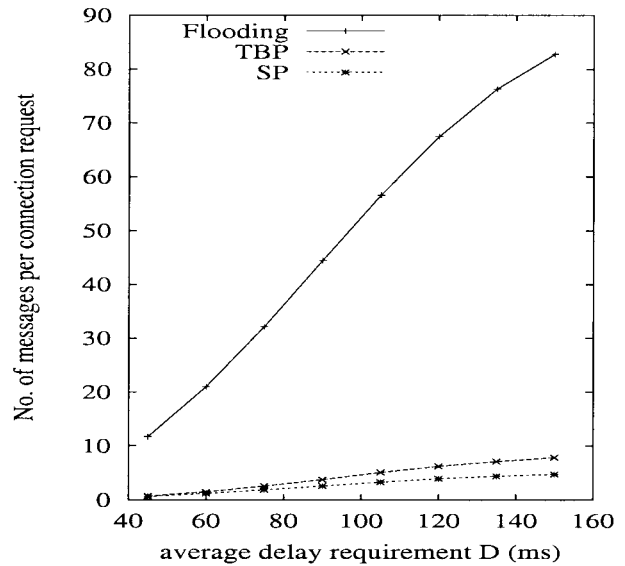Fig. 9.   Relative success ratio (imprecision rate: 10%).

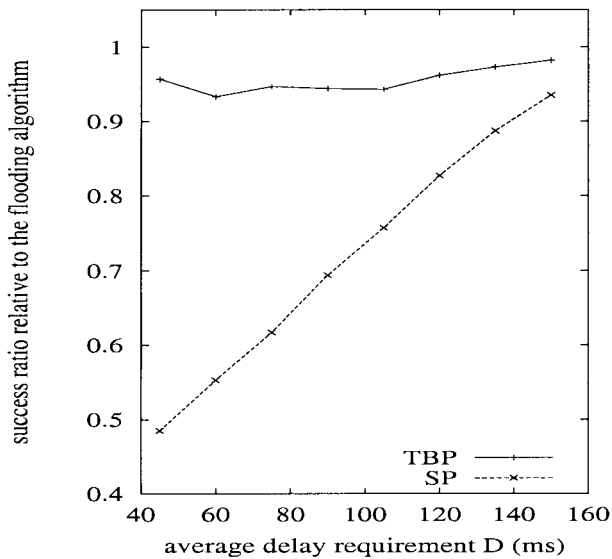Fig. 11.   Messages overhead (imprecision rate: 5%).

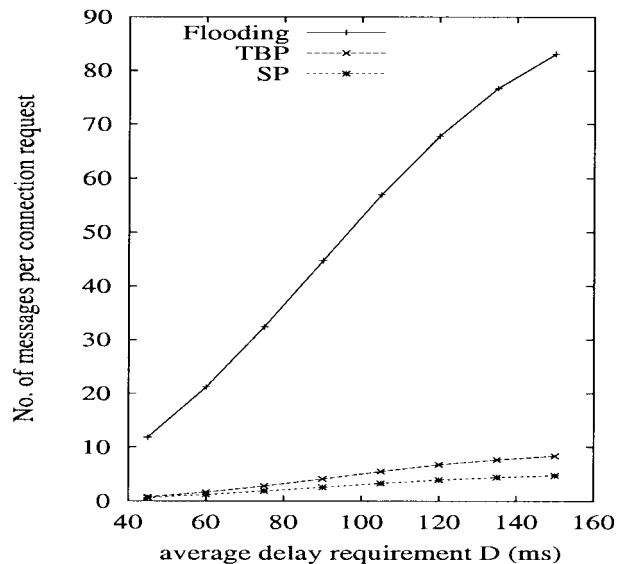Fig. 10.   Relative success ratio (imprecision rate: 50%).

Fig. 12.   Messages overhead (imprecision rate: 10%).

There is one exception in Fig. 18, that the average path cost of TBP is higher than that of SP when D is relatively low. That can be explained as follows: TBP has a much higher success ratio than SP when the imprecision rate is 50%. Those connections, which TBP is able to establish but SP is not, tend to have relatively long routing paths, as observed in the simulation. They also tend to have higher cost, which brings the average path cost up. A fairer comparison is made in Figs. 19 and 20, where only the connections that can be established by SP are considered. The average path cost of TBP is lower than that of SP.

D. Mobility Test

The goal of this test is to evaluate how the node mobility affects the QoS provision. Our mobility model is similar to the one used in [6]. Every node stays at its current location for a period, which is called the station time, and then it moves to a new location that is randomly selected. Each node continues this behavior, alternately staying and moving to a new location. The velocity of moving is randomly taken between 0.1–1 m/s.[13] The time it takes for the node to move to the new location is called the moving time. The mobility ratio of a connection is defined as follows:

$$\text{mobility ratio} = \frac{\text{total moving time}}{\text{total station time} + \text{total moving time}}.$$

By altering the station time, we can change the mobility ratio.

When a new connection arrives, it is routed along a feasible path, on which the required resources are reserved. The time during which the path remains unbroken is called the QoS time. Note that the QoS requirement is ensured during this period of time. However, when the path is broken, it must

---

[13]Recall that all nodes are placed within a 15×15 m² area, and every node has a transmission distance of 3 m.
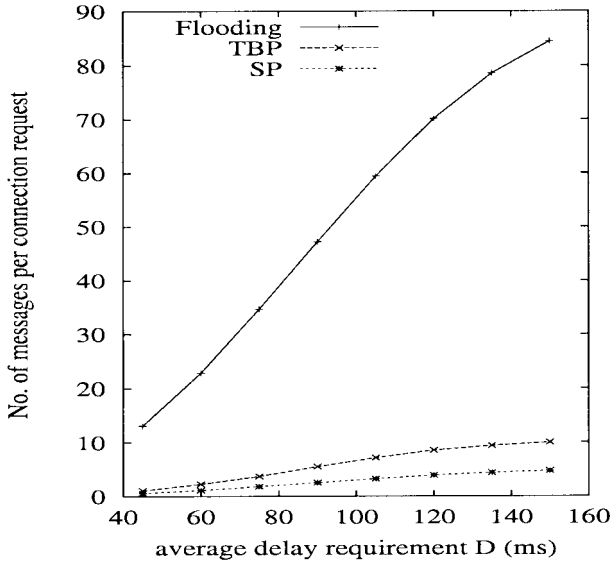
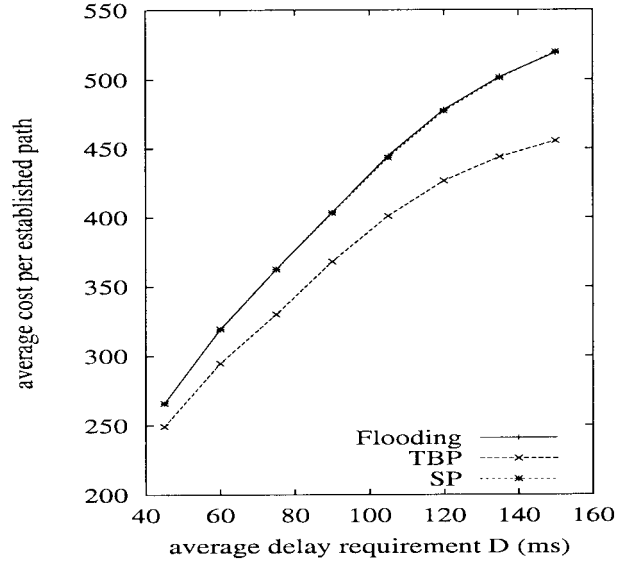Fig. 13. Messages overhead (imprecision rate: 25%).

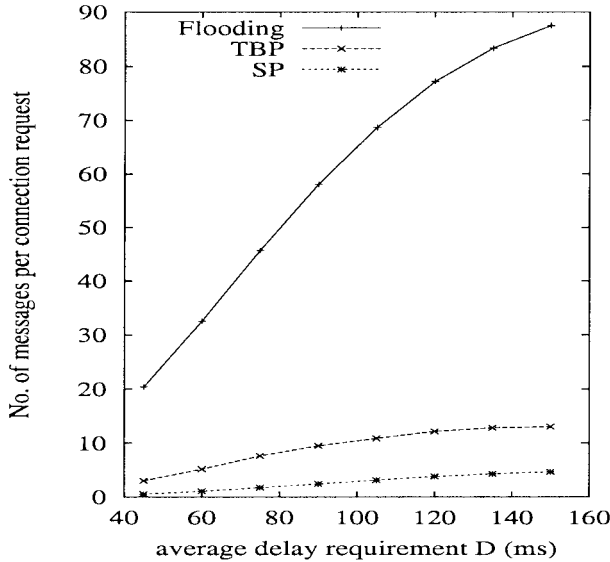Fig. 15. Cost per established path (imprecision rate: 5%).

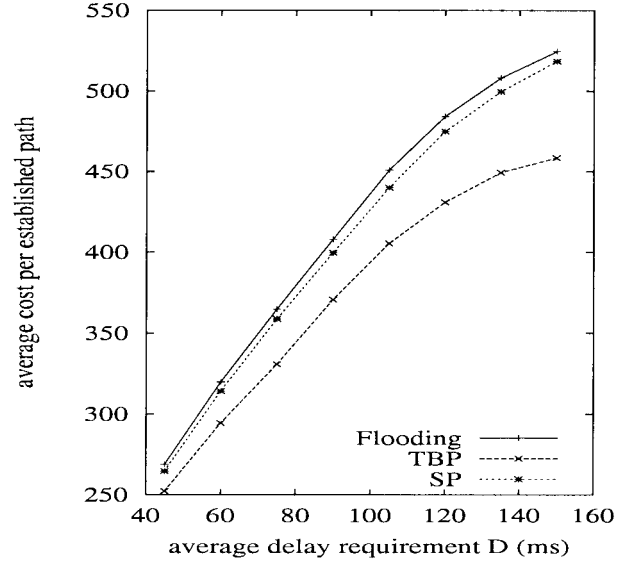Fig. 14. Messages overhead (imprecision rate: 50%).

Fig. 16. Cost per established path (imprecision rate: 10%).

be reestablished. The time it takes to reestablish a new feasible path is called the best-effort time, during which the data packets of are sent as best-effort traffic. We define a performance metric, the QoS ratio, to measure the percentage of a connection's lifetime during which the required QoS is ensured

$$\text{QoS ratio} = \frac{\text{total QoS time}}{\text{total QoS time} + \text{total best-effort time}}.$$

We study how the mobility ratio affects the QoS ratio.

Before we present the result, we shall briefly discuss some implementation decisions made in our simulation. After a routing path is broken, we first try to repair it (see Section VI-D). If the broken path cannot be repaired, we try to reroute it along a new feasible path. If the rerouting fails, we keep increasing the delay requirement by 20 ms each time until a feasible path is found. Hence, we assume the applications

under simulation have the ability to adapt to a looser QoS requirement.[14] We did not use path redundancy in the simulation. Five thousand connection requests are simulated for every mobility ratio. Each connection lasts for 5 min. We assume the control messages (e.g., probes) have a priority lower than the QoS traffic but higher than the best-effort traffic [18]. Note that delay-new(j, k) is the link delay for QoS traffic. The link delay for control messages should be larger. Because we did not implement a specific MAC protocol in the simulation, the delay of a control message over a link, including the queuing delay at the sending node, is randomly generated from [50 ... 200 ms]. The relatively large delay for control messages is a conservative choice because it will increase the rerouting time.

[14] The QoS adaptation is important in an ad hoc network because all feasible paths may be broken in the middle of data transmission due to the mobility of the intermediate nodes.
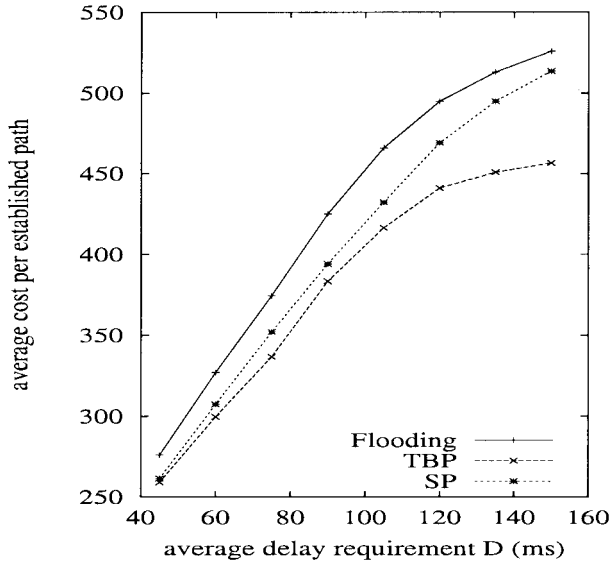
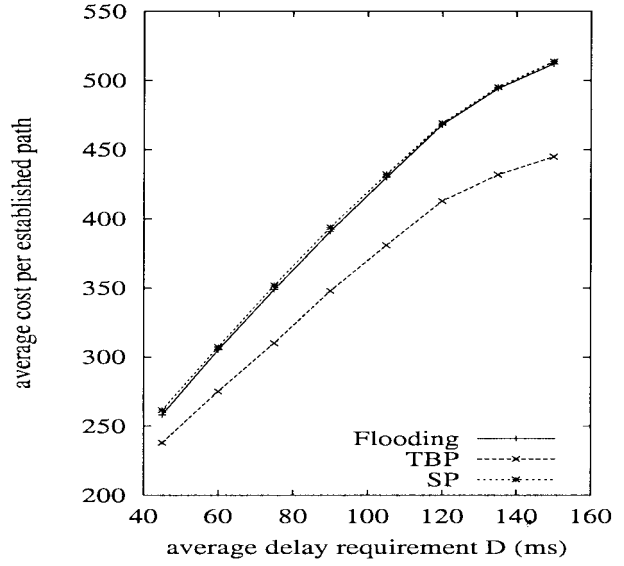Fig. 17.   Cost per established path (imprecision rate: 25%).

Fig. 19.   Comparing average cost by using the same set of established paths (imprecision rate: 25%).
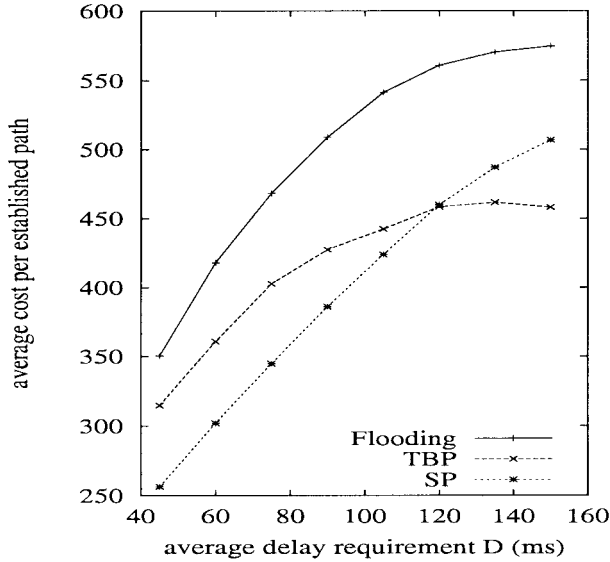

Fig. 18.   Cost per established path (imprecision rate: 50%).
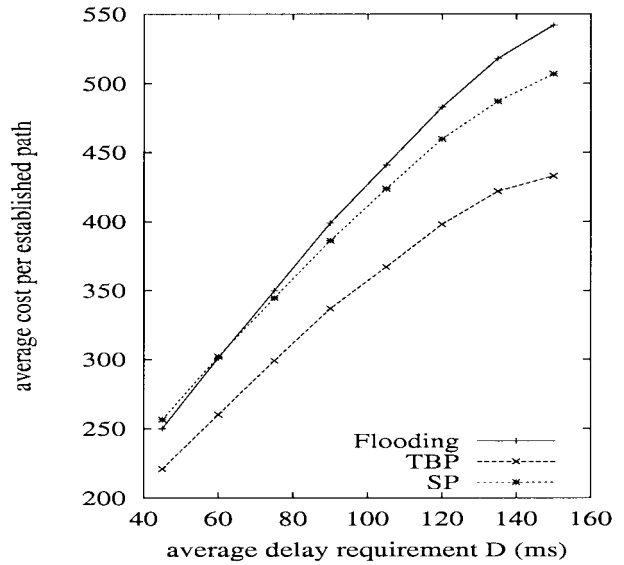

Fig. 20.   Comparing average cost by using the same set of established paths (imprecision rate: 50%).

Fig. 21 shows the QoS ratio with respect to the mobility ratio. The result shows that our routing algorithm only supports soft QoS.[15] The QoS ratio is high when the mobility ratio is low. For a mobility ratio less than 10%, the QoS ratio is above 95%. The QoS ratio decreases as the mobility ratio increases. For a mobility ratio more than 35%, the QoS ratio is below 80%. Hence, our routing algorithm should not be used in the networks with frequent node movements and fast changing topologies.

VIII. RELATED WORK

Much work has been done in two important research areas: QoS routing and ad hoc routing. Most existing QoS routing algorithms were proposed for the wireline networks, and most

---

[15]The guaranteed QoS has a QoS ratio of 100%.

ad hoc routing algorithms support only best-effort routing. The ad hoc QoS routing is a relatively new problem. In this section, we discuss the related work and compare our algorithm with the existing ones.

The problem of QoS routing in wireline networks has been attracting much attention in both academia and industry [34]. Many published algorithms [12], [17], [30], [35] assume the availability of precise global state at every node, which is based on a feasible path that is computed locally at the source node or by a distributed computation [10]. Such an assumption is not true in a dynamic network environment [13], especially if we try to apply these algorithms in a mobile ad hoc network. Guerin and Orda proposed a routing algorithm based on imprecise state information [13]. Their imprecision
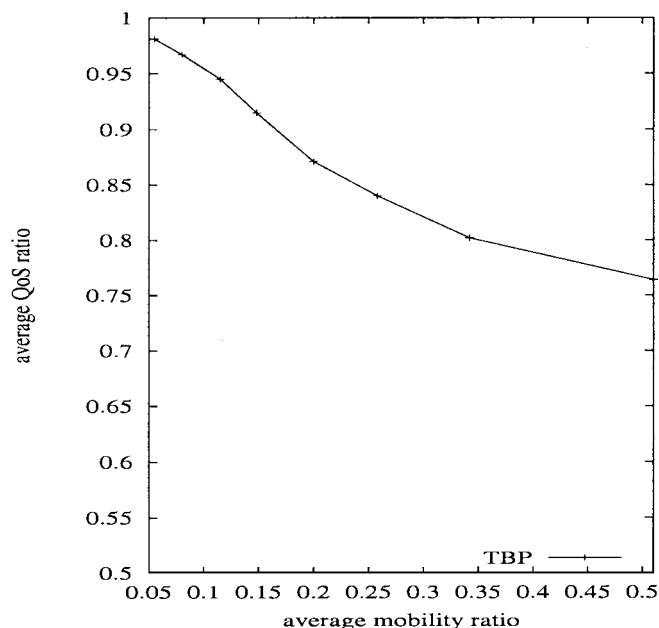
Fig. 21. QoS ratio with respect to mobility ratio.

Fig. 22. TBP does the multipath QoS routing without flooding.

model is based on the probability distribution functions. For instance, for every link l, every node maintains the probability $p_l(d)$ of link l having a delay of d units, where d ranges from zero to the maximum possible value. Such an imprecision model, however, is not suitable for an ad hoc network. It is very difficult to maintain the link-state probability distribution in an ad hoc network, considering that a link may be broken before any meaningful probabilistic data has been gathered. Some recent publications [16], [18], [36] avoid the imprecise state problem by relying only on the up-to-date local state maintained at every node. These algorithms use flooding to collectively utilize the local states to find a feasible path. However, the overhead of flooding is inhabitatively high for the algorithms to be used in the bandwidth-scarce wireless networks, considering that the QoS routing is typically done on a per-connection basis.

Many multipath routing algorithms were proposed for the best-effort data traffic in wireless ad hoc networks [4]–[7], [9]. Most of these algorithms use flooding to discover routing paths.[16] The found path(s) are cached at the source node [6] or at the intermediate nodes [4], [5], [7]. All data traffic from the source to the destination shares these paths. When the paths are broken, various path maintenance algorithms are used to repair the broken paths [5], [7] or reestablish a new one [6].

Our TBP inherits many novel approaches from the previous work. For example, TBP separates route discovery from route maintenance [28], uses stationary links [7], uses probe messages to discover routes hop-by-hop [6], [7], lets the receiver make the path selection [7], avoids the expensive rerouting by path repairing [7], [37], and reduces the chance of service-disruption by path redundancy [4], [5], etc. On the other hand,
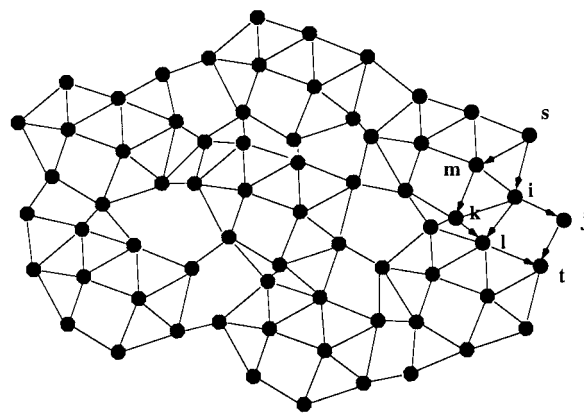
TBP is significantly different from the previous work, which is explained in the following.

First, TBP is designed to support QoS traffic. It finds QoS-constrained paths based on network connectivity and QoS state information. The previous algorithms are designed to support best-effort traffic. They find nonconstrained paths based only on network connectivity. Hence, TBP needs to address the imprecise state problem, which is important for QoS routing, while the previous work does not have this problem.

Second, TBP is executed for every connection request. Such a strategy is inherited from the QoS routing algorithms in wireline networks [13], [17], [19], [30]. The purpose is to increase the success ratio and avoid the traffic concentration. On the other hand, most previous ad hoc routing algorithms cache one or multiple routing paths,[17] which are shared by all data packets from the source to the destination.[18] Such a routing approach works well for best-effort traffic, but is not sufficient for QoS traffic. The reason is that the cached path(s) may not have the resources required by a QoS connection. At the same time, there are numerous other paths connecting the source to the destination that may have the required resources.

Third, TBP does multipath QoS routing without flooding, whereas most previous algorithms are flooding-based. The advantage is illustrated in Fig. 22. Let s and t be the source and the destination, respectively. Suppose s issues three tickets. TBP sends a small number of messages searching three paths. Fig. 22 shows an example where the searched paths are $s \rightarrow m \rightarrow k \rightarrow l \rightarrow t$, $s \rightarrow i \rightarrow l \rightarrow t$, and $s \rightarrow i \rightarrow j \rightarrow t$. On the other hand, a flooding path-discovery algorithm such as DSR [6] or ABR [7] will generate a large number of routing messages that may reach every node in the network, including those remote, less relevant nodes. The design philosophy of TBP is twofold: reducing the overhead by avoiding flooding and meanwhile preventing any significant performance degradation by intelligent hop-by-hop path selection.

---

[16] In [9], each node maintains the information about its local neighborhood, called a zone. The routing overhead is reduced by replacing the hop-by-hop flooding with the much coarse bordercasting [9], in which routing messages travel zone-by-zone.

[17] It is too expensive for the algorithms such as DSR [6] and ABR [7] to execute on a per-connection basis because their path-discovery mechanisms use flooding.

[18] When the paths are broken, all traffic will be swifted to one or multiple reestablished paths.

Chen and Gerla proposed a bandwidth-constrained routing algorithm for ad hoc networks [38]. The algorithm is similar to SP used in our simulation (see Section VII). Both the Chen–Gerla algorithm and TBP use a distance-vector protocol to collect the end-to-end QoS information. However, there are two important differences. First, the Chen–Gerla algorithm does not consider the problem of information imprecision while TBP does. Second, the Chen–Gerla algorithm considers only the shortest path recorded by the distance vector while TBP does multipath routing. We have shown in Section VII that TBP outperforms SP in a dynamic network where the available state information is inherently imprecise.

Sivakumar *et al.* recently proposed a core-extraction distributed routing algorithm (CEDAR) [37] for QoS routing in ad hoc networks. The idea is to maintain a self-organizing routing infrastructure, called the "core," which serves as the spline of the network. Each node in the core covers its n-neighborhood. All nodes in the core must cover the entire network. In the first step, the algorithm broadcasts a message along the core to find where the destination is. It also finds a path P within the core connecting the source to the destination. In the second step, with the searching direction set by P, the algorithm tries to find a bandwidth-constrained path among the nodes in the n-neighborhood of P. CEDAR avoids flooding by the help of P. However, it needs to "flood" within the core to find P, which still makes the routing a "global" operation. Due to the dynamic nature of an ad hoc network, the core may be broken at transient time periods during which the routing cannot be effectively done. Furthermore, searching for a QoS-constrained path is directed by the core. The tree structure of the core may not lead to the discovery of the shortest feasible path that often takes a shortcut between tree branches. On the contrary, TBP does not depend on any backbone structure (core) embedded in the network; its routing process does not involve any "global" operation, and it tries to optimize the routing path.

IX. Conclusion

In this paper, we proposed a ticket-based distributed QoS routing scheme for ad hoc networks. The existing single-path routing algorithms have low overhead but do not have the flexibility of dealing with imprecise state information [38]. On the other hand, the flooding algorithms can handle information imprecision but have prohibitively high overhead. Our ticket-based probing scheme achieves a balance between the single-path routing algorithms and the flooding algorithms. It does multipath routing without flooding. The basic idea is to achieve a near-optimal performance with modest overhead by using a limited number of tickets and making intelligent hop-by-hop path selection. Simulations showed that the proposed routing scheme has a high success ratio and finds low-cost feasible paths with an overhead significantly lower than that of a flooding algorithm.

Various approaches were proposed to detect and reestablish broken paths. While rerouting was used as the primary approach to deal with the path-breaking problem, other approaches were proposed to reduce the jitter in the QoS provision. In particular, multiple levels of path redundancy were used to tolerate the topology dynamics, and the dynamic path repairing was used to repair the broken paths at the breaking point.

References

[1] C. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance vector routing (DSDV) for mobile computers," in *Proc. ACM SIGCOMM'94*, pp. 234–244.

[2] S. Murthy and J. J. Garcia-Luna-Aceves, "An efficient routing protocol for wireless networks," *ACM Mobile Networks Applicat. J.*, Special Issue on Routing in Mobile Communication Networks, 1996.

[3] E. Gafni and D. D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Trans. Commun.*, vol. COMM-29, Jan. 1981.

[4] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wireless Networks*, vol. 1, pp. 61–81, 1995.

[5] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proc. IEEE INFOCOM'97*, pp. 1405–1413.

[6] D. Johnson and D. Maltz, "Dynamic source routing in *ad hoc* wireless networks," *Mobile Computing,* E. Imielinski and H. Korth, Eds. Norwell, MA: Kluwer, 1996.

[7] C.-K. Toh, "Associativity-based routing for *ad-hoc* mobile networks," *Wireless Personal Commun.*, vol. 4, pp. 103–139, 1997.

[8] R. Sivakumar, B. Das, and V. Bharghavan, "An improved spine-based infrastructure for routing in *ad hoc* networks," in *Proc. IEEE Symp. Computers and Communications*, 1998.

[9] Z. J. Haas and M. R. Pearlman, "The performance of query control schemes for the zone routing protocol," in *Proc. ACM SIGCOMM'98*.

[10] S. Chen and K. Nahrstedt, "An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions," *IEEE Network*, Special Issue on Transmission and Distribution of Digital Video, pp. 64–79, Nov./Dec. 1998.

[11] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts, "Throughput-competitive on-line routing," in *Proc. 34th Ann. Symp. Foundations of Computer Science,* Palo Alto, CA, Nov. 1993.

[12] S. Chen and K. Nahrstedt, "On finding multi-constrained paths," in *Proc. IEEE ICC'98*, pp. 874–879.

[13] R. Guerin and A. Orda, "QoS-based routing in networks with inaccurate information: Theory and algorithms," in *Proc. IEEE INFOCOM'97*, Japan, pp. 75–83.

[14] J. Moy, "OSPF Version 2," Internet RFC 1583, Mar. 1994.

[15] I. Cidon, R. Rom, and Y. Shavitt, "Multi-path routing combined with resource reservation," in *Proc. IEEE INFOCOM'97,* Japan, pp. 92–100.

[16] C. Hou, "Routing virtual circuits with timing requirements in virtual path based ATM networks," in *Proc. IEEE INFOCOM'96*, pp. 320–328.

[17] H. F. Salama, D. S. Reeves, and Y. Viniotis, "A distributed algorithm for delay-constrained unicast routing," in *Proc. IEEE INFOCOM'97*, Japan, pp. 84–91.

[18] K. G. Shin and C.-C. Chou, "A distributed route-selection scheme for establishing real-time channel," in *Proc. 6th IFIP Int. Conf. High Performance Networking (HPN'95).*

[19] ATM Forum, "Private Network Network Interface (PNNI) v1.0 Specifications," June 1996.

[20] D. H. Lorenz and A. Orda, "QoS routing in networks with uncertain parameters," in *Proc. IEEE INFOCOM'98*.

[21] S. Cen, C. Pu, R. Staehli, and J. Walpole, "A distributed real-time MPEG video audio player," in *Proc. NOSSDAV'95*.

[22] F. Goktas, J. Smith, and R. Bajcsy, "Telerobotics over communication networks: Control and networking issues," in *Proc. 36th IEEE Conf. Decision and Control,* 1997.

[23] N. Tran and K. Nahrstedt, "Active adaptation by program delegation in VOD," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems,* 1998, pp. 96–107.

[24] A. Alwan, R. Bagrodia, N. Bambos, M. Gerla, L. Kleinrock, J. Short, and J. Villasenor, "Adaptive mobile multimedia networks," *IEEE PCS Mag.*, 1996.

[25] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman, 1979.
[26] M. S. Corson, S. Papademetriou, P. Papadopoulos, V. D. Park, and A. Qayyum, "An internet MANET encapsulation protocol (IMEP) specification," Internet-Draft, draft-ietf-manet-imep-spec01.txt, Aug. 1998.
[27] J. Tsai and M. Gerla, "Multicluster, mobile, multimedia radio network," *ACM-Baltzer J. Wireless Networks,* vol. 1, no. 3, pp. 255–265, 1995.
[28] D. B. Johnson, "Routing in *ad hoc* networks of mobile hosts," in *Proc. IEEE Workshop Mobile Computing Systems and Applications*, Dec. 1994.
[29] Q. Ma and P. Steenkiste, "Quality-of-service routing with performance guarantees," in *Proc. 4th Int. IFIP Workshop Quality of Service*, May 1997, pp. 115–126.
[30] Z. Wang and J. Crowcroft, "QoS routing for supporting resource reservation," *IEEE J. Select. Areas Commun.*, Sept. 1996.
[31] C. Parris, H. Zhang, and D. Ferrari, "A mechanism for dynamic re-routing of real-time channels," International Computer Science Institute, Berkeley, CA, Tech. Rep. TR-92-053, Apr. 1992.
[32] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Network*, Sept. 1993, pp. 8–18.
[33] H. Zhang and S. Keshav, "Comparison of rate-based service disciplines," in *Proc. ACM SIGCOMM'91*, pp. 113–122.
[34] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the internet," Internet-Draft, draft-ietf-qosr-framework-06.txt, Aug. 1998.
[35] Q. Ma, P. Steenkiste, and H. Zhang, "Routing high-bandwidth traffic in max–min fair share networks," in *Proc. ACM SIGCOMM'96*.
[36] K. Carlberg and J. Crowcroft, "Building shared trees using a one-to-many joining mechanism," *ACM Comput. Commun. Rev.*, pp. 5–11, Jan. 1997.
[37] R. Sivakumar, P. Sinha, and V. Bharghavan, "Core extraction distributed *ad hoc* routing (CEDAR) specification," IETF Internet draft draft-ietf-manet-cedar-spec-00.txt, 1998.
[38] T. Chen, M. Gerla, and J. T. Tsai, "QoS routing performance in a multi-hop, wireless networks," in *Proc. IEEE ICUPC'97*.

Shigang Chen received the B.S. degree from the University of Science and Technology of China and the M.S. degree from the University of Illinois at Urbana-Champaign (UIUC), both in computer science. He will receive the Ph.D. degree in computer science from UIUC in May 1999.

His research interests include quality-of-service, multimedia networking, resource management, and distributed computing.

Klara Nahrstedt (S'93–M'95) received the A.B. and M.Sc. degrees in mathematics from Humboldt University, Berlin, Germany, and the Ph.D. degree in computer science from the University of Pennsylvania.

She is an Assistant Professor in the Computer Science Department at the University of Illinois at Urbana-Champaign, where she does research on quality-of-service (QoS)-aware systems with emphasis on end-to-end resource management, routing, and middleware issues for distributed multimedia systems. She is the coauthor of the widely used multimedia book, *Multimedia: Computing, Communications, and Applications* (Englewood Cliffs, NJ: Prentice-Hall, 1995).

Dr. Nahrstedt is the recipient of the Early NSF Career Award and the Junior Xerox Award for Research Achievements.