Distributed QoS Routing with Imprecise State Information *

Shigang Chen, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{s-chen5, klara} @cs.uiuc.edu

Abstract

The goal of Quality-of-Service (QoS) routing is to find a network path which has sufficient resources to satisfy certain constraints on delay, bandwidth and/or other metrics. The network state information maintained at every node is often imprecise in a dynamic environment because of non-negligible propagation delay of state messages, periodic updates due to overhead concern, and hierarchical state aggregation [6]. The information imprecision makes QoS routing difficult. The traditional shortest-path routing algorithm does not provide satisfactory performance with imprecise state information.

We propose a distributed routing scheme, called ticket-based probing, which searches multiple paths in parallel for a satisfactory one. The scheme is designed to work with imprecise state information. It allows the dynamic tradeoff between the routing performance and the overhead. The state information of intermediate nodes is collectively used to guide the routing messages along the most appropriate paths in order to maximize the success probability. The proposed algorithm consider not only the QoS requirements but also the cost optimality of the routing path. Extensive simulations show that our algorithm achieve high call-admission ratio and low-cost routing paths with modest overhead. The algorithm can tolerate high degree of information imprecision.

1 Introduction

The up-coming Gbps high-speed networks are expected to support a wide range of communication-intensive, real-time applications. The quality-of-service (QoS) requirements of these applications raise new challenges for the development of integrated-service network systems. One of the key issues is QoS routing. The goal of routing solutions is two-folded: (a) selecting network routes that have sufficient resources to meet the QoS requirements of every admitted connection and (b) achieving global efficiency in resource utilization.

Routing consists of two basic tasks. The first task is to collect the state information and keep it up-to-date. The second task is to find a satisfactory path for a new connection based on the collected information. Most published routing algorithms [2, 8, 9, 11] require every node to maintain a *global network state* either by a distance-vector protocol or by a link-state protocol. However, such a global state is inheritly *imprecise* in a dynamic network where the traffic load changes constantly. The imprecision is especially noticeable in large wide-area networks due to the following reasons. First, it takes non-negligible propagation delay for a local state change to be broadcasted to other nodes. Second, a distance-vector (or link-state) protocol updates the state information periodically or upon triggering when significant state change is detected. There exists a tradeoff between the update frequency and the overhead involved. For large scale networks, the excessive communication overhead often makes it impractical for the update frequency to be high enough to cope with the dynamics of network parameters such as bandwidth and delay. Third, the hierarchical approach is likely to be used to solve the scalability problem of routing in large networks [4]. However, the state aggregation in hierarchical routing increases the level of imprecision [6].

We propose a distributed QoS routing scheme that works with imprecise state information. While in this paper we only study the NP-complete *delay-constrained least-cost routing* problem, our scheme can also be applied to other routing problems such as *bandwidth-constrained least-cost routing* [1]. A path which satisfies the delay (or bandwidth) constraint is called a *feasible path*. The design goal of our heuristic algorithm is to find a low-cost feasible path by using only the available imprecise information.

The most related work was done by Guerin and Orda [6] and by Lorenz and Orda [7]. There are several important differences which distinguish our work from theirs.

First, the imprecision model in [6, 7] is based on the probability distribution functions. For instance, every node maintains, for every link l, the probability $p_l(w)$ of link l having a delay of d units, where d ranges from zero to maximum possible value. The problem of how to maintain the prob-

*This work was supported by the Airforce grant under contract number F30602-97-2-0121 and the National Science Foundation Career grant under contract number NSF CCR 96-23867.

ability distribution was not discussed. If the distribution is collected over time, it ignores the *current, specific* traffic context, that may be very useful to find the best path. Our imprecision model is much simpler. Take the delay metric for example. A node i maintains two values for every possible destination t. They are (1) $D_i(t)$, an estimation of the end-to-end delay from i to t, and (2) $\Delta D_i(t)$, an estimation of the maximum change of $D_i(t)$ for the next update period. $\Delta D_i(t)$ can be easily computed from the recent state history.

Second, the goal of [6, 7] is to maximize the probability of finding a feasible path. The optimality of such a path, which is probably equally important, is ignored. On the contrary, the goal of our algorithms is to find a *low-cost* feasible path.

Third, the algorithms in [6, 7] are source routing algorithms. The routing path is locally computed at the source node. Our algorithm does distributed routing. The path is computed by a restricted diffusion computation [3]. It avoids the expensive centralized computation at the source, and collectively utilizes the state information kept at intermediate nodes to find a path. Multi-path parallel routing is used to increase the probability of finding a low-cost feasible path. A technique, called *ticket-based probing*, is used to flexibly bound the overhead.

2 System Models

2.1 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. Each node i keeps the up-to-date local state about all outgoing links. The state information of link (i, j) includes 1) $delay(i, j)$, consisting of the queueing delay at node i and the propagation delay along the link, and 2) $cost(i, j)$, which can be simply one as a hop count or a function of the link utilization. The delay and cost of a path $P = i \rightarrow j \rightarrow ...k \rightarrow l$ are defined as follows

$$delay(P) = delay(i, j) + ... + delay(k, l)$$
$$cost(P) = cost(i, j) + ... + cost(k, l)$$

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 s.t. $delay(P) \leq D$.

When there are multiple feasible paths, we want to select the one with the least cost. Finding the delay-constrained least-cost path is NP-complete [5].

2.2 Imprecise state model

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

1) Connectivity: $R_i(t)$ is a routing table entry, keeping a subset of adjacent nodes that can be used to route data traffic to t.

2) Delay: $D_i(t)$ keeps the minimum end-to-end delay from i to t, i.e., the delay of the *least-delay path.*

3) Cost: $C_i(t)$ keeps the least end-to-end cost of the paths from i to t, i.e., the cost of the *least-cost path.*

The above information, especially $D_i(t)$, is inheritly *imprecise* in a dynamic network, as discussed in Section 1. We propose a simple imprecise state model which can be easily implemented. An additional state variable is required.

4) 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.

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

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

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

By the above 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 negligibly small is to enlarge $\Delta D_i(t)$. Hence, we shall modify the formula and introduce another factor β (> 1).

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

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

For the purpose of simplicity, we do not apply the imprecise model on $R_i(t)$ and $C_i(t)$. Such a simplification will not degrade the routing performance significantly because of the following reasons. (1) The topology ($R_i(t)$) of the network can change, but is relatively infrequent comparing to the QoS state such as delay. (2) The cost metric ($C_i(t)$) is used for optimization, in contrast to the delay metric used in QoS constraints. Since there is not a strict cost bound requirement, certain degree of imprecision for $C_i(t)$ is tolerable.

3 Routing by Ticket-Based Probing

We propose a multi-path [1] distributed routing scheme, called *ticket-based probing*. The basic idea is outlined in this section while the operational details about delay-constrained routing are discussed in Sections 4.

[1] Search multiple paths for a feasible one.

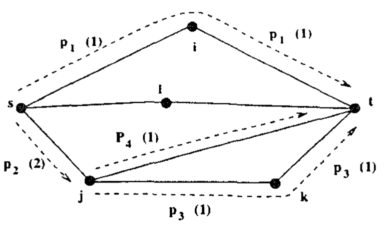

Figure 1. Two probes, p_1 and p_2, are sent from s. The number between the parentheses following a probe is the number of tickets carried. At node j, p_2 is split into p_3 and p_4, each of which has one ticket. There are at most three probes at any time. Three paths are searched and they are $s \to i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

Probes (routing messages) are sent from the source s toward the destination t to search for a low-cost path which satisfies the delay requirement. Certain number of tickets are issued at the source according to the contention level of network resources. Each probe is required to carry at least one ticket. Hence, 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 Figure 1 for an example. The routing scheme utilizes the state information at 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 *nice* 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. For those QoS requirements that can be easily satisfied according to the current state condition, a few or even one ticket are sufficient to find a feasible path. For other requirements that are harder to be satisfied, more tickets are issued.

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. Multi-path parallel search increases the chance of finding a feasible path and thus provides a means to tolerate information imprecision.

3) Our scheme considers not only the QoS requirement but also the optimality of the selected path. Low-cost paths are given preference in order to improve the overall network performance.

4 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. 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 tickets among the new probes, and then forwards them along selected outgoing links toward t. Probes can only travel along the paths that satisfy the delay requirement. Hence, any probe arriving at the destination detects a feasible path.

There are two problems: (1) how to determine N_0, and (2) how to distribute the tickets of a received probe among the new probes. The first problem is solved in Section 4.1 and the second problem is solved in Section 4.2.

4.1 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 (more precisely, probes carrying them) prefer paths with smaller delays, so that the chance of satisfying a given delay requirement is higher.

The number of yellow tickets, Y_0, is determined by 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 tickets are issued to search multiple paths for a feasible one. The curve of Y_0 is given in Figure 2 (upper curve), which will be explained in details in Section 4.1.1.

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

The number of green tickets, G_0, is also determined by the delay requirement. The curve of G_0 is given in Figure 2 (lower curve), which will be explained in details in Section 4.1.2.

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.

4.1.1 Number of yellow tickets, Y_0

Consider a connection request whose source, destination and delay requirement are s, t and D, respectively. Y_0 is determined by $D_s(t)$, $\Delta D_s(t)$ and D. The curve of Y_0 with respect to D is shown in Figure 2 (upper curve), which is explained in the following.

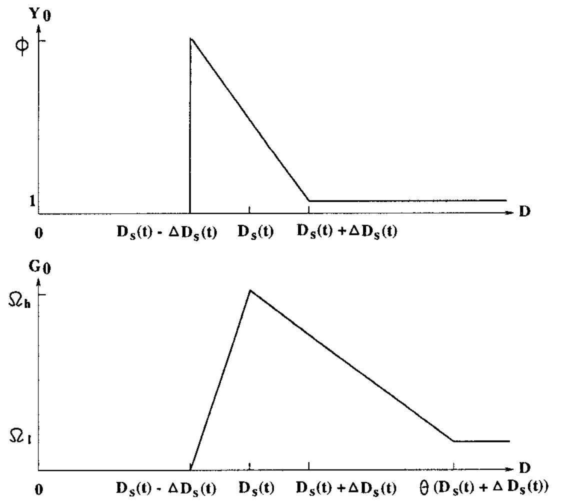

Figure 2. Curves of Y_0 and G_0 with respect to D

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 $(D_s(t) + \Delta D_s(t))$, [2] a single yellow ticket will be sufficient to find a feasible path. How to propagate the ticket will be discussed in Section 4.2.

2. If $D_s(t) - \Delta D_s(t) \leq D < D_s(t) + \Delta D_s(t)$, then $Y_0 = \lceil \frac{D_s(t)+\Delta D_s(t)-D}{2\times\Delta D_s(t)} \times \Phi \rceil$, where Φ is a system parameter specifying the maximum allowable number of yellow tickets. 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 not be satisfied. The connection request is rejected and the QoS negotiation process is activated for a relaxed delay bound.

4.1.2 Number of green tickets, G_0

The curve of G_0 with respect to D is shown in Figure 2 (lower curve), which is explained in the following.

1. If $D \geq \theta \times (D_s(t) + \Delta D_s(t))$, then $G_0 = \Omega_l$, where θ (> 1) and Ω_l (≥ 1) are system parameters. θ specifies a threshold, $\theta \times (D_s(t) + \Delta D_s(t))$, beyond which D is considered to be *sufficiently* large. When D is sufficiently large, the minimum number (Ω_l) of green tickets 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 values for θ and Ω_l. $\theta \in [1.5, 2.0]$ and $\Omega_l \in [1..3]$ worked fine in our simulation (Section 5).

2. If $D_s(t) \leq D < \theta \times (D_s(t) + \Delta D_s(t))$, then $G_0 = \lceil \frac{\theta\times(D_s(t)+\Delta D_s(t))-D}{\theta\times(D_s(t)+\Delta D_s(t))-D_s(t)} \times (\Omega_h - \Omega_l) \rceil + \Omega_l$, where Ω_h is the maximum allowable number of green tickets. As D decreases, G_0 increases.

3. If $D_s(t) - \Delta D_s(t) \leq D < D_s(t)$, then $G_0 = \lceil \frac{D-D_s(t)+\Delta D_s(t)}{\Delta D_s(t)} \times \Omega_h \rceil$. As D approaches to $D_s(t) - \Delta D_s(t)$, the delay requirement becomes increasingly harder to be satisfied. 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 $D < D_s(t) - \Delta D_s(t)$, $G_0 = 0$. The connection request is rejected.

Theoretically, Φ (or Ω_h) can be $+\infty$, which makes the ticket-based probing a flooding scheme. In practice, a value of 10 or less for Φ (or Ω_h) should be sufficient according to our simulation. We can also take the source-destination distance into account — Φ and Ω_h are larger when the distance is larger.

Other than Figure 2, there can be many different curves for Y_0 and G_0, depending on the tradeoff objective between the routing performance and the overhead.

4.2 Ticket-based probing

If $Y_0 + G_0 = 0$, the connection request is rejected. Otherwise, probes carrying the tickets are sent from s toward t. A probe proceeds only when the path has a delay no more than D. Hence, once the 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), $delay(p) := delay(p) + delay(i, j)$.

The distribution of the tickets is described as follows: 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. Let $R_i^p(t) = \{j \mid delay(p) + delay(i,j) + D_j(t) - \Delta D_j(t) \leq D, j \in R_i(t) - \{k\}\}$. There is no need to send any ticket to $j \in R_i(t) - R_i^p(t)$, because the best expected delay from j to t, which is $D_j(t) - \Delta D_j(t)$, plus $delay(p)$ and $delay(i,j)$ violates the delay requirement. If $R_i^p(t) = \emptyset$, invalidate all received tickets and discard them. Otherwise, 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 show how to calculate $Y(p_j)$ and $G(p_j)$ in the following.

[2]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.

4.2.1 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.

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

$Y(p_j)$ calculated by the above 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 $\sum_{j \in R_i^p(t)} Y(p_j) = Y(p)$.

In order to calculate $Y(p_j)$, node i must maintain the value of $D_{j'}(t)$, for every adjacent node j', which can be easily realized either in a distance-vector protocol or in a link-state protocol. Another possible implementation is to inquire $D_{j'}(t)$ from j' and cache the value locally. The same thing is true for $C_{j'}(t)$.

4.2.2 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. $\forall j \in R_i^p(t)$,

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

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

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

In the above 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,[3] (2) the number of hops a probe can traverse is bounded, i.e., over-aged probes will be discarded, or (3) a probe records its path to detect the possible loops.

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 a combination of delay and cost. We omit the discussion on this type of tickets due to the space limitation.

Our ticket-based probing algorithm nicely 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 along the *least-delay path* to the destination, and in addition, several (Ω_l) green tickets are issued, trying to find a feasible path with a lower cost.

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

The routing process is activated by the source node, which sends itself a probe with Y_0 yellow tickets and G_0 green tickets. New probes will then be created and sent to t. The termination of the routing process is discussed in Section 4.3.

4.3 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 [4] 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. [5] Timeout is used to handle the problem of message (tickets) losses. 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.

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 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. The second approach is appropriate for an ATM network where a probe with a constant, smaller size is more likely to be able to fit into a single cell.

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.

4.4 Data Structure

The data structure of a probe p is as follows.

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

The last five fields, k, $Y(p)$, $G(p)$, *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

[4]The tickets in a received probe are invalidated if $R_i^p(t) = \emptyset$ (Section 4.2).

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

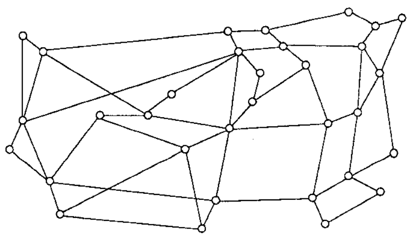

Figure 3. network topology

Φ	θ	Ω_l	Ω_h
4	1.5	2	6

Table 1. System parameters for TBP

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.

5 Simulation and Results

Extensive simulations were done to evaluate the proposed ticket-based probing algorithm. Three performance metrics, ***success ratio***, ***average message overhead*** and ***average path cost***, are defined as follows.

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

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

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

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

The network topology used in our simulation is shown in Figure 3, which expends the major circuits in ANSNET by inserting additional links to increase the connectivity. 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 $[30, 160ms]$. 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 $[0, 50ms]$, while *delay-new*(j,k) is uniformly distributed in $[(1-\xi) \times$ *delay-old*(j,k), $(1+\xi) \times$ *delay-old*$(j,k)]$, where ξ is a simulation parameter, called ***imprecision rate***, specifying the largest percentage difference of *delay-new*(j,k) from *delay-old*(j,k).

$$\xi = supremum\{\frac{|delay\text{-}new(j,k) - delay\text{-}old(j,k)|}{delay\text{-}old(j,k)}\}$$

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

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 [10], 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 there exists one 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 1. The values in the table are obtained by extensive simulation runs. See Section 4 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 $D \le D_s(t)$, the algorithm sends out one routing message along the least-delay path to check the current resource availability for possible connection establishment.

5.1 Success ratio

Figures 4-5 compare the success ratios of the three algorithms. The success ratio is a function of both ***average delay requirement*** D and ***imprecision rate*** ξ. The former is represented by the x axis and the later is shown by different figures. In each figure, as the delay requirement becomes larger, it becomes easier to be satisfied and thus the success ratio is higher. The flooding algorithm, as expected, has the best success ratio. The success ratio of TBP is very close

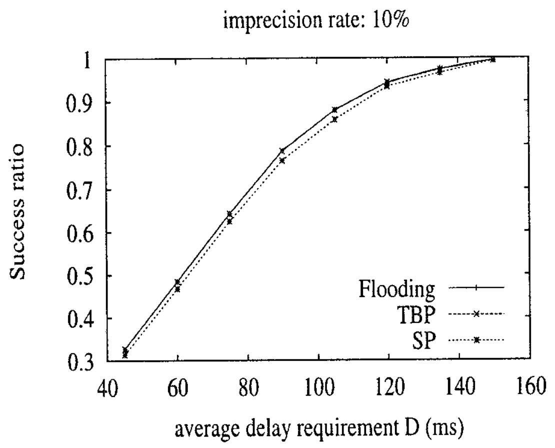

Figure 4. success ratio (imprecision rate: 10%)

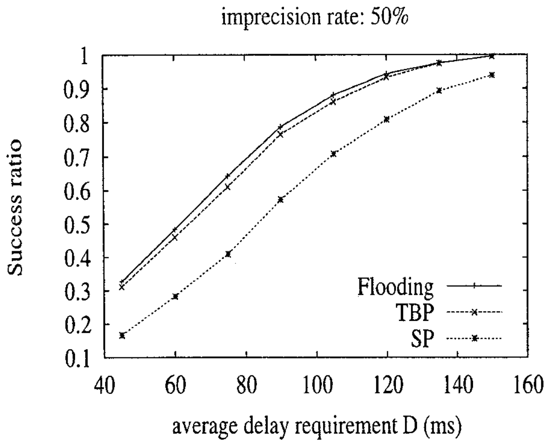

Figure 5. success ratio (imprecision rate: 50%)

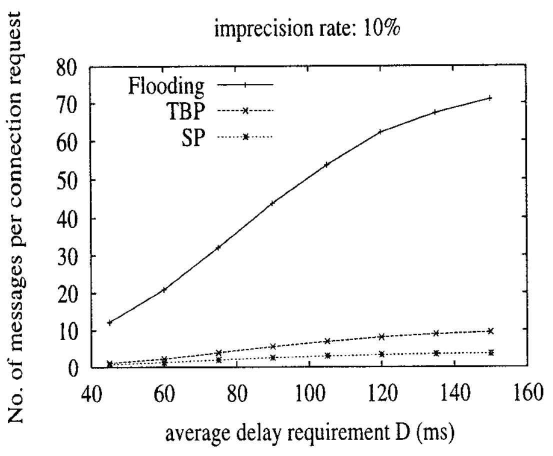

Figure 6. message overhead (imprecision rate: 10%)

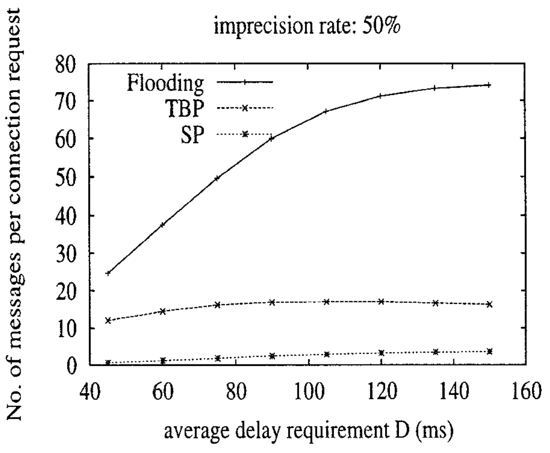

Figure 7. message overhead (imprecision rate: 50%)

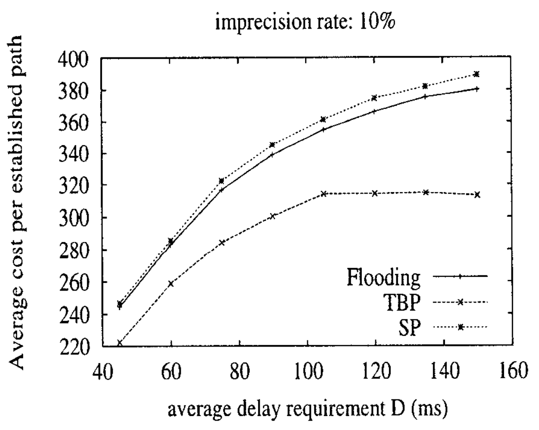

Figure 8. cost per established path (imprecision rate: 10%)

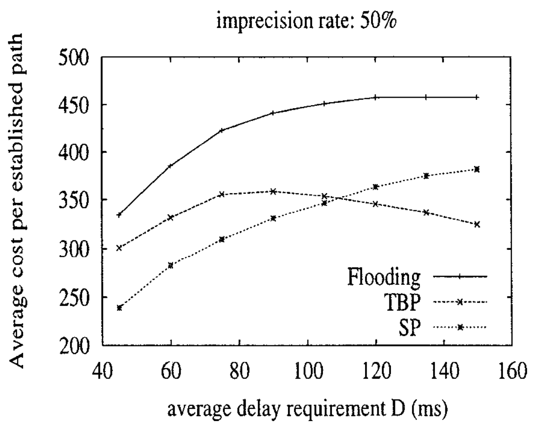

Figure 9. cost per established path (imprecision rate: 50%)

to that of the flooding algorithm, even when the imprecision rate is as high as 50%. This is because TBP searchs multiple paths and the number of paths searched is adjusted according to how hard 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 towards the destination. In contrast, the SP algorithm performs much worse when the imprecision rate is high.

5.2 Message overhead

Figures 6-7 compare the average message overhead of the three algorithms. The flooding algorithm has a very high message overhead. SP has the lowest overhead. TBP has an overhead higher than that of SP but much lower than that of the flooding algorithm. The overhead of TBP increases as the imprecision rate increases. Readers are referred to [1] for some simple heuristics which reduce the message overhead of TBP significantly.

5.3 Average path cost

Figures 8-9 compare the average path cost of the three algorithms. In Figure 8, TBP has a much lower 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.

In Figure 9, however, 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, that TBP is able to establish but SP is not able to, 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 Figure 10, where only the connections which can be established by SP are considered. The average path cost of TBP is lower than that of SP.

6 Conclusion

In this paper, we proposed a ticket-based distributed QoS routing scheme which works for dynamics networks where the global state information maintained at every node is imprecise. Our simulations show that the scheme achieves high success ratio and low-cost feasible paths with modest overhead. It can tolerate high degree of information imprecision.

Our on-going work includes (1) a theoretical approach to formally analyze the performance of the proposed routing algorithm, (2) a simulation study on how different system parameters affect the routing performance and how to choose their values and (3) an efficient way to record the routing paths and dynamically optimize these paths during the process of the ticket-based probing.

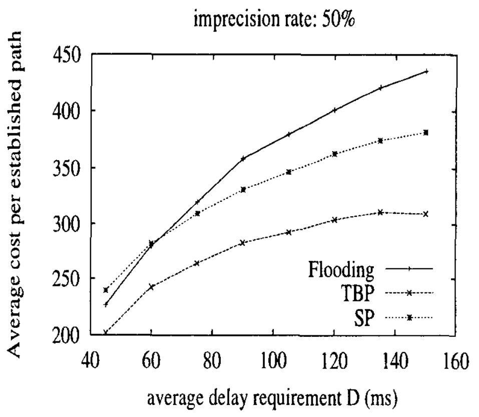

Figure 10. Using the same set of established paths to compare the average path cost (imprecision rate: 50%)

References

[1] S. Chen and K. Nahrstedt. Distributed qos routing with imprecise state information. *Technical Report, University of Illinois at Urbana-Champaign, Department of Computer Science*, 1998.

[2] S. Chen and K. Nahrstedt. On finding multi-constrained paths. *IEEE International Conference on Communications*, June 1998.

[3] E. W. Dijkstra and C. S. Scholten. Termination detection for diffusion computations. *Inform. Process. Lett.*, 11(1):833–837, August 1980.

[4] A. Forum. Private network network interface (pnni) v1.0 specifications. May 1996.

[5] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman and Co., 1979.

[6] R. Guerin and A. Orda. Qos-based routing in networks with inaccurate information: Theory and algorithms. *Infocom'97, Japan*, April 1997.

[7] D. H. Lorenz and A. Orda. Qos routing in networks with uncertain parameters. *Infocom'98*, March 1998.

[8] Q. Ma and P. Steenkiste. Quality-of-service routing with performance guarantees. *Proceedings of the 4th International IFIP Workshop on Quality of Service*, May 1997.

[9] H. F. Salama, D. S. Reeves, and Y. Viniotis. A distributed algorithm for delay-constrained unicast routing. *INFOCOM'97, Japan*, April 1997.

[10] K. G. Shin and C.-C. Chou. A distributed route-selection scheme for establishing real-time channel. *Sixth IFIP Int'l Conf. on High Performance Networking Conf. (HPN'95)*, pages 319–329, Sep. 1995.

[11] Z. Wang and J. Crowcroft. Qos routing for supporting resource reservation. *JSAC*, September 1996.