Distributed Quality-of-Service Routing in High-Speed Networks Based on Selective Probing [*]

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

Abstract

We propose an integrated QoS routing framework based on selective probing for high-speed packet-switching networks. The framework is fully distributed and depends only on the local state maintained at every individual node. By using controlled diffusion computations, the framework captures the common messaging and computational structure of distributed QoS routing, and allows an efficient implementation due to its simplicity. Different distributed routing algorithms (DRAs) can be quickly developed by specifying only a few well-defined constraint-dependent parameters within the framework. Our simulation shows that the overhead of the proposed algorithms is stable and modest.

1 Introduction

Distributed multimedia applications have quality-of-service (QoS) requirements specified in terms of constraints on various metrics such as bandwidth, delay, delay jitter, cost, etc. The task of QoS routing is to find a path from the source node to the destination node which has sufficient resources to support the required end-to-end QoS.

The recent work in QoS routing has been following two main directions: *source routing* and *distributed routing.* In the source routing, each node maintains an image of the *global network state*, based on which a routing path is centrally computed at the source. The global network state is typically updated periodically by a link-state algorithm. In the distributed routing, 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.

The source routing scheme [4, 6, 8] has several problems. First, the global network state has to be updated frequently enough to cope with the dynamics of network parameters such as bandwidth and delay, which makes the communication overhead excessively high for large scale networks. Second, the link-state algorithm commonly used in the source routing can only provide *approximate* global state due to the overhead concern and non-negligible propagation delay of state messages. The inaccuracy in the global state may cause the QoS routing fail. Third, the link-state algorithm has the scalability problem [1]. It is impractical for any single node to have access to detailed state information about all nodes and all links in large networks. The hierarchical routing is used as a solution [5]. However, the state aggregation increases the level of inaccuracy [6]. Fourth, the computation overhead at the source is excessively high, especially when multiple constraints are involved, considering that the QoS routing is typically done on a per-connection basis.

In the distributed routing, the path-selection computation is distributed among the intermediate nodes between the source and the destination. Hence, the routing response time can be made shorter and the algorithm is more scalable. However, most existing distributed routing algorithms [9, 11] still require each node to maintain a global network state, based on which the routing decision is made on a hop-by-hop basis. The routing performance heavily depends on the accuracy of the global state. Therefore, these algorithms more or less share the same problem of the source routing.

Multimedia applications tend to have diverse QoS requirements on bandwidth, delay, delay jitter, cost, path length, etc. From a network designer's point of view, it would be beneficial to accommodate different QoS routing algorithms in a single integrated framework, which captures the common messaging and computational structure. The framework should be *simple*, which enables an efficient implementation, and *extensible*, so that new QoS metrics can be easily added without affecting the existing ones. It should also support both unicast and multicast. Besides, a distributed framework whose correctness depends only on local states is preferred. To the best of our knowledge, all existing algorithms are tailored towards specific problem classes with a single or multiple specific routing constraints, and

[*]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.

none of them provides a framework with the above features.

In this paper, we present an integrated QoS routing framework, from which a family of fully distributed algorithms for unicast routing are derived. The framework is presented in the form of a *generic algorithm* based on *selective probing*. Every node in the network maintains its own local state and no global state is required. The algorithm uses a distributed computation to collectively utilize the most up-to-date local state at each node to find a path. It requires a single pass of messages to find a routing path which is always loop-free. A family of *concrete unicast routing algorithms* is derived from the generic algorithm by specifying a few well-defined open parameters that are constraint-dependent. We show the flexibility of our framework by discussing various algorithms on bandwidth, delay, cost and their combinations. We also discuss several optimization techniques which improve the routing performance. In addition, we describe a heuristic solution to the NP-complete multi-constrained routing problem. Our framework supports multicast routing as well, which was presented in [3].

The most related work is done by Shin and Chou [10] and by Hou [7][1]. Their flooding-based distributed algorithms also rely on local states for routing, but only the delay metric is considered and a large number of routing messages are required. Our framework provides an integrated approach to route connections with many different QoS constraints and improves the routing performance by optimizations such as topology-based routing table and iterative routing.

The rest of the paper is organized as follows. The network and QoS models are given in Section 2. A distributed QoS routing framework is proposed in Section 3, from which a family of unicast routing algorithms are derived in Section 4. Simulation results are presented in in Section 5. Finally, Section 6 concludes the paper. An early version of this paper can be found in [2].

2 Network and QoS Models

A network is modeled as a set V of nodes that are interconnected by a set E of communication links. Each node i has a routing table that has an entry $R_i(t)$ for every possible destination t. The routing table is constructed based on the connectivity (topology) of the network. For example, $R_i(t)$ can be $\{j \mid d_{i,t} = d_{j,t} + 1 \vee d_{i,t} = d_{j,t},\ \forall(i,j) \in E\}$, where $\forall x, y \in V$, $d_{x,y}$ is the distance between node x and node y. $R_i(t)$ consists of a subset of adjacent nodes of i.

The topology of the network can change, which however is *relatively infrequent* comparing to those QoS parameters such as bandwidth, delay and delay jitter. We assume that the topology information is updated by the distance-vector or link-state algorithm, which can be the same one used for non-QoS data-packet routing. That is, QoS routing and non-QoS routing are allowed to share state information and even routing tables. Throughout this paper, we only consider the logical network defined by the combination of the routing tables at all nodes. Those paths that are made possible by the underlying network but not by the routing tables are excluded.

[1] Hou's algorithm works in the context of the virtual path layout in an ATM network.

Definition 1 Concave and additive QoS metrics: Let $m(i,j)$ be a QoS metric for link (i,j). For a path $P = (s,i,j,...,l,t)$, metric m is *concave* if $m(P) = min\{m(s,i), m(i,j), ..., m(l,t)\}$. Metric m is *additive* if $m(P) = m(s,i) + m(i,j) + ... + m(l,t)$.

Bandwidth is concave; delay, delay jitter and cost are additive. A connection request is represented by a tuple (Qos, s, t, cid), where Qos, s, t and cid are the quality-of-service requirement, the source, the destination and a system-wide unique identifier, respectively. cid consists of the source IP address, the port number and a sequence number. When an unsuccessful connection request is retried, a different sequence number is assigned.

The quality-of-service requirement (Qos) consists of a set of constraints, $\bigcup_{i=1}^{n}\{m_i \ \circ \ M_i\}$, where m_i is a metric and M_i is a constant. The purpose of routing is to find a path P which satisfies, $\forall i \in [1..n]$, $m_i(P) \ \mathcal{R} \ M_i$, where $\mathcal{R}$ is $<, \leq, >$ or $\geq$. Wang and Crowcroft [11] proved that if Qos contains at least two additive metrics then the routing is a NP-complete problem.

3 Distributed QoS Routing Framework

Our distributed QoS routing framework defines a protocol for exchanging routing messages among nodes, and provides a common basis for distributed QoS routing regardless the specific metrics involved. We have two goals in mind when designing the framework: (1) It must be general enough to accommodate various QoS metrics, and (2) it must be simple enough to enable a relatively easy implementation with acceptable overhead in the context of high-speed networks.

We present the framework in the form of a *generic algorithm* which implements the following two-phase protocol.

3.1 Two-phase connection establishment protocol

The connection establishment protocol used in this paper consists of two phases: *probing phase and acknowledgement phase.* The probing phase does the QoS routing and the routing path it selects is called the *tentative path.* The acknowledgement phase does the resource reservation. Three types of control messages, *probe, ack and failure*, are used.

The probing phase is started when a connection request (Qos, s, t, cid) arrives. *Probes* are sent from the source s toward the destination t to select a tentative routing path. There are three problems: (1) detecting the *eligible* paths that have sufficient resources to support the required Qos, (2) selecting a tentative path from the eligible paths based on certain optimization criteria, and (3) minimizing the routing overhead introduced by probes. Our solution to the above problems is *selective probing*, in which probes are flooded selectively only along those paths which satisfy the Qos and the optimization requirements. Since a probe proceeds only when the nodes and the links on the way have sufficient resources, every probe arriving at t detects an eligible routing path, which is the one the probe has just traversed from s to t. Upon receipt of probes, the destination t selects a tentative path among the detected eligible paths based on the optimization information carried by the probes.

In the acknowledgement phase, an *ack* message is sent back along the tentative path from t to s, and reserves resources along the way. When s receives the ack, the connection is successfully established. However, since the available resources of every link or node are dynamically changing, an intermediate node on the tentative path may not be able to reserve the required amount of resources at the time when it receives the ack. In such a case, the ack is turned around as a *failure* message and sent back to t to release the reserved resources.

3.2 Data Structures

A simplified data structure for a control (probe, ack or failure) message is $[k, Qos, s, t, cid]$, where k is the sender of the message, which can be the source or an intermediate node. For example, $probe[k, Qos, s, t, cid]$ represents a probe sent by k for connection cid whose source, destination and QoS requirement are s, t and Qos, respectively. When details are not of interest, we simply use $probe[cid]$ to refer to a probe for connection cid. Messages belonging to different connection requests may be sent over the network simultaneously. They are distinguished by their cids and thus no message interference will occur among different requests.

As a probe is sent from the source to the destination, we want to record the path the probe traverses. There are two approaches: 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. We choose the second approach, which 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. Table 1 shows the variables declared at a node i. The variables for a connection cid are created upon the receipt of the first probe[cid]. The obsolete variables are deleted by a timeout mechanism.

3.3 Generic DRA

The function *genericDRA*() defined in the following implements the proposed two-phase protocol. It has two important parameters, ***forward condition*** in Line 8 and Δt in Line 9, which are constraint-dependent and thus left open. In Section 4, various *concrete DRAs* will be derived by specifying these parameters differently. The *forward condition* is a boolean function with three arguments (i, j, Qos), where i is the node with the DRA running, j is an adjacent node of i, and Qos is the quality-of-service requirement. A probe received by i is forwarded to j only when the forward condition is satisfied, which implements the *selective probing*. Δt is called the *probe waiting-time*. Let us assume it to be zero for the time being, which means that the probe is sent immediately to j without an artificially introduced delay. When we discuss the *additive* DRAs in Section 4.2, assigning non-zero values to Δt becomes crucial.

$\pi_i(cid)$	The node from which i receives a probe of connection cid. It is called the *predecessor* of i.
$n_i(cid)$	The node from which i receives an ack of connection cid. It is called the *successor* of i.
$f_i(cid)$	A boolean variable whose initial value is false. Whenever an intermediate node i forwards a received probe[cid], $f_i(cid)$ is set to be true.
$R_i(t)$	The routing-table entry indexed by t. $R_i(t)$ stores a subset of the adjacent nodes of i, to which data packets for t may be forwarded.

Table 1. Variable definition

genericDRA() at node i

```
1.  while true do
2.    block until receiving a message;
3.    switch (the received message)
4.    case 1: probe[$k, Qos, s, t, cid$]
5.      if $i \neq t$ and $f_i(cid) = false$ then
          /* $i$ has not forwarded a probe[$cid$] before. */
6.        $\pi_i(cid) := k$; /* record the predecessor */
7.        for every node $j \in R_i(t), j \neq k$ do
8.          if forward condition $(i, j, Qos)$ is true then
9.            send $j$ a probe[$i, Qos, s, t, cid$] after a delay $\Delta t$;
10.           $f_i(cid) := true$;
11.         endif
12.       endfor
13.     else if $i = t$ and $f_t(cid) = false$ then
          /* start the acknowledgement phase */
14.       send $k$ an ack[$i, Qos, s, t, cid$];
15.       $f_t(cid) := true$;
16.     else
17.       discard the probe;
18.     endif
19.   case 2: ack[$k, Qos, s, t, cid$]
20.     if node $i$ has the required resources then
21.       reserve resources for connection $cid$;
22.       $n_i(cid) := k$; /* record the successor */
23.       if $i \neq s$ then
24.         send $\pi_i(cid)$ an ack[$i, Qos, s, t, cid$];
```

```
25.         else
26.            connection $cid$ is successfully established;
27.         endif
28.      else
29.         send $k$ a failure[$i, Qos, s, t, cid$];
30.      endif
31.   case 3: failure[$k, Qos, s, t, cid$]
32.      release resources reserved at $i$ for connection $cid$;
33.      if $i \neq t$ then
34.         send $n_i(cid)$ a failure[$i, Qos, s, t, cid$];
35.      endif
36.   endswitch
37. endwhile
```

Case 1 describes the probing phase and cases 2 and 3 describe the acknowledgement phase. When a new connection request arrives, a probe is sent to the source to initiate the probing phase. By lines 5-12, *an intermediate node i forwards only the first received probe that satisfies the forward condition.* Lines 13-15 do the path selection. If two or more probes arrive at t, i.e., two or more paths satisfy the QoS requirement, only the first received probe counts and its path is selected as the tentative path, which is recorded by the $\pi_i(cid)$ variables at the intermediate nodes. Optimizing the path selection is discussed in Section 4. The probing phase terminates and the acknowledgement phase starts at Line 14.

The algorithm successfully establishes a connection when the source receives an ack and reserves the required resources successfully. If the source does not receive an ack during a timeout period, the rejection of the connection is assumed. The timeout period is set based on the time complexity of the algorithm (Section 3.4). An alternative approach for rejection indication is to send back a negative acknowledgement (nack) for each probe dropped. When an intermediate node receives a nack for every probe sent, it sends a nack to its predecessor. When the source receives a nack for every probe sent, it signals the rejection of the connection.

Theorem 1 If the generic DRA establishes a connection, the path of the connection must be loop-free.

Due to space limitation, we leave out the proof of all theorems. Interested readers can find the proof in [2].

3.4 Complexity analysis

The algorithm takes a single message round-trip time to establish a connection. If we assume that in normal conditions it takes at most one unit of time for a message to traverse one link including the buffering and processing time at nodes, then the time-complexity is $O(2l)$ units of time, where l is the length of the tentative path.[2]

The algorithm sends at most one probe per link in the sub-net consisting of all paths from the source to the destination. The total number of probes sent is thus bounded by e, where e is the total number of links in the sub-net. There are at most one ack and one failure message for each link on the tentative path. The total number of ack and failure messages is thus bounded by $2l$, where l is the length of the tentative path. Hence, the message complexity (number of control messages) of the algorithm is $O(e + 2l)$ for a single connection request.[3] The average message overhead is substantially lower than the worst-case one when optimization techniques are applied, which is shown by the simulation results in Section 5.

4 Distributed Unicast Routing Algorithms

From the generic DRA, we derive various concrete DRAs for different QoS metrics such as bandwidth, delay, cost, delay jitter, path length, etc. Concrete DRAs are classified into three categories: *concave-metric DRAs*, *additive-metric DRAs* and *multiple-metric DRAs.*

4.1 Concave-metric DRAs

We use the *bandwidth* metric as an example. DRAs on other concave metrics such as buffer space or *CPU time* can be defined similarly. For all concave DRAs, $\Delta t = 0$. The forward condition is specified as follows.

4.1.1 Bandwidth DRA

Let $bandwidth(i, j)$ be the residual (unused) bandwidth of link (i, j). Let B be the bandwidth requirement of a connection. The forward condition is

$$\textit{forward condition } (i, j, Qos\{bandwidth \circ B\}) : \\ bandwidth(i, j) \geq B$$

The resulting routing algorithm is called DRA($bandwidth$).

It is hard to analyze the performance of a routing algorithm in a dynamic network where the bandwidth of each link may change at any time. For simplicity, we assume for the following theorem that the local states at nodes do not change during the process of routing a connection. The same assumption is made also for Theorems 3-4

Theorem 2 Given a connection request ($Qos\{bandwidth \circ B\}, s, t, cid$), DRA($bandwidth$) finds a tentative path P from s to t such that $bandwidth(P) \geq B$, if such a path exists.

[2] According to Lines 13-15 of genericDRA(), only the first probe received by t counts for the time complexity. It defines the tentative path and terminates the probing phase. All probes received by t successively are simply discarded and do not add up to the time complexity.

[3] Only a single run of the algorithm is considered in the complexity. The overhead of failure-and-retry is not included.

4.1.2 Optimization

When there exist many paths that satisfy the QoS requirement, the *shortest* path in terms of delay or number of hops or the *widest* path in terms of residual bandwidth [11] is often preferred. However, DRA(*bandwidth*) finds a tentative path blindly through the competition among probes. Whichever probe reaching the destination first defines the path. This scheme does not guarantee that the shortest or widest path is always selected as the tentative path, although we argue that our algorithm is statistically in favor of the shortest path, because a probe sent along a shorter path will be more likely to reach the destination earlier than a probe sent along a longer path when the two paths have similar congestion conditions. In the following, we discuss some optimization techniques, which are also applicable to all DRAs proposed later.

We define the *age* of a probe as the number of hops it has traversed. When a node i receives and forwards a probe$[k, B, s, t, cid]$, it not only keeps k in $\pi_i(cid)$ as its predecessor but also records the age of the probe. If another probe with a less age is received from k', i changes the value of $\pi_i(cid)$ to k' so that when an ack is received by i later, the ack is forwarded to k' instead of k and thus follows a shorter path to the source. The above scheme can also be applied to bandwidth. We define the *bandwidth* of a probe as the minimum bandwidth of all links it has traversed so far. When i receives a probe from k' with a larger bandwidth than that of the previously received probe, i changes its predecessor to k' which leads to a path with larger residual bandwidth.

Instead of keeping a single predecessor, a node i can declare $\pi_i(cid)$ as a set which keeps a group of predecessors. For every received probe$[k, B, s, t, cid]$, i adds k to its predecessor set.[4] When i receives an ack, it chooses a predecessor from the set to forward the ack. If i receives a failure message from that predecessor, it chooses another predecessor from the set to forward the ack again. Only after i receives a failure message from every one in the set, it passes the failure message to $n_i(cid)$. Multiple predecessors improve the chance for the connection to be successfully established when the network state dynamically changes. Further improvement on the above approach can be done as follows. When i receives a probe, information besides the predecessor, such as the age and/or the bandwidth of the probe, is also recorded. Such information can help i make a better choice on which one in the predecessor set should be selected to forward the ack.

In DRA(*bandwidth*), the destination sends an ack immediately after it receives the first probe even though successive probes may suggest better paths. We can improve the algorithm as follows. When the destination receives a probe, it checks the age and the bandwidth of the probe. If the quality of the corresponding path is good, for example, the age is equal to the shortest distance from the source to the destination or the bandwidth is greater than some large value, an ack is replied immediately. Otherwise, the destination waits for a short period of time for more probes and then chooses the best one to reply.

[4] Still, only the first probe which satisfies the forward condition is forwarded.

4.2 Additive-metric DRAs

We define an abstract additive metric, *length*, which can be delay, delay jitter, cost or number of hops. Let p be a probe and P be the path which p has traversed so far. We define $length(p) = length(P) = \sum_{(i,j)\in P} length(i,j)$.

We extend the structure of a probe and add a new field to keep $length(p)$, so that the receiver of p knows the value of $length(p)$. Initially, $length(p) := 0$. Whenever p proceeds for another link (i, j), $length(p) := length(p) + length(i, j)$. The forward condition for *length* is specified as

$$\begin{gathered} \textit{forward condition } (i, j, Qos\{length \circ M\}): \\ length(i,j) + length(p) < M \end{gathered}$$

where p is the received probe and M is a constant.

4.2.1 Discussion of Δt

Besides the forward condition, specifying an appropriate non-zero Δt (Line 9 of genericDRA() in Section 3.3) is also important for an additive-metric DRA. If we let Δt be zero as we did for a concave-metric DRA, the algorithm does not work well sometimes. See Figure 1 for an example. The length of each link is labeled beside the link. Figure 1 (a) shows that the path $P = s \to i \to k \to h \to t$ satisfies the connection request $(Qos\{length \circ 5\}, s, t, cid)$ because $length(P) = 4$ and the length requirement is 5. In Figure 1 (b), suppose probe 2 arrives at node k earlier than probe 1. Probe 2 traverses $s \to j \to k \to h$, and is dropped by node h because $length(h, t) + length(probe\ 2) > 5$ and thus the forward condition is violated, where $length(probe\ 2) = length(s, j) + length(j, k) + length(k, h) = 5$ as defined previously. Probe 1 traverses $s \to i \to k$ and is dropped by node k because k has already forwarded probe 2.[5] Hence, no probe can reach t and the algorithm fails to find a tentative path.

We want probe 1 to arrive at node k earlier than probe 2. More generally, it should take less time for a probe to traverse a shorter path than a longer path. A simple approach is to introduce extra delay for each probe to traverse a link; the longer the length of the link, the longer the delay. That can be done by assigning an appropriate value for Δt (see Line 9 of genericDRA() in Section 3.3). How to specify Δt as well as the implementation issues will be discussed shortly when we study the delay DRA and the cost DRA. A performance advantage of doing so is that the shortest path will always be selected as the tentative path because the probe traversing the shortest path arrives at the

[5] By the construction of the algorithm, node k forwards at most one probe. After k forwards probe 2, it discards all successively received probes including probe 1.

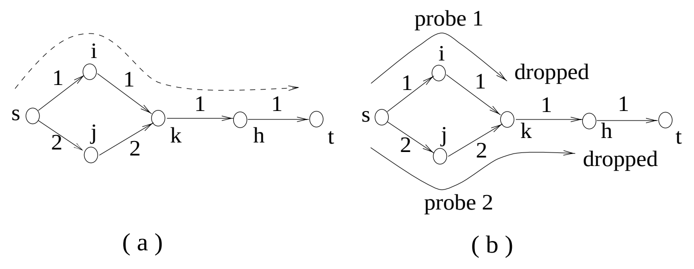

Figure 1. (a) There exists an eligible path for the request $(Qos\{length \circ 5\}, s, t, cid)$. (b) An additive-metric DRA fails in finding the path if $\Delta t = 0$.

destination first. The message (or computation) overhead for finding the shortest path is $O(e)$,[6] which is remarkably lower than $O(ve)$ of the distributed Bellman-Ford algorithm, where v is the number of nodes and e the number of links. However, the cost we pay is the time complexity, which is increased by the extra delay of the probes. Fortunately, as we will see shortly, the increase on the time complexity is controllable and can be adjusted to a modest range. In the following, we will use *delay* and *cost* as examples to show how to choose Δt.

4.2.2 Delay DRA

A quality-of-service requirement $Qos\{delay \circ D\}$ requires the end-to-end delay to be bounded by D. The end-to-end delay of a path is the summation of the *node-delay* at every node and the *link-delay* at every link on the path. *node-delay*(i, j) includes the protocol-processing time and the queueing delay of a normal data packet at node i for link (i, j). *link-delay*(i, j) is the propagation delay on link (i, j). The *delay* metric is defined as

$$delay(i, j) = node\text{-}delay(i, j) + link\text{-}delay(i, j)$$

Let p be a probe and suppose the path that p has traversed so far is $P = s \to k \to ... \to i \to j$.

$$delay(p) = delay(s, k) + ... + delay(i, j)$$

The forward condition for *delay* is

$$\begin{gathered} forward\ condition\ (i, j, Qos\{delay \circ D\}): \\ delay(i, j) + delay(p) < D \end{gathered}$$

Let $\Delta t = node\text{-}delay(i, j)$, and the resulting DRA is called DRA(*delay*). Line 9 of genericDRA() becomes

$$\begin{gathered} \text{send } j \text{ a probe}[i, Qos\{delay \circ D\}, s, t, cid] \\ \text{after a delay of } node\text{-}delay(i, j) \end{gathered} \tag{1}$$

The probe has to be forwarded **after** a period of *node-delay*(i, j). This can be easily implemented by treating probes as *normal data packets* whose processing and queueing delay at node i is *node-delay*(i, j) by definition. However, this approach is applicable only to DRA(*delay*) but not to other additive-metric DRAs where a value other than *node-delay*(i, j) is assigned to Δt, as we shall see shortly in DRA(*cost*). A more general implementation is described as follows: Two queues, *data packet queue* and *control message queue*, are maintained at node i for link (i, j). The control message queue has a higher priority than the data packet queue. The statement (1) is non-blocking. It completes by inserting the probe with a timestamp Δt into the control message queue, in which probes are placed in the order of increasing timestamps. A timer is set appropriately for the first probe in the queue. When the timer is timed out, the probe is transmitted immediately.

Since probes travel at speeds according to the delays, a probe traveling along the minimum-delay path arrives first. The time complexity of the probing phase is bounded by $min\{D, delay(P^m_{s,t})\}$, where $P^m_{s,t}$ is the minimum-delay path from s to t. After time D, $delay(p) \geq D$ for every existing probe p of the connection and thus none of the probes can satisfy the forward condition any more. After $delay(P^m_{s,t})$, the probe traversing $P^m_{s,t}$ has already arrived at the destination, which terminates the probing phase by selecting $P^m_{s,t}$ as the tentative path.

Theorem 3 Given a connection request $(Qos\{delay \circ D\}, s, t, cid)$, (1) DRA(*delay*) finds a tentative path P such that $delay(P) < D$, if such a path exists; (2) the tentative path must be the minimum-delay path from s to t.

4.2.3 Cost DRA

A quality-of-service requirement $Qos\{cost \circ C\}$ requires the end-to-end cost to be bounded by C. Let $cost(i, j)$ be the cost of link (i, j) in dollars or by some other measurement, and $cost(p)$ be the accumulated cost on the path that a probe p has traversed. The forward condition for *cost* is

$$\begin{gathered} forward\ condition\ (i, j, Qos\{cost \circ C\}): \\ cost(i, j) + cost(p) < C \end{gathered}$$

Let $\Delta t = f(cost(i, j))$ where f is a monotonic non-decreasing function, and the resulting DRA is called DRA(*cost*). An example f is

$$f(cost(i, j)) = max\{\frac{cost(i, j)}{C} \times T - link\text{-}delay(i, j),\ 0\}$$

where T is a time constant. The time for a probe to traverse link (i, j) is

$$\begin{aligned} & f(cost(i, j)) + link\text{-}delay(i, j) \\ & = max\{\frac{cost(i, j)}{C} \times T,\ link\text{-}delay(i, j)\} \end{aligned}$$

Ideally, we want to choose T such that $\frac{cost(i,j)}{C} \times T \geq$ *link-delay*(i, j), which makes the probing delay over (i, j)

[6] $O(e)$ is the message complexity of the probing phase. See Section 3.4.

directly proportional to $cost(i,j)$. However, there exists a tradeoff since a larger T makes the routing time longer. Let P be the tentative path. The time complexity of the probing phase can be measured by

$$\begin{aligned}
&\sum_{(i,j)\in P} max\{\frac{cost(i,j)}{C}\times T,\ link\text{-}delay(i,j)\} \\
&= max\{\frac{\sum_{(i,j)\in P} cost(i,j)}{C}\times T,\ \sum_{(i,j)\in P} link\text{-}delay(i,j)\} \\
&\le max\{T,\ \sum_{(i,j)\in P} link\text{-}delay(i,j)\}
\end{aligned}$$

Generally, the value of T can be selected based on the estimated end-to-end delay from s to t.

4.3 Multiple-metric DRAs

We use two examples. The first one has one concave metric and one additive metric, and the second one has two additive metrics.

4.3.1 Bandwidth and delay DRA

We consider $Qos\{bandwidth \circ B, delay \circ D\}$ where the bandwidth requirement is B and the end-to-end delay requirement is D. The forward condition is

$$\begin{aligned}
&\textit{forward condition } (i,j,Qos\{bandwidth \circ B, delay \circ D\}): \\
&\quad bandwidth(i,j) \ge B \ \wedge \ delay(i,j) + delay(p) < D
\end{aligned}$$

Let $\Delta t = node\text{-}delay(i,j)$ and the resulting DRA is called DRA($bandwidth, delay$).

Theorem 4 Given a connection request $(Qos\{bandwidth \circ B, delay \circ D\}, s, t, cid)$, (1) DRA($bandwidth, delay$) finds a tentative path P such that $bandwidth(P) \ge B$ and $delay(P) < D$, if such a path exists; (2) the tentative path has the minimum delay among all eligible paths from s to t.

4.3.2 Delay and cost DRA

It is a NP-complete problem to find a path that satisfies $Qos\{delay \circ D, cost \circ C\}$, which requires the end-to-end delay and the end-to-end cost to be bounded by D and C, respectively. DRA($delay, cost$) provides a heuristic solution for this problem. The forward condition is

$$\begin{aligned}
&\textit{forward condition } (i,j,Qos\{delay \circ D, cost \circ C\}): \\
&\quad delay(i,j) + delay(p) < D \ \wedge \ cost(i,j) + cost(p) < C
\end{aligned}$$

Determining Δt is tricky. A general form is

$$\begin{aligned}
\Delta t = max\{&\alpha \times delay(i,j) + \beta \times cost(i,j) - \\
&link\text{-}delay(i,j),\ 0\}
\end{aligned}$$

When the delay bound is hard to achieve but the cost bound is easy to achieve, we let $\alpha = 1$ and $\beta = 0$. Δt becomes the same as in DRA($delay$). Hence, when probing for a tentative path, the algorithm is in favor of paths with short delays. When the cost bound is hard to achieve but the delay bound is easy to achieve, let $\alpha = 0$ and $\beta = T/C$, where T is a constant. Δt becomes the same as in DRA($cost$). When both delay and cost bounds are difficult to achieve, which means D (C) is close to the minimum delay (cost) between the source and the destination, we let $\alpha = 1$ and $\beta = D/C$. [7] When the relative difficulties of achieving the two bounds are unknown, a general way of increasing the probability of finding a tentative path is to allow multiple executions of DRA($delay, cost$) with different Δt's. When a connection request $(Qos\{delay \circ D, cost \circ C\}, s, t, cid)$ arrives, two pseudo requests are created:

$$\begin{gathered}
(Qos\{delay \circ D, cost \circ C\}, s, t, cid') \\
(Qos\{delay \circ D, cost \circ C\}, s, t, cid'')
\end{gathered}$$

Hence, three instances of DRA($delay, cost$) are executed and each uses a different Δt. The three Δt's are

$$\begin{aligned}
&\Delta t = node\text{-}delay(i,j) \\
&\Delta t = max\{\frac{cost(i,j)}{C}\times T - link\text{-}delay(i,j),\ 0\} \\
&\Delta t = node\text{-}delay(i,j) + \frac{D}{C}\times cost(i,j)
\end{aligned}$$

4.4 Iterative DRAs

We use DRA($*$) as a general term referring to *any* of the above concrete DRAs. DRA($*$) finds a tentative path through the competition among probes. The first probe reaching the destination defines the path, which however may not be the optimal one. When there exist many paths that satisfy the QoS requirement, the shortest path in terms of *number of hops* is often desired due to a better statistical performance [8]. We propose *iterative* DRAs to find the shortest tentative path. As an example, let us design an iterative version of DRA($bandwidth$) first.

We define the *age* of a probe p as the number of hops it has traveled.[8] Initially, $age(p) := 0$. Whenever p passes a link, $age(p) := age(p) + 1$. We introduce a new forward condition for DRA($bandwidth$):

$$\begin{aligned}
&\textit{forward condition } (i,j,Qos\{bandwidth \circ B\}): \\
&bandwidth(i,j) \ge B \ \wedge \ age(p) + d_{j,t} + 1 \le L
\end{aligned}$$

where $d_{j,t}$ is the distance from j to the destination t and L is a constant no less than $d_{s,t}$.

Should p be forwarded to j, the length of the shortest possible path for p to reach t would be $age(p) + d_{j,t} + 1$, including the portion that p has already traversed from s to i. Hence, the above forward condition allows a probe

[7] The reason for β to be D/C instead of 1 is to balance the impacts of the two metrics on Δt so as to avoid one metric dominating the value of Δt.

[8] In Section 4.1.2, the ages of probes are used as an optimization technique for the intermediate nodes to choose their predecessors. Here, the same ages are used in an *end-to-end* manner to constrain the length of the tentative path by limiting the maximum number of hops the probes can travel.

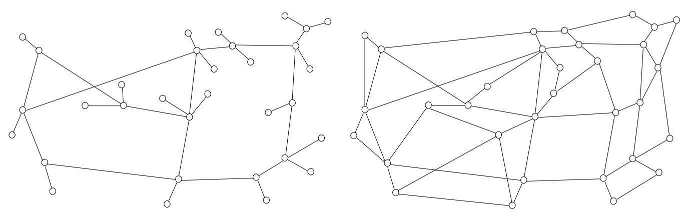

Figure 2. network topologies

to be forwarded only along those links which lead to paths whose lengths are bounded by L. When $L = d_{s,t}$, the algorithm forwards probes only along the shortest paths and the corresponding DRA is called

$$\text{DRA}(\textit{bandwidth}) \text{ WITH } L = d_{s,t}$$

A general form of our iterative DRA is

$$\text{DRA}(\textit{bandwidth}) \text{ ITERATIVE ON} \\ \{L = d_{s,t}, L = d_{s,t} + 1, ..., L = +\infty\}$$

It executes DRA(*bandwidth*) iteratively each time with an increasing value of L — $d_{s,t}, d_{s,t} + 1$, ..., and $+\infty$, in turn — until the connection is established.

Though many iterations are allowable, we only use two iterations in order to reduce the overhead. That is, we consider DRA(*bandwidth*) ITERATIVE ON $\{L = d_{s,t}, L = +\infty\}$ in this paper. DRA(*bandwidth*) WITH $L = d_{s,t}$ is first executed, which tries to find a shortest path P with $\textit{bandwidth}(P) \geq B$. If it succeeds, *the next iteration is canceled.* If it fails, DRA(*bandwidth*) WITH $L = +\infty$ is executed to search all possible paths.

For an arbitrary DRA($*$), let Fcondition($*$) be its original forward condition. A new forward condition is defined as

$$\textit{forward condition } (i, j, Qos): \\ \text{Fcondition}(*) \ \wedge \ \textit{age}(p) + d_{j,t} + 1 \leq L$$

The corresponding iterative DRA is denoted as

$$\text{DRA}(*) \text{ ITERATIVE ON } \{L = d_{s,t}, L = +\infty\}$$

Comparing to their non-iterative counterparts, the iterative DRAs not only find shorter routing paths but also reduce the average overhead significantly at the cost of longer routing time, which will be shown by the simulation results in the next section.

5 Simulation Results

5.1 Message overhead

The message overhead is one of the most important performance metrics for distributed routing algorithms. It has a direct impact on how applicable the algorithms are in the real world. The worst-case message overhead of our DRAs has been analyzed in Section 3.4. In this section, we study by simulation the *average* overhead, namely, the average number of probes sent per connection request.

Two network topologies, NET-1 and NET-2 (Figure 2), were used in the simulation. NET-1 is based on the major circuits in ANSNET and NET-2 increases the connectivity of NET-1 by additional links. Each link is full duplex with a bandwidth capacity of 155Mbps(OC3). The nodes are placed in a 3000×2400 Km^2 rectangle, roughly the size of USA, and the propagation speed through the links is two thirds of the light speed. The message overhead of DRA(*bandwidth*), DRA(*delay*) and their iterative versions will be presented. The results for other DRAs can be found in [2]. We use DRA($*$) ITERATIVE as an abbreviation of DRA($*$) ITERATIVE ON $\{L = d_{s,t}, L = +\infty\}$.

Figure 3 presents the results of four experiments: running DRA (*bandwidth*) and DRA(*bandwidth*) ITERATIVE on NET-1 and NET-2. The message overhead is plotted with respect to the average traffic load in the network. The source node, the destination node and the bandwidth requirement B of each connection request are randomly generated. B is uniformly distributed between 64Kbps $\sim$ 1.5Mbps. For each request, the background traffic load on every link is randomly generated from the range of [0.0, 155.0Mbps]. Each point in Figure 3 is taken by averaging the overhead of one thousand independent connection requests.

Both DRA(*bandwidth*) and DRA(*bandwidth*) ITERATIVE have higher overhead on NET-2 than on NET-1, because NET-2 is better connected and more probes are flooded during the routing. DRA(*bandwidth*) has a much higher average overhead than DRA(*bandwidth*) ITERATIVE though its worst-case overhead is lower (see Section 4.4). This can be explained by the following reasoning: When the network is not congested (average traffic load less than 145 Mbps in our simulation), DRA(*bandwidth*) ITERATIVE succeeds in its first iteration and thus probes are flooded only along the shortest paths. The overhead is certainly lower than DRA(*bandwidth*) which does a much broader flooding. When the traffic load grows heavier, the overhead of DRA(*bandwidth*) ITERATIVE may increase due to the second iteration.[9] However, the more the second iterations are required, the more the network is congested, which further implies that satisfying the forward condition becomes harder and harder due to the lack of bandwidth. The immediate result is more and more probes are discarded instead of being forwarded. Therefore, when the second iterations can not compensate the discarded probes, the average number of probes sent decreases, as observed in Figure 3. When the network is fully congested, hardly can any probes be sent out from the source nodes and the overhead reaches zero.

Figure 4 shows the results of running DRA(*delay*) and DRA(*delay*) ITERATIVE on NET-1 and NET-2. The de-

[9] The overhead increase is not observed for DRA(*bandwidth*) ITERATIVE on NET-1. This is because NET-1 is much sparser than NET-2 and thus the number of probes resulted from the second iterations are not many enough to offset the increasing number of discarded probes due to heavier background traffic.

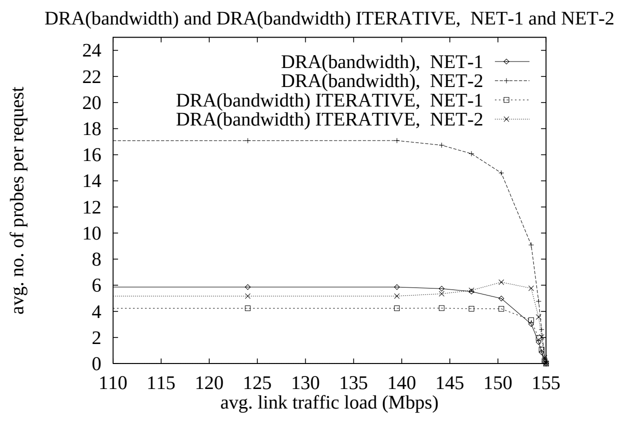

Figure 3. Average message overhead of DRA(bandwidth) and DRA(bandwidth) ITERATIVE

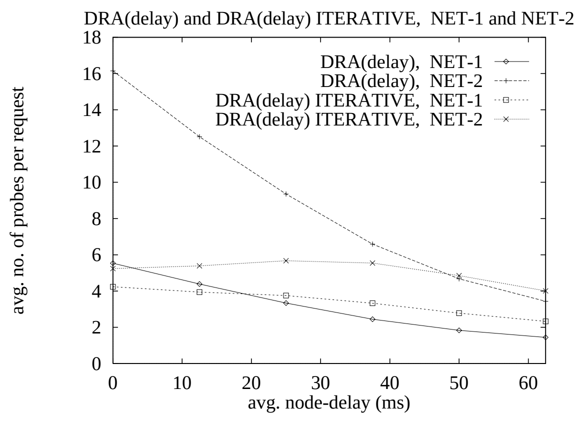

Figure 4. Average message overhead of DRA(delay) and DRA(delay) ITERATIVE

lay requirements of the experimental connection requests are uniformly distributed in the range of [0.0, 200.0ms]. The *node-delay* of each link is randomly generated from [0.0,125.0ms]; experiments show that larger *node-delay*'s make the forward condition hardly satisfied and thus of little interest to our simulation. Similar to Figure 3, the overhead of DRA(*delay*) ITERATIVE does not increase significantly with respect to the average *node-delay*. The reason is that although an increasing *node-delay* results in more second iterations, it also results in more discarded probes because the forward condition is harder to be saitisfied. On Net-2, DRA(*delay*) ITERATIVE has a much lower overhead than that of DRA(*delay*) when the average *node-delay* is relatively low. Experiments on other DRAs also reveal that the iterative DRAs have a better overall performance.

The average source-destination distance of all connection requests is 3.2 hops in our simulation. The average number of probes per request is always under 6.5 for any iterative DRA and any background traffic condition. Such a result indicates that the selective probing provides an efficient distributed routing approach, which has low overhead and does not rely on accurate global state.

5.2 Routing time

In this section, we use DRA(*delay*) and DRA(*delay*) ITERATIVE to study how long it takes to make the routing. For those connections that are successfully established, we measure the average time to find a tentative path, called the *average probing time*. For those connections that are rejected, we measure the average time to timeout and signal the rejection, called the *average rejection time*. DRA(*delay*) rejects a connection after a timeout period of $2D$ if no *ack* is received, where D is the delay requirement.[10] DRA(*delay*) ITERATIVE has two iterations: DRA(*delay*) WITH $L = d_{s,t}$ and DRA(*delay*) WITH $L = \infty$. Both of them have a timeout period of $2D$. Therefore, a connection will be rejected after $4D$.

[10]Recall that the time complexity of the probing phase is bounded by D (Section 4.2.2). A timeout period of $2D$ leaves enough margin for the ack to travel back in time.

Before presenting the simulation results, we make a simple analysis on the probing time of DRA(*delay*) first. Let Φ be the set of connection requests used in the simulation. Among the requests, the set of successful (accepted) ones is Φ_s and the set of rejected ones is Φ_r. Recall that probes in DRA(*delay*) travel at speeds according to the delays of their paths (Section 4.2.2). The probe traversing the minimum-delay path arrives at the destination first. Therefore, for a request $\{D, s, t, id\}$ in Φ, the probing time is $delay(P^m_{s,t})$, where $P^m_{s,t}$ is the minimum-delay path. The average probing time of successful requests is

$$\lambda = \frac{\sum_{\{D,s,t,id\}\in\Phi} delay(P^m_{s,t})}{|\Phi|}$$

Suppose s and t are uniformly selected from all nodes. If $\Phi_s = \Phi$, λ is expected to be equal to the *average end-to-end delay* between two nodes in the network, which is defined as

$$\eta = \frac{\sum_{s,t\in V, s\neq t} delay(P^m_{s,t})}{\sum_{s,t\in V, s\neq t} 1}$$

Now, let us consider the case $\Phi_s \subset \Phi$. A request $\{D, s, t, id\}$ with a large $delay(P^m_{s,t})$ is more likely to be rejected, because given the same D the constraint $delay(P^m_{s,t}) < D$ is harder to be satisfied. Therefore, Φ_s tends to have requests with smaller $delay(P^m_{s,t})$ and Φ_r tends to have requests with larger $delay(P^m_{s,t})$, and hence λ can be less than η, as we shall see in the simulation results.

The node-delay of a link is randomly generated from $[0.0, 60.0ms]$. Figure 5 shows the average probing times of DRA(*delay*) and DRA(*delay*) ITERATIVE with respect to the average delay requirement D. The probing time of DRA(*delay*) is bounded by η, 88.38ms in the figure. It converges to η as D increases. DRA(*delay*) ITERATIVE has

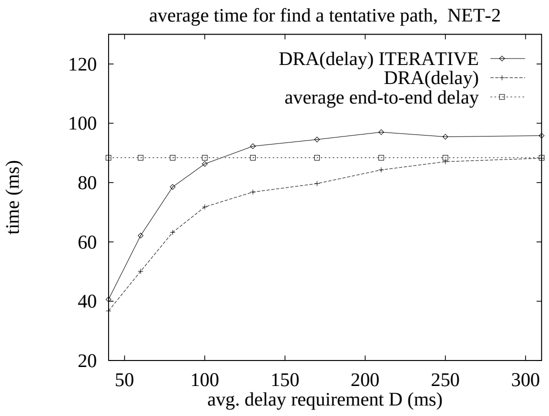

Figure 5. Average time to find a connection

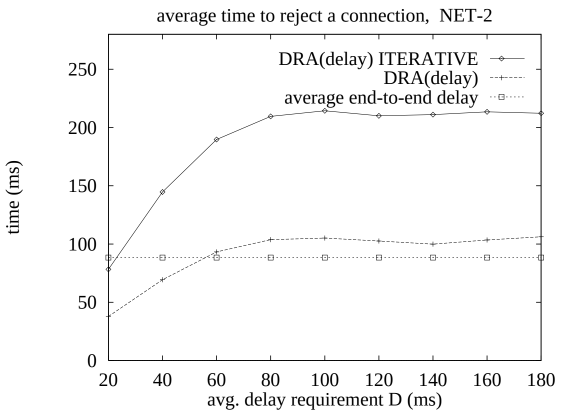

Figure 6. Average time to reject a connection

a larger probing time than DRA(*delay*) due to its two iterations. As D increases, the timeout between the two iterations increases and thus the probing time increases. On the other hand, larger D's allow more connections to be established by the first iterations, and less second iterations (less timeouts) are necessary. When the decrease in the number of timeouts compensates the increase in the length of timeout periods, the average probing time stops increasing. The probing time of DRA(*delay*) ITERATIVE goes slightly higher than η as shown in the figure.

Figure 6 shows the rejection times of DRA(*delay*) and DRA(*delay*) ITERATIVE. As D increases, the average rejection times increase due to longer timeouts. However, after D passes certain threshold (80ms in the figure), the average rejection times become stable and do not increase further. This is because connections with too large D will not be rejected and thus will not contribute to the rejection-time average. The rejection time of DRA(*delay*) ITERATIVE is about twice that of DRA(*delay*), which is comparable to the average end-to-end delay η between two nodes in the network. This can be illustrated by the following much simplied analysis. Consider a fictitious ideal simulation run where all connections with their delay requirements D_r less than η are rejected and all other connections are accepted. Suppose D_r's of the rejected connections are uniformly distributed in $[0..\eta]$. The average of D_r's will be $\overline{D_r} = \frac{1}{2}\eta$. Hence, it takes about $4\overline{D_r} = 2\eta$ on average for DRA(*delay*) ITERATIVE to timeout and signal the rejection, and it takes about $2\overline{D_r} = \eta$ on average for DRA(*delay*) to timeout.

In the actual simulation, not all D_r's fall in $[0..\eta]$. Some rejected connections have delay requirements larger than η, which makes $\overline{D_r}$ and thus the average rejection time larger than those in the above ideal case. In Figure 6, the rejection times of DRA(*delay*) ITERATIVE and DRA(*delay*) can go up to 2.4 and 1.2 times η, respectively.

6 Conclusion

In this paper, we described in detail a unique integrated and distributed QoS routing framework presented in the form of a generic DRA. From the framework, we derived three classes of unicast routing algorithms: concave-metric DRAs, additive-metric DRAs and multiple-metric DRAs. Each DRA has its iterative version. Simulations revealed that the iterative DRAs have a better overall performance than the non-iterative counterparts. The stable and modest overhead of the iterative DRAs makes them practical.

References

[1] J. Behrems and J. Garcia-Luna-Aceves. Distributed, scalable routing based on link-state vectors. *SIGCOMM*, pages 136–147, August 1994.

[2] S. Chen and K. Nahrstedt. Distributed qos routing. *Tech. Report UIUCDCS-R-97-2017, Department of Computer Science, University of Illinois at Urbana-Champaign*, July 1997.

[3] S. Chen and K. Nahrstedt. Distributed quality-of-service routing in high-speed networks based on selective probing. *Technical Report, University of Illinois at Urbana-Champaign, Department of Computer Science*, 1998.

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

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

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

[7] C. Hou. Routing virtual circuits with timing requirements in virtual path based atm networks. *INFOCOM'96*, 1996.

[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.