Routing by Distributed Recursive Computation and Information Reuse *

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

Abstract

Distributed multimedia applications have quality-of-service (QoS) requirements specified in terms of constraints on various metrics such as bandwidth and delay. The task of QoS routing is to find a path from the source node to the destination node with sufficient resources to support the required end-to-end QoS. We propose several distributed algorithms for the bandwidth-constrained routing and the delay-constrained routing. The algorithms are presented in the form of distributed recursive computation (DRC). DRC computes the global routing state in a distributed, recursive fashion and often leaves useful information at intermediate nodes during the process. An information-reuse scheme is studied to utilize such information in order to reduce the overall overhead. Our simulation shows that the overhead of the proposed algorithms is modest and stable.

1 Introduction

The task of quality-of-service (QoS) routing is to find a path from the source node to the destination node with sufficient resources to support the required end-to-end QoS.[4] 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 protocol. 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 [3, 7, 9] 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 protocol 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 protocol 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 [6]. However, the state aggregation increases the level of inaccuracy [7]. 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 [10, 12] 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.

In this paper, several distributed algorithms are proposed for the bandwidth-constrained routing and the delay-constrained routing. No global network state is required to be maintained by a distance-vector (or link-state) protocol. We define a new concept, *distributed recursive computation* (DRC), which computes the global state of a network upon the arrival of a routing request. DRC provides a nice/compact presentation of distributed algorithms and makes the induction proofs easier. All proposed distributed routing algorithms are presented in the form of DRC.

The basic idea of DRC is that every node does a small amount of local computation based on its local state and invokes child computations at a selected set of adjacent nodes, which recursively invokes their child computations to carry out the global computation gradually and distributedly. Global routing information is thus calculated in a distributed, recursive fashion. Various techniques are studied to reduce the overhead. During the process of distributed recursive computation, partial routing information is computed and distributed at intermediate nodes. An information-reuse scheme is proposed to utilize such information for future routing in order to reduce the overall overhead.

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

0-7803-5258-0/99 $10.00 © 1999 IEEE

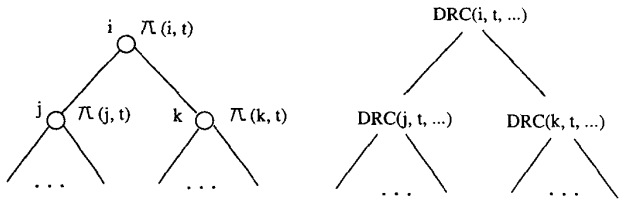

Figure 1. The parent node i has two child nodes j and k; the parent computation DRC$(i,t,...)$ has two child computations DRC$(j,t,...)$ and DRC$(k,t,...)$.

2 Distributed Recursive Computation and Information Reuse

2.1 Distributed recursive computation

A network consists of a set N of nodes which are fully connected by a set E of full-duplex, directed communication links. Each node keeps certain *local state* such as the queuing delay, the propagation delay and the *residual* (unused) bandwidth of its outgoing links. The combination of the local states of different nodes is called a *global state*. In this paper, we are particularly interested in the end-to-end global state between two nodes. Examples are the residual bandwidth of a routing path from one node to another, the maximum residual bandwidth among all paths between two nodes, and the minimum end-to-end delay between two nodes. A global state may change when the local state of any involved node changes. The solution to many distributed problems in a network environment relies on the knowledge of global states. Consider the routing of a video data stream whose delivery delay is required to be bounded. A useful global state will be the minimum end-to-end delay from the source node to the destination node.

An interesting question is how to compute a global state, assuming that each node always keeps its up-to-date local state. We propose a new approach, called *distributed recursive computation* (*DRC*). The idea is briefly illustrated below. More detailed discussion is left to Sections 2.2 and 2.3, where the concrete examples of DRC on QoS routing are studied.

Let DRC$(i,t,...)$ be a distributed recursive computation which computes an end-to-end global state, denoted as $\pi(i,t)$, between nodes i and t. DRC$(i,t,...)$ takes the source node i, the destination node t and other values as parameters. Node i is the place where DRC$(i,t,...)$ is executed. See Figure 1. Let j and k be two adjacent nodes of i. In order to compute $\pi(i,t)$, we first compute $\pi(j,t)$ and $\pi(k,t)$. This means that DRC$(i,t,...)$ invokes two *child computations*, DRC$(j,t,...)$ at j and DRC$(k,t,...)$ at k, which recursively invoke their child computations and upon completion return $\pi(j,t)$ and $\pi(k,t)$ to their *parent computation* DRC$(i,t,...)$. DRC$(i,t,...)$ calculates $\pi(i,t)$ by synthesizing $\pi(j,t)$, $\pi(k,t)$ and its local state.

Definition 1 *Parent-child relationship*: Given DRC$(i,t,...)$ and DRC$(j,t,...)$, if DRC$(j,t,...)$ upon its completion sends $\pi(j,t)$ in a reply message to DRC$(i,t,...)$, DRC$(j,t,...)$ is called a child computation of DRC$(i,t,...)$ and DRC$(i,t,...)$ is called a parent computation of DRC$(j,t,...)$.

A parent computation may have many child computations whereas a child is allowed to have *one* or *more than one* parents depending on the *invocation rule*, which will be discussed in Sections 2.2 and 2.3. When only one parent is allowed, all computations form a parent-child tree as shown in Figure 1. The algorithm of DRC is outlined below.

DRC$(i,t,...)$
1. Select a subset C of adjacent nodes based on local state.
2. Send every $j \in C$ an invocation message$[t,...]$, which causes the child computation DRC$(j,t,...)$ to be executed at node j.
3. Wait for the reply message$[\pi(j,t),...]$, $j \in C$, from the child computations.
4. Compute $\pi(i,t)$ based on $\pi(j,t), j \in C$, and send a reply message$[\pi(i,t),...]$ to the parent node(s).

We shall discuss the problem of how to select C in Sections 2.2 and 2.3. For convenience, we call $j \in C$ a child node of i and i a parent node of j. A message sent from a parent node to a child node is called an *invocation message*, and a message from a child to a parent is called a *reply message*. The notation *message*$[t,...]$ specifies the values carried by the message. The recursion terminates at DRC$(t,t,...)$, which does not have any child computation and thus is able to send a reply message based on its local state.

In the rest of this section, we shall discuss four DRCs for the QoS routing.

2.2 Bandwidth-constrained routing

A bandwidth-constrained routing request is represented by a tuple (s,t,B,id), where s, t, B and id are the source node, the destination node, the bandwidth requirement and the identifier of the request. The purpose of routing is to find a path from s to t such that the residual (unused) bandwidth of the path is not less than B; such a path is called *a solution path*. In a dynamic network, many different routing requests may exist simultaneously. In order to distinguish them, we assign each request a system-wide unique identifier, denoted as id, which consists of the source's identity and a sequence number. Given a path $p = s \rightarrow i \rightarrow j \rightarrow ... \rightarrow k \rightarrow t$,

$$bandwidth(p) = min\{bandwidth(s,i), ..., bandwidth(k,t)\}$$

Assume that each node maintains only local QoS state information — it only knows the residual bandwidths of its outgoing links.

Let $\delta_{i,t}$ be the distance (length of the shortest path) from node i to node t. The value of $\delta_{i,t}$ solely depends on the network topology. We assume that the network topology is relatively stable, comparing to the QoS state such as the residual bandwidth and the delay of each link in the network,

s	source node
t	destination node
i, j, k	intermediate nodes
B	bandwidth requirement
D	delay requirement
id	identifier of the routing request
C	set of child nodes
p	routing path
$\delta_{i,j}$	the distance (length of the shortest path) from i to j

Table 1. Basic notations

which may change at a much faster rate. We further assume i knows the values of $\delta_{i,t}$ and $\delta_{j,t}$, for every adjacent node j, which are often readily available as a result of the traditional link-state (or Bellman-Ford) algorithm in the current packet-switching networks.

Some basic notations, used throughout the rest of the paper, is summarized in Table 1 for quick reference.

2.2.1 The b-computation

A distributed recursive computation, *b-computation*(i, t, B, id), is designed to find a solution path p from i to t such that *bandwidth*$(p) \geq B$. The b-computation(i, t, B, id) returns TRUE if there exists a solution path and FALSE otherwise. The end-to-end global state $\pi(i, t)$ to be computed is thus a boolean value.

Basic algorithm: b-computation(i, t, B, id)

1. If $i = t$, then send the parent a reply message[TRUE]; otherwise, do the following steps.
2. $C := \{j \mid bandwidth(i, j) \geq B,\ j$ is adjacent to $i\}$
   If $C = \emptyset$, send the parent a reply message[FALSE]; otherwise, do the following steps.
3. Send every $j \in C$ an invocation message$[t, B, id]$.
4. Wait for reply messages from the child b-computations. If any reply message is TRUE, the current b-computation is TRUE; if all reply messages are FALSE, the current b-computation is FALSE. The result is sent to the parent in a reply message.

Step (4) of the b-computation completes when (i) a reply message[TRUE] is received or (ii) a reply message[FALSE] has been received from every child computation. In case that a reply message is lost, a timeout mechanism can be used to avoid infinite waiting.

Given a routing request (s, t, B, id), we start from b-computation(s, t, B, id) and send invocation messages recursively to all intermediate nodes i on paths with sufficient bandwidths to invoke b-computation(i, t, B, id). The idea is that each b-computation(i, t, B, id) only finds the next hop $j \in C$ of the routing path and lets the child b-computation(j, t, B, id) complete the rest of the routing by finding a solution path from j to t. However, in an arbitrarily-connected network, each node i may receive many invocation messages. If b-computation(i, t, B, id) is invoked whenever an invocation message$[t, B, id]$ is received, the total number of b-computations invoked in the entire system will grow exponentially. We introduce the *invocation rule* to reduce the overhead.

- Invocation Rule: When i receives its *first* invocation message$[t, B, id]$, b-computation(i, t, B, id) is invoked. For every *successively-received* invocation message$[t, B, id]$, a reply message[FALSE] is returned immediately without the actual execution of b-computation(i, t, B, id).

The purpose of the invocation rule is to allow b-computation(i, t, B, id) to be executed at any node i **at most once**. Given a routing request (s, t, B, id), let us consider the time complexity and the message complexity (number of messages sent) of b-computation(i, t, B, id). Let d_i be the number of outgoing links (adjacent nodes) of a node $i \in N$. The time complexity of Step 1 is $O(1)$; the time complexities of Steps 2, 3 and 4 are all $O(d_i)$. The message complexities of Steps 1 and 2 are zero; the message complexities of Steps 3 and 4 are both $O(d_i)$. Hence, the time and message complexities of b-computation(i, t, B, id) are both $O(d_i)$. For every $i \in N$, b-computation(i, t, B, id) is invoked at most once by the invocation rule. Therefore, the total time and message complexities in the entire network for a single routing request are both $\Sigma_{i \in N} O(d_i) = O(E)$.

Theorem 1 B-computation(s, t, B, id) finds a solution path if there exists one.

The proof of all theorems in the paper can be found in [5]. The found solution path is recorded by the reply message[TRUE] as it travels from the destination t back to the source s. When a node i receives the reply message[TRUE] from a child node j, the resource reservation may be performed and the required bandwidth B is reserved on link (i, j).

The above b-computation has some problems. The overhead may be excessively high as the invocation messages are flooded into the network, and the found solution path may be very long. It is often undesired to select a too-long path since it consumes too much network resource and may reduce the overall call admission ratio. One solution for the above problems is to add another parameter l to the b-computation. The new parameter specifies the maximum length a solution path may have. Steps 2 and 3 of the basic algorithm need to be modified. An additional field is added to the invocation message as well.

Revised algorithm: b-computation(i, t, B, id, l)

1. Same as in the basic algorithm.
2. $C := \{j \mid \delta_{j,t} \leq l - 1,\ bandwidth(i, j) \geq B,\ j$ is adjacent to $i\}$
   If $C = \emptyset$, send the parent a reply message[FALSE]; otherwise, do the following steps.
3. Send every $j \in C$ an invocation message$[t, B, id, l - 1]$, which causes b-computation$(j, t, B, id, l - 1)$ to be executed if it is the first invocation message$[..., id, ...]$

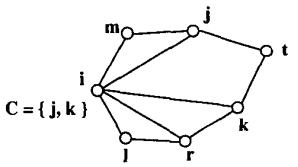

Figure 2. Given the topology, we have $\delta_{i,t} = 2$, $\delta_{j,t} = 1$, $\delta_{k,t} = 1$, $\delta_{m,t} = 2$, $\delta_{r,t} = 2$, and $\delta_{l,t} = 3$. By Step 2 of the w-computation, $C = \{j, k\}$, which is the set of adjacent nodes leading to the shortest paths from i to t.

received by j.

4. Same as in the basic algorithm.

The height of the parent-child tree rooted at b-computation(i, t, B, id, l) is bounded by l. Therefore, the length of the found solution path must be no more than l. In particular, when $l = \delta_{i,t}$, the found solution path must be a shortest path. The additional condition $\delta_{j,t} \leq l - 1$ in the calculation of C helps to reduce the number of child computations.

2.2.2 The w-computation

The shortest paths are often preferred in the bandwidth-constrained routing to save resources. The shortest path which has the maximum residual bandwidth among all shortest paths is called the *widest-shortest path* [12]. A distributed recursive computation, *w-computation*, is designed to calculate $b_{i,t}$ — the residual bandwidth of the widest-shortest path from i to t. For the purpose of convenience, we define $b_{t,t} = +\infty$. In Step 2, C is the set of adjacent nodes which leads to the shortest paths from i to t. See Figure 2 for an example. In Step 5, the variable $n_{i,t}$ keeps the successive node of i on the widest-shortest path.

Algorithm: w-computation(i, t)

1. If $i = t$, then $b_{t,t} := +\infty$ and return the parent a message$[b_{t,t}]$; otherwise, do the following steps.
2. $C := \{j \mid \delta_{j,t} = \delta_{i,t} - 1, \ j \text{ is adjacent to } i\}$
3. Send every $j \in C$ an invocation message$[t]$.
4. Wait for a reply message$[b_{j,t}]$ from every child w-computation(j, t), $j \in C$.
5. $b_{i,t} := \max_{j \in C}\{ min\{bandwidth(i, j), b_{j,t}\}\}$, $n_{i,t} := k$, where $k \in C$ and $b_{i,t} = min\{bandwidth(i, k), b_{k,t}\}$
6. Return the parent a reply message$[b_{i,t}]$.

- Invocation Rule: Whenever i receives an invocation message$[t]$, it invokes w-computation(i, t).

Theorem 2 W-computation(i, t) calculates the maximum residual bandwidth among all shortest paths from i to t.

After w-computation(i, t), the variable $n_{i,t}$ keeps the successive node of i on the widest-shortest path. By the recursive nature of w-computation(i, t), w-computation(j, t) is executed at every node j on the widest-shortest path and the variable $n_{j,t}$ keeps the successive node of j on the path. Therefore, the widest-shortest path can be recovered by tracing $n_{j,t}$ of each node j on the path till reaching t.

We propose a routing algorithm which combines the w-computation and the b-computation. If the bandwidth of the widest-shortest path is no less than the requirement, then the widest-shortest path is used as the solution path; otherwise, execute the b-computation to find a non-shortest solution path. If b-computation(s, t, B, id, l) is used in Step 2, l should be larger than $\delta_{s,t}$.

Algorithm: bandwidth-constrained-routing(s, t, B, id)

1. Execute w-computation(s, t) to calculate $b_{s,t}$.
2. If $b_{s,t} \geq B$, use the widest-shortest path as the solution path; if $b_{s,t} < B$, execute b-computation(s, t, B, id) or b-computation(s, t, B, id, l) to find a solution path.

2.2.3 Information reuse

One problem remains for the w-computation — the invocation rule causes the exponential time (message) complexity. Our solution to this problem is *information reuse*. Each node i maintains a variable $b_{i,t}$ for every other node t. When w-computation(i, t) is executed and $b_{i,t}$ is updated, a timestamp equal to the current clock time is attached to $b_{i,t}$. When i receives another invocation message for the execution of w-computation(i, t), the timestamp of $b_{i,t}$ is checked to see whether it has passed a pre-defined timeout period. If it has not been timed out, the value of $b_{i,t}$ is returned immediately to the parent w-computation; if it has been timed out, w-computation(i, t) is executed and the value of $b_{i,t}$ is updated before being returned. If an invocation message is received when w-computation(i, t) is in execution, then it simply waits for w-computation(i, t) to complete. In this case, the sender becomes an additional parent of w-computation(i, t). Therefore, multiple parents are allowed. During a timeout period, each node i invokes w-computation(i, t) at most once. The invocation rule of the w-computation is rewritten below.

- Invocation Rule: When node i receives an invocation message$[t]$ from node k,

1. if $b_{i,t}$ has not been timed out, send k a reply message$[b_{i,t}]$;
2. if $b_{i,t}$ has been timed out and there is no w-computation(i, t) executing, invoke w-computation(i, t) and k is a parent;
3. if $b_{i,t}$ has been timed out and w-computation(i, t) is in execution, add k as an additional parent.

When w-computation(i, t) completes, a reply message$[b_{i,t}]$ is sent to every parent.

The above information-reuse scheme is especially effective when the system has a relatively small set of servers and a relatively large set of clients. The w-computations recursively invoked for a routing request will distribute state information at many nodes on the shortest paths to a server.

That information can be utilized by future requests and thus help to reduce the overall overhead.

The overhead of w-computation depends on the timeout period, denoted as T. [1] For any pair of nodes i and t, w-computation(i,t) is executed at most once during a time period of T. Let the total number of nodes in the network be n. There are n^2 different node pairs in total if i and t are allowed to be the same node. It gives an upper-bound overhead of at most n^2 w-computations in the entire network for any given time period of T, no matter how many routing requests arrive. Each w-computation sends (receives) at most one invocation (reply) message along every adjacent link. The actual number of w-computations can be less than the upper bound. In particular, when there are no routing requests, there will be no w-computations invoked.

Similar to the link-state (or distance-vector) algorithm, the state information $b_{i,t}$ is updated periodically, which naturally introduces the probability of inaccuracy. In practice, a roughly (though not exactly) precise value of $b_{i,t}$ is still of great value, which is supported by our simulation. A larger T corresponds to a lower overhead and a less accurate value of $b_{i,t}$. A smaller T corresponds to a higher overhead and a more up-to-date $b_{i,t}$. When the inaccuracy of $b_{i,t}$ makes the w-computation fail, the b-computation is used as another attempt to find a solution path and special control messages can be sent along all shortest paths from i to t to clean up the inaccurate information. When a node j receives such a control message, $b_{j,t}$ is immediately timed out.

2.3 Delay-constrained routing

We study another routing problem, the *delay-constrained routing*. A delay-constrained routing request (s,t,D,id) is to find a path from s to t such that the end-to-end delay of the path is bounded by the delay requirement D. Given a path $p = s \to i \to j \to ... \to k \to t$,

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

Assume that each node knows the delays of its outgoing links but does not know those of the other links in the network.

2.3.1 The d-computation

D-computation(i,t,D_i,id) finds a solution path p from i to t such that $delay(p) \le D_i$. It returns TRUE if there exists a solution path and FALSE otherwise.

Basic algorithm: d-computation(i,t,D_i,id)

1. If $i = t$, then send the parent a reply message[TRUE]; otherwise, do the following steps.
2. $C := \{j \mid delay(i,j) \le D_i,\ j$ is adjacent to $i\}$
   If $C = \emptyset$, send the parent a reply message[FALSE]; otherwise, do the following steps.
3. For every $j \in C$, send an invocation message[t, $D_i - delay(i,j)$, id], which may cause d-computation(j, t, $D_i - delay(i,j)$, id) to be executed according to the invocation rule.
4. Wait for reply messages from the child d-computations. If any reply message is TRUE, the current d-computation is TRUE; if all reply messages are FALSE, the current d-computation is FALSE. The result is sent to the parent in a reply message.

- Invocation Rule: When i receives an invocation message[t, D_i, id], if it is the first invocation message with the identifier id, d-computation(i,t,D_i,id) is invoked; otherwise, a reply message[FALSE] is returned immediately.

Theorem 3 D-computation(s,t,D,id) finds a solution path if there exists one.

Note that $delay(i,j)$ is the delay that a *normal data packet* experiences on link (i,j). The invocation message is a *system message* and thus may use less time than $delay(i,j)$ to traverse (i,j). However, in order for the above theorem to hold, we require the delay of an invocation message to be $delay(i,j)$ as well [2, 5]. There are a number of ways to achieve this. The simplest approach is to treat the invocation messages as normal data packets whose delay on (i,j) is $delay(i,j)$ by definition. A more general approach is to set a timer for each invocation message [2]. The invocation message is sent immediately after the timer is expired. The timer is set appropriately so that the delay of the timer plus the propragation delay of the link is equal to $delay(i,j)$. Another approach was proposed by Shi and Chou [11]. They showed that when certain scheduling policies are used and the routing messages are set to the appropriate priority, it also takes $delay(i,j)$ for the routing message to be delivered along link (i,j).

Similar to the discussion in Section 2.2.1, we can add another parameter l to the d-computation and revise Steps 2 and 3 of the algorithm as follows. The length of the solution path found by the revised d-computation is bounded by l.

Revised algorithm: d-computation(i,t,D_i,id,l)

1. Same as in the basic algorithm.
2. $C := \{j \mid \delta_{j,t} \le l-1,\ delay(i,j) \le D_i,\ j$ is adjacent to $i\}$
   If $C = \emptyset$, send the parent a reply message[FALSE]; otherwise, do the following steps.
3. For every $j \in C$, send an invocation message[t, $D_i - delay(i,j)$, id, $l-1$], which causes d-computation(j, t, $D_i - delay(i,j)$, id, $l-1$) to be executed if it is the first invocation message[..., id, ...] received by j.

4-6. Same as in the basic algorithm.

2.3.2 The f-computation

The shortest path [2] which has the minimum delay among all shortest paths is called the *fastest-shortest* path. We

[1] The value of T depends on how often the load of the network changes.

[2] A *shortest* path is a path from the source to the destination with the minimum number of hops. Given two nodes, there may be multiple shortest paths.

design a distributed recursive computation, *f-computation*, to calculate $d_{i,t}$ — the delay of the fastest-shortest path from i to t. For the purpose of convenience, we define $d_{t,t} = 0$.[3]

Algorithm: f-computation(i, t)
1. If $i = t$, then $d_{t,t} := 0$ and return the parent a message$[d_{t,t}]$; otherwise, do the following steps.
2. $C := \{j \mid \delta_{j,t} = \delta_{i,t} - 1,\ j$ is adjacent to $i\}$
3. Send every $j \in C$ an invocation message$[t]$.
4. Wait for a reply message$[d_{j,t}]$ from every child f-computation(j, t), $j \in C$.
5. $d_{i,t} := \min_{j \in C}\{delay(i, j) + d_{j,t}\}$
   $n_{i,t} := k$, where $k \in C$ and $d_{i,t} = delay(i, k) + d_{k,t}$
6. Return the parent a reply message$[d_{i,t}]$.

- Invocation Rule: Whenever i receives an invocation message$[t]$, it invokes f-computation(i, t).

By using the above invocation rule, we have the following theorem.

Theorem 4 F-computation(i, t) calculates the minimum delay among all shortest paths from i to t

The information reuse scheme can used to reduce the overhead of the f-computation. The invocation rule is modified as follows:

- Invocation Rule: When node i receives an invocation message$[t]$ from node k,

1. if $d_{i,t}$ has not been timed out, send k a reply message$[d_{i,t}]$;

2. if $d_{i,t}$ has been timed out and there is no f-computation(i, t) executing, invoke f-computation(i, t) and k is a parent;

3. if $d_{i,t}$ has been timed out and f-computation(i, t) is in execution, add k as an additional parent.

Given a routing request (s, t, D, id), the following delay-constrained routing algorithm is proposed.

Algorithm: delay-constrained-routing(s, t, D, id)
1. Execute f-computation(s, t) to calculate $d_{s,t}$.
2. If $d_{s,t} \leq D$, use the widest-shortest path as the solution path; if $d_{s,t} > D$, execute d-computation(s, t, D, id) or d-computation(s, t, D, id, l) to find a solution path.

2.4 Multi-constrained routing

DRCs for multi-constrained routing can also be designed. For example, we can easily combine b-computation and d-computation (w-computation and f-computation) to solve the bandwidth-delay-constrained routing problem.

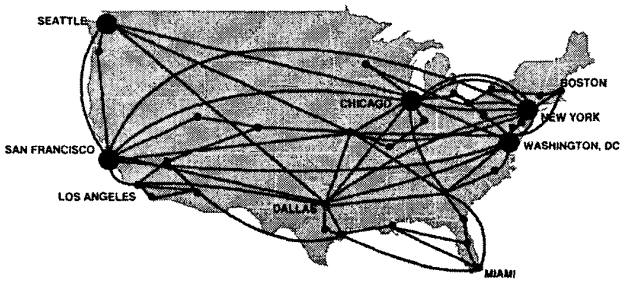

Figure 3. The network topology of the UUNET at the United States

3 Simulation

Simulations were done to evaluate the proposed routing algorithms. The results about the bandwidth-constrained routing is presented in this section. Two performance metrics, *average message complexity* and *call admission ratio*, are considered. The former is defined as the average number of invocation messages sent per routing request; the latter is defined as the percentage of routing reqests which are accepted into the network.

The network topology of the UUNET at the United States, as shown in Figure 3, is used in our simulation. [4] Each link is full duplex with a bandwidth capacity of 155Mbps(OC3). The source node, the destination node, the bandwidth (or delay) requirement and the background traffic are randomly generated independently for every routing request.

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. In Figure 4, we compare the message overhead of two algorithms: the *proposed bandwidth-constrained-routing* algorithm and the *bounded flooding* algorithm [8].

The bandwidth-constrained routing algorithm is a combination of the b-computation and the w-computation. In our simulation, b-computation(s, t, B, id, l) is used in Step 2 of bandwidth-constrained-routing(s, t, B, id), and $l = \delta_{s,t} + 2$. (See Section 2.2.1 for the revised algorithm of the b-computation.) The timeout period of the w-computation is 60 time units. The arrival rate of routing requests at each node is 1 request per time unit. The average bandwidth requirement of the routing requests is 0.1 Mbps.

Most existing distributed routing algorithms [10, 12] uses either a distance-vector protocol or a link-state protocol to maintain a global network state, based on which the routing decision is made on a hop-by-hop basis. Maintaining a

[3] In this paper, *delay* is defined only on links and paths with non-zero lengths. $d_{t,t}$ is undefined because the length of the shortest paths from t to itself is zero. Mathematically, we let $b_{t,t} = 0$ in order to uniforming the algorithm and eliminating the discussion of special cases.

[4] Only the network topology comes from the UUNET. All other parameters are assigned by our own assumptions, independent from those of the UUNET.

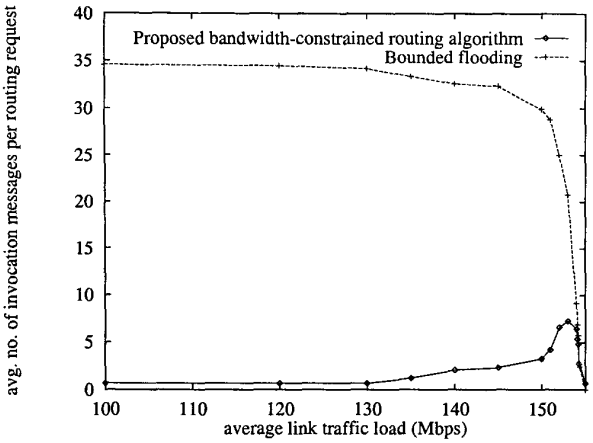

Figure 4. Average message overhead of the bandwidth-constrained routing

global state causes the scalability problem [1, 2]. Our algorithm requires every node to maintain only its local state.[5] The most related work was done by Kweon and Shin [8]. Their bounded flooding algorithm does not require a global network state to be maintained, either. It floods routing messages from the source to the destination. The routing messages check the bandwidth availability of the intermediate links as they traverse towards the destination. In order to reduce the overhead, routing messages proceed only along those paths whose length is bounded by certain number l. In our simulation, we choose $l = \delta_{s,t} + 2$. By simple analysis, it can be shown that the bounded flooding algorithm is equivalent to the revised b-computation.

Each point in Figure 4 is taken by averaging the overhead of one thousand requests. The average message overhead of our bandwidth-constrained routing algorithm is constantly modest; as shown from the figure, it is always bounded by 7, much lower than the average overhead of the bounded flooding algorithm, which can go up to 35 messages. The reason is that our bandwidth-constrained routing algorithm tries to route as many requests as possible by the w-computations, which are much cheaper because only the shortest paths are involved and the routing information is reused. On the contrary, the bounded flooding algorithm is equivalent to the revised b-computation, and it searches much more paths and thus has a much larger invocation tree than the w-computation does.

Our simulation also shows that the call admission ratios of the above two algorithms are the same. This is nothing surprising because the bounded flooding algorithm is equivalent to the revised b-computation, i.e., Step 2 of our bandwidth-constrained routing algorithm. The simulation result about the call admission ratios is not presented in this paper due to the lack of space.

[5]Note that $b_{i,t}$ and $d_{i,t}$ are not the results of a link-state (or distance-vector) protocol but the by-product of the routing process itself.

4 Conclusion

We proposed several distributed routing algorithms based on distributed recursive computation and information reuse. For the bandwidth-constrained routing, the b-computation finds a solution path if there exists one; the w-computation computes the maximum residual bandwidth among all shortest paths. For the delay-constrained routing, the d-computation finds a solution path if there exists one; the f-computation computes the minimum end-to-end delay among all shortest paths. An information-reuse scheme was proposed for the w-computation (f-computation), which result in a bounded overall overhead — at most n^2 w-computations (f-computations) for any given timeout period. Our simulation showed that the average overhead per routing request is modest and practical, especially when considering that QoS routing is done not on a per-data-packet basis but to set up a connection, along which thousands or even millions of data packets may be transmitted.

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 quality-of-service routing in high-speed networks based on selective probing. *LCN'98*, 1998.

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

[4] S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network*, Nov./Dec. 1998.

[5] S. Chen and K. Nahrstedt. Routing by distributed recursive computation and information reuse. *Technical Report, University of Illinois at Urbana-Champaign, Department of Computer Science*, 1998.

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

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

[8] S. Kweon and K. G. Shin. Distributed qos routing using bounded flooding. *Technical Report, Real-Time Computing Laboratory, University of Michigan*, 1998.

[9] 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.

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

[11] 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.

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