A Distributed Algorithm for Delay-Constrained Unicast Routing *

Douglas S. Reeves† Hussein F. Salama‡

February 18, 1998

Abstract

In this paper, we study the *NP*-hard delay-constrained least-cost path problem. A solution to this problem is needed to provide real-time communication service to connection-oriented applications, such as video and voice. We propose a simple, distributed heuristic solution, called the delay-constrained unicast routing (DCUR) algorithm. DCUR requires limited network state information to be kept at each node: a cost vector and a delay vector. We prove DCUR's correctness by showing that it is always capable of constructing a loop-free delay-constrained path within finite time, if such a path exists. The worst case message complexity of DCUR is $O(|V|^2)$ messages, where $|V|$ is the number of nodes. However, simulation results show that, on the average, DCUR requires much fewer messages. Therefore, DCUR scales well to large networks. We also use simulation to compare DCUR to the optimal algorithm, and to the least-delay path algorithm. Our results show that DCUR's path costs are within 10% from those of the optimal solution.

Keywords: Routing, Delay Constraints, Quality of Service, Distributed Algorithms

*This work was supported in part by the Center for Advanced Computing and Communication at North Carolina State University, and by AFOSR grant F49620-96-1-0061. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the AFOSR or the U.S. Government.

†Address: Departments of Computer Science and Electrical and Computer Engineering, North Carolina State University, Box 8206, Raleigh, NC 27695. Email: reeves@csc.ncsu.edu

‡Address: Cisco Systems, San Jose, CA 95134. Email: hsalama@cisco.com

1 Introduction

New distributed applications for packet- and cell-switched networks are emerging at a fast rate. Many of these applications are real-time in nature, including, for example, video and voice transmission, interactive simulation and gaming, distributed control, etc. Most of these applications result in a fairly steady flow of data over long periods of time between the participants. It is therefore appropriate to think of the participants in a distributed real-time application as engaging in an interactive session of significant duration.

Distributed real-time applications require the network to provide strict bounds on delay and other quality of service (QoS) metrics, such as loss and jitter. A common approach to providing QoS guarantees is to establish a connection between the participants that has the properties needed to meet the needs of the application. This connection must follow a path that has suitable properties, including adequate resources (e.g, bandwidth and buffer space). Real-time traffic in general, and especially video, often utilizes a significant amount of resources. Managing the network resources efficiently will reduce the cost of the network service, and allow more applications to run simultaneously.

The task of finding a path through a network is handled by routing protocols. A unicast routing protocol finds a path that connects the data source to a single destination, while a multicast routing protocol finds a path connecting the data source to two or more destinations. In this paper, we consider the class of applications having only a single destination. Unicast routing protocols can be classified into two general types: distance-vector protocols (such as the routing information protocol, RIP [1]), and link-state protocols (such as the open shortest path first protocol, OSPF [2]). Distance-vector protocols are based on a distributed version of Bellman-Ford's shortest path (SP) algorithm [3], which requires only a modest amount of information to be stored at each node. Link-state protocols use a non-distributed algorithm due to Dijkstra [3]. With a link-state protocol, each node maintains complete information about the network topology.

Considering the message complexity, distance-vector routing protocols scale well to large networks, since each node exchanges information only with its direct neighbors, and the amount of information stored at each node is smaller. Due to their distributed nature, distance-vector protocols may suffer from looping problems

when the network is not in steady state. Link-state protocols do not scale as well, because flooding is used to update the nodes' topology information, and the amount of information stored at each node is larger. They do not suffer from looping problems, however, because of their centralized nature. In 1995, Garcia-Luna-Aceves and Behrens [4] proposed a distributed protocol based on link vectors, that avoids looping problems and scales well to large networks.

Both Bellman-Ford's and Dijkstra's SP algorithms are exact and run in polynomial time. As the name indicates, an SP algorithm minimizes the sum of the lengths of the individual links on the path from source to destination, and thus optimizes a single additive metric. If the length of a link is a measure of the delay on that link, then an SP algorithm will compute the least-delay (LD) path; if the link length is a measure of the link cost, an SP algorithm will compute the least-cost (LC) path. For distributed real-time applications, it is desirable to find a path that considers two metrics at the same time. That is, the delay along the path should be acceptable to the application, and the cost of the path should be as low as possible. In this paper, we study the problem of unicast routing of real-time traffic subject to an end-to-end delay constraint in connection-oriented networks. We term this the Delay-Constrained Least Cost (DCLC) path problem. The DCLC problem has been shown to be NP-hard [5].

Widyono [6] proposed an optimal centralized delay-constrained algorithm to solve the DCLC problem. His algorithm, called the constrained Bellman-Ford (CBF) algorithm, performs a breadth-first search to find the optimal DCLC path. Due to its worst-case exponential running time, CBF is not practical for large networks. Jaffe [7] studied a variation of the problem in which the path cost and the path delay are defined as two constraints. He proposed a pseudo-polynomial-time heuristic and a polynomial-time heuristic for solving the problem. The polynomial-time algorithm minimizes ($L(p) + d * W(p)$), where $L(p)$ is the length of the path, $W(p)$ is the cost of the path, and d is a weighting factor. This variation of the shortest-path problem can fail to find a delay-constrained solution when one exists. Wang and Crowcroft [8] investigated routing subject to multiple quality of service constraints in datagram networks. They considered multiplicative and concave constraints in addition to additive constraints. They presented algorithms for delay-constrained, minimum-bandwidth-constrained routing. Aida et al. [9] presented a method for optimal delay-constrained

routing when the delay is a random variable. Cost, however, was not considered in this work. Rampal and Reeves [10] investigated routing algorithms for real-time traffic. Their focus was on the call admission probability, rather than the minimum cost.

Delay-constrained unicast routing is a special case of the delay-constrained multicast routing problem which has received a lot of attention in recent years [11]. Thus, delay-constrained multicast routing heuristics can be used to solve the DCLC problem. However, these delay-constrained multicast heuristics require complete information about the network topology to be available at every node, and their running times grow at fast rates with the network size [11]. Therefore, the delay-constrained multicast routing heuristics cannot be applied to large networks.

We present in this paper a new solution to the DCLC problem. Our solution is called the delay-constrained unicast routing (DCUR) algorithm. DCUR is distributed and requires only a modest amount of information at each node, similar to distance-vector protocols. DCUR is also a heuristic, and does not suffer from the excessive running times required by optimal solutions.

The remainder of this paper is organized as follows. In section 2, we state the DCLC problem formally. In section 3, we describe the routing information needed at each node by the DCUR algorithm. Then, in section 4, we present DCUR, prove its correctness, and analyze its complexity. In section 5, we evaluate DCUR's performance using simulation. Section 6 concludes the paper.

2 Problem Formulation

We represent a point-to-point communication network N as a set of nodes (switches, routers) connected by directed links,[1] where V denotes the set of nodes and E denotes the set of links. Any node is reachable from any other node in this network, and any two nodes are directly connected by at most one link in each direction. A directed link $e = (u, v) \in E$ has a cost $C(e)$ and a delay $D(e)$ associated with it. $C(e)$ and $D(e)$ may take any nonnegative real values. The link delay $D(e)$ is a measure of the delay a packet experiences when

[1]We study networks with asymmetric link costs and delays as the more general and realistic case.

traversing the link e. The link cost $C(e)$ may reflect the monetary cost of the link, or may represent some other metric which is desirable to optimize, such as the link's utilization.

We define a path as an alternating sequence of nodes and links $P(v_0, v_k) = v_0, e_1, v_1, e_2, v_2, \cdots, v_{k-1}, e_k, v_k$, such that $e_i = (v_{i-1}, v_i) \in E$, for $1 \leq i \leq k$. A path contains loops if not all its nodes are distinct. In the remainder of this paper, it will be explicitly mentioned if a path contains loops. Otherwise a "path" always denotes a loop-free path. We will use the following notation to represent a path: $P(v_0, v_k) = \{v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_{k-1} \rightarrow v_k\}$. For a given source node $s \in V$ and destination node $d \in V$, $\mathcal{P}(s, d) = \{P_1, \cdots, P_m\}$ is the set of all possible paths from s to d. The cost of a path P_i is defined as:

$$Cost(P_i) = \sum_{e \in P_i} C(e). \tag{1}$$

Similarly, the end-to-end delay along the path P_i is defined as:

$$Delay(P_i) = \sum_{e \in P_i} D(e). \tag{2}$$

Thus, both cost and delay are examples of additive metrics. The DCLC problem finds the LC path from a source node s to a destination node d such that the delay along that path does not exceed a delay constraint Δ. It is a constrained minimization problem that can be formulated as follows.

Delay-Constrained Least-Cost (DCLC) Path Problem: *Given a directed network $N = (V, E)$, a nonnegative cost $C(e)$ for each $e \in E$, a nonnegative delay $D(e)$ for each $e \in E$, a source node $s \in V$, a destination node $d \in V$, and a positive delay constraint Δ, the constrained minimization problem is:*

$$\min_{P_i \in \mathcal{P}'(s,d)} Cost(P_i) \tag{3}$$

where $\mathcal{P}'(s, d)$ is the set of paths from s to d for which the end-to-end delay is bounded by Δ. Therefore $\mathcal{P}'(s, d) \subseteq \mathcal{P}(s, d)$. If $P_i \in \mathcal{P}(s, d)$ then $P_i \in \mathcal{P}'(s, d)$ if and only if

$$Delay(P_i) \leq \Delta. \tag{4}$$

The DCLC problem is *NP*-hard [5] (p.214). It remains *NP*-hard in the case of undirected networks. However, it is solvable in polynomial time if all link costs are equal or all link delays are equal.

3 Routing Information

In this section, we discuss the routing information which needs to be present at any node in the network to assure successful execution of DCUR. Each node $v \in V$ must have the following information available during the computation of the delay-constrained path: the cost of each outgoing link of v, the delay of each outgoing link of v, a cost vector, a delay vector, and a routing table. The cost vector and delay vector structures are presented below, and the routing table structure will be described in the next section.

The cost vector at node v consists of $|V|$ entries, one entry for each node w in the network. Each entry in the cost vector holds the following information:

- the destination node ID, w;
- the cost of the LC path from v to w, denoted by $least\_cost\_value(v, w)$; and,
- the ID of the next hop node on the LC path from v to w, denoted by $least\_cost\_nexthop(v, w)$.

Similarly, the delay vector at node v has one entry for each node w in the network. Each entry in the delay vector holds:

- the destination node ID, w,
- the total end-to-end delay of the LD path from v to w, $least\_delay\_value(v, w)$, and
- the ID of the next hop node on the LD path from v to w, $least\_delay\_nexthop(v, w)$.

The cost vectors and delay vectors are similar to the distance vectors of some existing routing protocols [1]. Distance-vector-based protocols such as RIP discuss in detail how to update the distance vectors in response to topology changes, and how to prevent instability. These procedures are simple and require the contents of the distance vector at each node to be periodically transmitted to direct neighbors of that node only. The same procedures used for maintaining the distance vectors can be used for maintaining the cost vectors and delay vectors required by our algorithm. We will not discuss these procedures in this paper. We assume that the cost vectors and delay vectors at all nodes are up-to-date. We also assume that the link costs, the link delays, the contents of the cost vectors, and the contents of the delay vectors do not change during the execution of the routing algorithm.

The specification of RIP [1] discourages the use of dynamic metrics, such as link utilization. The reason is that dynamic metrics make it difficult for the nodes to converge to correct, consistent vector values. On the other hand, dynamic metrics based on the utilization of resources allow more efficient management of the network than fixed metrics which do not adapt to changing network conditions. This is particularly important for multimedia applications, which can be resource-intensive. DCUR can be implemented using either fixed metrics or dynamic metrics.

In addition to the cost vector and delay vector, each node v maintains a routing table and an invalid-link table. The contents of these two tables are described in the next section.

4 The Delay-Constrained Unicast Routing (DCUR) Algorithm

In this section we describe the DCUR algorithm, which is a heuristic solution to the DCLC problem. We explain how DCUR is executed in a distributed fashion. In the initial description, we ignore the possibility of loops. After presenting this basic algorithm, we show how loops may be created, and how DCUR detects and eliminates them. This completes the description of DCUR. We then prove its correctness, and analyze its complexity (number of messages exchanged, and running time).

DCUR is a source-initiated algorithm that constructs a delay-constrained path connecting source node s to destination node d. The path is constructed one node at a time, from the source to the destination. Any node v at the head of the partially-constructed path can choose to add one of only two alternative outgoing links. One link is on the LC path from v to the destination, while the other link is on the LD path from v to the destination. This limitation considerably reduces the amount of computation required at any node, at the expense of restricting DCUR's ability to construct the optimal path.

An application wishes to establish a connection from source node s to destination node d, with delay constraint Δ. The source node s assigns a unique identifier for the proposed session, denoted I. This allows the construction of different paths from node s to node d, for applications with differing requirements. The source node s initiates path construction by looking up the $least\_delay\_value(s, d)$ from its delay vector. If

this value is greater than the delay constraint Δ, then no delay-constrained paths exist between s and d; DCUR reports failure and stops execution. If, however, delay-constrained paths do exist, i.e.,

$$least\_delay\_value(s, d) \leq \Delta, \tag{5}$$

the algorithm continues execution until a path is found. The source s becomes the currently active node, denoted $active\_node$. At all times there is only one active node, located at the head of the path which is under construction. The active node determines the next link to add to the path, while other nodes merely respond to messages from the active node.

Each node v in the network maintains a routing table of all paths (either partially- or fully-constructed) which pass through it. Routing table entries are created while establishing a connection for a session. When a real-time session terminates, the path is torn down, and the routing table entry created for the session is deleted from each node along the path. A routing table entry can also store information unrelated to routing, such as the status of a connection, the resources reserved for it, etc. These uses are outside the scope of this paper.

As part of a routing table entry for session I, the active node stores the delay along the partially-constructed path from s to itself; this delay is denoted $delay\_so\_far$. The $active\_node$ reads the ID of the next hop node on the LC path towards d, $least\_cost\_nexthop(active\_node, d)$, from its cost vector. In the following, $least\_cost\_nexthop(active\_node, d)$ is abbreviated as lc_nhop for convenience. Then $active\_node$ sends a message to lc_nhop, requesting the LD value from lc_nhop to d. This message is termed a QUERY message. In response to the QUERY message, lc_nhop looks up the requested value, $least\_delay\_value(lc\_nhop, d)$, from its delay vector. This information is sent back to the $active\_node$ in a RESPONSE message. After $active\_node$ receives the RESPONSE message, it checks if

$$delay\_so\_far + D(active\_node, lc\_nhop) + least\_delay\_value(lc\_nhop, d) \ \leq \ \Delta \tag{6}$$

is satisfied. If it is satisfied, and link $(active\_node, lc\_nhop)$ is still valid for this session (as denoted by its absence from the invalid-link table; see section 4.1 below on loop removal), then there exist valid delay-constrained paths from $active\_node$ to d which use the link $(active\_node, lc\_nhop)$, and $active\_node$ selects

the direction of the LC path towards d. If the inequality is not satisfied, or an entry for this link and session exists in the invalid-link table, then $active\_node$ selects the direction of the LD path towards d. The LD path from $active\_node$ to d is guaranteed to be part of at least one delay-constrained path from s to d; otherwise, $active\_node$ could not have been selected in a previous step.

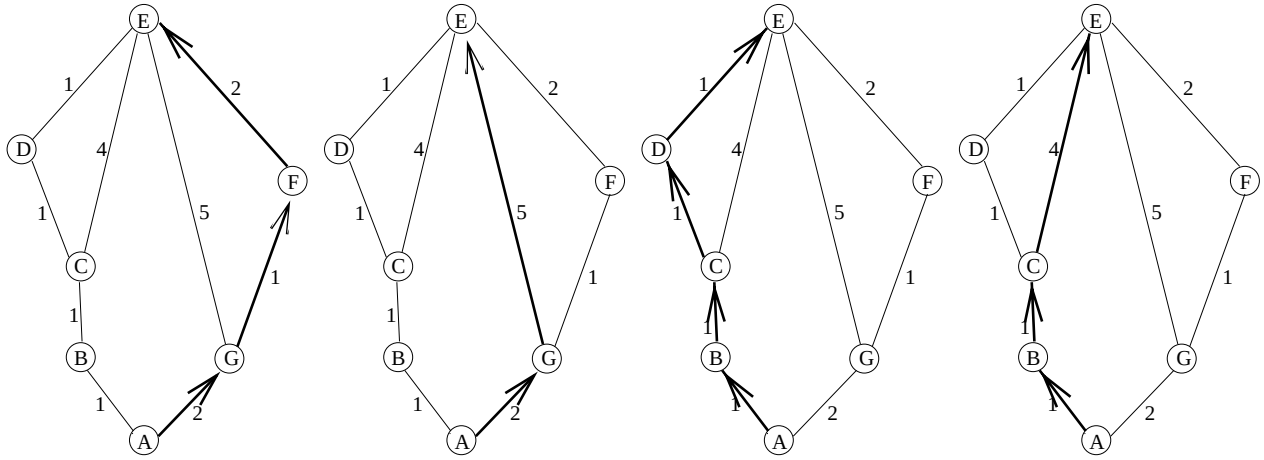
After deciding which direction to follow, $active\_node$ creates a routing table entry for this session. The entry for session I contains the following information:

- the ID of the session, I;
- the source of the session, s;
- the destination of the session, d;
- the $previous\_active\_node$ of this path, described below;
- $next\_node = \begin{cases} least\_cost\_nexthop(active\_node, d), & \text{if the LC path direction is chosen,} \\ least\_delay\_nexthop(active\_node, d), & \text{if the LD path direction is chosen,} \end{cases}$
- $previous\_delay = delay\_so\_far$, and
- $direction\_taken = \begin{cases} LCPATH & \text{if the LC path direction is chosen,} \\ LDPATH & \text{if the LD path direction is chosen.} \end{cases}$

For a node v which is part of a path for session I, $previous\_active\_node$ is the identity of the neighboring node which immediately precedes v on the path, while $next\_node$ is the identity of the neighboring node which immediately follows v. For the source node s of a session, $delay\_so\_far$ is set to 0 , and $previous\_active\_node$ is set to $null$.

After creating a routing table entry for the session, the $active\_node$ adds $D(active\_node, next\_node)$ to the variable $delay\_so\_far$ to create the sum $delay\_so\_far\_nexthop$. $delay\_so\_far\_nexthop$ represents the delay along the already constructed path from s to $next\_node$. $active\_node$ then sends a CONSTRUCT_PATH message to $next\_node$. The information contained in a CONSTRUCT_PATH message is the session ID I, the ID of the source s, the ID of the destination d, the value of the delay constraint Δ, and the value of $delay\_so\_far\_nexthop$. After sending out the CONSTRUCT_PATH message, $active\_node$ becomes inactive.

When a node $v \neq d$ receives a CONSTRUCT_PATH message, it becomes the new $active\_node$. The new



(a) Optimal DCLC path, cost = 5, delay = 3

(b) LD path, cost = 7, delay = 2

(c) Unconstrained LC path, cost = 4, delay = 4

(d) DCUR path, cost = 6, delay = 3

Figure 1: Paths constructed by different algorithms from source node A to destination node E. All link delays are equal to 1, and are not shown in the figure. Link costs are shown next to each link. The delay constraint, Δ, is equal to 3.

$active\_node$ makes a routing decision, creates a routing table entry, and continues path construction, as described above. As part of the routing table entry, $previous\_active\_node$ is set to the ID of the node which sent it a CONSTRUCT_PATH message.

When the destination node d receives a CONSTRUCT_PATH message, it creates a routing table entry. In this entry, $next\_node = null$ and $direction\_taken = null$; other fields in the entry are determined as described above. The destination then sends an acknowledgment back to the source. [2] When the source receives the acknowledgment message, it signals to the application that a connection has been established, so that data can begin being transmitted to the destination.

Figure 1 shows an example[3] of the paths obtained by different routing algorithms to connect source node

[2]The acknowledgment message can either travel the constructed path backwards or it can be sent over either the LC path or the LD path from the destination to the source. The choice is straight-forward and will not be described in this paper. Tearing down an existing path is also a simple operation that will not be discussed in this paper.

[3]Figures 1 and 2 show examples of undirected networks for simplicity. DCUR can be applied to both directed and undirected networks.

A to destination node E, with a delay constraint of 3. Subfigure 1(d) shows the path DCUR constructs. DCUR proceeds as follows. The source A adds the first link on the LC path towards E, link (A, B), after checking that there exist delay-constrained paths from A to E that utilize (A, B). Then node B adds the first link on its LC path towards E, link (B, C), after checking that there exist delay-constrained paths from A to E that utilize (A, B) and (B, C). Node C next determines that the first link on its LC path towards E, link (C, D), cannot be used. This is because the subpath $\{A \rightarrow B \rightarrow C \rightarrow D\}$ is not part of any delay constrained path from A to E. Thus C decides to continue via the LD path direction. It adds the first link in that direction, link (C, E); this completes the path to the destination.

An $active\_node$, does not send a QUERY if the next hop node is the same on both the LC path and the LD path from $active\_node$ to the destination, i.e., $least\_cost\_nexthop(active\_node, d) = least\_delay\_nexthop(active\_node, d)$. It is known in advance that the LD direction satisfies the delay constraint, so there is no need for the QUERY message. In this case, $active\_node$ sets the $direction\_taken$ in the routing table entry to $LDPATH$. The reason for that particular setting will be explained later in this section, when routing loops are discussed.

The paths constructed by existing distance-vector protocols are guaranteed to be loop-free if the contents of the distance vectors at all nodes are up-to-date and the network is in stable condition. However, up-to-date cost vector and delay vector contents and stable network condition are not sufficient to guarantee loop-free operation for DCUR. In DCUR, each node involved in the path construction operation selects either the LC path direction or the LD path direction as has been explained above. If all nodes choose the LC path direction, or all nodes choose the LD path direction, then no loops can occur, because the resulting paths are the LC path or LD path respectively. However, if some nodes choose the LC path direction while others choose the LD path direction, loops may occur. In the following subsection, we discuss how DCUR detects and eliminates loops.

4.1 Loop Removal

Figure 2 shows a scenario that results in a loop. The source node A initiates the construction of a path towards the destination node D with an imposed delay constraint value of 8. Subfigures 2(a), 2(b), and 2(c) show

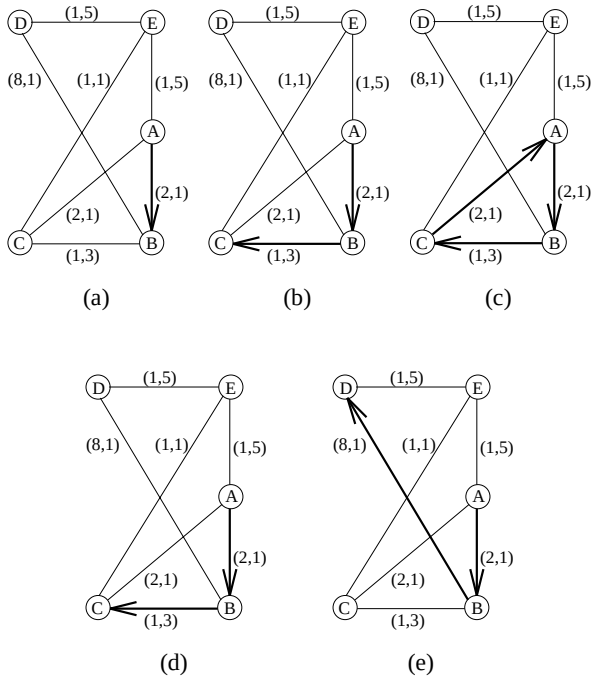


Figure 2: Scenario illustrating loop creation. A is the source and D the destination. Link costs and link delays are shown next to each link as (cost,delay). $\Delta = 8$.

successive stages of path construction until a loop is created. The source A follows the LD path direction towards the destination D and link (A, B) becomes the first link in the path. Node B follows the LC path direction towards D and adds link (B, C) to the path. Node C follows the LD path direction and adds link (C, A) to the path. This creates the loop $\{A \rightarrow B \rightarrow C \rightarrow A\}$, as shown in subfigure 2(c).

The occurrence of a loop may be easily detected. When a node receives a CONSTRUCT_PATH message and becomes the active node, it checks whether an entry exists in its routing table for the session specified in the message. If such an entry does exist, $active\_node$ has detected the existence of a loop, and we call it the closing node of the loop. The active node therefore initiates a loop removal operation, as follows. The contents of $active\_node$'s routing table entry are left unchanged. $active\_node$ sends a REMOVE_LOOP message to the previous node in the loop. For the closing node, this is the node that just sent it a CONSTRUCT_PATH message; for other nodes in the loop, it is recorded as $previous\_active\_node$ in the routing table entry for this session. The REMOVE_LOOP message only needs to contain the ID of this session. Then $active\_node$

becomes inactive.

The REMOVE_LOOP message traverses the loop backwards, removing routing table entries, until it finds a node w whose routing table entry's $direction\_taken$ is set to $LCPATH$. This indicates that w took the LC path direction towards the destination. There must be at least one node on the loop that follows the LC path direction, since loops can not be created if all nodes follow the LD path direction.

The REMOVE_LOOP message is not propagated any further after it arrives at such a node w, which becomes the active node. Node w reverses its prior decision to take the LC path, instead choosing the LD path to avoid the conditions that caused the loop. This new decision can never lead to any delay constraint violations. Thus w adjusts the contents of its routing table entry for this session such that $next\_node = least\_delay\_nexthop(w, d)$ and $direction\_taken = LDPATH$. The variables $previous\_node$, $previous\_delay$, and $delay\_so\_far$ remain unchanged. In addition, w creates an entry in its invalid-link table for link $(w, least\_cost\_nexthop(w, d))$ for session I. This removes this link from any further consideration during construction of the path from s to d for this session. [4] Then w sends a CONSTRUCT_PATH message to $next\_node$, and path construction continues.

For the example of figure 2, node A detects the existence of a loop. It reacts by sending a $Remove\_Loop$ message that traverses the loop backwards to node C. C is already following the LD path direction towards the destination, so all it does is send the REMOVE_LOOP message further backwards to B, and delete its routing table entry, thereby removing link (C, A) from the path (subfigure 2(d)). Node B receives the REMOVE_LOOP message. It is following the LC path direction towards the destination, so it decides to follow the LD path direction instead. To do so, B modifies its routing table entry to remove link (B, C) from the path and add link (B, D) instead. At the same time, B creates an entry for (B, C) in its invalid-link table, which prevents this link from being used any further. Then B continues constructing the path by sending a CONSTRUCT_PATH

[4] Note that removal of a routing table entry at a node is *not* accompanied by removal of any invalid-link entry for the same session. This is so the same loop will not be constructed twice during the construction of a path from the source to the destination. An entry in the invalid-link table can be safely removed when enough time has elapsed to be sure path construction for the session has completed; implementation is not discussed further in this paper.

message to D, which is the destination. The final delay-constrained path from A to D is the one shown in subfigure 2(e).

It was mentioned above that, at a node w, the routing table entry's $direction\_taken$ is set to $LDPATH$ when the LC path and the LD path use the same link to the next hop. If the $direction\_taken$ were set to $LCPATH$ and then w received a REMOVE_LOOP message, w would remove the link leading to the next node in the LC path direction and then add the same link to the path again, because that link also leads to the next node in the LD path direction. The result would be the same loop occurring twice. The correct setting of $direction\_taken$ prevents this from happening.

The description of DCUR is now complete. Complete pseudo code for the algorithm can be found in appendix. In the remainder of this section, we prove the correctness of DCUR and study its complexity.

4.2 Correctness of DCUR

We verify the correctness of DCUR by proving that it always constructs a loop-free delay-constrained path from the source s to the destination d, if such a path exists. We also prove that DCUR will always terminate.

A link is added each time a CONSTRUCT_PATH message is generated. Each link that is added creates a path. In the following, we denote the path created by the i^{th} successful (non-looping) CONSTRUCT_PATH message as P_i. If $P_i = v_0 \rightarrow \cdots \rightarrow v_i$, let $V_i = \{v_0, \cdots, v_i\}$ be the set of nodes connected by those links, where $s = v_0$ is the tail of the path, and v_i is the head of the path. Also let $\mathcal{P}_i = \{P_0, ..., P_i\}$ be the first i paths constructed by DCUR. DCUR constructs path P_{i+1} by extending a path in $\mathcal{P}_i$.

Theorem 1 *A path constructed by DCUR for session I with source s and destination d does not contain loops.*

Proof. A path P_{i+1} is created by adding a link to the head of a previously-constructed path $P_j \in \mathcal{P}_i$. A loop can only be created if this link joins v_j with another node $v_k \neq v_j$, $v_k \in V_j$. However, node v_k's routing table entry for I will indicate it is already part of this path, and DCUR will remove the link that was just added. Since loops are detected and broken, the final path cannot contain loops. □

Theorem 2 *Algorithm DCUR always terminates.*

Proof.

If there is no delay-constrained path between the source s and the destination d, DCUR immediately terminates. Otherwise, it constructs one or more paths, terminating when d is reached. We show below that every path constructed by DCUR is unique. Since there are a finite number of unique, loop-free paths in a graph of finite size, this will prove that DCUR always terminates.

The proof proceeds by induction on the number of paths constructed. For the basis of the induction, $i = 0$, $P_0 = \{s\}$, and $\mathcal{P}_0 = \{P_0\}$. Any path constructed by extending P_0 is obviously not in $\mathcal{P}_0$.

For the inductive step, assume every path in $\mathcal{P}_i$ is unique. If P_{i+1} extends P_i, it is the first extension of P_i, and P_{i+1} is therefore unique. The link used to extend P_i in this case can be either an LC link or an LD link. If the attempt to add a link to P_i would result in a loop, this loop is unrolled back to some path $P_j \in \mathcal{P}_i$, where $P_j \neq P_i$. Loop unrolling will only stop at a path which was extended by an LC link. Path P_j is instead extended by the LD link to construct P_{i+1}. P_j could not previously have been extended by an LD link, or loop unrolling would not have stopped at P_j. This guarantees that P_{i+1} is not in $\mathcal{P}_i$, so P_{i+1} is also unique in this case. □

Theorem 3 *DCUR will construct a delay-constrained path from a given source s to destination d if and only if such a path exists.*

Proof:

Only if part: If no delay-constrained path exists from s to d, then $least\_delay\_value(s, d) > \Delta$, and DCUR terminates immediately without constructing a path.

If part: The proof is done by induction on i, the number of paths constructed by DCUR. We show that all paths constructed by DCUR, including those which terminate at the destination d, meet the delay constraint. The basis for the induction is $i = 0$, $P_0 = \{s\}$, and $\mathcal{P}_0 = \{P_0\}$. Since $Delay(P_0) = 0$ and $least\_delay\_value(s, d) \leq \Delta$, the basis is proved.

For the inductive step, assume that

$$Delay(P_j) + least\_delay\_value(v_j, d) \leq \Delta, \quad 0 \leq j \leq i. \tag{7}$$

Let P_{i+1} extend a path in $\mathcal{P}_i$. If P_{i+1} extends P_i by adding the LC link from v_i, then inequality 6 must be satisfied. This inequality can be rephrased as follows after substituting $Delay(P_i)$ for $delay\_so\_far$, v_i for $active\_node$, and v_{i+1} for lc_nhop:

$$Delay(P_i) + D(v_i, v_{i+1}) \ + \ least\_delay\_value(v_{i+1}, d) \ \leq \ \Delta \tag{8}$$

$$Delay(P_{i+1}) \ + \ least\_delay\_value(v_{i+1}, d) \ \leq \ \Delta.$$

P_{i+1} can otherwise extend a path $P_j \in \mathcal{P}_i$ by adding the LD link from v_j. In this case,

$$least\_delay\_value(v_j, d) \ = \ D(v_j, v_{i+1}) + least\_delay\_value(v_{i+1}, d), \tag{9}$$

and we can restate inequality 7 as:

$$Delay(P_j) + D(v_j, v_{i+1}) \ + \ least\_delay\_value(v_{i+1}, d) \ \leq \ \Delta \tag{10}$$

$$Delay(P_{i+1}) \ + \ least\_delay\_value(v_{i+1}, d) \ \leq \ \Delta.$$

In either case, the delay of path P_{i+1} is not greater than the delay constraint, and the inductive step is proved.

□

4.3 Complexity of DCUR

The computational complexity of the proposed distributed algorithm at any node is $O(1)$. This is because each time a node receives a CONSTRUCT_PATH message or a REMOVE_LOOP message, it performs a fixed amount of computations, irrespective of the size of the network.

We now consider the worst case message complexity of DCUR, i.e., the number of messages needed in the worst case, in order to construct a loop-free path from a given source s to a destination d. The number of messages needed to construct a path (or loop) is proportional to the number of links in the path (or loop), because a node running DCUR exchanges at most three messages to add one link. [5] Likewise, the number of

[5]The need for the QUERY and RESPONSE messages can be completely eliminated by making use of the fact that a node transmits the contents of its cost vector and delay vector periodically to all its neighbors. Thus if a node saves a copy of the cost vector and delay vector from each of its neighbor nodes, then there is no need for the QUERY RESPONSE messages. However, this increases the storage requirements at each node.

messages needed to remove a loop is no greater than the number of links in the loop, because DCUR generates only one message to remove one link.

For a network of size $|V|$ nodes, the longest loop-free path contains at most $|V|$ nodes and $(|V|-1)$ links. Constructing this loop-free path requires O(V) messages in the worst case.

During construction of this path, loops may be created. Each loop which is created contains at least one LC link. Removal of a loop invalidates (removes from further consideration during the construction of the path) one such link. Since there are exactly $(|V|-1)$ LC links in the graph (one from each node other than the destination), and since each loop must contain at least one such link, it follows that no more than $(|V|-1)$ loops can be created. The maximum size of any loop created by DCUR is $|V-1|$ nodes and $(|V|-1)$ links, since no loop can contain the destination. Creating and removing a loop requires O(V) messages in the worst case.

The number of messages generated during the creation and removal of O(V) loops, each of which contains O(V) links, is O(V^2). It follows that DCUR needs O(V^2) messages to create a loop-free path to the destination, in the worst case. Fortunately, our simulation results show that DCUR's average performance is much better than this worst case. These results will be presented in the next section.

5 Simulation Results

We used simulation to evaluate the average performance of DCUR. Full duplex, directed, simple, connected networks of different sizes with homogeneous link capacities of 155 Mbps (OC3) were used in the experiments. The positions of the nodes were fixed in a rectangle of size $4000 * 2400$ Km^2, roughly the area of the continental USA. A random generator was used to create links interconnecting the nodes [12]. The output of this random generator is always a connected network in which each node's degree is at least 2. We adjusted the parameters of the random generator carefully to obtain realistic network topologies with an average node degree of 4, which is close to the average node degree of current internetworks.

The propagation speed through the links was taken to be two thirds the speed of light. Under this

assumption, the size of the rectangle enclosing our network is $20 * 12$ msec^2. In addition, we assumed a high-speed networking environment with small packet (cell) sizes and limited buffer space at each node. The link propagation delay was dominant under these assumptions, and the queuing component of the link delay was neglected. The link delays were symmetric, because the link lengths were symmetric; therefore, $D(u, v) = D(v, u)$. We defined the cost of link e, denoted $C(e)$, as a function of its utilization. Link costs were asymmetric, because $C(u, v)$ and $C(v, u)$ are independent.

We conducted two experiments to evaluate DCUR's performance. The first experiment measured the average number of messages generated by DCUR while constructing delay-constrained paths. The second experiment compared the quality (cost) of paths generated by DCUR with the quality of delay-constrained paths constructed by two other important routing algorithms.

5.1 The Average Message Complexity of DCUR

In the first experiment, we measured the average number of messages required to establish a delay-constrained path. In order for DCUR to be practical to use in large networks, the number of messages generated must grow slowly as the network size increases.

For network of a fixed size and a fixed delay constraint, we generated a random set of links to interconnect the fixed nodes, we selected a random source and destination, and we generated a random amount of background traffic on each link. The cost (utilization) of a link was a random variable uniformly distributed between 5 Mbps and 125 Mbps. This procedure was repeated, with network sizes ranging from 20 nodes up to 200 nodes, and delay constraints ranging from 15 to 55 msec. Relatively small values were chosen for the delay constraint; these are necessary to allow the higher level end-to-end protocols enough time to process the transmitted information, while maintaining the interactive quality of the real-time session.

In this experiment, we measured the average number of messages exchanged between the nodes which execute the distributed DCUR algorithm to construct a delay-constrained path. Note that any message generated by DCUR travels a distance of only one hop. Unless otherwise stated, DCUR was run repeatedly with new random numbers until confidence intervals of less than 5% of the mean value, using 95% confidence

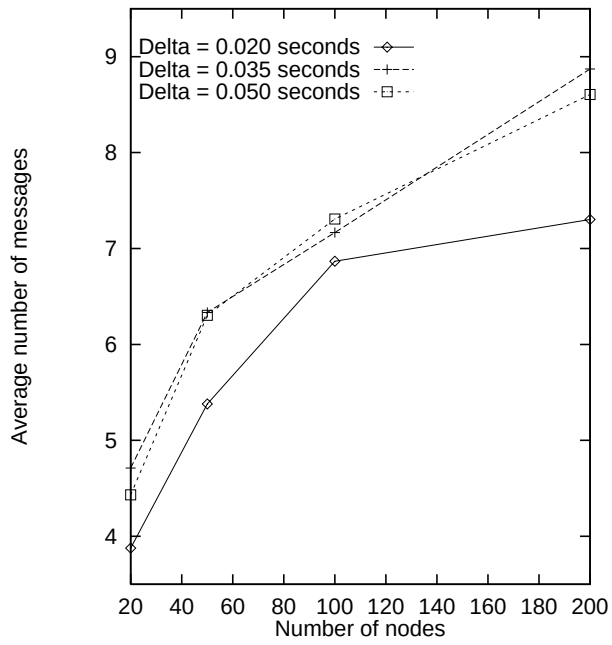


Figure 3: Average number of messages, variable network size, average node degree 4, three delay constraint settings: 20 msec, 35 msec, and 50 msec.

levels, were achieved for all measured values presented in this subsection and in the next subsection.

Figure 3 shows the average number of messages versus the size of the network for three different values of the delay constraint: a strict value of 20 msec, a moderate value of 35 msec, and a lenient value of 50 msec. All three curves of figure 3 indicate clearly that the average number of messages grows very slowly with the size of the network. For any of the delay constraint values shown in the figure, doubling the size of the network increases the average number of DCUR's messages by roughly one message only. Thus the average growth rate of the number of messages is approximately logarithmic in the network size.

A path that satisfies a strict delay constraint consists on the average of fewer links than a path that satisfies a lenient delay constraint. For a 200-node network the average number of links per path is 4.28 for a 20 msec delay constraint, 4.72 for a 35 msec delay constraint, and 5.12 for a 50 msec delay constraint. That is why the number of messages exchanged while constructing a path is smallest when the delay constraint value is small. In addition, when the delay constraint is strict, DCUR is forced to follow the LD path direction most of the time. Therefore, the probability of the occurrence of a loop is small, which also means fewer messages are

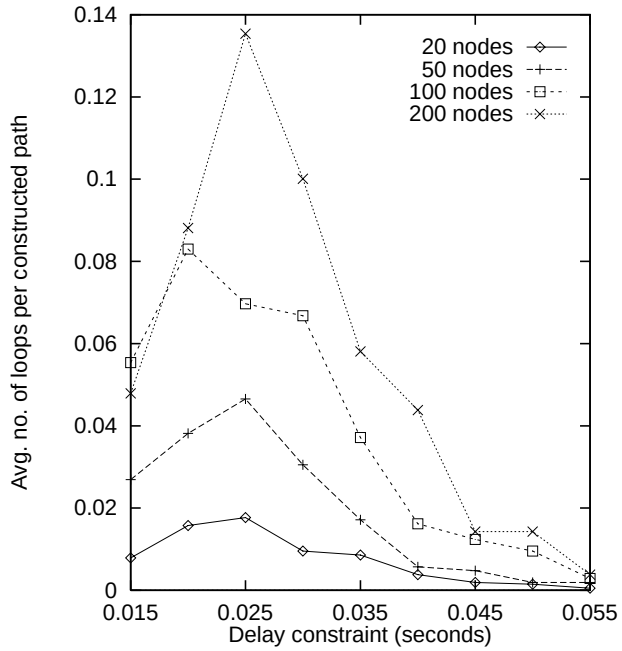


Figure 4: Average number of loops occurring while constructing a single delay-constrained path, network sizes of 20 nodes, 50 nodes, 100 nodes, and 200 nodes, average node degree 4, variable delay constraint.

generated. When the delay constraint is increased to 35 msec, the number of messages is largest. The reason is that 35 msec is a moderately strict delay constraint, and DCUR may be able to follow the LC path direction at some nodes and to follow the LD path direction at others. This toggling between LC path direction and LD path direction increases the probability of loop occurrence, and hence increases the average number of messages exchanged. Increasing the delay constraint further, from 35 msec to 50 msec, does not reduce the average number of messages. Under these conditions DCUR is able to follow the LC path direction most of the time without violating the delay constraint, which leads to slightly longer paths.

In order to verify our hypothesis that loops occur most frequently when the delay constraint is moderately strict, we collected additional data during the above experiment. For each successful run of DCUR (i.e., a run that successfully constructs a delay-constrained path), we measured the number of loop occurrences. Because loops occur infrequently (as will be shown), obtaining 5% confidence intervals for the average number of loop occurrences would have required an excessive amount of simulation time. One thousand successful runs of DCUR were simulated for each data point in figure 4. Figure 4 shows the average number of loop occurrences

per successful run of DCUR versus the delay constraint, for different network sizes. This experiment shows that loops occur most frequently when the delay constraint value ranges from 20 msec to 35 msec. When the delay constraint is lenient (larger than 35 msec) loop occurrences are very infrequent. The average number of loop occurrences also decreases when strict delay constraint values of less than 20 msec are used. Figure 4 also indicates that loops occur more frequently as the size of the network increases.

5.2 Comparison to Other Algorithms

In a second experiment we compared DCUR with two algorithms that are also applicable for delay-sensitive applications. The first algorithm is the LD path algorithm, or simply LDP. LDP is optimal with respect to the end-to-end delay, but it does not attempt to minimize the cost of the constructed path. Therefore, it may result in inefficient utilization of the link bandwidth. The other algorithm is CBF, which was briefly described in section 1. CBF constructs the optimal DCLC path, but its execution time grows exponentially with the network size.

The structure of the second experiment was similar to that of the first experiment. The only difference is that for each randomly selected source-destination pair we applied DCUR, LDP, and CBF, one at a time, to construct the delay-constrained path.

All three algorithms have the same success rate in satisfying the imposed delay constraint. This is because all of them are always capable of constructing a delay-constrained path, if one exists. The success rate for a network with 200 nodes is shown in figure 5. For each path constructed, the cost of the path was measured. Since the path cost of CBF is optimal, we present the costs of LDP and DCUR relative to CBF. More precisely, the inefficiency of an algorithm x is defined as:

$$inefficiency_x = \frac{(cost_x - cost_{CBF})}{cost_{CBF}} \tag{11}$$

Figure 6 shows the average inefficiency of LDP and DCUR relative to CBF for 200-node networks and a variable delay constraint. When the delay constraint is small, $<$ 20 msec, the number of alternate delay-constrained paths, available for the algorithms to choose from, is small, and therefore the differences between

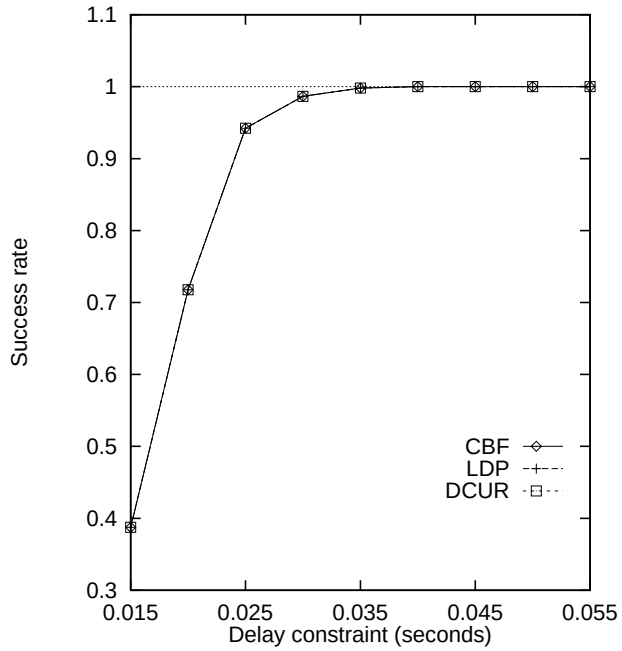


Figure 5: Success rate, 200-node networks, average node degree 4.

the algorithms are also small. For delay constraint values between 25 msec and 35 msec, DCUR is up to 10% worse than the optimal CBF. The reason is that, because of the tight delay constraint, DCUR can not always follow the unconstrained LC path direction. In some cases, it has to follow the LD path direction instead. The toggling between these two directions affects DCUR's ability to create low-cost paths. However, DCUR remains on the average more efficient than LDP. When the value of the delay constraint exceeds 45 msec, DCUR's inefficiency approaches zero. This is because it almost exclusively elects to follow the LC path direction. LDP, in contrast, becomes more and more inefficient as the delay constraint increases, and is always worse than DCUR. Since it optimizes delay rather than cost, this is not surprising.

Figure 6 indicates that DCUR's path costs are always within 10% from the path cost of the optimal CBF. Thus DCUR's cost performance is quite satisfactory, especially considering that CBF is a centralized algorithm that requires global information about the network topology, while DCUR is a distributed heuristic that requires only limited information to be maintained at each node (one cost vector and one delay vector).

In addition to 200-node networks, we simulated 20-node, 50-node, and 100-node networks as well. The results for the different network sizes are similar to results shown above for the 200-node case. This indicates

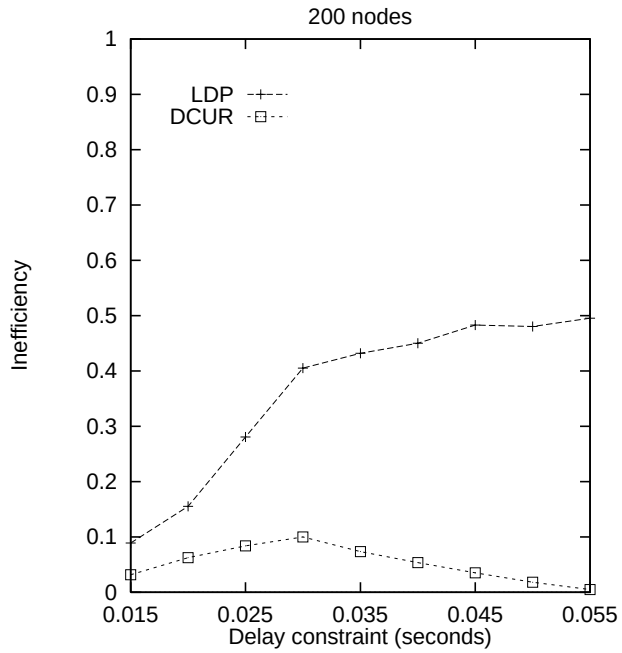


Figure 6: Inefficiency, 200-node networks, average node degree 4, variable delay constraint.

that the performance of the different algorithms relative to each other does not depend significantly on the network size.

6 Conclusions

We studied the delay-constrained routing problem in point-to-point connection-oriented networks. Our work was motivated by the fast evolution of delay-sensitive distributed applications. We formulated the problem as a delay-constrained least-cost (DCLC) path problem, which is known to be *NP*-complete. Therefore, we proposed a distributed, source-initiated heuristic solution, the delay-constrained unicast routing (DCUR) algorithm, to avoid the excessive complexity of the optimal solutions. DCUR requires only a limited amount of information at each node. The information at each node is stored in a cost vector and a delay vector. These vectors are constructed and maintained in exactly the same manner as the distance vectors which are widely deployed over current networks. The basic idea of DCUR is to restrict the amount of computation by limiting the number of links to choose from when constructing delay-constrained path for a given source-destination pair. We proved the correctness of DCUR by showing that it is always capable of constructing a loop-free

delay-constrained path within finite time, if such a path exists. The worst case message complexity of DCUR is dominated by the occurrence and removal of loops. It requires $O(|V|^2)$ messages in the worst case. Simulation results show that DCUR requires much fewer messages on the average, because loop occurrence is rare in realistic networks. We compared the performance of DCUR to CBF, which is an optimal DCLC path algorithm. We also compared DCUR to LDP, a shortest path algorithm that minimizes the end-to-end delay. Our evaluation of the cost performance of the algorithms showed that DCUR is always within 10% from the optimal CBF, while LDP is up to 50% worse than optimal in some cases. All three algorithms have identical success rates for constructing delay-constrained paths.

In summary, DCUR is a simple, efficient, distributed algorithm that scales well to large network sizes. This encourages us to use it as a starting point for implementing a routing protocol that is capable of providing QoS guarantees for real-time applications. Future work should focus on mechanisms to cope with transient situations when the contents of the cost vectors and the delay vectors at different nodes are not consistent. In addition, we would like to extend DCUR to routing of delay-constrained multicast trees.

References

[1] C. Hedrick, "Routing Information Protocol." Internet RFC 1058, `http://ds.internic.net/rfc/rfc1058.txt`, June 1988.

[2] J. Moy, "OSPF Version 2." Internet RFC 1583, `http://ds.internic.net/rfc/rfc1583.txt`, March 1994.

[3] D. Bertsekas and R. Gallager, *Data Networks*. Prentice-Hall, 2nd ed., 1992.

[4] J. Garcia-Luna-Aceves and J. Behrens, "Distributed, Scalable Routing Based on Vectors of Link States," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1383–1395, October 1995.

[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. Widyono, "The Design and Evaluation of Routing Algorithms for Real-Time Channels," Tech. Rep. ICSI TR-94-024, University of California at Berkeley, International Computer Science Institute, June 1994.

[7] J. Jaffe, "Algorithms for Finding Paths With Multiple Constraints," *Networks*, vol. 14, pp. 95–116, 1984.

[8] Z. Wang and J. Crowcroft, "Quality of Service Routing for supporting Multimedia Applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, September 1996.

[9] M. Aida, I. Nakamura, and T. Kubo, "Optimal Routing in Communication Networks with Delay Variations," in *Proceedings of IEEE INFOCOM '92*, pp. 153–159, 1992.

[10] S. Rampal and D. Reeves, "An Evaluation of Routing and Admission Control Algorithms for Multimedia Traffic," *Computer Communications*, vol. 18, no. 10, pp. 755–768, October 1995.

[11] H. Salama, D. Reeves, and Y. Viniotis, "Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 332–345, April 1997.

[12] H. Salama, *Multicast Routing for Real-time Communication on High-Speed Networks*. PhD thesis, North Carolina State University, Department of Electrical and Computer Engineering, November 1996. (also available from ftp://ftp.csc.ncsu.edu/pub/rtcomm/SalamaThesis.ps.Z).

Appendix Pseudo-Code of DCUR

First here is a list of all the control messages exchanged between nodes implementing DCUR:

CONSTRUCT_PATH(session, source node, destination node, delay constraint value, value of the delay from the source to the downstream node receiving the message)
QUERY(destination node)
RESPONSE(destination node, least delay value from the responding node to the destination)
REMOVE_LOOP(session, source node, destination node)

The following function is executed by the source node s when it receives a request from an application to construct a delay-constrained path to a destination node d.

```
Initiate_Path_Construction(session I, source node s, destination node d, delay constraint Δ) {
    if least_delay_value(s, d) > Δ send a failure indication to the application;
    else {
        active_node := s;
        previous_active_node := null;
        delay_so_far := 0;
        call Path_Construction(active_node, previous_active_node, I, s, d, Δ, delay_so_far);
    };
};
```

The following function is executed by node $active\_node$ when it receives a CONSTRUCT_PATH message from a node $previous\_active\_node$. It is also called the source node when initiating the path construction process.

```
Path_Construction(current node active_node, previous node previous_active_node, session I,
                  source node s, destination node d, delay constraint Δ, current delay delay_so_far) {
```

if $active\_node = d$ {
create a routing table entry with $session := I$, $source := s$, $destination := d$,
$previous\_node := previous\_active\_node$, $next\_node := null$, and $previous\_delay := delay\_so\_far$;
send an acknowledgement message (path construction is complete) back to s;
}
else {
if a routing table entry corresponding to session I, source s, and destination d already exists,
send a REMOVE_LOOP(I, s, d) message to $previous\_active\_node$;
else {
$use\_LDPATH := False$;
$lc\_nhop := least\_cost\_nexthop(active\_node, d)$;
$ld\_nhop := least\_delay\_nexthop(active\_node, d)$;
if $lc\_nhop = ld\_nhop$
$use\_LDPATH := True$;
if there exists an entry in the invalid-link table for link $(active\_node, lc\_nhop)$ for session I
$use\_LDPATH := True$;
if $use\_LDPATH = False$ {
send QUERY(d) message to lc_nhop;
wait to receive a RESPONSE($d, delay$) message from lc_nhop;
if $(delay\_so\_far + D(active\_node, lc\_nhop) + delay) \leq \Delta$ {
create a routing table entry with $session := I$, $source := s$, $destination := d$,
$previous\_node := previous\_active\_node$, $next\_node := lc\_nhop$,
$previous\_delay := delay\_so\_far$, and $direction\_taken := LCPATH$;
$delay\_so\_far := delay\_so\_far + D(s, lc\_nhop)$;
send a CONSTRUCT_PATH($I, s, d, \Delta, delay\_so\_far$) message to lc_nhop;
}
else $use\_LDPATH := True$;
};
if $use\_LDPATH = True$ {
create a routing table entry with $session := I$, $source := s$, $destination := d$,
$previous\_node := previous\_active\_node$, $next\_node := ld\_nhop$,
$previous\_delay := delay\_so\_far$, and $direction\_taken := LDPATH$;
$delay\_so\_far := delay\_so\_far + D(s, ld\_nhop)$;
send a CONSTRUCT_PATH($I, s, d, \Delta, delay\_so\_far$) message to ld_nhop;
};
};
};

The following function is executed by node n when it receives a QUERY message from node $active\_node$.

Process_Query(current node n, querying node $active\_node$, destination node d) {
send a RESPONSE($d, least\_delay\_value(n, d)$) message back to $active\_node$;
};

The following function is executed by node $active\_node$ when it receives a REMOVE_LOOP message.

```
Loop_Removal(current node active_node, session I, source s, destination d) {
    find the routing table entry corresponding to I;
    in that routing table entry, if direction_taken = LCPATH {
        create an invalid-link table entry for link (active_node,least_cost_nexthop(active_node, d)), session I;
        nhop := least_delay_nexthop(active_node, d);
        in the routing table entry corresponding to I, set {
            direction_taken := LDPATH;
            next_node := nhop;
        };
        delay_so_far := previous_delay + D(active_node, nhop);
                        (previous_delay is given in the routing table entry)
        send a CONSTRUCT_PATH(I, s, d, Δ, delay_so_far) message to nhop;
    }
    else {
        send a REMOVE_LOOP(I, s, d) message to the previous_node given in that routing table entry;
        delete that routing table entry;
    };
};
```