A Distributed Algorithm for Delay-Constrained Unicast Routing*[†]

Hussein F. Salama
ECE Department
N.C. State University
Box 7911, Raleigh, NC 27695
hfsalama@eos.ncsu.edu

Douglas S. Reeves
CSC and ECE Departments
N.C State University
Box 8206, Raleigh, NC 27695
reeves@eos.ncsu.edu

Yannis Viniotis
ECE Deparmtent
N.C. State University
Box 7911, Raleigh, NC 27695
candice@eos.ncsu.edu

Abstract

In this paper, we study the NP-hard delay-constrained least-cost path problem, and propose a simple, distributed heuristic solution: 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 $O(|V|^3)$ *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.*

1 Introduction

New distributed applications are emerging at a fast rate. These applications typically involve real-time traffic that requires quality of service (QoS) guarantees. Traffic streams carrying voice, video, or critical real-time control signals have particularly stringent end-to-end delay requirements. In addition, real-time traffic usually utilizes a significant amount of resources while traversing the network. Hence the need for routing mechanisms which are able to satisfy the delay requirements of real-time traffic and to manage the network resources efficiently.

Unicast routing protocols can be classified into two categories: distance-vector protocols, e.g., the routing information protocol (RIP) [1], and link-state protocols, e.g., 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]. Considering the message complexity, distance-vector routing protocols scale well to large networks, because each node sends periodical topology update messages only to its direct neighbors. Each node maintains only limited information about the shortest paths to all other nodes in the network. Due to their distributed nature, distance-vector protocols may suffer from looping problems when the network is not in steady state. In link-state protocols, on the other hand, each node maintains complete information about the network topology, and uses this information to compute the shortest path to a given destination centrally using Dijkstra's algorithm [3]. Link-state protocols have limited scalability, because flooding is used to update the nodes' topology information. 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 the sum of the lengths of the individual links on the path from source to destination. If the length of a link is a measure of the delay on that link, then an SP algorithm computes the least-delay (LD) path, and if the link length is set equal to the link cost, then an SP algorithm computes the least-cost (LC) path.

We study the problem of unicast routing of real-time traffic subject to an end-to-end delay constraint in connection-oriented networks. We formulate the problem as a Delay-Constrained LC (DCLC) path problem. This problem is *NP*-hard [5]. Therefore, we propose a distributed heuristic solution: the delay-constrained unicast routing (DCUR) algorithm. 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. Unfortunately, due to its optimality, CBF's worst case running times grow exponentially with the size of the network. Jaffe [7] studied a variation of the problem in which the path cost and the path delay are defined as two constraints, and he proposed a pseudo-polynomial-time heuristic and a polynomial-time heuristic for solving the problem. The path cost (and similarly the path delay) is an additive metric, i.e, it is equal to the sum of the costs of the links on the path. Wang and Crowcroft [8] investigated the routing problem subject to multiple quality of service

*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.

constraints in datagram networks. They considered multiplicative and concave constraints in addition to additive constraints.

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

2 Problem Formulation

A point-to-point communication network is represented as a directed, simple, connected network $N = (V, E)$, where V is a set of nodes and E is a set of directed links. Any 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 traversing the link e. The link cost $C(e)$ may be either a monetary cost or some measure of 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:

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

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:*

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

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]. 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. Every node $v \in V$ must have the following information available during the computation of the delay-constrained path: the costs of all outgoing links, the delays of all outgoing links, 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, $least\_cost\_value(v, w)$, and
- the ID of the next hop node on the LC path from v to w, $least\_cost\_nhop(v, w)$.

Similarly, the delay vector at node v has one entry for each node w in the network. However, 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, $least\_delay\_nhop(v, w)$.

The cost vectors and delay vectors are similar to the distance vectors of some existing routing protocols [1]. Distance-vector based protocols 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. 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.

4 The Delay-Constrained Unicast Routing (DCUR) Algorithm

We start by presenting a simple version of DCUR. Then we discuss how loops may be created, and how DCUR detects them and eliminates them. After completing the description of DCUR, we prove its correctness and derive its complexity.

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 end 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 restricts DCUR's ability to construct the optimal path, but it considerably reduces the amount of computation required at any node.

In the following, we describe a simple version of DCUR which assumes that no routing loops can occur. 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, and DCUR fails and stops. If, however, delay-constrained paths do exist, i.e.,

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

the algorithm proceeds. The source s becomes the current active node, denoted $active\_node$. At all times there is only one active node, at the end of the partially-constructed path. The variable $delay\_so\_far$ is set to 0, and the variable $previous\_active\_node$ is set to $null$.

The $active\_node$ reads the ID of the next hop node on the LC path towards d, $least\_cost\_nhop(active\_node, d)$, from its cost vector. $least\_cost\_nhop(active\_node, d)$ is denoted as lc_nhop for convenience. Then $active\_node$ sends a $Query$ message to lc_nhop, requesting the LD value from lc_nhop to d. lc_nhop looks up the requested value, $least\_delay\_value(lc\_nhop, d)$, from its delay vector, and sends a $Response$ message back to $active\_node$ with this information. After $active\_node$ receives the $Response$ message, it checks if

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

If the inequality is satisfied, then there exist 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, 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 (a proof is provided in subsection 4.2). After deciding which direction to follow, $active\_node$ creates a routing table entry with the following information:

- the ID of s,
- the ID of d,
- $previous\_node =$ ID of the $previous\_active\_node$,
- $next\_node = \begin{cases} lc\_nhop, \\ \quad \text{if LC path direction is chosen,} \\ least\_delay\_nhop(active\_node, d), \\ \quad \text{if LD path direction is chosen,} \end{cases}$
- $previous\_delay = delay\_so\_far$, and
- $flag = \begin{cases} LCPATH \\ \quad \text{if LC path direction is chosen,} \\ LDPATH \\ \quad \text{if LD path direction is chosen.} \end{cases}$

Then $active\_node$ adds $D(active\_node, next\_node)$ to the variable $delay\_so\_far$. Finally the $active\_node$ sends a $Construct\_Path$ message to $next\_node$ that contains: the ID of the source s, the ID of the destination d, the value of the delay constraint Δ, and the updated value of $delay\_so\_far$ which represents the delay along the already constructed path from s to $next\_node$. 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 $active\_node$ sets $previous\_active\_node$ to be the ID of the node which sent it a $Construct\_Path$ message. Then the new $active\_node$ executes the same procedure just described.

When the destination node d receives a $Construct\_Path$ message, it records the ID of the node which sent the message. d creates a routing table entry, with the following values: ID of the source s, ID of the destination d, $previous\_node = previous\_active\_node$, $next\_node = null$, and $previous\_delay = delay\_so\_far$. Then the destination sends an acknowledgment back to the source. When the source receives the acknowledgment message, it signals to the application that the path construction has been successfully completed, and traffic can be transmitted along that path.

An $active\_node$, does not send a $Query$ message 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., $least\_cost\_nhop(active\_node, d) = least\_delay\_nhop(active\_node, d)$. 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 $flag$ 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 vectors and delay vectors 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 1 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 1(a), 1(b), and 1(c) show 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 1(c).

DCUR detects loops as follows. When a node receives a $Construct\_Path$ message, it searches its routing table. A loop is detected if a routing table entry already exists for the source-destination pair specified in the $Construct\_Path$ message.

The active node, $active\_node$, that detects a loop initiates the loop removal operation. The contents of $active\_node$'s routing table entry are left unchanged. $active\_node$ sends a $Remove\_Loop$ message to the previous node on the loop, $previous\_active\_node$ (the node from which $active\_node$

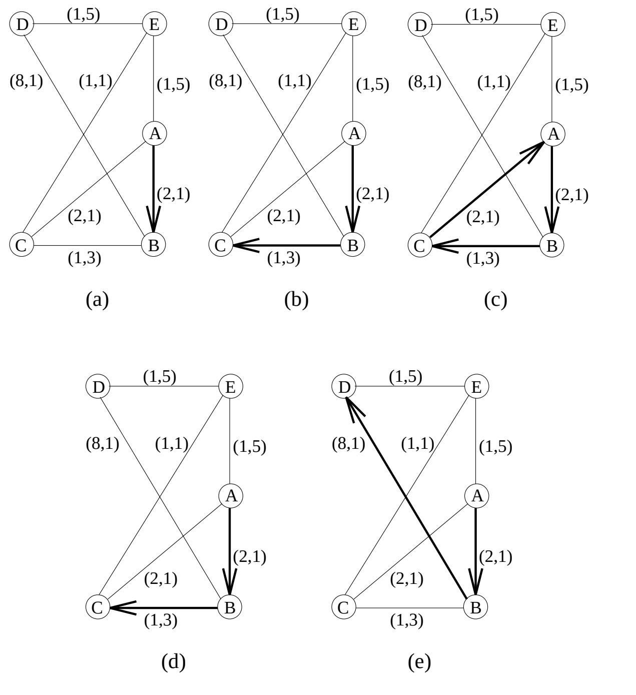

Figure 1: Example of a loop scenario. A is the source and D the destination. Link costs and link delays are shown next to each link as (cost,delay). $\Delta = 8$.

received the last $Construct\_Path$ message), and then $active\_node$ becomes inactive. The IDs of the source and destination nodes are all that needs to be included in the $Remove\_Loop$ message. The $Remove\_Loop$ message traverses the loop backwards, removing routing table entries, until it finds a node w whose routing table entry's $flag$ is set to $LCPATH$ indicating that this node is following the LC path direction towards the destination. There must be at least one node on the loop that follows the LC path direction, because, as we mentioned before, loops can not be created if all nodes follow the LD path direction. The $Remove\_Loop$ message is not sent any further backwards along the loop, after it arrives at w. Node w then decides to follow the LD path direction, instead of the LC path direction, in order to avoid the conditions that caused the loop. This decision can never lead to any delay constraint violations. Thus w adjusts the contents of its routing table entry so that $next\_node = least\_delay\_nhop(w, d)$ and $flag = LDPATH$. The variables $previous\_node$, $previous\_delay$, and $delay\_so\_far$ remain unchanged. Then w sends a $Construct\_Path$ message to $next\_node$, and path construction continues.

For the example of figure 1, node A detects the existence of a loop. It reacts by sending a $Remove\_Loop$ message that traverses the loop backwards. Node C receives the $Remove\_Loop$ message from A, but C is already following the LD path direction towards the destination, so all it does is to send the $Remove\_Loop$ message further backwards to B, and to delete its routing table entry, thereby removing link (C, A) from the path (subfigure 1(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, and modifies its routing table entry accordingly. Thus removing link (B, C) from the path and adding link (B, D) instead. Then B continues constructing the path by sending a $Construct\_Path$ message to D, which is the destination. The final delay-constrained path from A to D is the one shown in subfigure 1(e).

It was mentioned above that, at a node w, the routing table entry's $flag$ is set to $LDPATH$ when both the LC path direction and the LD path direction share the same link to the next hop. The reason is that if the $flag$ was set to $LCPATH$, and then w received a $Remove\_Loop$ message, it would have removed the link leading to the next node in the LC path direction, and then it would have added the same link to the path again, because that link leads also to the next node in the LD path direction. The result would have been the same loop occurring twice.

The description of DCUR is now complete. Complete pseudo code for the algorithm can be found in [9]. 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 can always construct a loop-free delay-constrained path within a finite time, if such a path exists.

Theorem 1 *DCUR always constructs a delay-constrained path for a given source s and destination d, if such a path exists.*

Proof. If no feasible path exists for a given source-destination pair, DCUR fails immediately at the source node after checking that the delay along the LD path exceeds the delay constraint, i.e., inequality 5 is not satisfied. If the LD path can not satisfy the delay constraint, no other path can. If at least one delay-constrained path from s to d exists, then inequality 5 will be satisfied, and path construction can start. Initially, the source s is the only member in the path. The rest of this proof is done by induction on j, where $P_j = \{v_0 \rightarrow \cdots \rightarrow v_j\}$ denotes the subpath constructed starting at the source, $s = v_0$, and ending at the current active node, $active\_node = v_j$, and j denotes the length of the path in hops. The basis for induction is $P_0 = \{v_0\}$. Since inequality 5 is satisfied, and $Delay(P_0) = 0$, it follows that

$$Delay(P_0) + least\_delay\_value(v_0, d) \leq \Delta. \tag{7}$$

Assume that

$$Delay(P_j) + least\_delay\_value(v_j, d) \leq \Delta. \tag{8}$$

Inequality 8 guarantees that the subpath P_j is part of at least one delay-constrained path from s to d. DCUR proceeds by adding either the first link along the LC path from v_j to d or the first link along the LD path from v_j to d. If DCUR adds the first link along the LC path, i.e., $v_{j+1} = lc\_nhop = least\_cost\_nhop(v_j, d)$, then inequality 6 must be satisfied. This inequality can be rephrased as follows after substituting $Delay(P_j)$ for $delay\_so\_far$ and v_j for $active\_node$ and v_{j+1} for lc_nhop:

$$\begin{aligned} & Delay(P_j) + D(v_j, v_{j+1}) + \\ & \quad least\_delay\_value(v_{j+1}, d) \quad = \\ & Delay(P_{j+1}) + \\ & \quad least\_delay\_value(v_{j+1}, d) \quad \leq \Delta. \end{aligned} \tag{9}$$

The other alternative for DCUR is to proceed from v_j by adding the first link along the LD path, i.e., $v_{j+1} = least\_delay\_nhop(v_j, d)$. In this case,

$$least\_delay\_value(v_j, d) = \\ D(v_j, v_{j+1}) + least\_delay\_value(v_{j+1}, d), \quad (10)$$

and we can restate inequality 8 as:

$$\begin{aligned} & Delay(P_j) + D(v_j, v_{j+1}) + \\ & \quad least\_delay\_value(v_{j+1}, d) \quad = \\ & Delay(P_{j+1}) + \\ & \quad least\_delay\_value(v_{j+1}, d) \quad \leq \Delta. \end{aligned} \quad (11)$$

In both cases, v_{j+1} becomes the next $active\_node$. It follows from inequalities 9 and 11 that the subpath from s to $active\_node$ is part of at least one delay-constrained path towards d. DCUR stops only when $active\_node = d$. □

Theorem 2 *The final path constructed by DCUR for a given source s and destination d does not contain any loops.*

Proof. We use the same notation used in the proof of theorem 1. Let $V_j = \{v_0, \cdots, v_j\}$ be the set of nodes in the subpath P_j. All nodes in V_j have a routing table entry for the source-destination pair, s and d. The active node, v_j, adds a link (v_j, v_{j+1}). If $v_{j+1} \in V_j$, a loop is created. Node v_{j+1} becomes the next active node. Node v_{j+1} searches its routing table for an entry corresponding to s and d. If $v_{j+1} \in V_j$, it will find such an entry, thus detecting a loop. We proved that when a link (v_j, v_{j+1}) is added that creates a loop, node v_{j+1} will always detect that loop.

Next we prove that when node v_{j+1} detects a loop, it calls a process that correctly breaks that loop. When v_{j+1} detects a loop, it sends a $Remove\_Loop$ message back to v_j. Node v_j's reaction to the receipt of the $Remove\_Loop$ message depends on the $flag$ in the routing table entry corresponding to s and d. In all cases, node v_j removes the link (v_j, v_{j+1}) from the path being constructed. This is sufficient to correctly break the detected loop. □

Theorem 3 *The execution time of DCUR for a given source s and destination d is always finite.*

Proof. If no delay-constrained paths exist, then DCUR fails immediately at the source after determining that inequality 5 is not satisfied. If inequality 5 is satisfied, then DCUR proceeds. If no loops occur, then, after adding at most $(|V| - 1)$ links, DCUR reaches the destination d. It remains to prove that even if loops occur, DCUR will still reach d within finite time. A subpath $P_j = \{v_0 \rightarrow \cdots \rightarrow v_i \rightarrow \cdots \rightarrow v_j\}$ ends with a loop if $v_j = v_i$ where $0 \leq i < j$ and $v_0 = s$. When the size of the network, $|V|$, is finite, the maximum number of distinct subpaths starting at s and ending with a loop is finite. Therefore, it is sufficient to prove that DCUR never attempts to construct the same subpath ending with a loop twice. When node $v_j = v_i$ detects a loop at the head of a subpath P_j^{LOOP}, it calls the loop removal procedure which traverses the path P_j^{LOOP} backwards removing links until a link $e_k^{LC} = (v_k, v_{k+1})$ is reached that is on the LC path direction from v_k towards d, where $i \leq k < j$. Link e_k^{LC} is removed from the path and path construction resumes by adding the link on the LD path direction from v_k towards d, link e_k^{LD}. One necessary condition to reconstruct P_j^{LOOP} is to readd link e_k^{LC} to the path being constructed. This means that a loop must occur, and to remove that loop DCUR removes link e_k^{LD}. However loop removal can not stop immediately after removing e_k^{LD}, because it is on the LD path direction towards d. Therefore loop removal must continue backwards until a link e_l^{LC} on the LC path direction from node v_l towards d is reached, where $0 \leq l < k$. DCUR removes link e_l^{LC}. Then path construction resumes and link e_k^{LC} may be readded to the subpath being constructed. Therefore, after a link e_k^{LC}, originally on a path P_j^{LOOP}, is removed from the path, it can be readded to the path only if a link e_l^{LC} is removed, where $0 \leq l < k < j$. The same holds for link e_l^{LC}. It follows that, the exact same subpath P_j^{LOOP} can not be reconstructed twice during the execution of DCUR. □

4.3 Complexity of DCUR

The computational complexity of the proposed distributed algorithm at any node is $O(1)$, 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 path for a given source-destination pair. If no loops occur, then the number of messages needed to construct a path is proportional to the number of links in the path, because a node running DCUR exchanges at most three messages to add one link. For a network size of $|V|$ nodes, the longest possible path from source to destination consists of $|V|$ nodes and $(|V| - 1)$ links. Therefore the number of messages needed in the worst case is $O(|V|)$, if it is guaranteed that no loops will occur. Unfortunately, the occurrence of loops complicates the analysis.

The tree of the LC paths from any node in the network to the destination node d, denoted *LCTREE*, consists of $(|V| - 1)$ links. Similarly, the tree of the LD paths from any node in the network to the destination d, denoted *LDTREE*, also consists of $(|V| - 1)$ links. The union of these two trees is a subnetwork $N' = (V, E')$, where $(|V| - 1) \leq |E'| \leq 2 * (|V| - 1)$, because some links may be members of both trees. Figure 2 shows an example of the union of an *LCTREE* and an *LDTREE*. In this example, the link (C, D) is a member of both trees. The $|E'|$ links are the only links considered by DCUR when constructing a path from a source s to the destination d, because, as has been explained before, at any node DCUR considers only the LC path direction and the LD path direction towards the destination.

Let the links of the *LCTREE* be called tree links. We add the links of the *LDTREE* to the *LCTREE* to obtain the subnetwork N'. The links of the *LDTREE* which are not already in the *LCTREE* will be classified into one of the following three link types.

- A back link which is traversed from a node to one of its ancestors[1]. A back link may result in a loop.

[1]A node v is an ancestor of a node w in the *LCTREE* if w is on the path

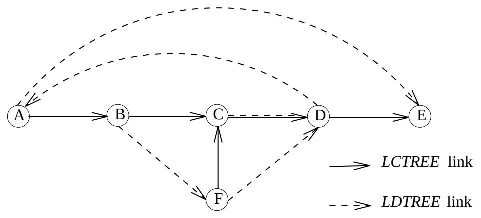

Figure 2: Example of a subnetwork constructed by taking the union of the *LCTREE* and the *LDTREE*. The destination is node E.

- A descendent link goes from a node to one of its descendants other than its child. A descendent link may provide one or more nodes with two alternate paths towards the destination.
- A cross link connects two nodes such that neither is a descendant of the other. A cross link may provide one or more nodes with two alternate paths towards the destination.

In the example of figure 2, links (A, B), (B, C), (C, D), (D, E), and (F, C) are tree links. The link (D, A) is a back link. Links (A, E) and (F, D) are descendent links, and the link (B, F) is a cross link.

A subnetwork N' has X back links, Y descendent links, and Z cross links where $0 \leq X, Y, Z \leq (|V| - 1)$ and $(X + Y + Z) \leq (|V| - 1)$. Adding a back link to a path under construction may or may not result in a loop. Since we are studying the worst case, we assume that adding a back link to a path always results in a loop. Consider a back link, e. Link e may be added and removed from the path being constructed several times, if it is reachable via multiple alternate paths from the source node. A loop results each time e is added. The back link e is reachable via $(Y + Z)$ alternate paths in the worst case. This happens when the $(Y + Z)$ descendent links and cross links are upstream from the back link e. In this case, each time DCUR attempts to use one of the $(Y + Z)$ resulting alternate paths, it may continue downstream and add the link e, thus creating a loop. If DCUR attempts to use all $(Y + Z)$ alternate paths while constructing the delay-constrained path, the link e will be added and removed $(Y + Z)$ times, which means that $(Y + Z)$ loops will be created and removed during path construction. The example of figure 2 is not a worst case scenario. However, it shows how the back link (D, A) can be reached via three alternate paths when node A is the source. The first alternative is the original path along the *LCTREE*: $\{A \rightarrow B \rightarrow C \rightarrow D\}$. The second alternative was created due to the addition of the cross link (B, F), and it is $\{A \rightarrow B \rightarrow F \rightarrow C \rightarrow D\}$. The final alternative is $\{A \rightarrow B \rightarrow F \rightarrow D\}$. This path was brought to existence by the descendent link (F, D).

So far we considered only one back link. However, the subnetwork N' contains X back links. In the worst case, each of the X back links is reachable via $(Y + Z)$ alternate paths. In this case we may end up with $X * (Y + Z)$ loops. Since $(X + Y + Z) \leq (|V| - 1)$, it follows that, in the worst case, DCUR may create and remove $O(|V|^2)$ loops before completing the construction of the delay-constrained path.

from v to the destination d. If v is an ancestor of w then w is a descendant of v. If the link (v, w) is a tree link, then w is v's child. In the *LCTREE* each node, other than d, has only one child.

The largest possible loop consists of $(|V| - 1)$ nodes and $(|V| - 1)$ links (the destination can not be part of a loop in DCUR). A maximum of three messages are needed to add one loop link. Thus it takes $O(|V|)$ messages to create the largest loop. One message is needed for removing one loop link, which means that at most $O(|V|)$ messages are needed if all loop links have to be removed before path construction resumes. Therefore, $O(|V|)$ messages are needed, to create and remove the largest loop. It follows that DCUR needs $O(|V|^3)$ messages to handle $O(|V|^2)$ loops in the worst case. Fortunately, our simulation results show that DCUR's average performance is much better than the worst case just studied. These results will be presented in the next section.

5 Simulation Results

We used simulation for our evaluation of 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 [9]. 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 queueing component of the link delay was neglected. The link delays were thus symmetric, $D(u, v) = D(v, u)$, because the link lengths were symmetric.

We defined the cost, $C(e)$, of link e, as a function of its utilization. We set the cost of a link to be equal to the sum of the equivalent capacities of the traffic streams traversing that link. Link costs were asymmetric, because $C(u, v)$ and $C(v, u)$ were independent. We conducted two experiments to evaluate DCUR's performance.

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. For each run of the experiment, we generated a random set of links to interconnect the fixed nodes, we selected a random source and a random destination, and we generated random background traffic to utilize each link. The cost of a link was a random variable uniformly distributed between 5 Mbps and 125 Mbps. The experiment was repeated with network sizes ranging from 20 nodes up to 200 nodes. We also varied the delay constraint value from 15 msec to 55 msec. We measured the average number of messages exchanged between the nodes which execute the distributed DCUR algorithm. Note that any message generated by DCUR travels a distance of one hop only. Unless otherwise stated, DCUR was run repeatedly until confidence intervals of less than 5% of the mean value, using 95% confidence level, were achieved for all measured values presented in this subsection and in the next subsection.

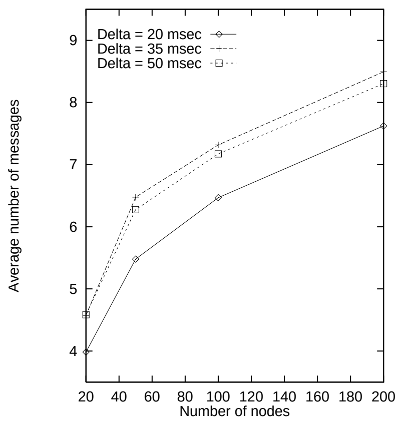

Figure 3: Average number of messages, variable network size, average node degree 4, three delay constraint settings: 20 msec, 35 msec, and 50 msec.

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 roughly 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, 20 msec. 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. As has been discussed in the previous section, the occurrence of loops increases in the number of messages.

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, leads to a reduction in the average number of messages, because for such a lenient value DCUR is able to follow the LC path direction most of the time without violating the delay constraint, and therefore it no longer toggles between the LC path direction and the LD path direction. The consequence is that loops occur rarely.

In order to verify our assumption, that loops occur most frequently when the delay constraint is moderately strict, we measured the average number of loop occurrences during one successful run of DCUR, i.e., a run that successfully constructs a delay-constrained path for a given source-destination pair. We found that loops do not occur frequently (less than 12 loops every 100 successful runs of DCUR). Therefore, it was not possible (due to the excessive simulation times) to repeat the experiment until small enough confidence intervals were achieved for the measured values of the average number of loop occurrences. 1,000 successful runs of DCUR were simulated for each 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. It shows that loops occur most frequently when the delay constraint value ranges from 20 msec to 45 msec. When the delay constraint is lenient (larger than 45 msec) loop occurrences are very infrequent, less than one loop every 100 successful runs of DCUR. The average number of loop occurrences also decreases when strict delay constraint values of less than 20 msec are used. Figure 4 indicates that loops occur more frequently as the size of the network increases.

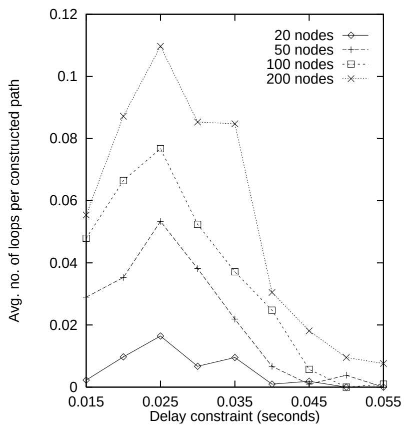

Figure 4: Average number of loops occurring while constructing a single delay-constrained path, network sizes of 20 nodes, 50 nodes, 100 nodes, and 200 nodes, average node degree 4, variable delay constraint.

5.2 Comparison to Other Algorithms

In this subsection, we show the results of the second experiment which compares DCUR with two algorithms that are also suitable 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 is 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. For each algorithm, we measured the average inefficiency relative to CBF. The average inefficiency of an algorithm x is defined as:

$$inefficiency_x = \frac{(cost_x - cost_{CBF})}{cost_{CBF}} \qquad (12)$$

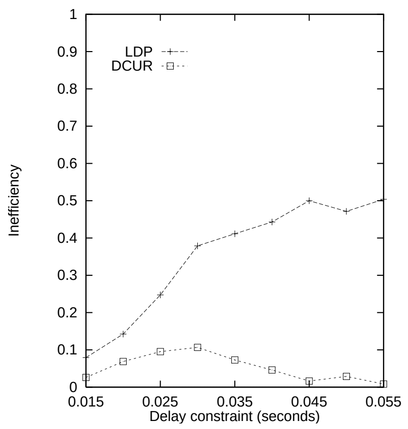

Figure 5: Inefficiency, 200-node networks, average node degree 4, variable delay constraint.

Figure 5 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 the algorithms are also small. For delay constraint values between 20 msec and 45 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, its effect on the constructed path is minimal. In that range, DCUR's inefficiency approaches zero, because it almost exclusively elects to follow the LC path direction. LDP does not attempt to minimize the path cost at all. That's why its inefficiency is up to 50% when the delay constraint value is large.

Figure 5 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 when 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).

Measurements from the same experiment indicate that the average end-to-end delays of DCUR and CBF are considerably larger than the minimal delays achieved by LDP. This is not a big advantage for LDP, though. More important is that all three algorithms are always capable of constructing a delay-constrained path, if such a path exists.

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 loop. It requires $O(|V|^3)$ messages in the worst case. Fortunately, however, our 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.

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 an routing protocol that is capable of providing QoS guarantees for real-time applications. Among others, future work should focus on specifying mechanisms that enable DCUR to cope with transient situations when the contents of the cost vectors and the delay vectors at different nodes are not consistent. In addition, future work should extend DCUR to address the multicast routing problem.

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, no. 1, pp. 95–116, Spring 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] H. Salama, *Multicast Routing for Real-time Communication on High-Speed Networks*. PhD thesis, North

Carolina State University, Department of Electrical and Computer Engineering, 1996. Available from `ftp://osl.csc.ncsu.edu/pub/rtcomm/rt-comm.html`.