Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Networks 47 (2005) 87–104

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

An efficient quality of service routing algorithm for delay-sensitive applications

Wei Liu [a,*], Wenjing Lou [b], Yuguang Fang [a]

[a] *Wireless Networks Laboratory (WINET), Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, United States*

[b] *Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609, United States*

Received 22 July 2003; received in revised form 13 May 2004; accepted 22 July 2004
Available online 11 September 2004

Responsible Editor: J. Hou

Abstract

It is well known that the Delay-Constrained Least-Cost (DCLC) unicast routing problem is NP-complete, hence various heuristic algorithms have been developed for this problem. In this paper, we propose a more efficient distributed algorithm, namely, Selection-Function-based DCLC (SF-DCLC), based on a novel selection function for the DCLC problem. The proposed SF-DCLC algorithm requires limited network state information at each node and is always able to find a loop-free path satisfying the delay bound if such paths exist. Simulation study shows that SF-DCLC is not as sensitive to the delay bound and the size of networks as some other DCLC routing algorithms, and has very low cost-inefficiency compared to the optimal one in various network scenarios we have studied. A noteworthy feature of SF-DCLC is that SF-DCLC has very high probability of finding the optimal solution in polynomial time with low computational complexity and message complexity.
© 2004 Elsevier B.V. All rights reserved.

Keywords: DCLC; QoS; Routing

1. Introduction

The emerging distributed real-time multimedia applications have diverse and stringent service requirements, defined by the quality of service (QoS) metrics in the service level agreements (SLA) between the service provider and the user applications. For example, the delay-sensitive applications such as real-time voice and video require the data streams to be received at the destination within a certain time deadline. Intensive research work has been carried out within the

* Corresponding author. Tel.: +1 352 392 8576.
E-mail addresses: liuw@ufl.edu (W. Liu), wjlou@ece.wpi.edu (W. Lou), fang@ece.ufl.edu (Y. Fang).

Internet engineering task force (IETF) in order to provide the support for such QoS requirements in the current computer networks, particularly over the Internet. Many service models and mechanisms, such as the integrated service (IntServ)/Resource Reservation Protocol (RSVP) model, the differentiated services (DiffServ) model, Multi-Protocol Label Switching (MPLS), traffic engineering, and QoS routing, have been proposed [1].

QoS routing is one of the most promising mechanisms developed in the current literature. The basic function of QoS routing is to find a *feasible path*, which has sufficient residual (unused) resources to satisfy the QoS requirements requested by a connection. Here, the QoS requirements are represented as a set of constraints, such as link constraints, end-to-end path constraints, or tree constraints for the entire multicast tree. The constraints can also be ones on bandwidth, delay, delay jitter, loss ratio, and so on. In addition, a QoS routing algorithm should also consider the optimization of resource utilization, which is usually measured by an abstract *cost* metric. The optimization of QoS routing is then to find the minimal cost path among all the feasible paths under the requested constraints [2].

Many QoS routing algorithms incorporating a variety of constraints have been proposed in the past few years. For unicast routing, the Multi-Constrained Optimal Path (MCOP, also known as PCPO) and the Multi-Constrained Path (MCP) problems are the most notorious ones for their NP-complete property [2]. MCOP routing is to find a path satisfying the required path constraints, meanwhile, the found path is optimized on another QoS metric. An example of MCOP is the delay-constrained least-cost (DCLC) routing problem, which is to find a least-cost path with bounded delay. MCP routing is to find a path satisfying multiple path constraints. An example of MCP is the delay/delay-jitter-constrained routing problem, which is to find a path with both bounded delay and bounded delay jitter. The two routing problems are related to each other, and MCP may be simpler than MCOP because MCP does not optimize on any metric, and instead, it only finds a path that meets all the constraints. Due to their NP-complete property, both routing problems are considered intractable for large networks. General approaches tackling the NP-complete problem are approximations or heuristic algorithms that guarantee to find a near-optimal solution with polynomial complexity.

In this paper, we propose an efficient distributed heuristic algorithm, namely, SF-DCLC, for DCLC unicast routing. The proposed algorithm makes use of two vectors, the least-delay path (LDP) vector and the least-cost path (LCP) vector. Our new algorithm uses a novel selection function, which leads to heuristics for finding a suboptimal path closer to the optimal one. This algorithm can easily find a loop-free delay-constrained path with only $O(|V|)$ message complexity in the worst case (where $|V|$ is the number of nodes in the network) and has very high probability of finding the optimal solution if such a path exists.

The rest of the paper is organized as follows. The related work on QoS routing is given in Section 2. Section 3 presents the proposed SF-DCLC algorithm. We start with the formulation of DCLC problem, and then describe the operations of SF-DCLC, followed by the correctness proof and complexity analysis. Simulations and performance evaluations are presented in Section 4. Finally, we conclude the paper in Section 5.

2. Related work

QoS routing has been extensively studied in recent years. In general, the QoS routing consists of two functions. The first one is the routing protocol whose task is to collect information about the states of the network and the available network resources and to distribute the information throughout the network. The second one is the routing algorithm whose task is to calculate the desired path with the information provided by the routing protocol. In this paper, we consider the latter one only. We assume that the first function is already in place and can always provide the most up-to-date information. Also, we only review works related to MCP and DCLC problems, a more detailed survey about QoS routing can be found in [2,32,33].

Many work has been carried out for the MCP problems. One large group of such algorithms is based on the calculation of a single metric aggregated from a combination of multiple weighted QoS constraints [3,6,19,21,22,24]. With the aggregated metric, a simple shortest-path algorithm can be used to find the shortest-path corresponding to the aggregated metric. Jaffe [3] proposed a pseudo-polynomial heuristic and a polynomial-time heuristic for the MCP problem under two constraints. Jaffe first defined an aggregated link cost as a linear combination function of the two link weights: $\omega(e) = \alpha\omega_1(e) + \beta\omega_2(e)$, where $\alpha, \beta \in Z^+$. With this aggregated cost, the path found by the Dijkstra's shortest-path algorithm does not necessarily satisfy the constraints. Thus, to appropriately choose the multipliers α and β such that the returned path is likely to satisfy the constraints is not a trivial task. Later, Jaffe proposed a nonlinear function $f(p) = \max\{\omega_1(p), c_1\} + \max\{\omega_2(p), c_2\}$ which guarantees to find a feasible path if such a path exists. Korkmaz et al. [19] proposed an algorithm that can dynamically adjust the values of α and β within a logarithmic number of function calls to Dijkstra's shortest-path algorithm. Andrew and Kusuma [25] extended Jaffe's algorithm to the case with more than two constraints. Iwata et al. [24] proposed a polynomial-time algorithm to solve the MCP problem. The algorithm first finds one (or more) shortest path(s) based on one aggregated cost and then checks if all the constraints are met. If it fails, it will repeat with another aggregated cost until an appropriate path satisfying all the constraints is found. Neve et al. in TAMCRA algorithm [6] and Mieghem et al. in SAMCRA algorithm [20] used the k-shortest path algorithm [21] with a nonlinear cost function to solve the MCP problem with more than two constraints. In addition, the concept of non-dominated paths is used to reduce the search space without compromising the solution. The performance of these two algorithms depends on the value of k. If k is large, the algorithm has good performance but with excessive computational cost. In [22], the authors proposed an algorithm, namely MEFPA, for MCP based on some linear energy function. For each node, it first constructs a number (B) of uniform coefficients to construct B linear energy functions, then B least energy trees rooted from this node are calculated. The MEFPA may be useful to implement the admission control at the router, but the algorithm itself does not guarantee the quality of the path it returns. Similar to TAMCRA and SAMCRA, whose performance relies on k, the performance of MEFPA highly depends on the choice of B. The group of algorithms discussed above shares the similar drawbacks: they are very sensitive to the selected aggregated weights, and also lack clear guideline on how the weights should be chosen. Chen et al. [5,23] proposed an approximate algorithm for the MCP. The algorithm first maps the $m-1$ unbounded link metrics (real) into the bounded integers as follows: $\omega_i^*(e) = \lceil \omega_i(e) \cdot x_i / c_i \rceil$ for $i = 2,3,\ldots,m$, where x_i are predefined positive integers. By doing this, the original MCP is reduced to a simpler one: finding a path P for which $\omega_1(p) \leqslant c_1$ and $\omega_i^*(p) \leqslant x_i$ for $i = 2,3,\ldots,m$. Thus, they solved the reduced problem using the extended Bellman–Ford (EBF) algorithm or the extended Dijkstra algorithm (EDSP). The algorithm proposed in [5] is of polynomial time complexity. However, it has to use high granularity in approximating the metrics, and it does not guarantee that the simplified problem has a solution if the original problem does. Yuan [26] presented two heuristics for the MCP problem. The first one, namely, *limited granularity heuristic*, is a generalization of the algorithm in [5]. The second heuristic, called *limited path heuristic*, requires each node to maintain k non-dominated paths (not necessarily the k shortest-paths) such that the time complexity of the EBF can be reduced. Liu and Ramakrishnan [27] proposed a so-called *A*Prune* to find not only one but multiple (K) shortest paths satisfying the constraints.

Meanwhile, the DCLC problem, one of the most famous MCOP problems, has attracted much research attention in the last few years. In 1994, Widyono [7] proposed a Constrained Bellman–Ford (CBF) algorithm that can be used to solve the DCLC problem optimally. The CBF performs a breadth-first search to discover the least-cost path while monotonically increasing delay. CBF maintains a list of least-cost paths for each delay value from the source to each other node.

Once the delay exceeds the constraint, CBF stops. CBF exactly solves the DCLC problem, unfortunately, the worst case running time of CBF grows exponentially with the network size. To overcome the worst-case complexity of CBF, several ε-optimal approximation algorithms were proposed [28–30] based on CBF. Another attempt to solve the DCLC problem is to map the DCLC problem into the possibly easier MCP problem. Guo et al. [8] introduced a cost bound based on the network state and then employed the k-shortest path algorithm [21] with a non-linear function of path delays and path costs to search a path that meets the delay constraint and cost constraint. In [9–11], the authors gave a few algorithms based on the Lagrange relaxation technique. The basic idea is first to construct an aggregated weight with a linear or non-linear function using Lagrange relaxation technique, then to use the Dijkstra algorithm repeatedly to find a feasible path. The problem of this kind of algorithms is how to choose appropriate multipliers for the Lagrange relaxation. Several researchers proposed distributed algorithms in order to alleviate the centralized computational overheads. Reeves and Salama [12] proposed a distributed algorithm called DCUR for the DCLC problem. The DCUR explores the network by choosing the node along the least-delay path or the least-cost path as the next node to be explored. Sun and Langendorfer [13] improved the DCUR such that no loop would be formed during the exploration of the network. Ishida et al. [14] and Sriram et al. [31] proposed two distributed algorithms similar to DCUR. Two interesting distributed algorithms, ticket-based routing [34] and enhanced ticket-based routing [35], use probes (routing messages) carrying colored tickets to explore the possible feasible paths. In ticket-based approaches, yellow tickets prefer paths with smaller delay, while green tickets prefer paths with smaller cost. And by properly choosing the number of green (yellow) tickets, ticket-based routing can find a feasible path with modest message overhead.

From the literature, we observe that there is a tradeoff between the quality of the path that an algorithm is able to find and the expense spent in finding the path—generally a better quality path could be found at the expense of increased complexity. By now there is no algorithm that can maintain a satisfactory tradeoff between these two. In other words, the above heuristics are either too costly in terms of computation or communication, or too complex in terms of the execution time, or inefficient in the sense that they fail to find the optimal solution with reasonable probability. Thus, more efficient QoS routing algorithms are still desired. Our proposed algorithm in this paper represents our contribution along this direction.

3. SF-DCLC routing algorithm

In this section, we present the proposed DCLC algorithm SF-DCLC which is based on a selection function, and we discuss a few properties of the proposed SF-DCLC algorithm.

3.1. Description of DCLC routing problem

Before we present our algorithm, we first describe the DCLC problem to facilitate our discussion. As a usual practice in the literature, a network is modeled as a connected, directed graph $G = (V, E)$, where V is the set of the network nodes and E is the set of edges representing physical or logical connectivities between nodes. Let R^+ denote the set of non-negative real numbers. Two non-negative functions are defined associated with each link e ($e \in E$): the delay function $delay(e)$: $E \rightarrow R^+$ and the cost function $cost(e): E \rightarrow R^+$. Each link may be asymmetric, that is, the costs and the delays of the link $e = (v_i, v_j)$ and the link $e' = (v_j, v_i)$ may have different values. We also define the non-negative delay and cost functions for any path p as

$$delay(p) = \sum_{e \in p} delay(e)$$

and

$$cost(p) = \sum_{e \in p} cost(e).$$

Given a source node $s \in V$, a destination node $d \in V$, and a positive delay constraint Δ, the DCLC routing problem is to find a path p from

s to d such that $\min\{cost(p), p \in P_d\}$ is achieved, where P_d is the set of all feasible paths from s to d that satisfy the delay constraint Δ, i.e., $delay(p) \leqslant \Delta$.

It has been proven that the DCLC problem is NP-complete even for undirected networks [4,18].

3.2. Routing information—the vectors

The traditional distance vector routing algorithms require each router to maintain a table (i.e., a vector), which gives the best known distance to each destination and which outgoing link to use to reach there. While in the DCLC routing algorithm, each node maintains two vectors, the least-delay vector and the least-cost vector, which provide the best known values based on two different metrics, delay and cost, respectively. Each vector is indexed by, and contains one entry for, each node in the network. One entry in the least-delay vector at one node (e.g., node v_i) contains the following information:

- v_j: the destination node identity;
- $delay(P_{ld}(v_i, v_j))$: the delay of the least-delay path $P_{ld}(v_i, v_j)$;
- $cost(P_{ld}(v_i, v_j))$: the cost of the least-delay path $P_{ld}(v_i, v_j)$;
- $nid(P_{ld}(v_i, v_j))$: the next hop on the least-delay path $P_{ld}(v_i, v_j)$;

the least-delay path $P_{ld}(s, d)$ is the path from s to d, which satisfies $delay(P_{ld}(s, d)) = \min\{delay(p), p \in P(s, d)\}$, where $P(s, d)$ is the set of all possible paths from s to d.

Similarly, the entry in the least-cost vector contains the following information:

- v_j: the destination node identity;
- $delay(P_{lc}(v_i, v_j))$: the delay of the least-cost path $P_{lc}(v_i, v_j)$;
- $cost(P_{lc}(v_i, v_j))$: the cost of the least-cost path $P_{lc}(v_i, v_j)$;
- $nid(P_{lc}(v_i, v_j))$: the next hop on the least-cost path $P_{lc}(v_i, v_j)$;

the least-cost path $P_{lc}(s, d)$ is the path from s to d, which satisfies $cost(P_{lc}(s, d)) = \min\{cost(p), p \in P(s, d)\}$, where $P(s, d)$ is the set of all possible paths from s to d.

The least-delay vector and the least-cost vector are similar to the vectors used in the existing distance vector routing protocols. We assume that each node knows the delay and cost to all its neighboring nodes. Then, the same procedure used to update and maintain the vectors in the existing distance vector routing protocols can be used to update and maintain these two vectors. As mentioned in Section 2, in this paper we focus on one of the two functions of QoS routing: the routing algorithms, and leave the problems that how to update the distant vectors in response to topology changes and how to prevent instability to the other function of QoS routing: the routing protocols. We further assume that the contents of the vectors are up-to-date and the contents of the two vectors do not change during the route setup period. Though in this paper we assume that some distance vector routing protocol is used in the network, we should note that, our SF-DCLC can be readily applied to networks using link state routing protocols such as OSPF. In link state routing protocols, every node has the complete topology of the network and is aware of the delay and the cost of each link, thus by using Dijkstra algorithm it is easy for a node to calculate not only its own but also all the other nodes' tables containing the required information for SF-DCLC. Then a centralized implementation of SF-DCLC can be used to search a feasible path.

3.3. Operation of SF-DCLC

The proposed SF-DCLC algorithm constructs the DCLC path node by node from the source node s to the destination node d. Each node chooses its subsequent node by evaluating a selection function *weight*() on all its neighbors. A special PATH_CONSTRUCTION message is sent by the node to the selected subsequent node that requests the continuing construction of the path till the destination. The PATH_CONSTRUCTION message contains the following information $\{d, \Delta, delaySoFar, P_{sf}(s, v)\}$, where d is the destination node identity, Δ is the delay bound, *delaySoFar* is the accumulated delay till the current node, and $P_{sf}(s, v)$ is the set of nodes indicating the partial found DCLC path till the current node v.

The operation of the algorithm is summarized in Fig. 1. Initially, the source node s checks if condition (1) is satisfied. If (1) is not satisfied, there exists no path that meets the given delay constraint from s to d and SF-DCLC stops. Further action could be interactive negotiation with the application for a looser delay constraint, which is out of the scope of this paper. If condition (1) is satisfied, there should exist at least one or more feasible paths that satisfy the delay constraint. Then, the source node s proceeds to check condition (2). If (2) is satisfied, the least-cost path $P_{lc}(s,d)$ is the optimal path. A PATH_CONSTRUCTION message $\{d, \varDelta, delay(s, nid(P_{lc}(s,d)), \{s\}\}$ is sent to the node $nid(P_{lc}(s,d))$ retrieved from its least-cost vector. If condition (2) is not satisfied, it evaluates the functions $weight()$ and $extract()$, based on which the subsequent node is chosen. Assume that the current node is v_i, for each neighboring node v_j, the selection function $weight()$ is defined as follows:

$$weight(v_i, v_j) = \begin{cases} cost(v_i, v_j) + cost'(v_j, d), \\ \quad delaySoFar + delay(v_i, v_j) \\ \quad + delay(P_{ld}(v_j, d)) \leqslant \varDelta, \\ +\infty, \quad \text{otherwise}, \end{cases} \tag{1}$$

where

$$cost'(v_j, d) = \begin{cases} cost(P_{lc}(v_j, d)), \\ \quad delaySoFar + delay(v_i, v_j) \\ \quad + delay(P_{lc}(v_j, d)) \leqslant \varDelta, \\ cost(P_{ld}(v_j, d)), \quad \text{otherwise}, \end{cases}$$

The function $extract()$ is to choose the node, say w, whose value of the selection function $weight(v_i, w)$ is the minimum one among all the neighboring nodes. If more than one nodes have the same minimum value, it chooses the one with the least $delaySoFar + delay(v_i, v_j) + delay(P_{ld}(v_j, d))$. The rationale of the choice of the selection function will be discussed later. Once the subsequent node has been chosen, a new PATH_CONSTRUCTION message is formed and sent to that node.

```
Step 1: (Initially at source node s)
 if (delay(P_ld(s,d)) ≤ Δ)                                  (1)
     delaySoFar = 0;
     P_sf = {s};
     goto Step 2;
 else
     stop with "the delay-constrained path does not exist" ;

Step 2: (Upon receiving a PATH_CONSTRUCTION message or at source
     node s)
 if (this_node != d )
     if (delay(P_lc(this_node,d)) + delaySoFar ≤ Δ)           (2)
        v = nid(P_lc(this_node,d));
        delaySoFar = delaySoFar + delay(this_node, v)));
        P_sf = P_sf + v;
        send PATH_CONSTRUCTION to v;
     else                                                     (3)
        for each neighboring node w and w ∉ P_sf
           calculate weight(this_node,w);
        end
        v = extract(this_node);
        delaySoFar = delaySoFar + delay(this_node,v);
        P_sf = P_sf + v;
        send PATH_CONSTRUCTION to v;
 else
     stop with "path found, P_sf" ;
```

Fig. 1. Pseudo-code for the SF-DCLC algorithm.

The new *delaySoFar* contained in the new message is equal to the old *delaySoFar* plus the delay of link v_i to w. The new $P_{sf}(s,w)$ is the old $P_{sf}(s,v_i)$ concatenating node w.

When a node different from the destination receives a PATH_CONSTRUCTION message, it will repeat the similar procedure as described in step 2 and send a PATH_CONSTRUCTION message to the next hop it selects. When the PATH_CONSTRUCTION message arrives at the destination d, the algorithm terminates and a feasible path $P_{sf}(s,d)$ has been found. Further action such as resource reservation can be performed.

To facilitate the computation of the selection function *weight*(), each node should cache the most recent least-delay vector updates and least-cost vector updates received from its neighbors. We also want to point out that our SF-DCLC is different from DCR proposed in [13] in that DCR chooses the next hop node from either the least-delay path or the least-cost path, while our algorithm takes advantage of the selection function to select the next hop from all neighboring nodes. The selection function approach used in our protocol is similar to those used in [17]. In [17] two selection functions are proposed to construct a Constrained Stein Tree. To find a Constrained Stein Tree is also known to be NP-complete [18]. A cost-delay selection function is used to add edge to the constructed subtree. The definition of the selection function in our protocol is different from the cost-delay selection function used in [17]. Basically, the selection function defined in [17] is as follows:

$$f_{CD}(v_i,v_j) = \begin{cases} \dfrac{cost(v_i,v_j)}{\Delta - delaySoFar - delay(v_i,v_j)}, & \\ \quad delaySoFar + delay(v_i,v_j) & \\ \quad +delay(P_{ld}(v_j,d)) \leqslant \Delta, & \\ +\infty, \quad \text{otherwise}, & \end{cases}$$

The basic idea of this selection function is to select the edge with lower cost and/or longer residual delay. However, as the numerator and denominator are two different measures, this cost-delay selection function is likely to mislead the selection of the next hop to a non-optimal direction when the two measures are of very different magnitudes. For example, it may select paths with delays far lower than Δ but of much higher cost. In our algorithm, we attempt to optimize the path at each intermediate node. Since the optimization objective of DCLC problem is the cost of the path, at each intermediate node v, our algorithm always chooses the next hop w with minimum $cost(v,w) + cost'(w,d)$, and meanwhile, without violating the delay constraint.

3.4. An example

Fig. 2 shows an example of the path constructed by the SF-DCLC algorithm from source $s = A$ to destination $d = B$ with $\Delta = 3.5$. The least delay path from A to B is the path $(A \rightarrow E \rightarrow B)$, the least cost path from A to B is the path $(A \rightarrow B)$. The SF-DCLC path $P_{sf}(A,B)$ found is the path $(A \rightarrow C \rightarrow E \rightarrow B)$ when $\Delta = 3.5$. Obviously, our algorithm leads to a better choice: a path with lower cost while still meeting the delay constraint.

3.5. Correctness of SF-DCLC

When the contents in the vectors at all nodes are up-to-date and do not change during the path construction period, our SF-DCLC can always find a feasible and loop-free path if such a path exists. To prove the correctness of the proposed SF-DCLC algorithm, we have the following theorems.

Theorem 1. *SF-DCLC can always find a feasible path from a source s to a destination d satisfying the given delay bound Δ if such feasible paths exist.*

Proof. If no feasible path exists for the given (s,d) pair and the delay bound, SF-DCLC would terminate immediately with failure notification at the source node after checking the condition (1): $delay(P_{ld}(s,d)) \leqslant \Delta$.Condition (1) is actually an indicator of whether or not feasible paths exist. As we know, the minimum possible delay from one node to a certain destination is the delay of the least-delay path to that destination. If the least-delay path cannot satisfy the delay constraint, then there exists no feasible path. On the contrary, if the least-delay path satisfies the delay

A' s Vectors

Dest	D_LD	C_LD	NH_LD	D_LC	C_LC	NH_LC
E	1	5	E	2	4	C
D	4	2	D	4	2	D
C	1	3	C	1	3	C
B	2	12	E	5	4	B
A	0	0	NULL	0	0	NULL

E' s Vectors

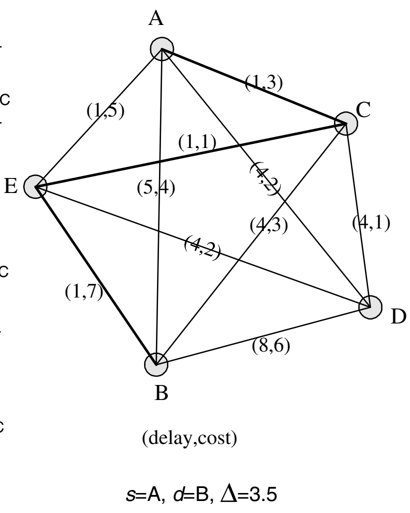
Dest	D_LD	C_LD	NH_LD	D_LC	C_LC	NH_LC
E	0	0	NULL	0	0	NULL
D	4	2	D	4	2	D
C	1	1	C	1	1	C
B	1	7	B	5	4	C
A	1	5	A	2	4	C

C' s Vectors

Dest	D_LD	C_LD	NH_LD	D_LC	C_LC	NH_LC
E	1	1	E	1	1	E
D	4	1	D	4	1	D
C	0	0	NULL	0	0	NULL
B	2	8	E	4	3	B
A	1	3	A	1	3	A

B' s Vectors

Dest	D_LD	C_LD	NH_LD	D_LC	C_LC	NH_LC
E	1	7	E	5	4	C
D	5	9	E	8	4	C
C	2	8	C	4	3	C
B	0	0	NULL	0	0	NULL
A	2	12	E	5	4	A



At Node A
weight (A,C) = 3+8 = 11
weight (A,E) = 5+7 = 12
weight (A,B) = +∞
weight (A,D) = +∞
extract (A) = C

At Node C
weight (C,E) = 1+7 = 8
weight (C,B) = +∞
weight (C,D) = +∞
extract (C) = E

At Node E
weight (E,D) = +∞
weight (E,B) = 7+0 = 7
extract (E) = B

At Node B
SF-DCLC terminates

Fig. 2. An example of the construction of the path from node A to node B with delay bound of 3.5 using SF-DCLC. Figures along links are (*delay*, *cost*) for both directions.

constraint, there exists at least one feasible path, which is the least-delay path. This condition is explicitly checked once at source node. If it is satisfied, feasible paths exist, and SF-DCLC starts. Then at each following node v_i, this condition is implicitly guaranteed by the definition of the *weight*() function or inherited from the preceding node, however, in a slightly different format, i.e., $delaySoFar + delay(P_{ld}(v_i,d)) \leqslant \Delta$, where the left side of the inequality is the summation of the delay accumulated from s to v_i and the minimum possible delay from node v_i to d. We now complete the proof by the induction on node v_i.At first, s is the only member in the partial path P_{sf}, $delaySoFar = 0$, and $delaySoFar + delay(P_{ld}(s,d)) \leqslant \Delta$. The source node is the basis for our induction. We assume that at an intermediate node v_i, $delaySoFar$ is already updated and $delaySoFar + delay(P_{ld}(v_i,d)) \leqslant \Delta$ still holds.Then SF-DCLC will check the condition (2): $delaySoFar + delay(P_{lc}(v_i,d)) \leqslant \Delta$. Condition (2) indicates whether or not the least-cost path $P_{lc}(v_i,d)$ satisfies the delay constraint $\Delta - delaySoFar$. If (2) is satisfied, then the partial path so far $P_{sf}(s,v_i)$ concatenating the least-cost path $P_{lc}(v_i,d)$ from v_i to d would be the desired path, and SF-DCLC will choose the nodes along the least-cost path $P_{lc}(v_i,d)$ one by one till a PATH_CONSTRUCTION message reaches the destination. In this case, it is obvious that once the condition (2) is satisfied at node v_i, it will be always satisfied at each node along the least-cost path from v_i to d. If condition (2) is not satisfied, then the selection function of each neighboring node is calculated. According to the definition of the selection function, only those nodes (v_j) satisfying the condition (4): $delaySoFar + delay(v_i,v_j) + delay(P_{ld}(v_j,d)) \leqslant \Delta$ and are not already in the partial path $P_{sf}(s,v_i)$, are considered in the selection function. This guarantees that when the selected next hop node, say w, receives the PATH_CONSTRUCTION message with updated $delaySoFar$, the condition $delaySoFar + delay(P_{ld}(w,d)) \leqslant \Delta$ still holds. On the other hand, among the neighboring nodes, there is always at least one node satisfying condition (4), which is the node along the least-delay path. One may wonder what if this node along the least-delay path has already been on the partial path P_{sf}, and since our algorithm excludes the nodes already in the partial path P_{sf} when evaluating the selection function *weight*(), our SF-DCLC would be stuck in this case. To facilitate our proof, here we consider a variant of SF-DCLC that when one node, say v_i, finds its only possible choice $w = nid(P_{ld}(v_i, d))$, the next node of the least-delay path from v_i to d, is already on the partial path $P_{sf}(s, v_i)$, node v_i still sends the PATH_CONSTRUCTION message to w to avoid the possible stuck situation. With this minor modification, though loops may be formed, the search for the feasible path will continue, and eventually SF-DCLC will terminate at destination d, where the condition $delaySoFar \leqslant \Delta$ is still valid. Thus SF-DCLC finds a feasible path satisfying the delay bound Δ. In all, SF-DCLC can always find a path from the source s to the destination d satisfying the given delay bound Δ if such a path exists. □

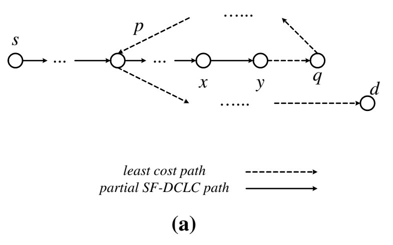
In the following proof of Theorem 2 we will prove that the aforementioned stuck situation will never happen and no loop will be formed. In other words, our algorithm is valid and does not need the minor modification we introduced for the proof of Theorem 1.

Theorem 2. *The path found by SF-DCLC contains no loop.*

Proof. We first proof this theorem based on the modified SF-DCLC as discussed above. According to the operation of SF-DCLC protocol (as shown in Fig. 1), we notice that if condition (2) is satisfied at any node v, then from that node onwards, condition (2) will always be satisfied. SF-DCLC will actually follow the least-cost path $P_{lc}(v, d)$. Since we assume the vectors at each node are consistent, there should exist no loop along the least-cost path. If condition (2) is not satisfied at any node, SF-DCLC will calculate the selection function *weight*() and select the next node based on the *weight*() function.

One possibility where loops might be formed is that, from the source node s to some node x SF-DCLC falls into condition (3), and from y (assume the next node after x on the P_{sf} is y) to d, SF-DCLC falls into condition (2), and chooses the nodes along $P_{lc}(y, d)$. The loop occurs when the least-cost path $P_{lc}(y, d)$ shares some node, e.g., node p, with the $P_{sf}(s, y)$. In addition, we have $P_{sf}(s, d) = P_{sf}(s, y) + P_{lc}(y, d)$. Fig. 3(a) shows such a scenario. Since node p is first added into $P_{sf}(s, y)$ under condition (3), it implies that condition (2) is not satisfied at node p when it receives the PATH_CONSTRUCTION message for the first time. While node p is also on the path $P_{lc}(y, d)$, which indicates condition (2) is satisfied when node p receives the PATH_CONSTRUCTION message for the second time. These contradict each other because the accumulated *delaySoFar* would be greater when SF-DCLC visits node p for the second time. Thus, this scenario does not exist. □

Recall that to avoid the "possible" stuck situation, our aforementioned minor modification allows SF-DCLC to send a PATH_CONSTRUCTION message to the only possible next node on the least-delay path, even when this next node is already on the partial P_{sf}, thus loops might be formed due to this modification. As shown in Fig. 3, we assume that at node C SF-DCLC runs across the stuck situation that the only choice of the next hop at node C is along the least-delay path $P_{ld}(C, d)$ and node $A = nid(P_{ld}(C, d))$ is already on the partial path $P_{sf}(s, C)$. Suppose the



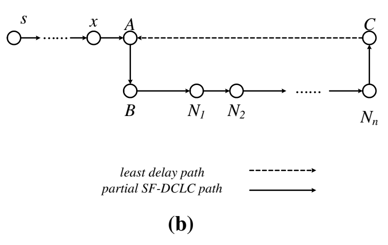


Fig. 3. The possible loop scenarios: (a) Scenario I and (b) Scenario II.

partial path is as follows: $P_{sf}(s,C) = \{s,\ldots,A, B,N_1,N_2,\ldots,N_n,C\}$. In this case a loop would have been formed by sending node A another PATH_CONSTRUCTION message. In what follows we will prove that this case is not possible and thus the loop would never happen.

With the proof of non-existence of the first loop scenario, we can assure that none of the nodes, $B,N_1,N_2,\ldots,N_n,C$, is on its preceding node's least-cost path. Therefore, node $B \neq nid(P_{lc}(A,d))$, node $N_1 \neq nid(P_{lc}(B,d))$, etc. According to our definitions of the *weight*() and *extract*() functions, node B, not necessary the next node $nid(P_{ld}(A,d)$ on the least-delay path from A to d, is selected because its value of *weight*() function is the minimum among node $A's$ neighbors. Thus we have

$$cost(P_{ld}(A,d)) \geqslant cost(A,B) + cost(P_{ld}(B,d)). \quad (2)$$

Similarly, we have the following inequations:

$$cost(P_{ld}(B,d)) \geqslant cost(B,N_1) + cost(P_{ld}(N_1,d)), \quad (3)$$

$$cost(P_{ld}(N_1,d)) \geqslant cost(N_1,N_2) + cost(P_{ld}(N_2,d)), \quad (4)$$

$$cost(P_{ld}(N_n,d)) \geqslant cost(N_n,C) + cost(P_{ld}(C,d)). \quad (n+2)$$

In addition, since $A = nid(P_{ld}(C,d))$, we have the following equation:

$$cost(P_{ld}(C,d)) = cost(C,A) + cost(P_{ld}(A,d)). \quad (n+3)$$

Furthermore, the nodes $(A,B,N_1,N_2,\ldots,N_n,C)$ cannot be the nodes along the least-delay path $P_{ld}(x,d)$ all at the same time, since no loop should be formed with consistent routing information. Thus, the equal conditions in the above inequations cannot be satisfied all at the same time. Then after summing up the above $n+3$ (in)equations, we have

$$\begin{aligned} cost(P_{ld}(A,d)) > {} & cost(P_{ld}(A,d)) + cost(A,B) \\ & + cost(A,B) + cost(B,N_1) \\ & + cost(N_1,N_2) + \cdots + cost(N_n,C) \\ & + cost(C,A). \end{aligned} \quad (n+4)$$

Obviously, the inequation $(n+4)$ does not hold. Thus, we demonstrate that the stuck situation does not exist and the minor modification we introduced to facilitate the proof of Theorem 1 is redundant. SF-DCLC operations described in Fig. 1 are completely correct and need no further enhancement.

With the above loop-freedom proof, now we are safe to claim that the path found by SF-DCLC contains no loop.

Theorem 3. *SF-DCLC always terminates in finite time.*

If no feasible path exists for the given source–destination pair and the delay bound Δ, SF-DCLC should terminate immediately with failure notification at the source node after checking condition (1). If condition (1) is satisfied, SF-DCLC will proceed. Since no loop will be formed, after running at $|V| - 1$ (a finite number) nodes in the worst case, SF-DCLC will terminate at the destination d, thus a feasible path $P_{sf}(s,d)$ will be found in finite time.

3.6. Complexity of SF-DCLC

The proposed SF-DCLC maintains two vectors, the least-cost vector and the least-delay vector. For a network $G = (V,E)$, using the similar procedures as distance vector routing protocols, the worst case computation complexity for each node to compute the two vectors is $O(|V|^3)$ [15]. Since we assume that our SF-DCLC is based on the existing distance vector routing protocols, the overhead for routing vector maintenance has already been consider in the distance vector routing, and thus in the following we only consider the extra overhead introduced by SF-DCLC.

As discussed before, in SF-DCLC each path is constructed in an "on demand" manner. For each path finding, a node should evaluate the link selection function *weight*() at most $|V|$ times, and should compare at most $|V|$ values to find out the minimum *weight*(), thus, in the worst case the extra computational complexity for a node to select the next hop is $O(|V|)$. Since the worst case path length would be $|V|$, the computational complexity for finding a SF-DCLC path in the worst case is $O(|V|^2)$.

The messages for updating and maintaining the vectors are exchanged only between the neighboring nodes. In a stable network, the messages transmitted for the path finding procedure is one PATH_CONSTRUCTION message per node (except the destination node). In the worst case, the longest path from the source to the destination contains $|V|$ nodes, then the worst case message complexity for a path finding by SF-DCLC is $O(|V|)$.

Each node caches the most up-to-date least-delay vector and least-cost vector received from its neighbors. Since a node at most has $|V|$ neighbors and a vector from one neighbor contains $|V|$ entries, the worst case memory complexity at each node is $O(|V|^2)$.

4. Performance evaluation

We carried out the performance evaluation for this new routing algorithm. Comparison study was made by using simulations.

4.1. Simulation environment

We developed a unicast routing simulator to carry out the simulations. To generate the network topology, we adapted BRITE [36] into the simulator. Two kinds of network topologies were generated in our simulations. The first one is based on the Waxman model [16]. In the Waxman model, two nodes u and v are connected with probability $p(u,v) = \alpha \cdot e^{-d(u,v)/(\beta L)}$, where $0 < \alpha, \beta \leqslant 1$, $d(u,v)$ is the Euclidean distance between u and v, and L is the maximum distance between any two nodes. In our implementation, we set $\alpha = 0.15$, and $\beta = 0.2$. The second kind of graph is based on the Barabasi–Albert model proposed in [37]. In the Barabasi–Albert model, the networks are formed by the continual addition of new nodes and a newly joining node has high probability to connect to existing nodes that are highly connected or popular. For example, a node v joins the network and the probability that it connects to a node u already belonging to the network is given by: $p(u,v) = d_u/\sum_{k\in V} d_k$, where d_u is the outdegree of node u, V is the set of nodes that have already joined the network and $\sum_{k\in V} d_k$ is the sum of outdegrees of all nodes that previously joined the network. It is shown that, the topologies generated with the Barabasi–Albert model bear a power law distribution in terms of outdegrees [38]. In fact, recent studies reveal that in the Internet, the nodal's outdegree also has such power-law distribution. In addition, the topologies generated with the Barabasi–Albert model have some features of "small world", e.g., higher clustering coefficient than those random topologies based on the Waxman model [39]. In our simulation, the random graph generator can always generate connected graph with average node degree of 4, which is close to the average node degree of the current Internet. The cost value of a link varies from 1 to 8 with a uniform distribution. As for the delay in our simulation, we only considered the propagation delay, which depends on the distance between the communication nodes. In order to capture the delay characteristics of the national wide network, the delay values of the links were selected from three ranges [9]. Seventy-five percent of the delay values were selected from the first range (1–5 ms), which represents the short local links. Twenty percent were selected from (5–8 ms), which represents the long local links. The remaining 5% were from (20–30 ms), representing the continental links. In our simulation, we assumed that the links are undirected. The simulation was repeated with network size ranging from 20 nodes up to 200 nodes. The delay constraint Δ was randomly selected from a range corresponding to the *delay level* ranging from 1 to 5, where the *delay level* is a new comparison index we introduce in this paper and will be described in the next subsection. For a specific network size, five network instances were used, and for each network instance, 100 routing requests were generated.

For comparison purpose, we also implemented three other algorithms, LDP, CBF, and DCR. The LDP algorithm is used to find the least-delay path. As we mentioned in Section 1, when there exists a feasible path, the CBF algorithm can always find the optimal DCLC path from source s to destination d. The DCR algorithm proposed in [13] is very similar to the DCUR algorithm proposed in [12], while DCR improves the worst-case performance of DCUR by avoiding, instead of detecting and removing, loops. DCR is a well-performed DCLC algorithm, so we also compared the performance of our SF-DCLC algorithm with DCR.

4.2. Performance metrics

In the literature of the performance comparison of DCLC algorithms, arbitrary Δ value is usually used for any two nodes, regardless the source, destination and the actual delay between them. However,according to the operation of the DCLC operation, when $\Delta < delay(P_{ld}(s,d))$, there is no feasible path. All the algorithms would not be able to find a path satisfying that bound. When $\Delta \geqslant delay(P_{lc}(s,d))$, CBF, DCR, and SF-DCLC should all find the same path $P_{lc}(s,d)$. Only when $delay(P_{ld}(s,d)) \leqslant \Delta < delay(P_{lc}(s,d))$, there exists one or more feasible paths and it depends on the routing algorithm to find out the optimal feasible path. Thus, using arbitrary Δ may not a good way for comparison, because for some network instance, this bound might be too loose or too stringent that fails to reveal the sophistication of the algorithms. We introduced a new comparison index metric in our simulation, *delay level*, which is related to the actual delay between each source and destination. The range between $[delay(P_{ld}(s,d)), delay(P_{lc}(s,d))]$ was divided into five equal length periods, and each period corresponds to a *delay level* (1–5). Thus the smaller the delay level is, the more stringent the bound is. In our simulation, Δ was randomly selected from the period corresponding to the five delay levels. The simulation results reported in the paper do not count the cases where $delay(P_{ld}(s,d)) = delay(P_{lc}(s,d))$.

Since the CBF algorithm can always find the optimal DCLC path from source s to destination d, the cost of the path found by CBF, $cost(P_{CBF})$, can be viewed as the lower bound of the cost of feasible DCLC paths. On the other hand, the least-delay path $P_{ld}(s,d)$ is always a feasible path if the delay bound Δ is appropriately chosen. The cost of $P_{ld}(s,d)$ can be viewed as a sort of upper-bound (although theoretically there should not be a upper-bound) of the cost of feasible DCLC paths if such a path exists.

We define the following two performance metrics to compare the proposed algorithm with other algorithms:

- *Cost inefficiency* (CI)

  $$\delta_A = \frac{cost(P_A) - cost(P_{CBF})}{cost(P_{CBF})}$$

  where A represents the algorithm by which the path is found. In our simulation, A could be LDP, DCR or SF-DCLC. This metric is used to evaluate the quality of the paths found by these algorithms.
- *Optimality Miss Ratio* (OMR). The probability that the path found is not the optimal one, i.e., the cost is different from that of the path found by CBF. This measurement is used to evaluate an algorithm's capability to find the optimal path.

In addition to these two metrics, we also study the *Average Number of Messages (ANM)* for each connection request to represent the average message complexity of SF-DCLC. Here, we only consider the messages engaged in the path-finding process, not those for updating the vectors.

4.3. Simulation results

In this section, we present the performance of our SF-DCLC algorithm and compare it with other DCLC algorithms. The evaluations are done by simulations. As we mentioned before, the Waxman model is the classical network topology generation model while the Barabasi–Albert model was recently proposed and has been identified as more closely resembling the practical Internet topology. To demonstrate the adaptivity and applicability of our algorithm, we carried out simulations on the topologies randomly generated by both of the models.

4.3.1. Efficiency

We compare the quality of the path found by different algorithms in this subsection. First, we examine the performance metrics with various delay constraint levels. Fig. 4 compares the performance of the *Cost Inefficiency* (CI) versus delay level while Fig. 5 presents the results of *Optimality Miss Ratio* (OMR) versus delay level. We only present the cases where the size of the network are 40 and 100. In both figures (and also in subsequent three figures), subfigure (a) shows the results from the Waxman model while subgraph (b) is from the Barabasi–Albert model. We should pointed out that all the three routing algorithms under study are capable of finding a feasible path if such a path exists. However, the path found by different algorithms might be different and of different costs. It is observed that, when delay bound is very stringent (i.e., small DL), both the CI and the OMR of the three compared algorithms are very close and all of them are small. However, the CI and the OMR of LDP grow faster than those of DCR and SF-DCLC with the increase of the delay level (i.e., delay requirement becomes less stringent). This results can be explained as follows. When the delay constraint is stringent, the number of feasible paths is very limited. All the algorithms are likely to choose the least-delay path P_{LDP}, which results in low inefficiency and close performance. For the same reason, the OMR of these algorithms is very close as well. However, when the delay constraint becomes loose, the number of feasible paths becomes larger and the algorithms have more room to choose from, therefore their

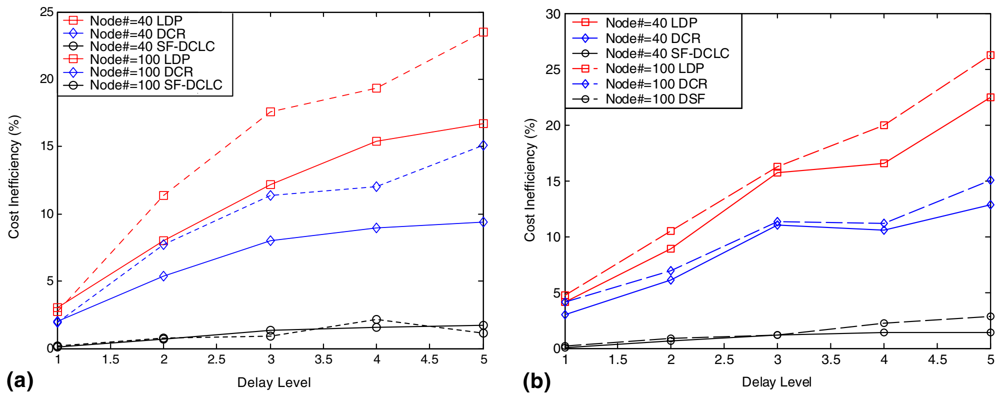


Fig. 4. CI vs. delay level (size of network = 40, 100): (a) Waxman model and (b) Barabasi–Albert model.

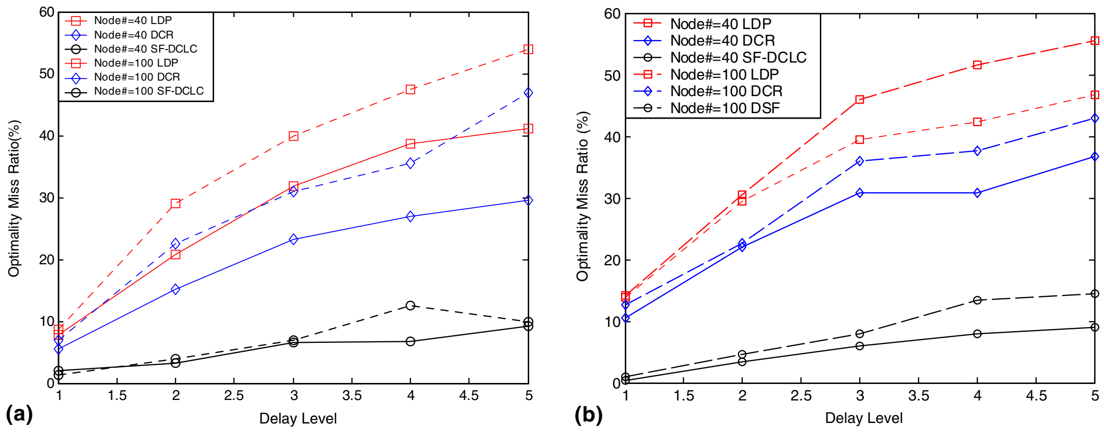


Fig. 5. OMR vs. delay level (size of network = 40, 100): (a) Waxman model and (b) Barabasi–Albert model.

performance starts to diverge. It is observed that the paths found by the proposed SF-DCLC algorithm remains very close to the optimal one at all levels of delay constraints, e.g., less than 3% *CI* and less than 12% OMR, compared with 15% CI and 47% OMR for DCR algorithm, and 23% CI and 54% OMR for LDP algorithm. The comparison clearly indicates that the proposed SF-DCLC algorithm is more likely to find the optimal path and the path found is more cost efficient.

By examining the paths returned by each of the algorithms, we noticed that in the cases where P_{CBF} is found to be the same as P_{LDP}, P_{DCR} and $P_{SF-DCLC}$ would also be the same as P_{LDP}. This implies that when LDP path is the optimal DCLC path, both DCR and SF-DCLC are able to select it, the optimal one. We also observe that SF-DCLC has better capability to find the optimal path than DCR. Actually, in our simulation (when the Waxman model is used, the size of network is 100 and the delay level is 4), 54% of optimal DCLC paths are not LDP paths. In these cases, SF-DCLC has the capability to find the optimal paths with probability of 78% (1−12%/54%), while DCR is only able to find the optimal paths with 12% (1−47%/54%) of them. In fact, in DCR the source node *s* or a node receiving a PATH_CONSTRUCTION message only checks two neighbors along its least-delay path and least-cost path, so the path returned by DCR may be either the least-delay path $P_{ld}(s,d)$ or a path consisting of $P_{ld}(s,x)$ concatenating $P_{lc}(x,d)$, where x is an intermediate node where the *path_direction* changes from *LD* to *LC* [13]. In our SF-DCLC, source node *s* or a node receiving a PATH_CONSTRUCTION message checks not only its two neighbors along the least-delay path and the least-cost path, but also the other neighbors, and subsequently chooses one node with the minimum *weight*() as the next hop. In this sense, DCR can be viewed as a special case of our SF-DCLC when SF-DCLC only considers outgoing links along the least-delay path and the least-cost path. Obviously, the definition and the design of the *weight*() and the *extract*() functions make it possible to locally optimize the overall cost in a hop-by-bop manner. Since the checking of a neighbor only involves some trivial computation (i.e., calculate *weight*() function), this performance gain is achieved at almost no additional cost.

Next, we examine the performance of the routing algorithms on networks with various sizes. Fig. 6 shows the CI versus size of the network for the cases that the delay levels (DL) are 2 and 4; Fig. 7 shows the OMR versus the size of the network for the same cases.

It is observed that the performance of all the three algorithms is not sensitive to the network size. The CI and the OMR of the proposed SF-DCLC algorithm are relative steady in all sized networks, while those of LDP and DCR increase slightly with the increase of the network size. Another observation is that the CI and the OMR of SF-DCLC are very low and always lower than

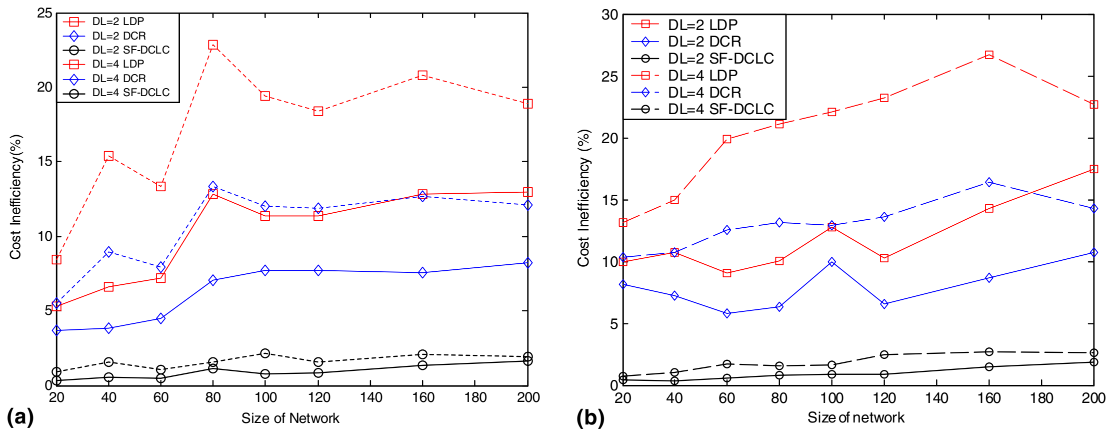


Fig. 6. CI vs. size of network (delay level (DL) = 2, 4): (a) Waxman model and (b) Barabasi–Albert model.

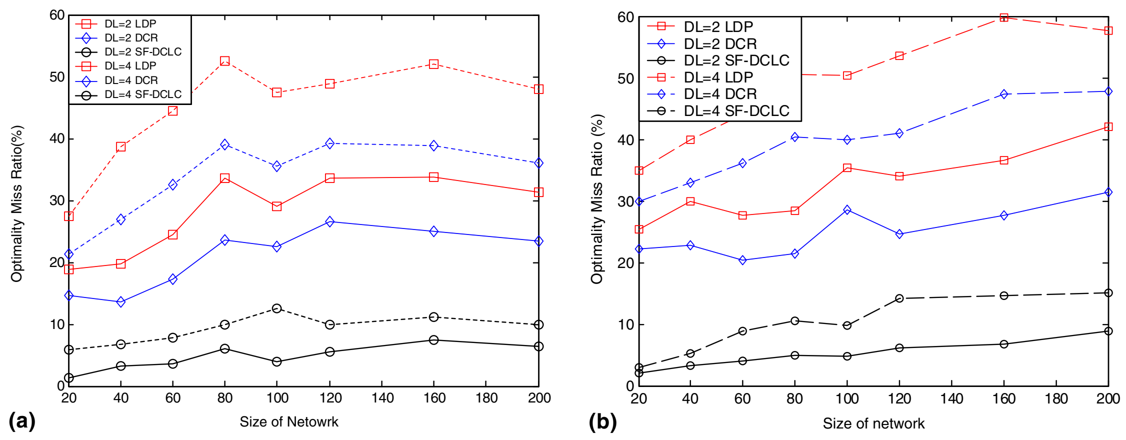


Fig. 7. OMR vs. size of network (delay level (DL) = 2, 4): (a) Waxman model and (b) Barabasi–Albert model.

those of LDP and DCR, indicating that the cost of a path found by SF-DCLC is very close to the optimal one and better than paths found by the other two algorithms. Thus, the SF-DCLC algorithm has better ability to find better path than the other two. If we look back to Fig. 2, we can see an example that our SF-DCLC can find a better path than the other two. In fact, the path found by SF-DCLC in that example is $A \rightarrow C \rightarrow E \rightarrow B$ with cost 11, while the path found by DCR is $A \rightarrow E \rightarrow B$ with cost 12.

4.3.2. Overhead

We examine the protocol overhead in this subsection. For communication overhead, we measure the number of messages exchanged between nodes when SF-DCLC is used. The messages counted here are the PATH_CONSTRUCTION messages for SF-DCLC. Messages for updating and maintaining routing vectors are not counted. Fig. 8 gives the average number of messages per path found versus the size of the network. It is observed that no direct relation exists between the ANM and the delay level. Since only m messages are required for finding a m-hop path, it is not surprising to see that the average growth of the number of messages is approximately logarithmic to the network size which implies that the proposed SF-DCLC scales well. We also observe that the ANM in the networks based on the

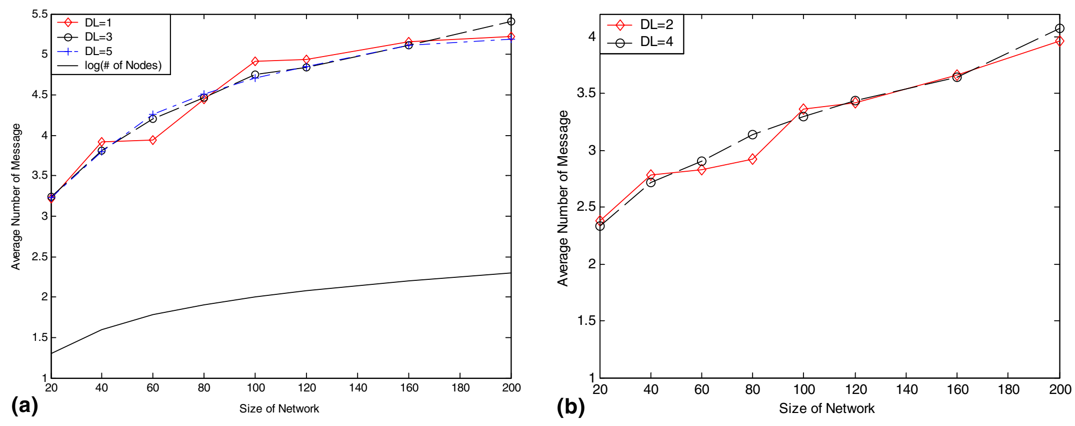


Fig. 8. ANM vs. size of network: (a) Waxman model and (b) Barabasi–Albert model.

Table 1
Average execution time for each connection request (1 s = 1000 ms)

Size of network	20	40	60	80	100	120	160	200
CBF (s/req)	0.012	0.226	1.787	293.46	292.783	29.647	119.183	62.781
DCR (ms/req)	0.062	0.032	0.032	0.032	0.05	0.03	0.21	0.522
SF-DCLC (ms/req)	0.096	0.092	0.126	0.504	0.57	0.79	0.78	0.858

Barabasi–Albert model is smaller than that for the networks based on the Waxman model. We can explain this observation as follows. On one hand, ANM of SF-DCLC is closely related to the path length of $P_{SF-DCLC}$; on the other hand, networks based on the Barabasi–Albert model has larger clustering coefficient than those based on the Waxman model, and this fact leads to shorter average path length. Thus, it is no surprise to observe that our SF-DCLC has smaller ANM in networks based on the Barabasi–Albert model.

To demonstrate the computational overheads that the different routing algorithms incur, we present in Table 1 the average simulation time for finding one path by different algorithms when running in a same computer. As expected, we observe that CBF algorithm, although theoretically providing the optimal solution to the DCLC problem, requires excessive computation time (a magnitude of 2 or 3), which makes it inapplicable in the practical use. Compared to CBF, DCR and SF-DCLC incur much shorter execution time. As we mentioned before, SF-DCLC incurs longer execution time than DCR because our SF-DCLC considers all the neighbors rather than the two neighbors along directions of LCP or LDP [13]. However, we argue that with the fast advancement in microprocessor technologies, the computation overhead incurred by SF-DCLC is trivial.

In summary, via simulation, we validated that SF-DCLC has high probability to find the optimal solution while keeping the overhead low, striking a very good balance between the path quality and complexity.

5. Conclusions and future work

In this paper, we studied the DCLC problem, which is crucial for the emerging delay-sensitive applications. We proposed a distributed unicast routing algorithm, namely, SF-DCLC, based on a special heuristic selection function. This algorithm is always able to find a loop free path if such a path exists. The worst case computational complexity and the worst case memory complexity of SF-DCLC are $O(|V|^2)$. In addition, the worst case message complexity for the path finding is $O(|V|)$, which does not grow exponentially with the size of the network, thus our proposed SF-DCLC scales well with the increase of the size of the network. We proposed a new comparison index, *delay level*, other than the arbitrary delay bound Δ value that is commonly used in other papers on DCLC problem. We also evaluated our algorithm by comparing it with DCR, LDP and CBF in terms of path cost and optimality. Our simulation results from both networks based on the Waxman model and the Barabasi–Albert model show that SF-DCLC has much better performance than DCR and LDP. SF-DCLC is insensitive to network sizes and delay levels. More specifically, for the networks based on the Waxman model, the cost inefficiency of SF-DCLC compared to CBF, the optimal one, is less than 3% with different delay levels and different sizes of networks. Our SF-DCLC also performs pretty well in the networks based on the Barabasi–Albert model. Furthermore, the optimality miss ratio of SF-FCLC is far less than those of DCR and LDP. Thus, the most attractive feature of the SF-DCLC algorithm is its high efficiency in the sense that *it has very high probability of finding the optimal path with very low complexity*.

A possible improvement to SF-DCLC is to modify the selection function to take the delay into consideration. Since our algorithm can always find a delay-constrained path with promising cost, our future work is to extend the algorithm to support multi-path or multicast routing for the delay-sensitive multimedia applications.

Acknowledgments

This work was supported in part by the Office of Naval Research Young Investigator Award under grant N000140210464 and the Office of Naval Research under grant N000140210554. The authors would like to thank all the anonymous reviewers for their valuable suggestion to improve the quality of this paper.

References

[1] X. Xiao, L.M. Ni, Internet QoS: a big picture, IEEE Network 13 (2) (1999) 8–18.

[2] S. Chen, K. Nahrstedt, An overview of quality of service routing for next-generation high-speed networks: problems and solutions, IEEE Networks 12 (6) (1998) 64–79.

[3] J.M. Jaffe, Algorithms for finding paths with multiple constraints, Networks 14 (1984) 95–116.

[4] Z. Wang, J. Crowcroft, Quality-of-service routing for supporting multimedia applications, IEEE Journal on Selected Areas in Communications 14 (7) (1996) 1228–1234.

[5] S. Chen, K. Nahrstedt, On finding multi-constrained paths, in: Proceedings of ICC'98, Atlanta, GA, 1998, pp. 874–879.

[6] H. De Neve, P. Van Mieghem, A multiple quality of service routing algorithm for PNNI, in: Proceedings of the ATM Workshop, May 1998, pp. 324–328.

[7] R. Widyono, The design and evaluation of routing algorithms for real-time channels, Technical Report ICSI TR-94-024, International Computer Science Institute, UC, Berkeley, June 1994.

[8] L. Guo, I. Matta, Search space reduction in QoS routing, Computer Networks 41 (1) (2003) 73–88.

[9] A. Juttner, B. Szviatovszki, I. Mecs, Z. Rajko, Lagrange relaxation based method for the QoS routing problem, in: Proceedings of the IEEE INFOCOM'2001, AK, 2001.

[10] T. Korkmaz, and M. Krunz, Multi-constrained optimal path selection, in: Proceedings of IEEE INFOCOM'2001, AK, 2001.

[11] G. Feng, C. Doulgeris, K. Makki, N. Pissinou, Performance evaluation of delay-constrained least-cost routing algorithms based on linear and nonlinear Lagrange relaxation, in: Proceedings of ICC'02, New York, April 2002, pp. 2273–2278.

[12] D.S. Reeves, H.F. Salama, A distributed algorithm for delay-constrained unicast routing, IEEE/ACM Transactions on Networking 8 (2) (2000) 230–250.

[13] Q. Sun, H. Langendorfer, A new distributed routing algorithm for delay-sensitive application, Computer Communications 21 (6) (1998) 572–578.

[14] K. Ishida, E. Amana, N. Kannari, A delay-constrained least cost path routing protocol and the synthesis method, in: Proceedings of the Fifth IEEE International Conference on Real-Time Computer System and Applications, October 1998, pp. 58–65.

[15] D. Bertsekas, R. Gallager, Data Networks, second ed., Prentice-Hall, Englewood Cliffs, NJ, 1992, p. 399.

[16] B.M. Waxman, Routing of multipoint connections, IEEE Journal on Selected Areas in Communications 6 (9) (1988) 1617–1622.

[17] V.P. Kompella, J.C. Pasquale, G.C. Polyzos, Multicast routing for multimedia communication, IEEE/ACM Transactions on Networking 1 (3) (1993) 286–292.

[18] M.R. Garey, D.S. Johnson, Computer and Intractability: A Guide to the Theory of NP-completeness, Freeman, San Francisco, 1979.

[19] T. Korkmaz, M. Krunz, S. Tragoudas, An efficient algorithm for finding a path subject to two additive constraints, in: Proceedings of the ACM SIGMETRICS'00, vol. 1, 2000, pp. 318–327.

[20] P. Van Mieghem, H. De Neve, F.A. Kuipers, Hop-by-Hop quality of service routing, Computer Networks 37 (2001) 407–423.

[21] E.I. Chong, S. Maddila, S. Morley, On finding single-source single-destination *k* shortest paths, in: Proceedings of International Conference on Computing and Information (ICCI)'95, July 1995, pp. 40–47.

[22] Y. Cui, K. Xu, J. Wu, Performance for multi-constrained QoS routing in high-speed networks, in: Proceedings of IEEE INFOCOM'03.

[23] S. Chen, K. Nahrstedt, On finding multi-constrained path, Technical Report UIUCDCS-R-97-2026, Department of Computer Science, UIUC, August 1997.

[24] A. Iwata, R. Izmailov, D.-S. Lee, B. Sengupta, G. Ramamurthy, H. Suzuki, ATM routing algorithm with multiple QoS requirements for multimedia internetworking, IEICE Transactions and Communications E70-B (8) (1998) 999–1006.

[25] L.H. Andrew, A.N. Kusuma, Generalized analysis of a QoS-aware routing algorithm, in: Proceedings of IEEE GLOBECOM 1998, vol. 1, 1998, pp. 1–6.

[26] X. Yuan, Heuristic algorithm for multi-constrained quality-of-service routing, IEEE/ACM Transactions on Networking 10 (2) (2002) 244–256.

[27] G. Liu, K.G. Ramakrishnan, *A*Prune*: an algorithm for finding *K* shortest paths subject to multiple constraints, in: Proceedings of IEEE INFOCOM 2001.

[28] R. Hassin, Approximation schemes for the restricted shortest path problem, Mathematics of Operational Research 17 (1) (1992) 36–42.

[29] D.H. Lorenz, A. Orda, QoS routing in networks with uncertain parameters, IEEE/ACM Transactions on Networking 6 (6) (1998) 768–778.

[30] D. Raz, Y. Shavitt, Optimal partition of QoS requirements with discrete cost functions, in: Proceedings of the of IEEE INFOCOM 2000.

[31] R. Sriram, G. Manimaran, C.S. Murthy, Preferred link based delay-constrained least-cost routing in wide area networks, Computer Communications 21 (1998) 1655–1669.

[32] F.A. Kuipers, T. Korkmaz, M. Krunz, P. Van Mieghem, An overview of constraint-based path selection Algorithms for QoS routing, IEEE Communications Magazine 40 (12) (2002) 50–55.

[33] O. Younis, S. Fahmy, Constraint-based routing in the Internet: basic principle and recent research, IEEE Communications Surveys and Tutorials 5 (1) (2003) 2–13.

[34] S. Chen, K. Nahrstedt, Distributed quality-of-service routing in ad-hoc networks, IEEE Journal on Selected Areas in Communications 17 (8) (1999) 1488–1505.

[35] L. Xiao, J. Wang, K. Nahrstedt, The enhanced ticket-based routing algorithm, in: Proceedings of the of ICC'02, New York, April 2002.

[36] Available from: <http://www.cs.bu.edu/brite/>.

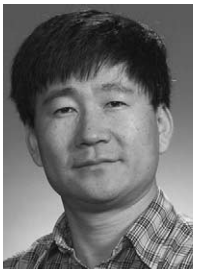
[37] A.L. Barabasi, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509–512, October.

[38] A. Medina, A. Lakhina, I. Matta, J. Byers, Brite: an approach to universal topology generation, MASCOTS, 2001.

[39] D.J. Watts, S.H. Strogatz, Collective dynamics of small-world networks, Nature 393 (4) (1998) 440–442.

Wei Liu received the B.E. and M.E. degrees in Electrical Engineering from Huazhong University of Science and Technology, Wuhan, China, in 1998 and 2001, respectively. He is currently pursuing the PhD degree in the Department of Electrical and Computer Engineering, University of Florida, Gainesville, where he is a research assistant in the Wireless Networks Laboratory (WINET). His research interest includes QoS, Secure, and Power Efficient Routing, and MAC protocols in Mobile Ad Hoc Networks and Sensor Networks.

Wenjing Lou is currently an assistant professor in the Electrical and Computer Engineering department at Worcester Polytechnic Institute. She obtained her Ph.D degree in Electrical and Computer Engineering from University of Florida in 2003. She received the M.A.Sc. degree from Nanyang Technological University, Singapore, in 1998, the M.E degree and the B.E degree in Computer Science and Engineering from Xi'an Jiaotong University, China, in 1996 and 1993 respectively. From December 1997 to July 1999, she worked as a Research Engineer in Network Technology Research Center, Nanyang Technological University. Her research interests are in the areas of ad hoc and sensor networks, and network security.



Yuguang Fang received a Ph.D degree in Systems and Control Engineering from Case Western Reserve University, Cleveland, Ohio, in January 1994, and a Ph.D degree in Electrical Engineering from Boston University, Massachusetts, in May 1997. From September 1989 to December 1993, he was a teaching/research assistant in Department of Systems, Control and Industrial Engineering at Case Western Reserve University, where he held a research associate position from January 1994 to May 1994. He held a post-doctoral position in Department of Electrical and Computer Engineering at Boston University from June 1994 to August 1995.

From September 1995 to May 1997, he was a research assistant in Department of Electrical and Computer Engineering at Boston University. From June 1997 to July 1998, he was a Visiting Assistant Professor in Department of Electrical Engineering at the University of Texas at Dallas. From July 1998 to May 2000, he was an Assistant Professor in the Department of Electrical and Computer Engineering at New Jersey Institute of Technology, Newark, New Jersey. In May 2000, he joined the Department of Electrical and Computer Engineering at University of Florida, Gainesville, Florida, where he got the early promotion with tenure in August 2003 and has been an Associate Professor since then. His research interests span many areas including wireless networks, mobile computing, mobile communications, automatic control, and neural networks. He has published over one hundred (100) papers in refereed professional journals and conferences. He received the National Science Foundation Faculty Early Career Award in 2001 and the Office of Naval Research Young Investigator Award in 2002.

He has actively engaged in many professional activities. He is a senior member of the IEEE and a member of the ACM. He is an Editor for IEEE Transactions on Communications, an Editor for IEEE Transactions on Wireless Communications, an Editor for IEEE Transactions on Mobile Computing, a technical editor for IEEE Wireless Communications Magazine, an Editor for ACM Wireless Networks, and an Area Editor for ACM Mobile Computing and Communications Review. He was an Editor for IEEE Journal on Selected Areas in Communications: Wireless Communications Series from May 1999 to December 2001, an Editor for Wiley International Journal on Wireless Communications and Mobile Computing from April 2000 to January 2004 and the Feature Editor for Scanning the Literature in IEEE Personal Communications (now the IEEE Wireless Communications) from April 2000 to April 2003. He has also actively involved with many professional conferences. He is the Program Co-Chair for the Global Internet and Next Generation Networks Symposium in IEEE Globecom'2004, was the Program Vice Chair for 2000 IEEE Wireless Communications and Networking Conference (WCNC'2000) where he received the IEEE Appreciation Award for the service to this conference. He has been serving on many Technical Program Committees such as IEEE INFOCOM (2005, 2004, 2003, 2000, and 1998), IEEE ICC (2004), IEEE Globecom (2004, 2003 and 2002), IEEE WCNC (2004, 2002, 2000 and 1999), and ACM MobiCom (2001). He served as the Committee Co-Chair for Student Travel Award for 2002 ACM MobiCom. He was the Vice-Chair for the IEEE Gainesville Section in 2002 and 2003, and is the Chair in 2004.