Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT•

Computer Communications 26 (2003) 1570–1580

computer communications

www.elsevier.com/locate/comcom

Algorithms and protocols for stateless constrained-based routing

Baoxian Zhang[a], Marwan Krunz[b,*]

[a]*Department of ECE, Queen's University, Kingston, Ont., Canada K7L 3N6*
[b]*Department of ECE, University of Arizona, Tucson, AZ 85721, USA*

Received 5 February 2002; revised 23 September 2002; accepted 31 October 2002

Abstract

Quality of service (QoS) routing has generally been addressed in the context of reservation-based network services (e.g. ATM, IntServ), which require explicit (out of band) signaling of reservation requests and maintenance of per-flow state information. It has been recognized that the processing of per-flow state information poses scalability problems, especially at core routers. To remedy this situation, in this paper we introduce an approach for *stateless* QoS routing in IP networks that assumes no support for signaling or reservation from the network. Instead, our approach makes use of the currently unused two bits in the DiffServ (DS) byte of the IP packet header. Simple heuristics are used to identify a low-cost delay-constrained path. These heuristics essentially divide the end-to-end path into at most two 'superedges' that are connected by a 'relay node'. Routers that lie on the same superedge use either the cost metric or the delay metric (but not both) to forward the packet. Standard hop-by-hop forwarding is performed with respect to either metric. Two different approaches are presented for implementing the relay-based forwarding. In the first approach, a *probing* protocol is used to identify the relay node and the routing metrics of the superedges. Tunneling and packet encapsulation are then used to forward packets from the source node to the relay node and then from the relay node to the destination node. The second approach does not require probing, but instead relies on the time-to-live (TTL) field in the header of the IP packet. Simulations are presented to evaluate the cost performance of the various approaches.

Keywords: QoS routing; Delay-constrained path selection; Stateless routing

1. Introduction

1.1. Motivation

The growing popularity of real-time and multimedia applications over the Internet has stimulated strong interest in extending QoS support to existing routing protocols (e.g. OSPF, BGP) [3,5,15,17,49]. Several recent studies have acknowledged the need for scalable QoS routing solutions [4,11,27,29,48] and have given the stimulus for a number of proposals on how to integrate QoS routing into a Differentiated Services (DiffServ) framework [21,24,43]. For example, in one proposal that advocates integrating the traffic engineering and scalable reservation aspects of MPLS into the DiffServ architecture [21], QoS routing is suggested as a mechanism to establish 'MPLS paths' onto which aggregates of IP flows can be transported with given QoS. In this context, MPLS paths are dynamically established between ISPs as part of their bilateral Service Level Agreements (SLAs).

So far, most work on QoS routing has been carried out under the assumption that the underlying QoS architecture is *reservation based*. In such architecture, routers maintain per-flow state information (e.g. flow identity, the amount of allocated bandwidth, priorities, etc.) and end systems use explicit reservation messages (e.g. RSVP messages, ATM PNNI signaling) to indicate their QoS requirements. In this case, the goal of a QoS routing algorithm is to identify a resource-efficient path for routing the *reservation message*. To reduce the connection setup time, the path to be identified should be likely to satisfy the requested QoS requirements (i.e. the reservation request is likely to pass the admission control test at all routers along the selected path). The advantage of this explicit-reservation model is that it allows the service provider to guarantee the requested QoS on an end-to-end basis with a high degree of accuracy by implementing admission control, traffic conditioning, and reservation protocols. On the other hand, it has been widely recognized that maintaining per-flow state information

* Corresponding author. Tel.: +1-520-621-8731; fax: +1-520-621-3862.
E-mail address: krunz@ece.arizona.edu (M. Krunz).

could lead to scalability problems at core routers. Accordingly, the focus of the research community has shifted to *stateless* QoS frameworks (e.g. DiffServ [9,10]) that rely on per-class service differentiation at core routers (with class information encoded in the header of the data packet), leaving the maintenance of the per-flow state information to edge routers. A stateless service model seems particularly appealing to router vendors, whose priority is to reduce the processing burden at the router to achieve high-speed packet forwarding. Thus far, the research on stateless QoS has mainly focused on packet scheduling issues (e.g. Refs. [31, 37–39]). In fact, it has been noticed that existing QoS routing solutions have been developed 'at some distance from the task of development of QoS architectures' [22]. In particular, current QoS architectural models, including the DiffServ, seem to implicitly assume that various classes of traffic are forwarded along the same (best effort) path, with service differentiation being achieved *locally* through appropriate packet scheduling. Decoupling routing and QoS provisioning can lead to 'inefficient' selection of routes, hence reducing the likelihood of meeting the applications end-to-end QoS requirements.

In this paper, we propose a simple approach for stateless QoS routing in IP networks. This approach is intended to complement, and be part of, existing stateless QoS architectures, such as the DiffServ. It relies on *dynamic packet forwarding mechanisms* that make novel use of the two unused bits in the DiffServ (DS) byte and, optionally, the Time-to-Live (TTL) field in the header of an IP packet. Historically, the DS byte used to be called the TOS (Type of Service) byte, which was originally intended for something very similar to QoS routing. However, the original TOS byte was rarely used for this purpose. Recently, this byte has been renamed as the DS byte, with six of the eight bits being used to encode the class information in the DiffServ framework. In this paper, we propose to use the remaining two bits to encode the routing metric according to which packet forwarding is to be performed. We refer to these two bits as the *routing metric identifier* (RMI) field. We present a stateless QoS routing approach in which routers maintain routing entries *for each concerned routing metric* (e.g. delay, cost, etc.). Our approach involves the *injection of probe messages* for exploration of viable QoS paths. No state information is maintained at a router, which performs the forwarding function based on information contained in the packet header. Protocols and algorithms for supporting the proposed stateless QoS routing framework are presented. Our approach requires very small extra computational overhead beyond what is currently used in best-effort routing.

The rest of this paper is structured as follows. In the remaining of this section, we present the network model and an algorithmic statement of the QoS routing problem. In Section 2 we present a series of source-based and distributed heuristics for delay-constrained path selection. These heuristics are based on the *relay* philosophy, in which the routing metric of interest can be switched at *relay nodes*, thus enabling stateless QoS routing. The proposed heuristics can always find a feasible (delay constrained) path if one exists. Two approaches for QoS based packet forwarding are presented in Section 3. In the first approach, a probing protocol is outlined for the discovery of a relay node. Once a relay node is identified, tunneling and packet encapsulation are used to forward packets along a low-cost QoS path with known delay. The second approach relies on a novel use of the TTL field in the header of an IP packet. Simulation results are presented in Section 4, followed by concluding remarks in Section 5.

1.2. Network model and problem formulation

The goal of QoS routing, in general, is to identify a resource efficient path (or paths) that satisfies one or more constraints on the maximum packet delay, available bandwidth, etc. Different formulations of this problem have been used in the literature (e.g. multi-constrained path selection [23,25,26], restricted shortest path [20], etc.). However, in many practical situations, the problem reduces to finding a low-cost delay constrained path, which can be stated as follows:

Definition 1. *Delay-constrained least-cost (DCLC) problem*: Consider a network that is represented by a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of possibly asymmetric links. Each link $e = (i, j) \in E$ is associated with a cost value $C(e)$ and a delay value $D(e)$. The cost of a link can be assigned in various ways (e.g. link utilization, inverse of available bandwidth, etc.). Given a delay constraint Δ, the problem is to find a path P from a source node s to a destination node d such that:

1. $D(P) \equiv \sum_{e\in P} D(e) \le \Delta$
2. $C(P) \equiv \sum_{e\in P} C(e)$ is minimized over all paths satisfying the first condition.

The DCLC problem is known to be NP-hard [14], necessitating the reliance on heuristic solutions. Several computationally efficient heuristics have been proposed in the literature (e.g. [20,35,40,41,46]). In Ref. [46] the author proposed an algorithm that performs a breadth-first search to find the optimal solution for the DCLC problem. However, the running time of this algorithm grows exponentially with the network size. Fully polynomial approximation schemes (FPAS) are already known for many NP-complete problems that can be solved by pseudo-polynomial algorithms. Two efficient ϵ-optimal approximation algorithms were proposed by Hassin [20] with running times of $O(\log\log B(|E|(|V|/\varepsilon) + \log\log B))$ and $O(|E|(|V|^2/\varepsilon)\log(|V|/\varepsilon))$, respectively, where B is an upper bound on the optimal cost ($|V| - 1$ times the maximum link cost). These algorithms produce a path with a cost that is at most ϵ-factor from the optimal one. Despite the algorithmic elegance of these algorithms, they are still too complex to be applied in large-scale networks. Furthermore, they all are source based.

Distributed solutions to the DCLC problem have been proposed in Refs. [35,40]. These solutions have the following properties in common: (1) each node knows the minimum cost and minimum delay to every other node in the network (disseminated using path-vector protocols or computed by executing Dijkstra's algorithm on link-state information); (2) the cost and delay vectors at all nodes are up-to-date (or at least consistent), a condition that may not always hold in a dynamic network; and (3) the path finding process added one node at a time, where in each time the added node lies on either the least-cost (LC) path or on the least-delay (LD) path. The last restriction is aimed at reducing the computational complexity of the search algorithm. The worst-case message complexities of the heuristics in Refs. [35,40] are $O(|V|^2)$ and $O(|V|)$, respectively. In Ref. [41] the authors attempt to combine the benefits of probing and backtracking based algorithms (better adaptiveness and wider search) with those of distance-vector algorithms (lower setup time). However, the worst-case message complexity of their heuristic grows exponentially with the size of the network.

2. Path selection algorithms

To achieve simple QoS-based forwarding, we restrict our scope to path selection algorithms in which the computed path consists of several concatenated *superedges*. A superedge is defined as a connected segment of the path on which all routers use the same routing metric for packet forwarding. A router that connects two superedges is referred to as a *relay node*. First, we consider source based path selection heuristics.

2.1. Source-based heuristics

Consider the DCLC problem with delay constraint Δ. Suppose that a cost-efficient delay-constrained path is to be found between a source node s and a destination node d. Let $P_{lc}(u,v)$ and $P_{ld}(u,v)$ indicate, respectively, the LC and LD paths from node u to node v, where u and v are any two nodes in V.

Heuristic 1. A trivial heuristic for DCLC is to either choose the LC path or the LD path, i.e. the computed path consists of only one superedge. Clearly, if $D(P_{lc}(s,d)) \leq \Delta$ then the optimal solution is given by $P_{lc}(s,d)$. Otherwise, the LD path is chosen provided that $D(P_{ld}(s,d)) \leq \Delta$ (if $D(P_{ld}(s,d)) > \Delta$, there is no feasible path). The computational complexity of this simple heuristic is twice that of Dijkstra's if link-state routing is used. If distance-vector routing is used, the complexity is simply $O(1)$.

Heuristic 2. Going one step further, we allow the computed path to consist of up to two superedges. For each node $v \in V$, the algorithm considers the four possible paths: $P_{ld}(s,v) \cup P_{ld}(v,d)$, $P_{ld}(s,v) \cup P_{lc}(v,d)$, $P_{lc}(s,v) \cup P_{ld}(v,d)$, and $P_{lc}(s,v) \cup P_{lc}(v,d)$. Of the approximately $4|V|$ possible paths, the algorithm selects the one with the minimum cost provided that this path satisfies the delay constraint. The node that connects the two superedges on the selected path is called the *relay node*. Note that for an *arbitrary* node v, the path $P_{ld}(s,v) \cup P_{ld}(v,d)$ is not necessarily the LD path from s to d. More specifically, $P_{ld}(s,v) \cup P_{ld}(v,d)$ may contain a loop (i.e. node v may not be on the LD path from s to d). Likewise, $P_{lc}(s,v) \cup P_{lc}(v,d)$ is not necessarily the same as $P_{lc}(s,d)$. However, as shown later, the minimum-cost path returned by the algorithm is guaranteed to be loop free. The complexity of Heuristic 2 is four times that of Dijkstra's.

Heuristic 2 can be implemented as follows:

Step 1: Compute the LD path from node s to every node $v \in V$.
Step 2: If $D(P_{ld}(s,d)) > \Delta$, return FAILURE (there is no feasible path). Otherwise, go to Step 3.
Step 3: Compute the LC path from node s to each node $v \in V$.
Step 4: If $D(P_{lc}(s,d)) \leq \Delta$, return the path $P_{lc}(s,d)$. Otherwise, continue.
Step 5: Compute the LD path from every node $v \in V$ to node d.
Step 6: Compute the LC path from every node $v \in V$ to node d.
Step 7: From the resulting $4|V|$ paths, $P_i(s,v) \cup P_j(v,d)$ where $i,j \in \{\mathrm{LC}, \mathrm{LD}\}$ and $v \in V$, choose the one with the smallest cost.

Steps 1 and 3 require two runs of Dijkstra's algorithm, while Steps 5 and 6 require two runs of Reverse Dijkstra's [7]. The running time of Step 7 is $O(|V|)$. Therefore, the overall complexity of Heuristic 2 is $O(|V|^2)$. If the Floyd–Warshall algorithm is used instead of Dijkstra's algorithms, then the *one-time* complexity jumps to $O(|V|^3)$, but only Step 7 needs to be executed online (in the Floyd–Warshall algorithm, each node knows the LC and LD values for the paths between all pairs of nodes in the network).

The following results can be stated on the performance of Heuristic 2. Let $\mathrm{PATH}_{\mathrm{opt}}(\Delta)$ and $\mathrm{PATH}_{\mathrm{algo2}}(\Delta)$ denote the optimal path and the path returned by Heuristic 2, respectively.

Result 1. For a node $v \in \mathrm{PATH}_{\mathrm{opt}}(\Delta)$, if $P_{lc}(s,v) \cup P_{lc}(v,d)$ is a feasible path, then $C(\mathrm{PATH}_{\mathrm{opt}}(\Delta)) = C(\mathrm{PATH}_{\mathrm{algo2}}(\Delta))$, i.e. the path returned by Heuristic 2 is optimal. The proof is trivial and is omitted for brevity.

Result 2. For any node $v \in V$, if both $P_{lc}(s,v) \cup P_{ld}(v,d)$ and $P_{ld}(s,v) \cup P_{lc}(v,d)$ are feasible, then

$$D(P_{lc}(s,v) \cup P_{lc}(v,d)) \leq 2\Delta - D(P_{ld}(s,d)). \qquad (1)$$

Result 2 follows from the fact that $D(P_{lc}(s,v))+D(P_{lc}(v,d))+D(P_{ld}(s,v))+D(P_{ld}(v,d)) \leq 2\Delta$, so that

$$\begin{aligned} &D(P_{lc}(s,v) \cup P_{lc}(v,d)) \\ &\quad \leq 2\Delta - [D(P_{ld}(s,v)) + D(P_{ld}(v,d))] \\ &\quad \leq 2\Delta - D(P_{ld}(s,d)). \qquad (2) \end{aligned}$$

Corollary.

(i) If $P_{lc}(s,v) \cup P_{lc}(v,d)$ is a feasible path with cost that is larger than the cost of the path returned by Heuristic 2, then it must be that $v \notin \text{PATH}_{\text{opt}}(\Delta)$.

(ii) If both $P_{lc}(s,v) \cup P_{ld}(v,d)$ and $P_{ld}(s,v) \cup P_{lc}(v,d)$ are feasible, where $v \in \text{PATH}_{\text{opt}}(\Delta)$, then we have

$$\begin{aligned} &C(\text{PATH}_{\text{algo2}}(2\Delta - D(P_{ld}(s,d))) \\ &\quad \leq C(\text{PATH}_{\text{opt}}(\Delta)). \qquad (3) \end{aligned}$$

That is, if the path returned by Heuristic 2 satisfies the premise in (ii) under a delay constraint $2\Delta - D(P_{ld}(s,d))$, then the cost of this path cannot be worse than the cost of the optimal path under a delay constraint Δ.

Heuristic 3. Heuristic 3 represents a tradeoff between the previous two heuristics (less computational complexity than Heuristic 2 but better performance than Heuristic 1). In here, each node v in the network maintains a *delay table* and a *cost table*, each consisting of $|V| - 1$ entries (one entry for every other node). The entry for node v_j at node v consists of:

- The address of node v_j;
- The delay of the LD path from v to v_j;
- The cost of the above path (i.e. $C(P_{ld}(v,v_j))$; and
- The predecessor of v_j on the LD path from v to v_j, $ld\_lhop(P_{ld}(v,v_j))$.

Similar information is maintained in the cost table but with the LD path replaced by the LC path. For distance-vector routing, the information in these tables can be disseminated as distance vectors. If link-state routing is being used, then this information can be obtained after two executions of Dijkstra's algorithm (with an extra label to keep track of the predecessor to each destination).

Given the above information, the algorithm first verifies that there is a feasible path in the network by checking whether $D(P_{ld}(s,d)) \leq \Delta$. If so, the algorithm checks if the LC path $P_{lc}(s,d)$ is feasible, in which case it returns it as the optimal solution. Otherwise, the algorithm proceeds in two phases, as shown in Fig. 1. In Phase 1, the search proceeds backward from the destination node d to the source node s along the LD path $P_{ld}(s,d)$ until a relay node v is found for which $D(P_{lc}(s,v))+D(P_{ld}(v,d)) \leq \Delta$. If such a node is found, Phase 1 returns the path $P_{lc}(s,v) \cup P_{ld}(v,d)$. Otherwise, the returned path from this phase defaults to $P_{ld}(s,d)$. Phase 2 attempts to improve upon the outcome of Phase 1 by searching for a feasible path that consists of a LD segment from s to a relay node v followed by a LC segment from node v to node d (i.e. the search starts from node d and proceeds along the LC path). Intuitively, the combined path $P_{lc}(s,v) \cup P_{ld}(v,d)$ for a node $v \in P_{ld}(s,d)$ has a lower cost than the path $P_{ld}(s,d)$. Similarly, the combined path $P_{ld}(s,v) \cup P_{lc}(v,d)$ for a node $v \in P_{lc}(s,d)$ has a smaller delay than the path $P_{lc}(s,d)$. It is worth noting that

```
/* Phase 1: The path search process proceeds from destination node $d$ backwards along the LD path $P_{ld}(s,d)$ */
1.  active_node =: d      /* active_node is the node being tested for a relay node */
2.  PATH =: $P_{ld}(s,d)$      /* PATH is the path returned by the algorithm; $P_{ld}(s,d)$ is the first known feasible path */
3.  while active_node≠s, do
4.        $D(P_{ld}(active\_node,d)) =: D(P_{ld}(s,d)) - D(P_{ld}(s,active\_node))$
5.        if $D(P_{lc}(s,active\_node)) + D(P_{ld}(active\_node,d)) \leq \Delta$
6.              PATH =: $P_{lc}(s,active\_node) \cup P_{ld}(active\_node,d)$
7.              Go to Line 11
8.        else set active_node to predecessor of active_node on $P_{ld}(s,active\_node)$
9.        end if-else
10. end while
/* Phase 2: The path search process proceeds from destination node $d$ backwards along the LC path $P_{lc}(s,d)$ */
11. active_node =: d
12. while active_node≠s, do
13.       $D(P_{lc}(active\_node,d)) =: D(P_{lc}(s,d)) - D(P_{lc}(s,active\_node))$
14.       if $D(P_{ld}(s,active\_node)) + D(P_{lc}(active\_node,d)) \leq \Delta$
15.          if $C(P_{ld}(s,active\_node)) + C(P_{lc}(active\_node,d)) < C(\text{PATH})$   /* C(PATH) is obtained from Phase 1 */
16.             PATH =: $P_{ld}(s,active\_node) \cup P_{lc}(active\_node,d)$  /* a feasible path with a lower cost is found */
17.          else set active_node to predecessor of active_node on $P_{lc}(s,active\_node)$; continue while loop
18.          end if-else
19.       else break while loop
20.       end if-else
21. end while
22. return PATH
```

Fig. 1. Pseudo-code for Heuristic 3.

the combined path $P_{ld}(s,v) \cup P_{lc}(v,d)$ for a node $v \in P_{ld}(s,d)$ and the path $P_{lc}(s,v) \cup P_{ld}(v,d)$ for a node $v \in P_{lc}(s,d)$ can also be used to further improve the performance of the algorithm. However, this requires executing Reverse Dijkstra's algorithm for each session with a different destination, which will increase the $O(|V|)$ computational complexity of Heuristic 3.

In line 19 of the algorithm, the backward search along the LC path is terminated once a node, say v, along the LC path from s to d is found for which $D(P_{ld}(s,v)) + D(P_{lc}(v,d)) > \Delta$. This is because for any node w on the path $P_{lc}(s,v)$, the path $P_{ld}(s,w) \cup P_{lc}(w,d)$ cannot be feasible since

$$\begin{aligned} &D(P_{ld}(s,w)) + D(P_{lc}(w,d)) \\ &\quad = D(P_{ld}(s,w)) + D(P_{lc}(w,v)) + D(P_{lc}(v,d)) \\ &\quad \geq D(P_{ld}(s,v)) + D(P_{lc}(v,d)) > \Delta \end{aligned}$$

So there is no use in continuing to search for a relay node along the LC path from s to v.

2.2. Correctness of the Proposed Heuristics

Since Heuristics 1 and 3 are special cases of Heuristic 2, it is sufficient to prove the correctness of Heuristic 2. It is obvious that the algorithm always returns a feasible path, if one exists. So we only need to prove that the returned path is loop-free. Note that a path that consists of two superedges (e.g. an LC superedge and an LD superedge) may contain a loop.

If the LD path from s to d is feasible, then Heuristic 2 will check whether the LC path from s to d, $P_{lc}(s,d)$, is feasible. If so, this path is returned, which is clearly loop-free. The other cases to be considered are as follows:

1. If the combined path through every relay node is not feasible or if the costs of all such paths are greater than or equal to $C(P_{ld}(s,d))$, then the algorithm will return $P_{ld}(s,d)$, which is also loop-free.
2. Now suppose that the algorithm returns a path that consists of two superedges. This path can be expressed as $P_x(s,v) \cup P_y(v,d)$ for some relay node $v \in V$, where $x, y \in \{\text{LC,LD}\}$. There are four cases to be considered, depending on x and y. Without loss of generality, consider the case when both x and y are equal to LC. We need to prove that the returned path $P_{lc}(s,v) \cup P_{lc}(v,d)$ is loop-free. Assume to the contrary that there exists a loop in this path; suppose that there exists a node w on the LC path from s to v that is also on the LC path from v to d (see Fig. 2). Since $P_{lc}(s,v) \cup P_{lc}(v,d)$ is the returned path by Heuristic 2, we have

$$C(P_{lc}(s,w) \cup P_{lc}(w,d)) \geq C(P_{lc}(s,v) \cup P_{lc}(v,d)) \quad (4)$$

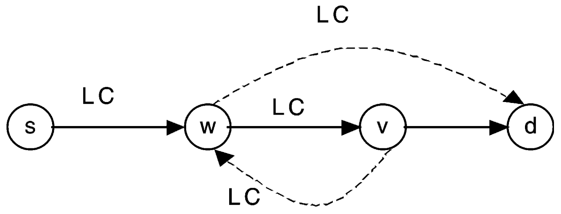


Fig. 2. Hypothetical scenario for the occurrence of a loop in a returned path with two superedges.

But from Fig. 2 it is clear that

$$\begin{aligned} &C(P_{lc}(s,v) \cup P_{lc}(v,d)) \\ &\quad = C(P_{lc}(s,w) \cup P_{lc}(w,v)) + C(P_{lc}(v,w) \\ &\quad\quad \cup P_{lc}(w,d)) \end{aligned} \quad (5)$$

which leads to an obvious contradiction. Thus, the returned path must be loop-free.

2.3. Probe-based distributed heuristic

The previously introduced heuristics are source based. In source-based routing, the source node performs the path computation and inserts the full path in the connection request (or in each packet) before sending the request to the next hop. Packet-based source routing is available in IP, but is rarely used. Instead, the vast majority of IP implementations rely on hop-by-hop (distributed) routing, in which all nodes along the path participate in the path computation task. To maintain compatibility with the current IP infrastructure, we present a distributed implementation of our stateless QoS routing approach, which is based on hop-by-hop packet forwarding. This implementation, called *distributed delay-constrained algorithm* (DDCA), uses the previously discussed relay strategy. In essence, DDCA is an extension of the DCR algorithm [40]. DCR uses the following procedure to construct a delay-constrained path from a source node to a destination node. The reservation message travels along the LD path until reaching a node from which the delay of its LC path satisfies the delay constraint. From that node and on, the message travels along the LC path all the way to the destination. In DDCA, we replace the path construction process by a path *probing* process and extend the probing direction to include both the LC and LD directions.

In DDCA each node in the network maintains a *delay table* and a *cost table*. These tables are similar to those maintained in Heuristic 3 except that the entry for the predecessor node to a destination is replaced by the next hop along the LD (LC) path, *ld_nhop* (*lc_nhop*). Similar information is also maintained in the algorithms in Refs. [35,40]. Note that in DDCA a node need not maintain the network topology and link-state information, as in source-based routing algorithms. The information in the delay and cost tables can be distributed to nodes using distance (or path) vector protocols [18].

Initially, the algorithm checks the feasibility of the LC path from s to d. If $D(P_{lc}(s,d)) \le \Delta$, the algorithm returns this path. Otherwise, the algorithm checks if a feasible path is available (by verifying that $D(P_{ld}(s,d)) \le \Delta$). If so, a *probing protocol* is used to discover an appropriate relay node that results in a low-cost feasible path. According to this protocol, the source node constructs two probe messages and sends them to a destination node d. One of these messages is sent along the LD path to node d, while the other is sent along the LC path. Each probe message contains the following fields:

- *probe_direction*: Forwarding direction of the probe message (LC or LD).
- *next_node*: Address of next hop on the path of the probe message.
- *relay_node*: Address of relay node (initially set to NULL).
- *delay_so_far*: Accumulated delay of path traversed by the probe message from the source node up to the current router.
- *cost_so_far*: Accumulated cost of path traversed by the probe message from the source node up to the current router.
- *delay_constraint*: Δ.
- *total_cost*: Cost of the best-known feasible path, initially set to $C(P_{ld}(s,d))$. Once a relay node is discovered, the value in this field is adjusted (reduced) to reflect the cost of the new path.
- *type*: Type of probe message, which can be a *probe query* message or a *probe reply* message.

Starting from the source, nodes along the LD (LC) path to d are probed, one at a time. Consider, for example, the probe message that is sent along the LD path. Node s sets the fields in this message as follows:

probe_direction ← LD (bit 0)
next_node ← *ld_nhop* (read from the routing tables at node s)
relay_node ← NULL
delay_so_far ← *D(s, ld_nhop)*
cost_so_far ← *C(s, ld_nhop)*
delay_constraint ← Δ
total_cost ← $C(P_{ld}(s,d))$ (cost of the first known feasible path)

The probe message that is sent along the LC direction is initialized in an analogous manner, but with *lc_nhop* replacing *ld_nhop*. Probe messages are sent to the next hop along the LD and LC paths, respectively. When a node v receives a probe message, it executes the algorithm in Fig. 3.

In lines 7 and 23 Fig. 3, the *probe reply* message contains the direction of the probe (LD or LC), the identity of the relay node v, and the total cost of the discovered path. Unless the cost of the new path is less than $C(P_{ld}(s,d))$, the value in the *relay_node* field will stay at NULL. For the probe message that is sent along the LD direction, a *probe reply* is generated by the *first* relay node, say v, that satisfies $D(P_{ld}(s,v)) + D(P_{lc}(v,d)) \le \Delta$. This is because it is not possible to obtain a lower-cost path than $P_{ld}(s,v) \cup P_{lc}(v,d)$ by selecting another relay node on the LD path from v to d. To see that, let w be a node on the LD path from v to d

```
1.  if probe_direction=LD
2.     if delay_so_far + $D(P_{lc}(v,d)) \le \Delta$ /* node v is a relay node */
3.        if cost_so_far + $C(P_{lc}(v,d))$ < total_cost  /* path is better than the LD path */
4.           relay_node ← v
5.           total_cost ← cost_so_far + $C(P_{lc}(v,d))$
6.        end if
7.        Send a probe reply message to node s
8.     else /* node v is not a relay node */
9.        delay_so_far ← delay_so_far + D(v, ld_nhop)
10.       cost_so_far ← cost_so_far + C(v, ld_nhop)
11.       Forward probe query message to ld_nhop
12.    end if-else
13. else   /* probe_direction=LC  */
14.    if delay_so_far + $D(P_{ld}(v,d)) \le \Delta$ /* node v is a relay node */
15.       if cost_so_far + $C(P_{ld}(v,d))$ < total_cost /* a better path is found */
16.          relay_node ← v
17.          total_cost ← cost_so_far + $C(P_{ld}(v,d))$
18.       end if
19.       delay_so_far ← delay_so_far + D(v,lc_nhop)
20.       cost_so_far ← cost_so_far + C(v,lc_nhop)
21.       Forward probe query message to lc_nhop
22.    else /* node v is not a relay node */
23.       Send a probe reply message to node s
24.    end if-else
25. end if-else
```

Fig. 3. Pseudo-code for DDCA.

and let $D(P_{ld}(s,w)) + D(P_{lc}(w,d)) \leq \Delta$. Then,

$$C(P_{ld}(s,w) \cup P_{lc}(w,d))$$
$$= C(P_{ld}(s,v)) + C(P_{ld}(v,w)) + C(P_{lc}(w,d))$$
$$\geq C(P_{ld}(s,v)) + C(P_{lc}(v,d)) \qquad (6)$$

So there is no point is continuing the search beyond node v. This does not apply to the probe message sent along the LC path, where in this case the search continues for a possibly better relay node. However, in this case, the search terminates unsuccessfully (line 23) if the probe message sent along the LC path encounters a node v for which $D(P_{lc}(s,v)) + D(P_{ld}(v,d)) > \Delta$. This is because for any subsequent node w on the path $P_{lc}(s,d)$, the path $P_{lc}(s,w) \cup P_{lc}(w,d)$ cannot be feasible since

$$D(P_{lc}(s,w)) + D(P_{ld}(w,d))$$
$$= D(P_{lc}(s,v)) + D(P_{lc}(v,w)) + D(P_{ld}(w,d))$$
$$\geq D(P_{lc}(s,v)) + D(P_{ld}(v,d)) > \Delta$$

So there is no use in continuing to search for a relay node along the LC path from v to d.

When node s receives the two probe reply messages, it selects the path with the lower cost (if both messages contain NULL in the relay node field, then node s selects the LD path between s and d). Note that the probe direction is needed in the reply to distinguish between LD + LC and LC + LD paths. A *probe query* message may visit up to $|V-1|$ nodes. Few computations are needed at each node to process a probe message. Hence, the worst-case message complexity of DDCA is $O(|V|)$.

There are two possible approaches to encode the probe messages. The first one is to encode these messages as ICMP (Internet Control Message Protocol) packets. Currently, intermediate routers do not process the payload portion of an ICMP packet. But it is possible to define a new ICMP packet type and assign to it one of the unused values in the *type* field. A QoS-capable router that receives an ICMP packet with this new type interprets this packet as a probe message, so it processes it (e.g. it inserts the IP address of the relay node). Routers that do not support QoS routing can simply ignore probe messages. The second, and perhaps more viable, approach is to define a new protocol type, called Internet Probe Message Protocol (IPMP), which is somewhat similar to ICMP except that it requires routers to process the payload portion of the IPMP packet. Currently, the *protocol field* byte in the IP header has several unassigned values [34], and one of these values can be used for IPMP. Both of the above approaches are backward-compatible with the existing IP.

One important issue in the probing protocol is how often probe messages are generated. One possibility is to send these messages on demand (whenever a real-time flow is to be 'established'). Another possibility is to send them on a periodic basis (e.g. at the same rate as that of link-state or distance-path update messages), and implement path caching at the source. A combination of both approaches can also be used. Probe messages may also be sent during the life of a flow for the purpose of identifying a better route (i.e. rerouting the flow of packets). In this case, the source may rely on end-to-end delay measurements to infer the need for rerouting. The effectiveness of such approaches is currently under investigation, and will be reported in a future paper.

3. Dynamic forwarding mechanisms

Once a cost-effective constrained path has been identified using DDCA, or a similar algorithm, the next step is to design appropriate forwarding mechanisms that implement the relay concept in a distributed manner. In here, we present two such mechanisms. The first one requires the use of a probing protocol, while no probing is needed in the second mechanism. Both mechanisms are suitable for stateless QoS routing.

3.1. Tunneling and encapsulation-based forwarding

Suppose that a relay node v has been identified. Without loss of generality, assume that $v \notin \{s,d\}$. To forward a packet from source node s to relay node v along, say, the LD path, and then from node v to destination node d along, say, the LC path, the original IP packet is encapsulated into another (outer) packet (see Fig. 4). The destination address of the outer packet is set to the address of the relay node, while the destination address of the inner packet is set to the address of node d. Let f_{rm} be a function that maps the routing metric (e.g. delay, cost, etc.) into an appropriate numeric value that is encoded in the 2-bit RMI field. For the outer packet f_{rm}(delay) is inserted, while for the inner packet f_{rm}(cost) is inserted. The packet is then forwarded in a standard hop-by-hop manner. The outer packet will be forwarded along the LD path to node v, which in turn strips off the outer header and forwards the inner packet to node d along the LC path. Tunneling the original packet via node v can be achieved using the IP-within-IP encapsulation technique, originally defined for Mobile IP [32]. It is also possible to use *minimal encapsulation* techniques such as the one defined in Ref. [33].

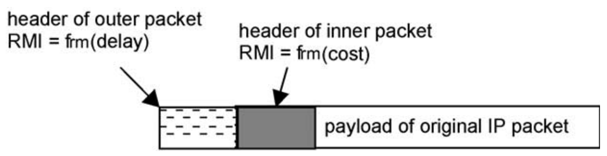


Fig. 4. Use of IP-within-IP encapsulation in QoS routing.

An advantage of the above approach is that once the relay node is identified, routers forward packets in the same way as in best-effort routing (the relay node need not process the 2-bit RMI field, which is set at the source). However, the previously discussed probing protocol is needed to identify a relay node. Also, there is some overhead associated with packet encapsulation and tunneling and with sending probe messages. Finally, because the outer destination address is not that of the true destination, it is relatively difficult to perform resource management and distribution with respect to the destination address. This last problem may be remedied by using IP loose-source routing (LSR), currently used for debugging purposes, with the address of the relay node(s) included in the option field.[1]

It should be emphasized here that the concept of tunneling has already been used for multicast routing in the Internet multicast backbone (MBONE) [13], and has also been proposed in the context of ATM [2]. However, in both applications the purpose of tunneling is not to reduce the maintenance and routing burden at the multicast nodes, but to connect multicast-capable nodes through the general IP network (if all Internet routers had multicast capability, tunneling would not have been used). Tunneling has also been used in Ref. [1] to avoid scalability problems by viewing each multicast tree as a collection of unicast links (tunnels) and locating only the multicast source and destination nodes on the junction of the trees.

3.2. TTL-Based forwarding

An alternative forwarding approach is to make use of the time-to-live (TTL) field in the IP header. In current IP networks, each router decrements the TTL value by the number of seconds that the packet spends in that router (in IPv4), or by one (in IPv6). If the TTL value reaches zero, the packet is discarded. We propose an alternative approach for handling the TTL field, which makes it possible to achieve stateless QoS routing. In this approach, no probing is needed, as the relay node is identified on a per-packet basis. Starting from the source node, each router along the path of a given packet computes the *remaining time* before the packet becomes overdue. Initially, the remaining time is Δ. Once a QoS-capable router receives a packet, it updates the TTL value by subtracting from it the delay of the link from which the packet was received and the expected queueing and processing delays at the current router. The resulting value is *nonlinearly* quantized (see below) to fit into the 8-bit TTL field, and the packet is forwarded to the next hop along the LC or the LD path, depending on the RMI identifier. More specifically, a router u that receives a packet from router v executes the following algorithm before forwarding the packet to the next hop:

```
if RMI = $f_{rm}$(delay) or RMI = $f_{rm}$(cost)
    TTL = TTL − $D(v,u) - D_{queueing}(u)$ TTL
    if TTL ≥ $D(P_{lc}(u,d))$ then
        compute checksum for IP header
        forward IP packet to lc_nhop
    else if TTL ≥ $D(P_{ld}(u,d))$ then
        compute checksum for IP header
        forward IP packet to ld_nhop
    else // there is no feasible path
        drop packet or forward it with no delay guarantee
        send ICMP message (‘path infeasible’) to originating source
    end if-else
end if-else

end if
```

In the above procedure, $D_{queueing}(u)$ is the average queueing and processing delay at router u. This value is computed dynamically by router u, and can be specified for each output interface at that router.

Basically, the TTL field is being used to carry the due-date of a packet. The packet will be forwarded along the LC path if, based on the router’s information, this path can deliver the packet before its due-date. Otherwise, the router will try the LD path. If even the LD path is incapable of meeting the packet’s due-date, then there is no need to keep forwarding the packet, so the packet may be dropped or forwarded with no guarantees. Note that there is an implicit assumption here that QoS routing is used for real-time applications (e.g. voice and video), which are typically transported using UDP/IP. Hence, dropped packets will not be retransmitted by the end system, and dropping overdue packets in the middle of the path is clearly advantageous. The 2-bit RMI allows for the encoding of up to four metrics. One of these metrics can be used to indicate ‘best effort routing,’ in which case the packet is treated as in current IP networks. When a packet with RMI $= f_{rm}$(delay) or RMI $= f_{rm}$(cost) is dropped at a router, the router may send a special ICMP packet to the sender, indicating the infeasibility of the path. The sender may then decide to abort the session or to change its delay constraint (if such ICMP packet is not sent, the router will continue to drop the packets of that flow, wasting the network resources between the source and the dropping router).

As mentioned above, the remaining delay before the packet becomes overdue has to be quantized to fit the 8-bit TTL field. Nonlinear quantization techniques such as the ones suggested in Ref. [5] can be used for this purpose. In Ref. [5] the authors described a technique for

[1] However, LSR is known to have some security problems.

exponentially encoding the link bandwidth and delay values into the 16-bit TOS (Type of Service) field of OSPF link-state-advertisement (LSA) packets. Such non-linear encoding allows for the representation of a wide range of bandwidth and delay values. In brief, the encoding in Ref. [5] is done as follows. The 16-bit TOS field is divided into a mantissa part (13 bits) and an exponent part (3 bits). An appropriate base is chosen for the exponent (in Ref. [5] the authors suggested using base 8 for bandwidth and base 4 for delay). A bandwidth (or delay) value is then expressed using scientific notation. For example, a bandwidth value of 8 Gbps is equal to $4096*8^6$ bits, so the mantissa and exponent are given by 4096 and 6, respectively (both encoded in binary using 2's complement).

In our case, the TTL field consists of 8 bits. So we suggest using 4 bits for the mantissa and 4 bits for the exponent, with base 2. The values produced by such encoding consist of all the numbers $X*2^Y$, where X and Y are any numbers between 0 and 15 (note that some values can have multiple representations). The encoded values are in units of milliseconds. The maximum value that can be encoded is $15*2^{15} = 491{,}520$ ms, which is sufficiently large. When the TTL value is updated at a router, the resulting value may not exactly match the resolution produced by the above nonlinear encoding. So the router truncates the computed TTL value into the next *lower* value supported by the nonlinear encoding (i.e. the router takes a conservative approach). For example, suppose that the router receives a packet with TTL value $= 15*2^2 = 60$ ms. Suppose that the delay over the link from which the packet was received plus the average queueing and processing delay at the current router is 22 ms. So the new TTL value is $60 - 22 = 38$ ms. Such value cannot be exactly encoded, the nearest available encoding (36) is used instead. So the router inserts the value $9*2^2 = 36$ in the TTL field (mantissa = 1001 and exponent = 0100 in binary).

The main advantage of the TTL approach is that it does not require a supporting probing protocol. Also, the relay node is determined dynamically without relying on possibly outdated information at the source (as in the probing approach). On the other hand, the TTL approach requires additional processing at the routers for updating the TTL value and, possibly, the RMI field. This may turn out to be insignificant, given that routers currently process the TTL field (albeit in a different manner). Updating the RMI field may require defining a new protocol type to maintain backward compatibility with existing best-effort routing. The quantization of the remaining delay value at a router will slightly impact the performance (since the delay values may be conservatively quantized, it is possible for a router to unnecessarily reject the LC path).

4. Simulation results

We studied the cost performance of the previously presented heuristics using simulated random topologies that were generated using Waxman's method [45]. Nodes were placed randomly (with uniform distributions) on a rectangular grid of dimensions 2400 times 4000 (in simulated kilometers). Links were added such that the probability that two nodes u and v are connected by a link is given by $\beta\exp(-d(u,v)/(\alpha L))$, where $d(u,v)$ is the Manhattan distance between nodes u and v, and L is the maximum possible distance between any two nodes in the graph. The parameters α and β are selected from the range (0,1]. A large value for β results in nodes with a high average degree, and a small α gives long paths. Both parameters were varied to obtain appropriately sparse networks. All simulations were conducted on 100-node random topologies. Link costs were sampled from a uniform distribution in the range [1,10], whereas link delays were made proportional to their Manhattan distance in the coordinate grid. For each experiment, 500 random graphs were created. In each random network, we tried to set up delay-constrained paths between all possible source-destination pairs.

In the first experiment, we study the efficiency of the source-based path selection heuristics, which include Heuristics 1–3 and LDP (Least Delay Path). We use the optimal (but exponentially complex) CBF algorithm [46] as a point of reference. More precisely, we define

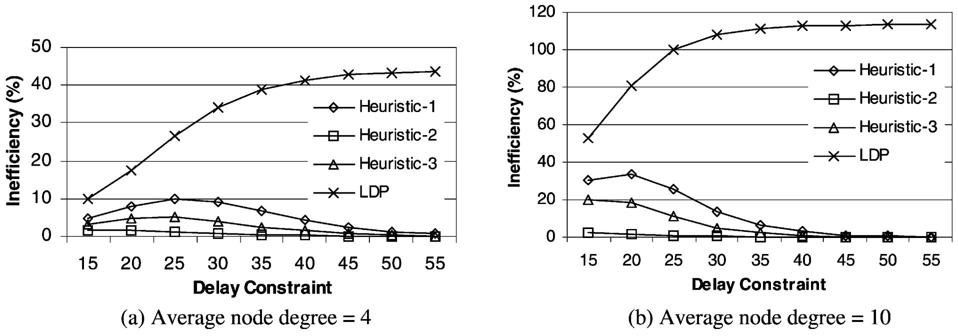
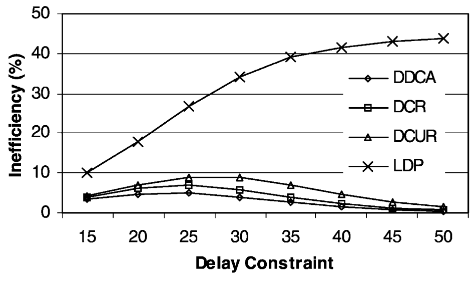
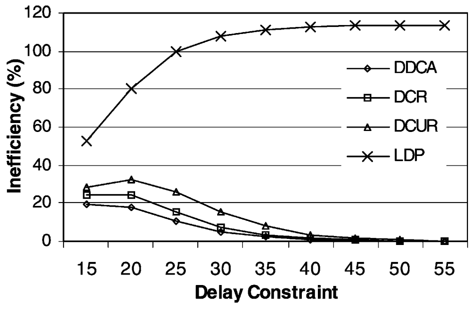


Fig. 5. Cost inefficiency of various source-based path selection heuristics.



(a) Average node degree = 4



(b) Average node degree = 10

Fig. 6. Inefficiency of distributed path selection algorithms (normalized by the cost of CBF).

the *inefficiency* of an algorithm X as:

$$\mathit{inefficiency}_{\mathrm{X}} = \frac{\mathrm{cost}_{\mathrm{X}} - \mathrm{cost}_{\mathrm{CBF}}}{\mathrm{cost}_{\mathrm{CBF}}} \quad (7)$$

Fig. 5 depicts the inefficiency of various source-based heuristics. Clearly, LDP results in a very costly path. The three other heuristics have reasonable costs, with Heuristic 2 being the most efficient and Heuristic 1 the least efficient of the three. The inefficiency in Heuristics 1 and 3 relative to CBF increases with the average degree of a node (the connectivity of the network). Heuristic 2 seems to be less sensitive to the node degree (the average additional costs of Heuristic 2 compared to CBF are 2% and 3% for average node degrees of 4 and 10, respectively). The reason why Heuristic 2 achieves good average cost performance is due to its larger search space. Note that all three source-based algorithms (and also DDCA) have the same success rate since they all return a path if one exists.

In the second experiment, we compare the cost performance of four distributed heuristics: DDCA, DCR [40], DCUR [35], and LDP. As before, the cost of CBF is used as a reference. Note that our proposed TTL-based forwarding approach has the same cost performance as that of the DCR algorithm (which assumes a connection-oriented network). The relative inefficiency of these algorithms is depicted in Fig. 6 for two values for the average node degree. Except for LDP (which is here implemented in a distributed manner), the three tested algorithms provide satisfactory cost performance, with DDCA being the most efficient, followed by DCR, and finally DCUR.

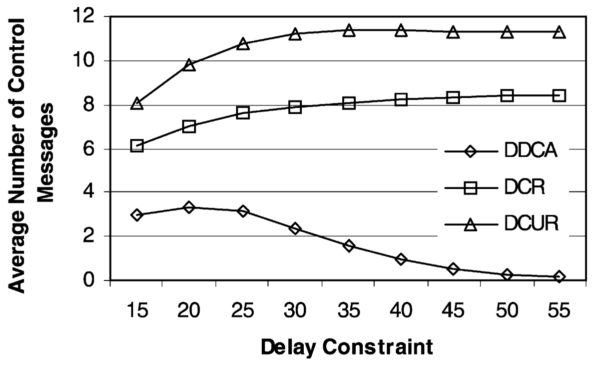


Fig. 7. Control-message overhead in distributed QoS routing algorithms versus the delay constraint (average node degree = 4).

Fig. 7 depicts the average control-message overhead in various distributed QoS routing algorithms as a function of the delay constraint. In here, the overhead is measured in the average number of exchanged messages (e.g. *probe query* and *probe reply* messages). For DCUR and DCR, the control message overhead includes *Path Construction* messages and *Acknowledge* messages (an *Acknowledge* message is sent by the destination node to notify the source node that the delay-constrained path has been successfully set up). Of the three algorithms, DDCA involves the least exchange of control messages. Moreover, it is the only algorithm that allows for stateless routing. For DDCA, increasing the delay constraint leads to a reduction in the average number of control messages. This is because when the delay constraint is large, the LC path from s to d will probably satisfy this constraint, avoiding the need for further probing

5. Conclusions

In this paper, we presented an approach for stateless QoS routing in IP networks. Our approach is based on simple heuristics for finding a low-cost delay-constrained path in a network. The returned path consists of at most two *superedges* that are connected at a *relay node*. Within a superedge, a packet is routed hop-by-hop using a given routing metric (cost or delay). The routing metric may be switched at the relay node. We presented simple source-based and distributed path selection heuristics that implement the relay strategy. Theoretical performance bounds for these heuristics were presented. We also proved the algorithmic correctness of these heuristics. To implement the relay strategy in an IP network, we provided two approaches, one relies on a probing protocol and the other makes use of the TTL field in the IP packet header. Simulation results were presented to evaluate the performance of the proposed heuristics and contrast them with previously proposed heuristics.

Acknowledgements

An abridged version of this paper was presented at the *IEEE GLOBECOM 2001 – Global Internet Symposium*, San Antonio, Dec. 2001. This work was supported in part by the National Science Foundation under grants ANI 9733143, CCR 9979310, and ANI 0095626; by the State of Arizona (Proposition 301); and by the Center for Low Power Electronic (CLPE) at the University of Arizona.

References

[1] E. Aharoni, R. Cohen, Restricted dynamic Steiner trees for scalable multicast in datagram networks, IEEE/ACM Transactions on Networking 6 (2) (1998) 286–297.

[2] M.H. Ammar, S.Y. Cheung, C. Scoglio, Routing multipoint connections using virtual paths in an ATM network, Proceedings of INFOCOM '93 March/April (1993) 98–105.

[3] G. Apostolopoulos, R. Guerin, S. Kamat, Implementation and performance measurements of QoS routing extensions to OSPF, Proceedings of the IEEE INFOCOM '99 Conference, New York, NY March (1999).

[4] G. Apostolopoulos, R. Guerin, S. Kamat, S.K. Tripathi, Quality of service based routing: a performance perspective, Proceedings of the ACM SIGCOMM '98 Conference September (1998) 17–28.

[5] G. Apostolopoulos, et al., QoS routing mechanisms and OSPF extensions, IETF-RFC 2676 August (1999).

[6] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network flows: Theory, Algorithms, and Applications, Prentice Hall, Englewood Cliffs, 1993.

[7] Y. Bernet, et al., A framework for differentiated services, IETF Internet Draft (draft-ieft-diffserv-framework-01.ext) October (1998).

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, An architecture for differentiated services, IETF RFC 2475 December (1998).

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

[10] H. Eriksson, Mbone: the multicast backbone, Communications of the ACM 27 (8) (1994) 54–60.

[11] M.R. Garey, D.S. Johnson, Computer and Intractability A guide to Theory of NP-Completeness, CW. H. Freeman and Company, New York, 1979.

[12] E. Crawley, R. Nair, B. Rajagopalan, H. Sandick, A framework for QoS-based routing in the Internet, Internet Engineering Task Force-RFC 2386 (Informational) August (1998).

[13] R. Guerin, A. Orda, D. Williams, QoS routing mechanisms and OSPF extensions, Proceedings of the IEEE Globecom '97-Global Internet Symposium, Phoenix, AZ November (1997) 1903–1908.

[14] C. Hedrick, Routing information protocol, IETF RFC June (1988).

[15] F. Hao, E.W. Zegura, On scalable QoS routing: performance evaluation of topology aggregation, Proceedings of the IEEE INFOCOM 2000 Conference, Tel Aviv, Israel March (2000) 147–156.

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

[17] J. Heinanen, Differentiated services in MPLS networks, IETF Internet Draft June (1999) (draft-heinanen-diffserv-mpls-00.txt).

[18] G. Huston, Next steps for the IP QoS architecture, IETF Internet Draft (draft-iab-qos-02.txt) August (2000).

[19] J. Jaffe, Algorithms for finding paths with multiple constraints, Networks 14 (1) (1984) 95–116.

[20] B. Jamoussi, et al., Constrained-based LSP setup using LDP, IETF Internet Draft August (1999).

[21] T. Korkmaz, M. Krunz, A randomized algorithm for finding a path subject to multiple QoS constraints, Proceedings of the IEEE GLOBECOM '99 Conference-Symposium on Global Internet: Application and Technology December (1999) 1694–1698.

[22] T. Korkmaz, M. Krunz, S. Tragoudas, An efficient algorithm for finding a path subject to two additive constraints, Proceedings of the ACM SIGMETRICS 2000 Conference, Santa Clara, California June (2000) 318–327.

[23] W.C. Lee, M.G. Hluchyi, P.A. Humblet, Routing subject to quality of service constraints in integrated networks, IEEE Network 9 (4) (1995) 46–55.

[24] Q. Ma, P. Steenkiste, Routing traffic with quality-of-service guarantees in integrated services networks, Proceedings of NOSS-DAV'98 July (1998).

[25] T. Nandagopal, V. Venkitaraman, R. Sivakumar, V. Bharghavan, Delay differentiation and adaptation in core stateless networks, Proceedings of the IEEE INFOCOM 2000 Conference March (2000) 421–430.

[26] C. Perkins, IP encapsulation within IP, IETF RFC 2003 (Standards Track) October (1996).

[27] C. Perkins, Minimal encapsulation within IP, IETF RFC 2004 (Standards Track) October (1996).

[28] J. Postel, Assigned numbers, IETF RFC 790 September (1981).

[29] D.S. Reeves, H.F. Salama, A distributed algorithm for delay-constrained unicast routing, IEEE/ACM Transactions on Networking 8 (2) (2000) 239–250.

[30] R. Sivakumar, T.-E. Kim, N. Venkitaraman, J.-R. Li, V. Bharghavan, Achieving per-flow weighted rate fairness in a core stateless network, Proceedings of the International Conference on Distributed Computing Systems (2000) 188–196.

[31] I. Stoica, S. Shenker, H. Zhang, Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks, Proceedings of the ACM SIGCOMM '98 Conference September (1998) 130–188.

[32] I. Stoica, H. Zhang, Providing guaranteed services without per flow management, Proceedings of the ACM SIGCOMM'99 Conference September (1999) 81–94.

[33] Q. Sun and H. Langendoerfer, A new distributed routing algorithm for supporting delay-sensitive applications, Internal Report, Institute of Operating Systems and Computer Networks, TU Braunschweig, Bueltenweg 74/75, 38106 Braunschweig, Germany, March 1997.

[34] R. Sriram, G. Manimaran, C. Siva Ram Murthy, Preferred link based delay-constrained least cost routing in wide area networks, Computer Communications 21 (1998) 1655–1659.

[35] C. Villamizar, MPLS optimized multipath (MPLS–OMP), IETF Internet Draft February (1999).

[36] B.M. Waxman, Routing of multipoint connections, IEEE Journal on Selected Area in Communications 6 (9) (1988) 1617–1722.

[37] R. Widyono, The design and evaluation of routing algorithms for real-time channels, Technical Report TR-94-024, Tenet Group, Department of EECS, University of California, Berkeley, 1994.

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

[39] Martin van der Zee, Quality of service routing-state of the art report, Ericsson Technical Report 1/0362-FCP NB 102 88 Uen (http://searchpdf.adobe.com/proxies/0/9/25/62.html), 1999.