Stateless QoS Routing in IP Networks

Baoxian Zhang* Marwan Krunz** Hussein T. Mouftah* Changjia Chen***

* Department of Electrical & Computer Engineering, Queen's University, Kingston, Ontario K7L 3N6, Canada.

** Department of Electrical & Computer Engineering, University of Arizona, Tucson, AZ 85721.

*** School of Electronics and Information Engineering, Northern Jiaotong University, Beijing 100044, P.R. China.

Abstract: 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. Simple heuristics are proposed 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. Simulations are presented to evaluate the cost performance of the proposed approach.

I INTRODUCTION

A. 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) [1][5]. Several recent studies have acknowledged the need for scalable QoS routing solutions [3][20] and have given the stimulus to a number of proposals on how to integrate QoS routing into a Differentiated Services (DiffServ) framework [7][10].

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. 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. On the other hand, it has been widely recognized that maintaining per-flow state information could lead to scalability problems at core routers. Accordingly, the focus of the research community has shifted to *stateless* QoS frameworks (e.g., DiffServ) that rely on per-class service differentiation at core routers, leaving the maintenance of the per-flow state information to edge routers. A stateless service model seems particularly appealing to router vendors. Thus far, the research on stateless QoS has mainly focused on packet scheduling issues (e.g., [11][16]). 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" [8]. 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. The proposed approach requires each router to maintain routing entries *for each concerned routing metric* (e.g., delay, cost, etc.). It 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. An approach for QoS based packet forwarding is presented in Section 3. Simulation results are presented in Section 4, following by concluding remarks in Section 5. Due to space limitation, proofs are omitted from this paper, and they can be found in an accompanying technical report.

B. Network Model and Problem Formulation

In many practical situations, the goal of QoS routing 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

0-7803-7208-5/01/$17.00 (C) 2001 IEEE

problem is to find a path P from a source node s to a destination node d such that:

1. $D(P) \equiv \sum_{e \in P} D(e) \le \Delta$, and
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 [4]. Several computationally efficient heuristics have been proposed in the literature (e.g., [19][15][17][6]). In [19] the author proposed an algorithm for finding the optimal solution to the DCLC problem, but with worst-case running time that grows exponentially with the size of the network. Two ε-optimal approximation algorithms were proposed by Hassin [6] with running times of $O(\log 1 \log B(|E|(|V|/\varepsilon)+\log\log B))$ and $O(|E|(|V|^2/\varepsilon)\log(|V|/\varepsilon))$, respectively, where B is an upper bound on the optimal cost. 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 have been proposed in [15][17], with worst-case message complexities of $O(|V|^2)$ and $O(|V|)$, respectively. In these heuristics, the path finding process constructs 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.

II PATH SELECTION ALGORITHMS

To achieve simple QoS-based forwarding, we restrict our scope to path selection algorithms in which the computed path consists, at most, of two 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. First, we consider source based path selection heuristics.

A. Source-based Heuristics

Consider the DCLC problem. 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 $D(P_{lc}(s,d)) \le \Delta$ then the optimal solution is given by $P_{lc}(s,d)$. Otherwise, the LD path is chosen provided that $D(P_{ld}(s,d)) \le \Delta$ (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 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, it is easy to show that 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 every node $v \in V$.

Step 4: If $D(P_{lc}(s,d)) \le \Delta$, return the path $P_{lc}(s,d)$. Otherwise, continue.

Step 5: Compute the LD path from every node $v \in V$ to d.

Step 6: Compute the LC path from every node $v \in V$ to d.

Step 7: From the resulting 4|V| paths, $P_i(s,v) \cup P_j(v,d)$ where $i, j \in \{LC, LD\}$ and $v \in V$, choose the one with the smallest cost provided that this path satisfies the delay constraint.

Steps 1 and 3 require two runs of Dijkstra's algorithm, while Steps 5 and 6 require two runs of Reverse Dijkstra's [2]. The running time of Step 7 is $O(|V|)$. Therefore, the overall complexity of Heuristic 2 is $O(|V|^2)$.

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.

Given the above information, the algorithm first verifies that there is a feasible path in the network by checking whether $D(P_{ld}(s,d)) \le \Delta$. 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 Figure 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 $D(P_{lc}(s,v)) + D(P_{ld}(v,d)) \le \Delta$. 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).

```
/* Phase 1 */
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)) ≤ Δ
6.            PATH =: P_lc(s,active_node)∪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 */
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)) ≤ Δ
15.       if C(P_ld(s,active_node)) + C(P_lc(active_node,d)) < C(PATH)
                    /* C(PATH) is obtained from Phase 1 */
16.           PATH =: P_ld(s,active_node)∪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
```

Figure 1: Pseudo-code for Heuristic 3.

B. Probe-Based Distributed Heuristic

The previous heuristics are source based. We now present a distributed heuristic, called *distributed delay-constrained algorithm* (DDCA), which is more suitable for stateless routing. Similar to the previous heuristics, DDCA is also based on the previously discussed relay strategy. In essence, DDCA is an extension of the DCR algorithm [17], which 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 [15][17]. The information in the delay and cost tables can be distributed to nodes using distance (or path) vector protocols.

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 node s up to the current node;
- *cost_so_far*: Accumulated cost of path traversed by the probe message from node s 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)

```
1.   if probe_direction=LD
2.        if delay_so_far + D(P_lc(v,d)) ≤ Δ /* 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)) ≤ Δ /* 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
```

Figure 2: Pseudo-code for DDCA.

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 Figure 2.

In lines 7 and 23 Figure 2, 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. 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.

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). A *probe query* message may visit up to $|V-1|$ nodes. Hence, the worst-case message complexity of DDCA is $O(|V|)$.

Two approaches can be used to encode the probe messages. The first one is to encode these messages as ICMP packets. Currently, intermediate routers do not process the payload portion of an ICMP packet, so a new ICMP packet "type" needs to be defined to allow routers to process probe messages (e.g., insert the IP address of the relay node). The second, and perhaps more viable, approach is to define a new protocol type, call it 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 [14], and one of these values can be used for IPMP.

III DYNAMIC FORWARDING MECHANISM

Once a cost-effective constrained path has been identified using DDCA, or a similar algorithm, the next step is to design an appropriate forwarding mechanism that implements the relay concept in a distributed manner. In here, we present one such mechanism that is based on tunneling and packet encapsulation. 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. 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 numerical value that is encoded, say, in some of the unused codepoints in the DS byte. 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 [12]. It is also possible to use *minimal encapsulation* techniques such as the one defined in [13].

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 forwarding direction field, which is set at the source). Of course, this comes at the expense of some overhead associated with packet encapsulation and tunneling and with sending probe messages.

IV SIMULATION RESULTS

We studied the cost performance of the previously presented heuristics using simulated random topologies that were generated using Waxman's method [18]. 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 performance of the source-based path selection heuristics, which include Heuristic 1, Heuristic 2, Heuristic 3, and LDP (Least Delay Path). We use the optimal (but exponentially complex) CBF algorithm [19] as a point of reference. More precisely, we define the *inefficiency* of an algorithm X as the difference between the cost of the path returned by algorithm X and the cost of the path returned by CBF, normalized to the cost of the path returned by CBF. Figure 3 depicts the performance of various source-based heuristics for an average node degree of 10. 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. We noted that the relative inefficiency in Heuristics 1 and 3 relative to CBF increases with the average degree of a node (the connectivity of the network), while Heuristic 2 seems to be less sensitive to the node degree. 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 [17], DCUR [15], and LDP. As before, the cost of CBF is used as a reference. The relative inefficiency of these algorithms is depicted in Figure 4. 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. 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.

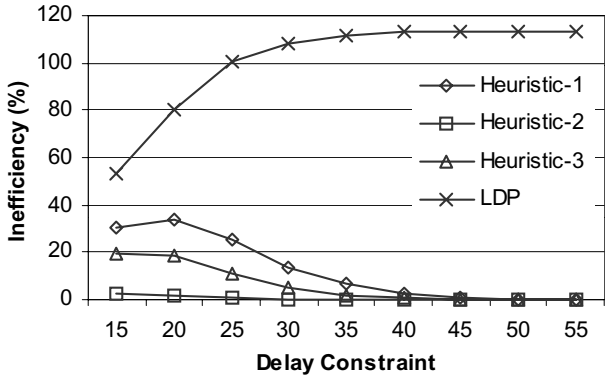

Figure 3: Performance of source path selection algorithms.

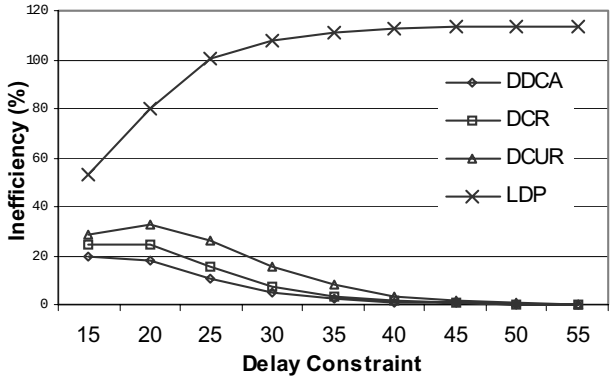

Figure 4: Performance of distributed path selection algorithms.

V 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. To implement the relay strategy in an IP network, we provided an approach that relies on a probing protocol and that uses tunneling and packet encapsulation. Simulation results were presented to evaluate the performance of the proposed heuristics and contrast them with previously proposed solutions.

REFERENCES

[1] G. Apostolopoulos et al., "QoS routing mechanisms and OSPF extensions," IETF RFC 2676, Aug. 1999.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows*: *Theory, algorithms, and applications*. Prentice Hall, Inc., 1993.

[3] S. Chen and K. Nahrstedt, "An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions," *IEEE Network*, Vol. 12, No. 6, pp. 64-79, November-December 1998.

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

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

[6] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, Vol. 17, No.1, pp. 36-42, February 1992.

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

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

[9] J. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, Vol. 14, No. 1, pp. 95-116, 1984.

[10] B. Jamoussi et al., "Constrained-based LSP setup using LDP," IETF Internet Draft, Aug. 1999 (draft-ietf-mpls-cr-ldp-02.txt).

[11] T. Nandagopal, V. Venkitaraman, R. Sivakumar, and V. Bharghavan, "Delay differentiation and adaptation in core stateless networks," in *Proceedings of the IEEE INFOCOM 2000 Conference*, pp. 421-430, March 2000.

[12] C. Perkins, "IP encapsulation within IP," IETF RFC 2003 (Standards Track), Oct. 1996.

[13] C. Perkins, "Minimal encapsulation within IP," IETF RFC 2004 (Standards Track), Oct. 1996

[14] J. Postel, "ASSIGNED NUMBERS," IETF RFC 790, Sep. 1981.

[15] D.S. Reeves and H.F. Salama, "A distributed algorithm for delay-constrained unicast routing," *IEEE/ACM Transactions on Networking*, Vol. 8, No. 2, pp. 239-250, April 2000.

[16] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proceedings of the ACM SIGCOMM '99 Conference*, pp. 81-94, September 1999.

[17] 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.

[18] B.M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Area in Communications*, Vol. 6, No. 9, pp. 1617-1722, December, 1988.

[19] 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 at Berkeley, 1994.

[20] X. Xiao and L. M. Ni, "Internet QoS: A big picture," *IEEE Network*, Vol. 13, No. 2, pp. 8-18, March/April 1999.