A LOAD-SENSITIVE QOS ROUTING ALGORITHM IN BEST-EFFORT ENVIRONMENT

Anirudha Sahoo

IEEE Member
1001 S. Main Street
Milpitas, CA – 95035.

ABSTRACT

The increase in real-time applications such as Voice over IP, audio and video streaming in the public Internet has warranted QoS based routing. But today's Internet largely supports best effort traffic. Thus, a QoS routing in the best effort environment is required to support real-time applications. Flooding of some QoS attributes (e.g. available bandwidth) have been used for QoS routing. But we believe a better way of implementing QoS routing is to localize the QoS routing changes to the region where QoS has deteriorated, instead of flooding the entire network. We present a load sensitive routing (LSR) algorithm that tries to route packets through an alternate path when congestion is reported on a link. LSR routing is contained locally i.e. only the neighboring nodes of a congested node perform LSR routing. Hence it has much less overhead than other QoS routing protocols reported. The LSR algorithm is designed carefully to avoid any looping. We have Simulation results of LSR algorithm which shows its average performance to be better than OSPF algorithm in terms of delay and jitter.

1 INTRODUCTION

Real-time applications such as voice over IP (VoIP), video and audio streaming applications are on the rise in the Internet. Performance of these applications over the Internet is very much dependent on the quality of service (QoS) provided in the Internet for these applications. Thus QoS routing has become very important for integrated services in IP networks. There have been many recent results published on QoS routing [1, 5, 9, 12, 13] for achieving improved network utilization and providing better performance to applications. QoS routing entails additional cost due to computational and protocol overhead. To support QoS, routers have to perform more complex path computation and they are required to send more attributes of state of the network, e.g., available bandwidth, frequently.

The new Internet Service Providers (ISP) are providing very inexpensive voice services over IP in the public Internet. But today's Internet largely supports best effort service. The Internet routing system routes packets based solely on destination address; the QoS requirement of connections are not taken into account. Thus, all packets between a source and destination pair go through the same intermediate path for a given network topology. Hence, if a particular path gets congested, all the flows going through the path encounter bad performance in terms of delay, jitter and packet loss. Clearly, this hampers the performance of real-time applications. Routers in the Internet compute paths based on shortest path algorithms [4]. Although these algorithms are simple and efficient, they cannot route packets along alternate paths, which may provide better QoS during period of congestion in the shortest path. Due to mesh like topologies of the routing backbones the chances of finding such alternate paths are quite high. Furthermore, routers today are much more powerful in terms of computing power and memory. Hence they are capable of implementing more complex routing algorithms and storing more routing information. Thus, modern routers should be able to provide better QoS by routing packets through alternate paths during period of congestion.

There have been quite a few studies reported on QoS routing. In [3] a cheapest path algorithm from one source to all destination when links have two weights (cost and delay) is presented. Several works have analyzed costs associated with QoS routing [7, 11]. Some other solutions use source routing along with shortest path routing to achieve the goal [10]. But security is a major concern for allowing source routing in the Internet. Also, most of the solutions proposed so far use flooding mechanism to update the link states with the available resources [2, 10]. Thus, protocol overhead and convergence are of concern in those cases. We believe that a better way of implementing QoS routing is to confine the routing change information to the region where QoS has deteriorated. This reduces the protocol overhead and convergence time of the algorithm. In this paper, we present a load sensitive routing (LSR) algorithm based on Dijkstra's shortest path algorithm. When load on the outgoing link of a router reaches a certain threshold, then the LSR algorithm will be invoked. This will represent congestion in the router for that outgoing link. The LSR algorithm will try to find an alternate path for the packets that may be transiting through the congested link. Thus, our algorithm will try to provide a better QoS in terms of delay, jitter and packet loss in a best effort routing infrastructure. We acknowledge that LSR algorithm is dependent on topology in the sense that existence of alternate path depends on if a node can find a different next hop node such that the number of hops in the alternate path is less than the OSPF path. We have run simulation of LSR algorithm. We show that LSR algorithm performs better, on an average, than the OSPF [6] algorithm. The congested state of a link is only notified to the neighbors, there is no flooding. Our algorithm is designed carefully to avoid loops. The LSR algorithm exhibits the following advantageous properties:

- *Better average performance*: The LSR algorithm tries to find alternate path to route packets when there is congestion in the OSPF path. Hence, packets get better QoS in terms of delay, jitter and packet loss.
- *Scalability*: Our algorithm does not use flooding mechanism to communicate congestion of a link. Rather the congestion notification is only contained to the neighbors of the node. Thus, our solution can be scaled to a network of any size without much performance hit.
- *No Change in Packet Forwarding Engine*: Our algorithm does not require any change in the logic of packet forwarding engine. Packet forwarding logic remains exactly the same as OSPF router. This makes the implementation easier.

- *Inter-operability with OSPF router*: Our algorithm can be implemented easily with an extension to the framework of OSPF standard [6] by creating a new LSA type. Routers running our algorithm can inter-operate with routers running vanilla OSPF (without our algorithm). When vanilla OSPF routers get this new LSA types, they will simply drop the LSA. Thus, our LSR algorithm can be implemented in the Internet in phases.

2 SYSTEM MODEL

2.1 NETWORK

We model a network consisting of N nodes. A node can either be a host or a router. A node *i* is identified by its id Node(i) ($0 \leq i < N$). Nodes in a network are connected by physical links along which packets can be transmitted. Node(i) and Node(j) are considered neighbors of each other if there is a physical link between them. The physical link between them is identified by the Link(i, j). Cost of transmitting a packet over Link(i, j) is denoted by Cost(i, j).

2.2 ROUTING TABLE

Each node builds a routing table from the network topology. Given a network topology, a node runs Dykstra's shortest path algorithm with itself as the source to find shortest path from itself to all other nodes in the network. We refer to these paths as OSPF path. Then, it builds a routing table so that it can forward packets destined for any node in the network. This routing table is called *active* routing table. In addition, it also runs Dijkstra's algorithm with each of its neighbors as source node and builds similar routing tables. We refer to these routing tables as *passive* routing table. Every entry in the routing table is a sextuple consisting of destination node, next hop node, total cost, Hop count, OSPF next hop and LSR flag. Destination node is the destination of a packet, next hop node is the next hop where a packet destined for destination node should be forwarded. Total cost is the total cost from the node to the destination node along the OSPF path. Hop count is the number of hops in the OSPF path from a node to the destination node. OSPF next hop is the next hop as per OSPF shortest path calculation. LSR flag when TRUE it means this entry in the routing table is forwarding packets according to LSR algorithm. Thus, when this flag is TRUE the next hop node would be the one calculated by the LSR algorithm, otherwise it would be the one found by shortest path algorithm of OSPF. Hence, when an entry in the routing table is routing packets along OSPF nexthop, then next hop node and OSPF next hop will be the same and the LSR flag will be FALSE. Routing table at node Node(i) calculated with Node(j) as source is denoted by RT(i, j). An entry in a routing table RT(i, j) is identified by the destination node of the entry and is denoted by Entry(Node(p), i, j), where Node(p) is the destination node of the entry. If shortest path from Node(i) to Node(j) is through Node(p) and Node(q), then corresponding entry in RT(i, i) is given by

Entry(Node(j), i, i) = {(Node(j), Node(p), TotCost(i, j),
HopCount(i, j), Node(p), FALSE}, (1)

if this entry is forwarding packets through OSPF path. TotCost(i, j) is given by

TotCost(i, j) = Cost(i, p) + Cost(p, q) + Cost(q, j), (2)

The next hop, in this example, is denoted by NextHop(i, j) = Node(p).

At Node(i), active routing table RT(i, i) is used for packet forwarding. Passive routing tables (e.g. RT(i, j) $i \neq j$) are not used for packet forwarding, but only used during execution of the LSR algorithm to find out alternate paths.

2.3 LSR TABLE

Each node also has a LSR table which stores information as to whether the node is getting packets from a previous node due to LSR routing. An entry in the LSR table is a tuple consisting of LSR previous node, Destination Node. LSR table at Node(p) is denoted by LSRTbl(p) and an entry in this table is given by

LSREntry(p, k) = {Node(k), Node(q)}, (3)

where Node(k) is the LSR previous node and Node(q) is the Destination node. This entry signifies that Node(k) is temporarily rerouting packets destined for Node(q) to Node(p) using LSR algorithm. Data in this table is used to prevent loop.

2.4 MESSAGES

We introduce some new messages to implement LSR algorithm. A message is a control packet used to communicate some information from one node to its neighbors to facilitate distributed execution of our algorithm. Following are the new messages needed to implement our algorithm:

- *Congestion Notification*: This message is sent by a node to its neighbors, when it detects congestion in one of its outgoing links. Each node will have a threshold set to detect congestion. Typically this threshold will be set to a value greater than 100% of the link capacity. When the load on an outgoing link is above the set threshold, then a congestion notification message will be sent out by the node to its neighbors. We denote this message by Congestion(i, j) which signifies that a congestion is experienced on the Link(i, j) by node Node(i).
- *Reroute Request*: When a node receives a congestion notification message, it may send out a reroute request to one of its neighbor nodes. When a reroute request message RerouteReq(i, j, k, l) is sent from node Node(k) to node Node(p), then this message is meant to let Node(p) know that Node(k) is going to temporarily reroute packets destined to Node(l) to Node(p) instead of its OSPF neighbor Node(i), because of a congestion reported in Link(i, j).
- *Congestion Over*: When a link is no longer congested, the associated node sends this message to its neighbors to inform about this change of state. We denote this message by CongestionOver(i, j) which signifies that a congestion is no longer experienced on the Link(i, j) by node Node(i).
- *Reroute Over*: This message is sent out by a node when it receives CongestionOver message from its neighbor and if this node was temporarily routing packets using LSR algorithm because of an earlier congestion notification. If Node(k) is temporarily rerouting packets destined to Node(p) to Node(q) because of congestion in Link(i, j) , then Node(k) will send RerouteOver(i, j, k) to Node(q) when it receives CongestionOver(i, j) from its neighbor Node(i).

2.5 ASSUMPTIONS

Our algorithm is based on the following assumptions:

- All the nodes in the network have access to the complete topology of the network i.e. the routing protocol is either OSPF or its derivative. Since majority of Intradomain routing use OSPF or some similar link-state routing protocol this assumption does not pose any major limitation.
- There are fairly good number of alternate paths between any two node pairs so that our algorithm can reroute packets through an alternate path when congestion occurs. This assumption is

reasonable when a real world network such as UUnet backbone is considered [10].

- We assume that the messages specific to LSR algorithm are sent through a reliable channel. Note that this can be achieved by implementing Acknowledgement for each message if an unreliable channel is used.

3 THE LSR ALGORITHM

3.1 FORMAL PRESENTATION

A. When Node(i) detects congestion on Link(i, j) it sends Congestion(i, j) message to all its neighbors except Node(j).

B. When Node(k) receives congestion message Congestion(i, j), it executes the following: (Note that since Congestion message is only sent to neighboring nodes, this means that Node(k) is a neighbor of Node(i))

```
B1  Congestion_Notification (i, j)
B2  {
B3     Let D = {set of all destination nodes in the routing table
         RT(k, k) for which Node(i) is the OSPF next hop node};
B4     Let D' = {Node(p) | (Node(p) ∈ D) and (Node(j) is the
                  OSPF next hop node in the routingtable RT(k, i)
                  for destination node Node(p))}
       /* D' contains all the destinations for which packets
        * forwarded from Node(k) to Node(i) would go out on
        *  congested link Link(i, j) */
B5     for each node Node(p) ∈ D' do {
          /* find all the nodes eligible for LSR forwarding for
           *  destination Node(p) */
B6        R={set of all neighboring nodes of Node(k)}–{Node(i)};
B7        Q = {Φ};
B8        For each node Node(q) ∈ R do {
B9           if (HopCount(q, p) < HopCount(k, p)) {
B10             Q = Q + {Node(q)};
B11          }
B12       } /* end of for each node Node(q) */
B13       while ( Q != {Φ} ) do {
B14          Node(r) = a randomly selected node in the set Q;
B15          If (LSREntry(k, r) == {Node(r), Node(p)}) {
          /* Node(r) is sending LSR packets destined to Node(p) to
           * this node Node(k), so do not send packets for
           * destination Node(p) to Node(r) to avoid looping.*/
B16             Q = Q – {Node(r)};
             }
B17          else break;
B18       } /* of while */
B19       if (Q == {Φ}) continue;
B20       Send RerouteRequest(i, j, k, p) message to Node(r);
B21       Set next hop node of Entry(Node(p), k, k) to Node(r);
B22       Set the LSR flag of Entry(Node(p), k, k) to TRUE;
B23    } /* for each node Node(p) */
B24 } /* Congestion_Notification() */
```

C. When Node(r) receives RerouteRequest(i, j, k, p) from Node(k) it executes the following:

```
C1 Process_RerouteReq (i, j, k, p)
C2 {
C3    if ( (next hop of Entry(Node(p), r, r) is Node(k) )  &&
C4          (LSR flag of Entry(Node(p), r, r) is  TRUE)) {
           /* Node(r) is temporarily routing packets destined for
            *Node(p) to Node(k) due to LSR algorithm, a direct
            * loop detected */
C5          if ( Node(r) > Node(k) ) {
               /* Fall back to OSPF routing */
C6             set next hop of Entry(Node(x), r, r) to OSPF next
               hop of Entry(Node(x), r, r);
C7             set LSR flag of Entry(Node(x), r, r) to FALSE;
C8          }
C9    }
C10    set LSREntry(r, k) = {Node(k), Node(p)};
C 11 } /* Process_RerouteReq() */
```

D. When Node(i) detects that congestion is over on Link(i, j), then it does the following:

```
D1  Detect_Congestion_Over(i, j)
D2  {
D3     Let S = {set of all neighbors of Node(i)} –
                                          {Node(j)};
D4     For every node Node(p) ∈ S do {
D5        Send CongestionOver(i, j) to Node(p);
D6     }
D7  }
```

E. When Node(k) receives CongestionOver(i, j), it does the following:

```
E1  Process_Congestion_Over (i, j)
E2  {
E3    Let D = {set of all destination nodes in the routing table
      RT(k, k) for which Node(i) is the OSPF next hop node};
E4    Let D' = {Node(p) | (Node(p) ∈ D) and (Node(j) is the
        OSPF next hop node in the routing table RT(k, i) for
        destination node Node(p))}
       /* D' contains all the destinations for which packets
        * forwarded from Node(k) to Node(i) would go out on
        * congested link Link(i, j) */
E5     for each node Node(p) ∈ D' do {
E6        if ( LSR flag of Entry(Node(p), k, k) is TRUE ) {
             /* This entry in RT(k,k) is doing LSR routing, reset it
              *  back to OSPF routing */
E7           Node(r) = next hop node of Entry(Node(p), k, k);
E8           Set next hop node of Entry(Node(p), k, k) equal to
             OSPF next hop;
E9           Send RerouteOver(i, j, k, p) to Node(r);
E10       }  /* if */
E11    } /* for */
E12 }
```

F. When Node(r) receives RerouteOver(i, j, k, p) from Node(k), it deletes entry LSREntry(r, k).

3.2 LOOP-FREE PROPERTY

The LSR algorithm does not use flooding to calculate an alternate path for a destination. Rather a congested node sends the congestion notification to all its neighboring nodes and the notification stops there. The neighboring node may change the next hop of packets going through the congested link by applying LSR algorithm. Thus, when some nodes are routing packets using LSR algorithm, other nodes in the network will have a different view of the routing topology of the network due to the local nature of the LSR algorithm. Thus, making LSR algorithm loop free is of utmost importance. In this section we will provide a formal proof that the LSR algorithm does not introduce looping of packets.

Theorem 1 The LSR algorithm does not give rise to loop.

Proof: There can be two kinds of looping that can happen: direct looping and indirect looping. Direct looping happens when a node Node(p) sends packets with destination, say Node(r), to Node(q) and Node(q) sends packets to Node(p) for the same destination. In this case, there is a direct loop between Node(p) and Node(q). An indirect loop, on the other hand, involves at least one intermediate node between two nodes to form a loop. As an example, for a particular destination node Node(r), Node(p) may forward packets to Node(x), Node(x) forwards to Node(q) and Node(q) forwards to Node(p).

Let us take the case of direct loop. There are two cases to consider for direct loop. First, when a node Node(p) has already received RerouteRequest from previous node Node(q) and then it tries to reroute packets to Node(q) due to LSR algorithm. Condition B15 in the algorithms identifies this case and prevents direct loop. Second, it is possible that a node Node(p) is already forwarding packets to Node(q) for a particular destination using LSR routing, and Node(q) is also forwarding packets to Node(p) for the same destination due to LSR routing. But the two nodes have not yet received RerouteRequest message from each other. But this looping is only short lived until the node with higher node id comes out of LSR forwarding due to condition C5 in the algorithm.

Now let us take the case of indirect loop. We prove this by contradiction. Let us say that there is an intermediate node Node(x) between Node(p) and Node(q) through which there is a loop for packets going to a particular destination Node(r). That is Node(p) is forwarding packets to Node(x) and Node(x) is forwarding packets to Node(q) and Node(q) is forwarding packets to Node(p).

There can be eight combinations of packet forwarding between these three nodes, since each could do OSPF forwarding or LSR forwarding. This is illustrated in the table below

Case #	Node(p)	Node(x)	Node(q)
1	OSPF	OSPF	OSPF
2	OSPF	OSPF	LSR
3	OSPF	LSR	OSPF
4	OSPF	LSR	LSR
5	LSR	OSPF	OSPF
6	LSR	OSPF	LSR
7	LSR	LSR	OSPF
8	LSR	LSR	LSR

Case 1 : All the nodes are forwarding packets by OSPF next hop. Since OSPF routing does not have loops, this looping scenario is not possible.

Case 2 : Node(q) is forwarding packets to Node(p) using LSR, hence condition B9 of the LSR algorithm should be satisfied i.e.

$$HopCount(p, r) < HopCount(q, r) \quad (4)$$

Since Node(p) is forwarding packets to Node(x) using OSPF,

$$HopCount(p, r) = HopCount(x, r) + 1 \quad (5)$$

Since Node(x) is forwarding packets to Node(q) using OSPF,

$$HopCount(x, r) = HopCount(q, r) + 1 \quad (6)$$

Using (5) and (6) in (4) we get

$$HopCount(q, r) + 2 < HopCount(q,r) \quad (7)$$

Since (7) cannot be true, this indirect loop is not possible.

Case 3: This is similar to case 2 and the same proof applies by replacing Node(q) by Node(x).

Case 4: Since Node(x) is forwarding to Node(q) using LSR, using condition B9 of LSR algorithm, we have

$$HopCount(q, r) < HopCount(x, r) \quad (8)$$

and since Node(q) is forwarding to Node(p) using LSR, using condition B9 of LSR algorithm, we have

$$HopCount(p, r) < HopCount(q, r) \quad (9)$$

From (8) and (9),

$$HopCount(p, r) < HopCount(x, r) \quad (10)$$

But since Node(p) is forwarding to Node(x) using OSPF,

$$HopCount(p, r) = HopCount(x, r) + 1 \quad (11)$$

Since (10) and (11) are contradicting each other, so such a loop is not possible.

Case 5 is similar to case 3, Case 6 is similar to Case 4 and Case 7 is similar to case 4.

Case 8: Since Node(p) is LSR forwarding to Node(x), using condition B9 of LSR algorithm, we have

$$HopCount(x, r) < HopCount(p, r) \quad (12)$$

Since Node(x) is LSR forwarding to Node(q), using condition B9 of LSR algorithm, we have

$$HopCount(q, r) < HopCount(x, r) \quad (13)$$

Since Node(q) is LSR forwarding to Node(p), using condition B9 of LSR algorithm, we have

$$HopCount(p, r) < HopCount(q, r) \quad (14)$$

But combining (12) and (13) we get

$$HopCount(q, r) < HopCount(p, r) \quad (15)$$

Since (14) and (15) are contradicting each other, a loop is not possible. Thus, LSR algorithm does not give rise to loop. Q.E.D.

4 PERFORMANCE EVALUATION

In this section, we report the performance of LSR algorithm. We compare its performance against OSPF algorithm. We have simulated a network of nodes running LSR and OSPF algorithm to route packets and measured various performance parameters.

4.1 TOPOLOGY

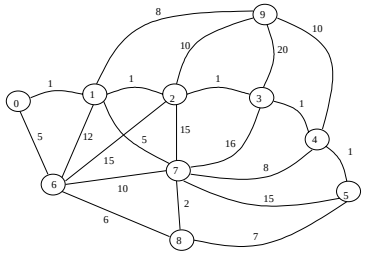

Figure 1 Topology of the Simulation Network

The topology of our simulation is shown in Figure 1. There are ten nodes in the network. Cost of direct paths between nodes are shown in the figure. Utilization of every link is varied uniformly from 0% to a maximum load (L_{max}) (in percentage) and average delay (d_{avg}) and jitter (J_{avg}) are measured between node 0 and node 5. The link from node 3 to 4 is kept under scrutiny i.e. reading of performance metrics are taken when this link experienced congestion. The congestion threshold for this link is set at 100%. The duration for which utilization of a link remains in effect is exponentially distributed. The mean duration of this distribution is denoted by t_{mean}.

4.2 RESULTS

Figure 2 shows the d_{avg} versus L_{max} when t_{mean} is 50msec. It is clear from the plot that the average delay for LSR algorithm is

much better than that for OSPF. When t_{mean} is 100msec, LSR algorithm also performs better in terms of average delay as shown in Figure 3. In both the cases, delay of OSPF routing is at least 50% more than that of LSR algorithm for all Load values. Figure 4 and Figure 5 compares average jitter for the two algorithms when t_{mean} is 50msec and 100msec respectively. In these cases also the LSR algorithm performs much better than OSPF. So, on an average, LSR algorithm performs better than OSPF algorithm in terms of delay and jitter.

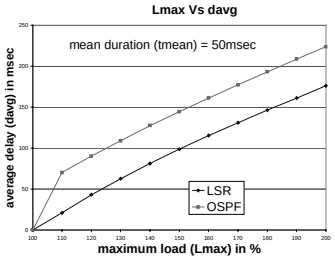

Figure 2

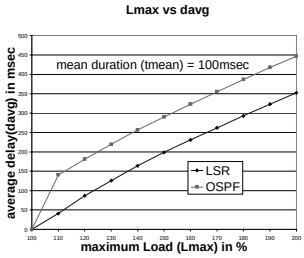

Figure 3

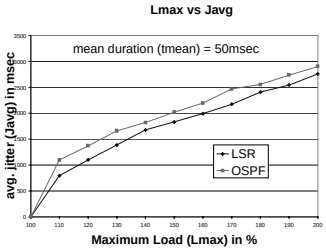

Figure 4

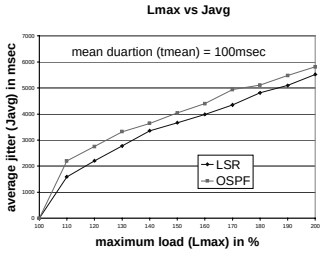

Figure 5

5 CONCLUSION AND FUTURE WORK

We have introduced a Load Sensitive Routing (LSR) algorithm based on Dijkstra's shortest path algorithm. The LSR algorithm does not use flooding to notify congestion over a link, rather the notification is limited only to the neighboring nodes. Thus, this algorithm uses less network resources than that proposed in OSPF extension for QoS support [2], since the latter uses flooding. Only overhead added for LSR algorithm is to calculate passive routing table of all the neighboring nodes, store those tables and send LSR messages to the neighboring nodes. Since modern routers have abundant memory and more computational power, this overhead should be easy to accommodate. Moreover, the computation of new active and passive routing table would only happen when topology of the network changes. The LSR forwarding is transparent to the forwarding engine, i.e., forwarding engine does not know whether it is sending packet along OSPF next hop or LSR next hop. Another advantage of LSR algorithm is that it can easily be implemented in the framework of OSPF by using a new LSA type. Also, nodes running LSR algorithm can inter-operate with nodes running vanilla OSPF. Thus, LSR nodes can be deployed in phases in the Internet.

In this study, we have chosen LSR next hop node randomly from all the eligible nodes. Other intelligent methods of picking next hop node can be studied. For example, a next hop node which has the fastest links in the alternate path or which has least utilization can be chosen. Our LSR algorithm statically selects the LSR next hop without taking the current condition of the alternate path into consideration. Further study can be done to see how LSR next hop can be chosen with some dynamic state information such as current delay along the alternate path to improve its performance. In this case, if an alternate path is found to have worse QoS than the OSPF path, then the OSPF next hop should be maintained in the routing table as the next hop. Route flapping is another aspect of this algorithm that can be studied in the future. When packets are routed through alternate paths due to LSR algorithm, the original congested path may soon overcome congestion and ask the neighbors to forward packets through the original (OSPF) path. This may lead to congestion of the original path again. This would give rise to route flapping. A route flap damping mechanism which could be similar to that described in [8] needs to be studied for LSR algorithm.

6 REFERENCES

[1] G. Apostolopoulos, R. Gu'erin, and S. Kamat, "Implementation and Performance Measurements of QoS Routing Extensions to OSPF." Proceedings of IEEEINFOCOM'99, New York, April 1999.

[2] G. Apostolopoulos, A. Orda, D. Williams, R. Gu'erin, T. Przygienda, and S. Kamat, "QoS Routing Mechanisms and OSPF Extensions." Internet Request for Comments, RFC 2676, August, 1999.

[3] A. Goel, K. G. Ramakrishnan, D. Katatria, D. Logothetis, "Efficient Computation of Delay-sensitive Routes from One Source to All Destinations." Proceeding of IEEE Infocom 2001.

[4] C. Huitema, "Routing in the Internet." Englewood Cliffs, New Jersey, Prentice Hall PTR, 1995.

[5] W. C. Lee, M.G. Hluchyj, and P. A. Humblet, "Routing Subject to Quality of Service Constraints in Integrated Communication Networks." IEEE Networks, pp. 46-55, July/August 1995.

[6] J. Moy, "OSPF Version 2, Internet Request for Comments." RFC 2178, July 1997.

[7] Q. Ma and P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees." Proceedings of IEEE International Conference on Network Protocols (ICNP), Atlanta, Georgia, October 1997.

[8] C. Villamizar, R. Chandra, R. Govindan, "BGP Route Flap Damping." RFC2439, November 1998.

[9] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, A. Przygienda, and D. Williams, "QoS routing mechanisms and OSPF extensions." Internet Request for Comments (RFC 2676), April 1999.

[10] A. Segall, P. Bhagwat, A. Krishna, "QoS routing using alternate paths." Journal of High Speed Networks, 7(2): pages 141-158, 1998.

[11] A. Shaikh, J. Rexford, and K. Shin, "Efficient precomputation of quality-of-service routes." Proceedings of Workshop on Network and Operating Systems Support for Digital Audio and Video, July 1998.

[12] Z. Wang, and J. Crowcroft, "Quality of Service Routing for Supporting Multimedia Applications." IEEE Journal Selected Areas in Communications, 14(7):1228-1234, 1996.

[13] R. Widyonon, "The Design and Evaluation of Routing Algorithms for real-time channels." Technical Report TR-94-024, University of California at Berkeley, June 1994.