# An OSPF Based Load Sensitive QoS Routing Algorithm using Alternate Paths

Anirudha Sahoo
IEEE Member
1001 S. Main Street,
Milpitas, CA-95035

*Abstract*- **Real-time applications such as Voice over IP, audio and video streaming require Quality of Service (QoS). Such applications are being executed over the public Internet. Since today's Internet largely supports best effort traffic, QoS routing in the best effort environment is required to support real-time applications. Some QoS routing use source routing and others use flooding of some QoS attibutes of the nodes. There were also some variants of shortest path algorithm reported in the literature. But those algorithms require changes to packet forwarding engine and logic for loop detection. We believe a better way of implementing QoS routing is to localize the QoS routing changes to the *region* where QoS has deteriorated and choose loop-free alternate paths. We present such an algorithm based on OSPF called LSR algorithm. In LSR algorithm, congestion notification is limited to neighbors of the congested node and the neighbors try to use alternate next hops to route packets. Alternate LSR next hop is chosen in such a way that it preserves the next hop property of OSPF routing which enables LSR algorithm to avoid loop. We present three such methods to choose an alternate LSR next hop and prove that these methods provide loop free routing. Our simulation results based on the three methods show that on an average LSR algorithm performs better than OSPF algorithm in terms of delay and jitter.**

## I. INTRODUCTION

Of late, more and more real-time applications such as Voice over IP, audio and video streaming are being executed over the public Internet. These applications require Quality of Service (QoS) from the Internet to have satisfactory performance. Thus QoS routing has become very important for integrated services in IP networks. There have been many recent results published on QoS routing [1, 6, 10, 15, 16] for achieving improved network utilization and providing better performance to applications. Despite these benefits, feasibility of implementing QoS routing protocols in IP networks is still an uncertainty [1]. Supporting QoS routing implies additional costs in terms of computational and protocol overhead. 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.

There has been an upsurge of next generation Internet Service Providers who are building integrated voice and data network based on IP. So they are providing inexpensive voice services over the public Internet. Since public Internet mostly support *best effort* traffic, there is no QoS provided to real-time applications such as voice over IP. The routing infrastructure is based on shortest path algorithm [5] and is based solely on destination address. Hence, when a packet experiences congestion, the routing subsystem cannot send it through an alternate path, even though there may be an alternate path available which can provide better QoS. If routing subsystem can temporarily divert packets through alternate paths, applications can get better QoS. Since modern routers have more powerful CPU and more memory, it should be able to implement this alternate path routing algorithm and store more information required for that.

QoS routing has been studied for a while. In [4] a cheapest path algorithm from one source to all destinations when links have two weights (cost and delay) is presented. Several works have analyzed costs associated with QoS routing [8, 13]. Some other solutions use source routing along with shortest path routing to achieve the goal [12]. 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, 12]. Thus, protocol overhead and convergence are of concern in those cases. Routing on alternate paths based on shortest path first method also exists [14]. But the drawbacks of this algorithm is that the packet forwarding engine needs to be more intelligent and a loop detection mechanism is needed in the algorithm. 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. We reported the first version of the LSR algorithm in [11]. But in this paper we have improved the LSR algorithm with three different methods of finding alternate paths. 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 next hop 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. The strength of our algorithm lies with the method used for finding alternate next hop. The method is based on the next hop properties of OSPF next hop. Hence our method prevents loops. We have shown three methods of finding alternate next hops and have given a formal proof to show that our algorithm does not give rise to loops. We have implemented the LSR algorithm in a simulation environment of a network. We present some performance comparison between the LSR algorithm and Open Shortest Path First (OSPF) [7] which is based on Dijkstra's algorithm. We show that our algorithm performs better, on an average, than the OSPF algorithm. Following are the advantages of our LSR algorithm:

- *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.
- *Less overhead and 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, it has less overhead and it can scale easily to large networks.
- *Coexistence with OSPF router*: Our algorithm can be implemented easily with an extension to the framework of OSPF standard [7] by creating a new LSA type. Routers running our algorithm can coexist 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.

- *Loop free property*: Since alternate next hop is chosen based on next hop property of OSPF routing, LSR algorithm is loop-free. Hence there is no need for loop detection mechanism.

## II. SYSTEM MODEL

### A. Network

We model a network consisting of *N* nodes. A node *i* is identified by its id Node(i) ($0 \le i < N$). 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).

### B. 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, ospf 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. Ospf cost (OC) is the cost from the node to the destination node along the OSPF path. Hop count (HC) 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 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), OC(i, j), HC(i, j), Node(p), FALSE}, (1)

when this entry is not forwarding packets through alternate path due to LSR algorithm. OC(i, j) is then given by

$$OC(i, j) = Cost(i, p) + Cost(p, q) + Cost(q, j). \quad (2)$$

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.

### C. 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

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

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 to Node(q) to Node(p) using LSR algorithm. Information stored in this table is used to prevent loop.

### D. Messages

We introduce some control messages to implement our algorithm. It is 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 out RerouteOver(i, j, k) to Node(q) when it receives CongestionOver(i, j) from its neighbor Node(i).

## III. THE LSR ALGORITHM

### A. OSPF NextHop Properties

As mentioned earlier, eligible nodes for LSR routing is chosen such a way that packets will not end up in a loop due to LSR routing. This loop-free property of LSR algorithm is designed based on the properties of ospf next hop. If Node(q) is the ospf next hop of Node(p) for destination node Node(r) then we have the following two properties

*Property 1* : For a given destination, number of hops from next hop is less than the number of hops from the current node i.e.

$$HC(q, r) < HC(p, r). \quad (4)$$

*Property 2* : For a given destination, ospf cost of next hop is less than the ospf cost of the current node i.e.

$$OC(q, r) < OC(p,r). \quad (5)$$

From inequality (4), we have

$$a * HC(q, r) < a * HC(p, r), \quad (6)$$

where $a > 0$. From inequality (5), we have

$$b * OC(q, r) < b * OC(p,r), \quad (7)$$

where $b > 0$. Thus, combining (6) and (7), we have

$$a * HC(q, r) + b * OC(q, r) < a * HC(p, r) + b * OC(p,r), \quad (8)$$

where $a \ge 0$ and $b \ge 0$ and $(a, b) \neq (0, 0)$.

Notation $(a, b) \neq (0, 0)$ means both a and b cannot be 0. In (8) a and b can be 0 separately, but they cannot be 0 at the same time. Inequality (8) is the next hop property of OSPF next hop and we will

use the same inequality for choosing LSR next hop. When a node looks for an LSR next hop for a particular destination, it should know the HC and OC values for all its neighbors. Since our algorithm is based on OSPF, every node has knowledge of entire topology of the network. Hence this information should be available to all the nodes. A neighbor will be considered an *eligible* LSR next hop if inequality (8) holds and if it is not the OSPF next hop. However, the nodes in a network has to decide what should be the values of *a* and *b*. We will refer to these two parameters as the *LSR coefficient pair* and will be represented by an ordered pair (a, b). For a particular destination node all the nodes in the network will use the same LSR coefficient pair and for each destination there will be a separate LSR coefficient pair. This condition ensures that there are no routing loops in the network.

### *B. Calculation of LSR Coefficient Pair*

In this section we provide some methods of calculating LSR coefficient pair.

*Method 1*: One trivial value for LSR coefficient pair is (1, 0), that is a=1, b=0. Basically, this method will find alternate route based on Property 1 i.e. based on hop count only. Obviously, this method may not always find the maximum number of alternate paths. We will refer to this method as *LSR_a* method.

*Method 2*: Another trivial value for LSR coefficient pair is (0, 1), that is a=0, b=1. Basically, this method will find alternate route based on Property 2 i.e. based on ospf cost only. This method also may not always find the maximum number of alternate paths. We will refer to this method as *LSR_b* method.

*Method 3*: This method is more involved than the previous two trivial methods. In this method, we try to find the value pair (a, b) such that the number of alternate paths (for a particular destination) is maximized. We will refer to this method as *LSR_ab* method.

### *C. Calculation of LSR Coefficient Pair for LSR_ab Method*

In this section, we will outline the method to calculate LSR coefficient pair for LSR_ab method. We need the following two theorems for that :

***Theorem 1*** Let $x_i$ and $y_i$ be hop count and ospf cost of a Node(i) respectively. Let $x_j$ and $y_j$ be the corresponding values for its neighbor Node(j) and that Node(i) is being considered as an alternate LSR next hop at Node(j). Then Node(i) should be rejected as LSR next hop of Node(j) if any of the following condition is true:

Condition 1 : $x_i > x_j$ and $y_i > y_j$, Condition 2 : $x_i = x_j$ and $y_i > y_j$

Condition 3 : $x_i > x_j$ and $y_i = y_j$, Condition 4 : if $x_i = x_j$ and $y_i = y_j$.

***Proof*** : According to inequality (8) Node(i) will be accepted as alternate LSR next hop if

$$a * x_i + b * y_i < a * x_j + b * y_j, \quad \text{(A1)}$$

where $a \geq 0$ and $b \geq 0$ and $(a, b) \neq 0$, i.e.

$$a * (x_i - x_j) + b * (y_i - y_j) < 0. \quad \text{(A2)}$$

If Condition 1 is true, then $(x_i - x_j) > 0$ and $(y_i - y_j) > 0$. Since $a \geq 0$ and $b \geq 0$ and $(a, b) \neq 0$, (A2) cannot be true in this case. Hence if Condition 1 is true, then Node(i) should be rejected as LSR next hop.

If Condition 2 is true, then putting $x_i = x_j$ from this condition in (A2)

$$b * (y_i - y_j) < 0. \quad \text{(A3)}$$

But $b \geq 0$ and for Condition 2, $(y_i - y_j) > 0$. Hence (A3) can never hold for Condition 2. Hence if Condition 2 is true, then Node(i) should be rejected.

If Condition 3 is true then applying this condition to (A2) we have

$$a * (x_i - x_j) < 0. \quad \text{(A4)}$$

But $a \geq 0$ and for Condition 3, $(x_i - x_j) > 0$. Hence (A4) can never hold for Condition 3. Hence if Condition 3 is true, then Node(i) should be rejected.

If Condition 4 is true, then putting these equalities in (A2), we verify that (A2) cannot hold for this condition. Hence in this case, Node(i) should be rejected. Q. E. D.

***Theorem 2*** Let $x_i$ and $y_i$ be hop count and ospf cost of a Node(i) respectively. Let $x_j$ and $y_j$ be the corresponding values for its neighbor Node(j) and that Node(i) is being considered as an alternate LSR next hop at Node(j). If Node(i) is not rejected as LSR next hop due to Theorem 1, then Node(i) can be chosen as LSR next hop as follows :

Case 1 : if $x_i < x_j$ and $y_i \leq y_j$ then Node(i) can be accepted as LSR next hop if

$$a > 0 \text{ and } b \geq 0. \quad \text{(9)}$$

Case 2 : if $x_i < x_j$ and $y_i > y_j$ then Node(i) can be accepted as LSR next hop if

$$b < p * a, \quad \text{(10)}$$

where $$p = (x_j - x_i) / (y_i - y_j). \quad \text{(11)}$$

Case 3 : if $x_i \geq x_j$ and $y_i < y_j$ then Node(i) can be accepted as LSR next hop if

$$b > q * a, \quad \text{(12)}$$

where $q = (x_i - x_j) / (y_i - y_j)$. (13)

***Proof*** : From (A1) we have

$$a * (x_j - x_i) + b * (y_j - y_i) > 0. \quad \text{(A5)}$$

Case 1 : if $x_i < x_j$ and $y_i \leq y_j$ then $(x_j - x_i) > 0$ and $(y_j - y_i) \geq 0$. Thus, (A5) will hold for any value of a and b, $a > 0$, $b \geq 0$. Hence for this case, Node(i) will be accepted as LSR next hop if $a > 0$, $b \geq 0$.

Case 2 : From (A5) we have,

$$b < a * (x_j - x_i) / (y_i - y_j), \quad \text{(A6)}$$

that is $b < p * a$, where $p = (x_j - x_i) / (y_i - y_j)$.

Since in this case, $x_i < x_j$ and $y_i > y_j$, $p > 0$. Thus, for this case Node(i) will be accepted as LSR next hop if $b < p * a$.

Case 3 : From (A5) we have,

$$b > a * (x_i - x_j) / (y_j - y_i), \quad \text{(A7)}$$

that is $b > q * a$, where $q = (x_i - x_j) / (y_j - y_i)$.

Since in this case, $x_i \geq x_j$ and $y_i < y_j$, $q \geq 0$. Thus, for this case Node(i) will be accepted as LSR next hop if $b > q * a$. Q.E.D.

Note that Theorem 1 and Theorem 2 cover all the possible combinations of inequalities between $(x_i, x_j)$ and $(y_i, y_j)$. Without loss of generality we can take the value of a=1 for all destinations. Then the task is to determine the value of b such that number of alternate path (AP) is maximized. Case 1 of theorem 2 will be taken care of as long as $b \geq 0$. Hence we only concentrate on Case 2 and Case 3 and make sure that $b \geq 0$. For a particular destination, when a node wants to find an LSR next hop, it considers all neighbors, except the ospf next hop for LSR next hop. If a neighboring node satisfies any of the conditions outlined in Theorem 1 then the node is rejected. Otherwise, cases in Theorem 2 are tried. If it falls under Case 1, then it is not considered (for determining value of b). If it falls under Case 2, then it is marked as *p-node*. If it falls under Case 3, it is marked as *q-node*. This procedure is repeated for every node in the network except for the destination node. Thus, for a particular destination p-nodes and q-nodes in the entire network are identified. Let there be *m* number of p-nodes and *n* number of q-nodes. Let the *p* values of the p-nodes be sorted in ascending order and also the *q* values of q-nodes be sorted in ascending order i.e.

$$p_1 \leq p_2 \ldots . \leq p_m \quad \text{and} \quad q_1 \leq q_2 \ldots . \leq q_n.$$

Remember that we have chosen a=1. The value of b will decide which of these p-nodes and q-nodes will become *eligible* LSR nodes. Number of eligible LSR nodes is a measure of total Alternate Paths (AP) available for the destination. Now b is determined depending on what is the possible scenario :

*Scenario 1* : m =0 and n = 0, in this case choosing any value of b will not change the number of alternate paths. So we choose b = 1.

*Scenario 2* : m > 0 and n = 0. Then the number of Alternate Paths (AP) will be maximized if $b < p_1$, since b will be less than all the 'p' values and according to (10) all p-nodes can be *eligible* LSR nodes.

*Scenario 3* : m = 0 and n > 0. Then according to (12) the number of AP will be maximized if $b > q_n$.

*Scenario 4* : m > 0 and n > 0. In this case, it is not trivial to determine the value of b so as to maximize the number of APs (i.e. total number of eligible p-nodes and q-nodes is maximized) . For this scenario, there can be different cases as follows:

**Case 1** : In this case $q_n < p_1$. Obviously, the number of APs will be maxmimum if

$q_n < b < p_1$.

**Case 2** : In this case $p_m < q_1$. b is chosen as follows

$b < p_1$ if $m \geq n$,

$b > q_n$ if $n > m$.

**Case 3** : For all other cases the following algorithm should be followed. Basically, this algorithm checks each p-node and q-node to see which one gives rise to the maximum number of alternate paths. Then it chooses the value of b accordingly.

```
complex_method_calc_b()  {
    alt_path = 0;
    for all points p_k in p-node family  do {
        Let  z = p_k; alt_path_p = 0;
        for all q_i for which z > q_i do {
            alt_path_p++; /* add this as an eligible node */
        } for all p_j for which z ≤ p_j  do {
            alt_path_p++;    /* add this as an eligible node */
        } if (alt_path_p > alt_path) {
            alt_path = alt_path_p;  node_type = 'P';node_indx = k;
        }
    } for all points q_k in q-node family  do {
        Let  z = p_k; alt_path_q = 0;
        for all q_i for which z ≥ q_i do {
            alt_path_q++; /* add this as an eligible node */
        } for all p_j for which z < p_j  do {
            alt_path_q++; /* add this as an eligible node */
        } if (alt_path_q > alt_path) {
            alt_path = alt_path_q; node_type = 'Q'; node_indx = k;
        }
    }
/* Now 'alt_path' contains maximum number of alternate paths */
    if (node_type == 'P') {
        find the largest q_i such that q_i < p_node-index.
        If such a q_i exists {
            find the largest p_j  such that p_j < p_node-index.
            If such a p_j exists {
                If (q_i ≥ p_j) choose 'b' such that q_i < b < p_node-index.
                else choose 'b' such that p_j < b < p_node-index.
            }else choose 'b' such that q_i < b < p_node-index.
        } else { /* such a q_i does not exist */
            find the largest p_j  such that p_j < p_node-index.
            If such a p_j exists choose 'b' such that  p_j < b < p_node-index.
            else choose 'b' such that  0 < b < p_node-index.
        } /* else of If such a q_i exists */
    }  else {  /* node_type == 'Q' */
        find the smallest p_i such that  p_i > q_node-index.
        If such a p_i exists {
            find the smallest q_j  such that q_j > q_node-index.
            If such a q_j exists {
                If (q_j ≥ p_i) choose 'b' such that p_i > b > q_node-index.
                else choose 'b' such that q_j > b > q_node-index.
            } else  choose 'b' such that p_i > b > q_node-index.
        } else { /* such a p_i does not exist */
            find the smallest q_j  such that q_j > q_node-index.
            If such a q_j exists choose 'b' such that  q_j > b > q_node-index.
            else choose 'b' such that  b > q_node-index.
        }
    }
}
```

While calculating 'b' we assumed a=1 for LSR_ab method. So it covers the case of (1, 0). But it does not cover the case of (0, 1). Hence number of alternate paths is found using LSR coefficient pair (0, 1). Whichever pair gives maximum number of alternate paths is accepted as the final LSR coefficient pair.

### *D. The LSR Algorithm*

**A.** When Node(i) detects congestion on Link(i, j) it sends Congestion(i, j) message to all its neighbor except Node(j) and sets up LSR routing for all destinations for which ospf next hop is 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 theOSPF next hop node};
B4       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) */
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 ( inequality (8) holds ) {
                    /* substitute p for r, k  for p and q for q  in (8) */
B10               Q=Q+{Node(q)}; /* Node(q) is an eligible LSR node */
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 the 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)};
C11 } /* Process_RerouteReq() */
```

**D.** When Node(i) detects that congestion is over on Link(i, j), then it sends CongestionOver(i, j) to all its neighbors except Node(j).

**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).

### *E. 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 3*** The LSR algorithm does not give rise to looping.

***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).

First we 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 |

**Table 1**

Remember that LSR next hop is chosen such that inequality (8) is satisfied.

*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 inequality (8) should be satisfied i.e.

$$a * HC(p, r) + b * OC(p, r) < a * HC(q, r) + b * OC(q, r) \quad (14)$$

Since Node(p) is forwarding packets to Node(x) using OSPF, using inequality (8)

$$a * HC(x, r) + b * OC(x, r) < a * HC(p, r) + b * OC(p, r) \quad (15)$$

And since Node(x) is forwarding packets to Node(q) using OSPF, using inequality (8)

$$a * HC(q, r) + b * OC(q, r) < a * HC(x, r) + b * OC(x, r) \quad (16)$$

Using (14) in (15) we get

$$a * HC(x, r) + b * OC(x, r) < a * HC(q, r) + b * OC(q, r) \quad (17)$$

Inequality (17) contradicts (16), so this indirect loop is not possible. Proof for all other cases are the same, since OSPF next hop and LSR next hop are based on the same property (inequality (8)). The same proof can be extended to multiple number of intermediate nodes. Thus, LSR algorithm does not give rise to loop. Q.E.D.

## IV. PERFORMANCE EVALUATION

In this section, we evaluate the performance of LSR algorithm. We compare performance of LSR algorithm when it uses different methods of calculating LSR coefficients and also against OSPF

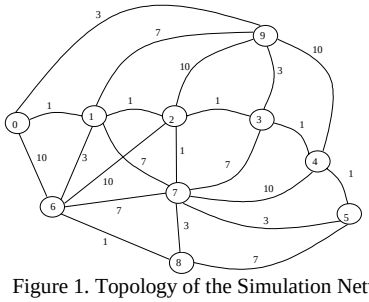


Figure 1. Topology of the Simulation Network.

algorithm. We have simulated a network of nodes running LSR and OSPF algorithm to route packets and measured various performance parameters. The simulation is written in C and run on a SUN/Solaris

environment.

*A. Topology*

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. Cost of the links are assigned using the rule given in [3] as follows.

$$\text{cost} = \lceil (100000000/\text{bandwith in bps}) \rceil \quad (18)$$

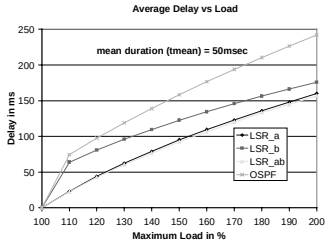


Figure 2.

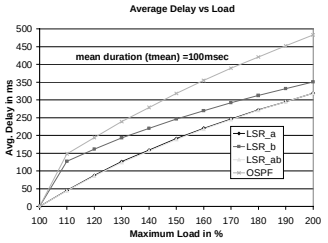


Figure 3.

We have assumed FDDI (100Mbps), T3 (45Mbps) Ethernet (10Mbps) and token ring (16Mbps) links for our simulation topology. 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 FDDI link from node 4 to 5 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 $t_{mean}$.

*B. 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 using LSR_a, LSR_b and LSR_ab coefficients are all much better than that for OSPF. Delay for LSR_ab is the lowest among the LSR methods, because it is based on finding the LSR coefficient so that number of APs is maximum. However, delay of LSR_ab method is very close to that of LSR_a method because the number of APs in those two methods are close to each other. As load increases performance of all the LSR methods becomes even better than that of OSPF. Delay in OSPF method is at least 80% worse than LSR_ab method. The same trend is observed for average delay when $t_{mean}$ is 100msec as shown in Figure 3. Figure 4 and Figure 5 compares average jitter for different LSR algorithms and OSPF algorithm when $t_{mean}$ is 50msec and 100msec respectively. In these cases also all LSR algorithms perform better than OSPF. Among different LSR algorithms, again LSR_ab method has the lowest jitter. Jitter value of OSPF algorithm is worse by at least 20% compared to LSR_ab method at different load values. So, on an average, all LSR algorithms perform better than OSPF algorithm in terms of delay and jitter and LSR algorithm using LSR_ab method performs the best.

## V. CONCLUDING REMARKS

A Load Sensitive Routing (LSR) algorithm based on Dijkstra's shortest path algorithm is presented in this paper. In LSR algorithm, notification of congestion is limited only to the neighboring nodes. Hence 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 and calculate LSR coefficients. This overhead should not be a concern for modern routers which have lot of memory and powerful CPU. Moreover, the computation of new active and passive routing table and LSR coefficients would only happen when topology of the network changes. One advantage of LSR routing is that the forwarding engine need not know whether it is sending packet along OSPF next hop or LSR next hop. Thus, no change in the forwarding engine is required to implement LSR algorithm. Another advantage of LSR algorithm is that it can easily be implemented in the framework of OSPF by using a new LSA type, so routers running LSR algorithm can coexist with those running OSPF. This will allow deployment of LSR algorithm in the Internet in phases.

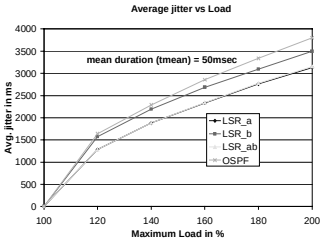


Figure 4.

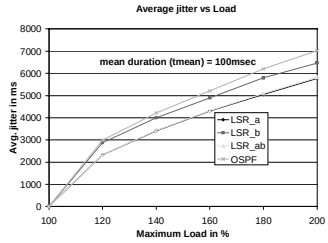


Figure 5.

In this study, we have developed LSR_ab method in which number of alternate path is maximized. Another way LSR coefficients may be calculated is to maximize *coverage* of alternate paths i.e. find LSR coefficients such that the number of nodes which has at least one alternate LSR next hop is maximized. Route flapping in LSR algorithm and effectiveness of route flap damping method to overcome route flapping similar to [9] may be studied.

## REFERENCES


[1] G. Apostolopoulos, R. Gu'erin, and S. Kamat, "Implementation and PerformanceMeasurements of QoS Routing Extensions to OSPF." Proceedings of IEEE INFOCOM'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] "OSPF Design Guide." http://www.cisco.com/warp/public/104/2.html, 2002.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] C. Villamizar, R. Chandra, R. Govindan, "BGP Route Flap Damping." Internet Request for Comments (RFC 2439), November 1998.

[10] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, A. Przygienda, and D. Williams,"QoS routing mechanisms and OSPF extensions." Internet Request for Comments (RFC2676), April 1999.

[11] A. Sahoo, "An OSPF Based Load-Sensitive QoS Routing Protocol in Best-Effort Environment." IEEE Milcom, 2002, in press.

[12] A. Segall, P. Bhagwat, A. Krishna, "QoS Routing Using Alternate Paths." Journal of High Speed Networks, 7(2): pages 141-158, 1998.

[13] 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.

[14] Z. Wang, and J. Crowcroft "Shortest path first with emergency exits." ACM SIGCOMM '90, pp. 166-176, Philadelphia, PA, Sept. 1990.

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

[16] 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.