QoS Routing Using Alternate Paths

Adrian Segall *
Dept. of Computer Science
Technion, Israel

Pravin Bhagwat Arvind Krishna
IBM, Thomas J. Watson Research Center
Yorktown Heights, NY

Abstract

A QoS routing protocol based on alternative path selection is described. Using a combination of source routing and shortest path routing, packets are routed along alternate paths when available resources along the shortest path cannot satisfy the QoS requirements of end to end flows. For path selection, the QoS routing protocol makes use of the information already available in the OSPF link information database. Packet forwarding uses existing mechanisms within the Internet protocols. Thus, support for QoS routing is provided without requiring major changes to the existing installed base of OSPF routers. Our proposal provides QoS routes for unicast as well as multicast connections. The major virtues of the solution are that of providing an evolutionary approach, whereby only a subset of the routers need to be enabled, and that the additional algorithms are computationally simple without requiring state maintenance at every router. Simulations of our proposed routing protocols on the UUnet backbone show that they perform well, almost optimal in terms of delivering the desired QoS for end to end flows.

1 Introduction

While Internet Service Providers (ISPs) and corporate Intranets are adding more bandwidth and switching power to their backbones, the end host population is growing exponentially, thereby stressing the shared resources. Making efficient use of the available capacity is becoming increasingly important, especially in networks operated by independent ISPs who want to maximize returns for their investment. At the same time, interest in Quality of Service (QoS) based routing is growing as evidenced by the Internet Engineering Task Force's (IETF) chartering a working group in 1997 [1] to develop a solution framework. Improving network utilization is not the only objective; QoS routing is also being viewed as a means for providing various quality of service levels to end users.

Today's Internet only support one type of service: *best effort service to all.* The Internet routing system routes unicast and multicast packets based only on the destination addresses contained in IP datagrams; the QoS demand of connections are not factored in making routing decisions. The

*Currently on sabbatical at IBM

result is that all packets between a source destination pair follow the same path. If the path chosen by the routing protocol is congested, all sessions flowing through the congested link suffer service degradation. It is true that most Internet applications are designed to be elastic with respect to the available network resources. Yet, we share the view that the routing should make the best effort of providing a service that is consistent with the applications needs. Making hard guarantees are difficult, but doing more than what is supported today is certainly desirable.

Routers in the Internet today compute paths based on shortest path algorithms [2]. While this algorithm is simple and efficient, the limitation is that it cannot route packets along alternate paths. Due to mesh like topologies of the routing backbones, several choices of routing paths now exist between most source and destination pairs. Furthermore, routers today are equipped with sufficient computational power to compute those paths. Thus, what was considered a limitation few years ago is gradually becoming less of an issue.

It is well known that the design of optimal routing under QoS constraints is a very complex problem[3], especially for large, complex, heterogeneous networks as the Internet. This paper discusses a proposed scheme for QoS routing in the Internet, referred to as QoS Alternate Path routing (QoS-AP), that requires minimal changes to the existing routing protocols and the infrastructure. In addition, it allows for gradual upgrading of nodes to QoS capability and for independence of the intra- and inter-domain QoS routing. The latter means that autonomous domains that choose to implement a different routing than QoS-AP can still participate in our protocol, provided that the border nodes are QoS-AP capable. For unicast, we base our proposal on OSPF routing, whereas for multicast we offer a Core-Based-Tree (CBT) based protocol. The link state architecture requires routers to broadcast the local resource status and the local topology information to all intra-domain routers. In our scheme we assume that intra-domain routers are updated at a reasonable rate about changes in resource availability and in topology. In addition, our protocol allows for variable delays in the updates and includes provisions for situations when the actual available resources met by the routed messages are different from the ones known by the origin at the time when message propagation starts. Simulations of our proposed routing protocols on the UUnet backbone, described in Section 7, show that they perform well, almost optimal in terms of delivering the desired QoS for end to end flows.

2 Background

A method for specifying type of service already exists in the IP protocol specification [4], though this feature has largely been unused by end hosts. Need for supporting multiple QoS levels is evident as virtual private networks (see [5] for an example) are being overlayed over the Internet. ISPs are interested in supporting service differentiation; by pricing each service type differently they have a potential for increasing revenue, while maximizing customer satisfaction.

Traffic prioritization is one means of providing QoS support in routers today. The basic idea is to provide multiple levels of priority queues [6]. Priorities can be assigned to packets based on their protocol, source network, destination network, packet type, and other protocol-specific fields. Weighted Fair queuing [7] is another method that is supported in Cisco and IBM routers [8, 9, 10]. These enhancements can be thought of as link sharing mechanisms which enable network administrators to subdivide link capacity into different classes, assign traffic to each class, and serve each class with a different priority. Packet classifiers can either be statically configured or applications can dynamically communicate their QoS requirements to the routers using a signaling protocols such as RSVP [11]. However, note that traffic prioritization only improves the QoS seen by a class of traffic on a given link, and that link is chosen by the (shortest path) routing mechanisms independent of the QoS.

While traffic prioritization is a good start, it is not a long term solution. When links along the shortest path are congested, new connections requests must be denied even though network may have spare capacity available along alternate paths. The answer lies in integrating QoS support with the routing layer. The IETF has recently formed a new working group [1, 12] to develop a framework for supporting QoS routing in the Internet. The goal of the working group is not to invent new QoS routing solutions, but to instead focus on fostering inter-operability between them. Different autonomous systems will have freedom to choose the QoS routing protocol of their choice. The group will only specify requirements so that different solutions can interact at the inter-domain level through simple, consistent interfaces.

The problem of QoS routing has already been investigated in the realm of circuit-switched networks, and more recently ATM networks. Interested readers can find a good summary of related work in [12]. While many of the concepts therein apply to QoS routing for the Internet, the need to have an evolutionary approach prohibits a direct application of these prior solutions. The complete design space of QoS routing solutions is very large, and perhaps there is no solution that can be labeled optimal. Throughout the following discussion we limit our discussion to only those design choices that are feasible to incorporate within the existing Internet routing infrastructure.

To perform QoS routing, routers must be capable of determining the current resource availability in the network, as well as the QoS requirements of flows. Based on this information, the routing algorithm can then choose a path that has a good chance of meeting the QoS requirements of flows. We assume that each router can monitor usage of links directly connected to it, and using the existing OSPF flooding mechanisms inform other routers when resource availability on those links change. We do not attempt to answer the question as to how frequently should this information be propagated. Readers interested in this issue should look at the work of Guerin et. al. [13] which analyses the impact of information inaccuracy on the performance of path selection process. We now explore three basic design choices in the context that full topology information as well as approximate link utilization is known at each node.

- **Pre-compute:** Each node can independently compute the shortest path subject to the QoS constraints, i.e., nodes first remove those links from the topology map that do not meet QoS demand and then compute the shortest path. Routers keep a table of pre-computed routes for every possible QoS value and the destination. This computation has to be repeated for every significant change in any link utilization. Some disadvantages of this approach are that every router has to be QoS capable, routing tables are large, and route flapping may occur. The basic advantage is that connection setup delay is low.
- **On-demand compute:** When the size of the routing table is an issue the QoS route for a particular destination can be computed purely on demand and the results cached. Again, every router has to be QoS capable. Scaling with the increasing number of QoS granularity is also an issue. Besides this design does not allow distinct flows with the same QoS and destination to take distinct paths.
- **Stateful Routing:** Each router maintains connection state, specifying the next hop for the given connection. Assuming that each node has access to the link state database, each node can independently choose the next hop. This approach is essentially combining RSVP with routing and suffers the first two disadvantages of the previous method. The additional disadvantage is that the amount of state in each router is large, being proportional to the total number of active flows.

The alternative to suffering many of the disadvantages described above is to resort to source routing. However, if explicit source routing is used, the size of the source routing header is large (e.g., for a 15 hops route IPv4 source routing header will consume 60 bytes. This overhead will be 240 bytes in the case of IPv6). A second drawback of explicit source routing is that every router has to process the header for every data packet, leading to unacceptable computational requirements.

3 Solution Overview

For unicast routing, our proposal offers a compromise that is computationally efficient and requires less overhead than the explicit (strict) source routing method. Moreover not all routers in the domain need to implement our protocol, and thus it allows for a gradual upgrade of the routers.

When resources along the shortest path are available OSPF routing is used to carry packets from the origin to the destination. Otherwise, the QoS-AP algorithm is used to compute an alternate path. The alternate path is chosen so as to avoid congested links in the network. Instead of listing every node on the chosen path, we list only a subset. This set is chosen such that the path constructed by connecting routers in the list (using shortest paths) satisfy the QoS demand of the flow. When a data packet is transmitted from the origin, its destination is set equal to the first router in the list. Using standard OSPF routing, the packet follows the shortest path to the specified router, which, after performing protocol processing, forwards the packet to the next address in the list. This process is repeated until all addresses in the list are visited, at which time the packet has reached at its final destination.

Our design reflects the tradeoff that provides an efficient path, but not necessarily the optimal path, while keeping the routing overhead within acceptable limit. The QoS routing algorithm only increases the likelihood that packets are routed along paths that meet QoS demands of connections. The proposed algorithm does not reserve resources. It is the responsibility of the higher layer protocol to do admission control and bandwidth reservation. However, if the alternate path routing protocols described in this paper do their job well, reservations may often be unnecessary. Roughly speaking, we provide a *best-effort QoS routing* protocol.

The multicast proposal is based on the Core-Based-Tree (CBT) idea introduced in [14]. The tree is built so that QoS requirements are met on the branches of the tree. We assume that a tree is built for every QoS requirement. As in the unicast case, we do not require all routers to be upgraded to implement our protocol, but we assume that all routers are CBT capable. In addition, we give provisions for routers to rejoin the tree in case of a branch or node failure and for coping with changes in link availability that may affect the QoS perceived by the multicast connection.

Nodes join the multicast connection by sending messages that propagate in the same fashion as the unicast routing until they reach a node that is already on the tree. At that point the join request is acknowledged and the branch is attached to the tree. After a tree branch or node failure, the node below the failure tries to rejoin the tree by sending a similar message. The latter propagates in the same fashion as the join message, except that it keeps propagating on the tree after it reaches it until it reaches the core or the originator of the rejoin. In the first case, the rejoin is successful, in the second a loop has formed and the disconnected portion of the tree is dismantled.

The protocol for adjustment to changes in link availability calls for receivers to monitor their quality of traffic and to send signals up-tree. The signals are processed at each tree node and again sent up-tree until the location of the overload that causes the deterioration in QoS is determined.

Throughout the design we have made conscious effort to meet the following design goals. As we'll show later in this paper, our algorithm satisfies all of the following properties.

- Phased Deployment: Deployment of QoS routing capability will happen in steps. Therefore the QoS routing algorithm should not require cooperation from every router.
- Decoupling of inter and intra domain routing.
- Scalability: routers will not maintain state about connections, their QoS requirements, or their associated QoS paths. This is true only for unicast connections. Routers will maintain state for multicast connections.

Before we present details of the QoS-AP protocol, we list all the assumptions that we make:

- Throughout this paper we assume that QoS is specified in units of bandwidth. Focus on bandwidth is for illustration purposes only, our QoS alternate path algorithm is not tied to any particular definition of QoS.
- Routers have access to a complete topology map of their routing domains, i.e., the routing protocol is either OSPF, or its derivative. The database that holds the domain topology at each router will be referred to as the *OSPF routing database*.
- The source of the QoS connection, its destination and the iter-domain border nodes that participate in the protocol must be QoS-AP capable. Nodes that are QoS-AP capable hold the information about link and node QoS-availability in their domain in a database called *QoS-availability database*. The later can be an extension to the OSPF database, but for clarity will shall refer to the two as different databases.

4 QoS-AP Unicast Routing

In this section, we describe the protocol used in QoS-AP in order to construct a routing path from origin node to destination node that satisfies the QoS requirements. For a given connection, the routing path is from the *origin node* to the destination node of the connection. The latter is denoted by *dest*. There are several types of nodes on the routing path:

- *non-QoS capable nodes* - those are nodes that have not been upgraded to QoS-capability; they use normal OSPF routing for both control and data packets; they do not implement the QoS-AP code.
- *intermediate destinations* or in short *idest*'s are QoS capable nodes that serve as intermediate destinations in the source routing header of data packets for the considered connection.
- *border nodes* are *idest*'s that are also on the border between two domains.
- *transit nodes* - nodes that implement the QoS-AP code for the route setup control packets, but are not *idest*'s or border nodes.

The protocol sends control messages $SEEK$ along the constructed path while the $SEEK$ accumulates the IP addresses of all *idest*'s in a parameter called *list*. When the $SEEK$ reaches the destination node, the *list* parameter will contain the list of all *idest*'s. The destination node sends this list to the origin node, that will use it as the source routing header for all data packets of this connection. The control messages used by the protocol are IP packets, that are sent between neighboring nodes and will be denoted in short by $SEEK(dest, idest, list, origin)$. In addition to the *list* parameter, $SEEK$ contains *dest*, the IP address of the destination node corresponding to the connection, *idest*, the IP address of the current intermediate destination and *origin*, the IP address of the origin node.

4.1 Intra domain routing

In this section we describe intra domain routing, namely the case when the origin and destination nodes of the connection are in the same domain.

The simplest case is when, according to the origin's OSPF routing database, the shortest path to the destination *dest* is composed of links that, according to the QoS-availability database, have the required QoS <A7> [1]. In this case, the origin tries to establish a direct non-source routed connection to the destination, by sending a $SEEK(dest, dest, \emptyset, origin)$ on the next link, according to the OSPF routing database. If the OSPF- and QoS-availability databases are consistent at all nodes on the route, the SEEK message will be sent on each link of the direct route, will arrive to the destination with an empty *list* variable and the established route will be the shortest OSPF path from origin to destination. If one of the databases at the origin node is not up to date or conditions change in the network while the $SEEK$ is being propagated, $SEEK$ may encounter at an transit node a link that is not QoS-available. In this case, the transit node next to the link declares itself an intermediate destination <A13> and tries an alternate route to the destination node, as described presently.

If the origin or an *idest* finds out from its QoS database <A8> that the direct OSPF route to the destination is not QoS-available, then the $try{-}detour()$ function is invoked <A8>. This function is also invoked when the next link to the destination is not QoS-available at a transit node on the route of the $SEEK$ <A14>. In the presented protocol, we propose three detour strategies, $QoSap()$, $PQoSap()$ and RQoSap() (see Fig. 1), but in general more can be constructed. All three functions look for paths with one *idest*. $QoSap()$ constructs the shortest path, such that both the path from the current node to *idest* and the path from *idest* to the destination satisfy the QoS requirements according to the current node's QoS-database. If such a path is not available, then the $PQoSap()$ function is invoked. The latter looks for an *idest*, k say, closest to the destination such that only the path from the current node to k is QoS-available, provided that k is closer to the destination than the current node. This with the hope that k will find a path that is QoS available to the destination with one additional *idest*. Finally, if both previous functions cannot find a path, $RQoSap()$ is invoked. This function is similar to the previous one, except that the restriction that k be closer to the destination than the current node is removed. The principle behind QoS-AP is related to deflection routing [15].

4.2 Inter-domain routing

Inter-domain routing is completely independent of the intra-domain routing. Domains that do not implement QoS routing or implement a different version than QoS-AP can still participate in the protocol.

[1]The notation $< \bullet >$ refers to the corresponding line in the algorithm code of Appendix A.

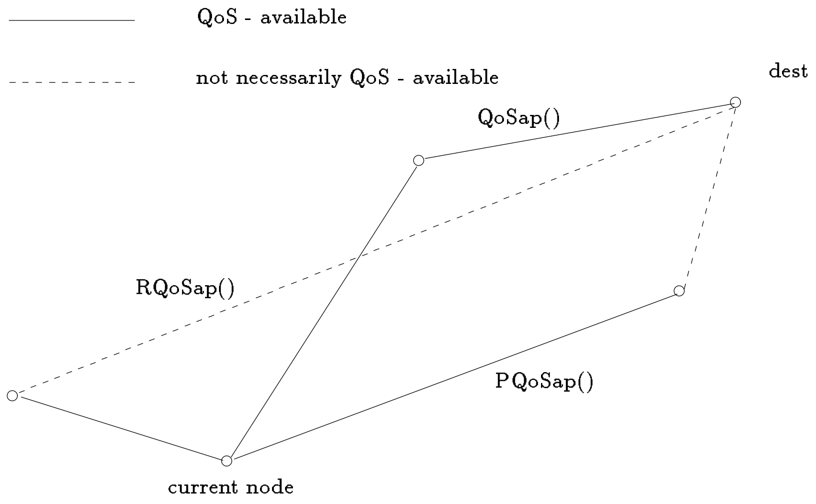

Figure 1: The three detour functions

In a domain that implements QoS-AP, inter-domain routing is performed in the same way as for intra-domain routing, except that the routing is to the border node on the OSPF route towards the destination <F1> rather than to the destination itself. In a domain that does not implement QoS-AP, the only requirement is that the intra-domain routing forwards the $SEEK$ message from the incoming border node to the outgoing border node, and that the border nodes implement QoS-AP. The border nodes will be included in the source routing *list* and this will allow data packets to be transmitted on the domain from one border node to the other.

4.3 Loop Avoidance and Topological Changes

Sometimes routing loops in QoS-routing are unavoidable. For example, suppose that in Fig. 2 , the destination node is C and the only QoS-available route is to $idest1$ via A and then from $idest1$ to C again via A. Since A is not upgraded to implement any QoS routing, the direct route origin-A-C cannot be implemented. In this case, the routing of the connection will contain an unavoidable loop. However note that we will never get an infinite loop.

Next suppose that the destination node in Fig. 2 is *dest* and consider the loop through a node that is QoS-AP capable, e.g. B. This loop can and will be avoided if the databases at all nodes are updated and consistent: $idest1$ will perform the QoS-AP algorithm, will find out that the route $idest1$-B-*dest* is shorter than *idest*-C-*dest* (via B) and therefore will not select the latter. However, if the databases are not consistent, the loop via B is still possible: $idest1$ thinks that the best route is via C and C finds out that the only route is via B. Another kind of loop may form if the $RQoSap()$ is used, since the *idest* selected by the current node may be further away from the destination than the current node.

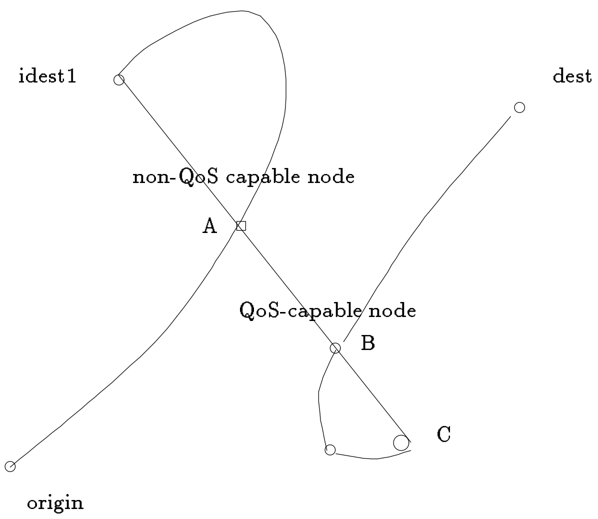

Figure 2: Loop Avoidance

The direct way to avoid loops is for every transit node to check whether it appears in the *list* carried by the $SEEK$ message <A15>. However not all loops are detected by this check, since only the *idest*'s are collected in *list*. In order to resolve other loops, the origin is required to verify the loop free property of the selected path. In our example, the selected path is *origin*-*idest*1-*C*-*dest*, and when the OSPF-database at *origin* will be updated, the latter will find out that there is an avoidable loop in the path. It will restart the route discovery procedure, and, if the database at *idest*1 will also be up-to-date, the new path will be *origin*-*idest*1-*B*-*dest*. Another loop avoidance strategy is implemented in the $PQoSap()$ function by requiring the current node to route only to a node that is closer than itself to the destination <D8>.

Even if the selected path is loop-free when established, loops can form afterwards due to routing transients or topological changes. For example, suppose that in Fig. 2, the established route is $origin - idest1 - C - dest$ via direct lines (not appearing in the Figure). Consequently, the routing changes or the two direct links $origin - idest1$ and $C - dest$ fail and the two loops via A and B form. As before, the loop via A cannot be avoided, but the one via B can. The origin will find out about this loop when its OSPF routing database is updated and will issue a new $SEEK$ massage. In steady-state, the database at *idest*1 will also be up to date and the later will route via B to dest.

A similar technique to the one described above is used in order to cope with topological changes. If the changes in topology warrants a change in the route, a new SEEK packet is issued from the origin.

5 QoS-AP Multicast Routing

Multicast in QoS-AP is based on a QoS-Core-Based-Tree (QoS-CBT) protocol, which is a QoS version of the Core-Based-Tree protocol [14]. We assume that all nodes are CBT-capable, namely have been upgraded to understand the CBT-protocol. However, as said in Sec. ??, we assumea gradual updating to QoS-capability, so not all nodes must be QoS-AP capable. However, only QoS-AP capable nodes can join QoS-CBT multicast groups.

QoS-CBT is invoked whenever a node is required to join a multicast group. At that time, the node sends a JOIN control message to the core of the group. The JOIN message is sent according to the unicast QoS-AP routing protocol, where the QoS availability is measured here according to the direction of the expected traffic between the node and the core: if the node is a source in the multicast group, the direction is away from the node, if it is a receiver, the direction is towards the node and if it is both then both directions must be considered <A6> – <A22> [2]. Not as in QoS-AP unicast, a node traversed by the JOIN records the interfaces over which it has received and sent the message, respectively as its $current - child$ and $parent$ in the QoS-CBT tree. It can do that because it is CBT-capable. Also, the JOIN does not collect the *idest*'s in a list. In QoS-CBT, the JOIN-ACK collects the entire node-list that includes all nodes, not only the *idest*'s.

The JOIN is propagated until it reaches the core or a node that is part of the tree. A node that propagates a JOIN message enters *pending* state, where it puts on hold any other JOIN message that may come in (see <A2>). When the propagated JOIN reaches a node that is already on the tree, the latter returns a JOIN-ACK, that propagates on the same route as the respective JOIN <B1> – <B9>. A node that propagates the JOIN-ACK enters *active* state, namely joins the tree and initializes its variable $child - set$ to contain the current child <B6>. The JOIN-ACK collects in *node-list* the route over which JOIN was sent. The origin of the JOIN keeps this list in a variable *tree-branch* <B9> and uses it to check the route for loops. This takes care of the possibility of transient loops on the route of the JOIN. If on the other hand the JOIN loops indefinitely, a JOIN-ACK will never be generated and nodes in pending state will eventually time-out (time-outs are omitted from the pseudo-code).

The multicast tree is not altered if the underlying unicast routing tables change. However, as explained below, there are instances when a node has to detach itself from the tree and rejoin. The rejoin mechanism is explained below. The detach mechanism consists of a node sending a QUIT message to its parent. The parent responds with a QUIT-ACK message, and if this was its last child, it relays the QUIT message to its own parent. The QUIT-ACK message returns a node to the inactive state <F1> – <G5>.

Adapting to link and node failures

[2]The notation $< \bullet >$ refers to the corresponding line in the algorithm code of Appendix B.

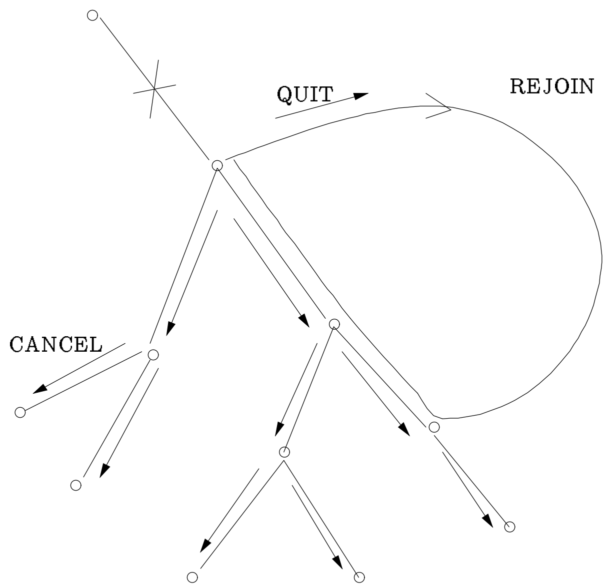

Figure 3: Loop due to REJOIN

When a link or a node on the CBT fails, the downstream node directly connected to the failed node or link initiates a rejoin process, provided it is QoS-capable <C5>. Extra care is required for processing REJOIN messages, since it is possible that the REJOIN message will hit a CBT node in the subtree below the node that issued the REJOIN message (see Fig. 3), thus forming a loop rather than reattaching to the CBT. This is the reason why we need to distinguish between JOIN and REJOIN by a similar algorithm to [14]. The REJOIN message travels in the same way as the JOIN, except that when it hits a CBT node, no ACK is generated and the REJOIN continues its journey on the tree <D5>. In its journey, the REJOIN will either reach the core of the tree or the originator of the JOIN. In the first case <D3>, the REJOIN has hit the tree connected to the core and the core generates a REJOIN-ACK, that propagates back to the origin of the REJOIN <E9>,<E6>. In the second case Fig. 3, the REJOIN has hit the subtree below the node that has generated the REJOIN, and QUIT and CANCEL messages are generated to take the subtree down <D9>, <D10>.

If a link or a node on the CBT fails and the downstream node is not QoS-AP capable, the latter starts taking down the subtree below itusing CANCEL messages <C2>. This will require a change in the code in non-QoS-AP capable nodes for QoS trees. The subtree is taken down until the process hits a QoS-AP capable node, which triggers a REJOIN <C7>.

Coping with changes in Link Availability: M-adjust

After the CBT is constructed, available bandwidth on some links may vary. The purpose of the M-adjust protocol is to identify links that can no longer satisfy QoS demands of the multicast connection, and reroute traffic around the points of congestion. We give a description of this protocol. We did not include it in the pseudo-code.

- Each receiver is responsible for monitoring the quality of traffic that it receives.
- Periodically receivers will notify their first hop router whether their QoS demands are met or not. This message will be multicast in the local LAN so that other receivers will not retransmit this message unnecessarily. The first hop router thus will always know the status of its receivers. The value 1 means the receiver's QoS demands are met and 0 means that QoS demands have been violated. A 0-1 message is sent to the parent router, which after determining the status of its children, may decide to relay either a 0 or 1 to its parent, and so on.
- The first hop router will periodically notify its status to its parent router.
- If a CBT router receives all 1's from its children it means that QoS requirements of all receivers in the subtree rooted at the router are satisfied. On the other hand, all 0's means that the source of the problem is elsewhere uptree. In this case, the CBT router forwards a 0 message to its parent.
- When some messages are 0's while others are 1's, the routers which have sent 0's are instructed to find alternate paths. Note that this can only be done after the parent has received at least a "1" from one of its children. If the parent router has only received 0's, it must wait until it hears the first "1".
- In the case of all 0's, the parent router must hear from all directly attached routers before it can relay the message to its parent.
- If 0-1 messages are lost, the parent may keep on waiting indefinitely. One possible solution is for the parent to start a timer when it hears the first status update message from one of its children. If updates from other children are not received before the timeout expires, the parent can directly send a query and request another status update. If again updates are not received, the parent will omit those children from 0-1 calculations.

6 Simulations

We have simulated the QoS-AP protocol on the UUNET High-Speed Backbone [3] (see Fig. 4) that consists of 33 nodes and 128 links. For simplicity, we assume that all links are homogeneous with a normalized bandwidth capacity of 100 units. The load on each link is chosen randomly between 1 and 99 units according to the two distributions described below. In an approximate sense, this loading of the links represents a snapshot in time of the network, and we vary the distributions to approximate a best-case and a worst-case loading. The performance of the algorithms is then

[3] topology map is available from `http://www.uunet.net/lang.en/network/us.shtml`

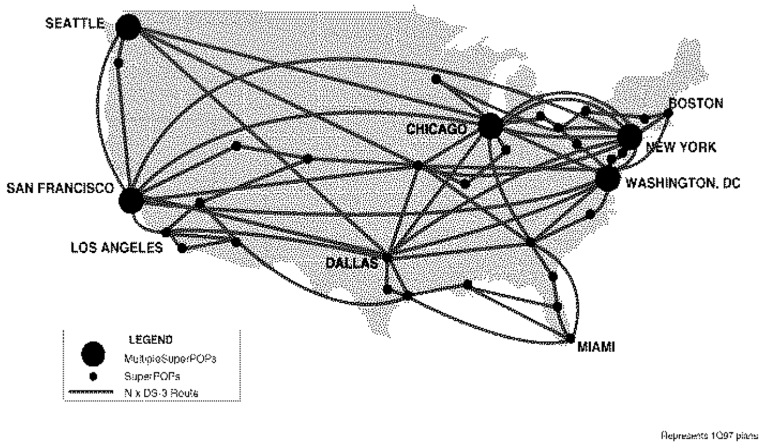

Figure 4: UUnet backbone topology

studied by examining how a typical request for a session with a desired bandwidth availability fares in this network.

In the first set of simulations, (Fig. 5), we have assumed a uniform distribution of load between 1 and 99 on each link and have plotted the fraction of node pairs that could not find a path with the required QoS, i.e., the "blocking" probability in the network with the proposed algorithms. The QoS requirement for sessions is varied from 1 to 50 units and the graphs represent an average over 100 independent runs, i.e., 100 different snapshots of the network. The curve labeled "disconnected" shows the average number of node pairs for which there is no path with the required QoS in the corresponding network. This is a lower bound on the blocking performance that can be achieved with any algorithm that attempts to route for QoS availability. The curve labeled "no QoSAP" shows the average number of connections for which the shortest path does not satisfy the QoS requirements. The other three curves give the average number of connections that cannot find a QoS available path, with progressive use of the detour functions $QoSap()$, $PQoSap()$ and $RQoSap()$. One can see that even with use of only $QoSap()$ we achieve a significant performance improvement, and the addition of the other two functions brings the performance close to the lower bound.

The uniform distribution of load represents an approximation to situations when the network is heavily loaded and is probably more pessimistic than the reality. In order to approximate more realistic situations, we repeated the simulation with a triangular distribution with reduced probability of high load on the links (Fig. 6), i.e., $\Pr(\text{load} \geq 100 - x) = \left(\frac{x}{99}\right)^2, 1 \leq x \leq 99$. Roughly speaking, such a distribution limits congestion to a few spots in the network, thereby improving the chances that alternate routing will reduce blocking. The corresponding behaviour seen in the graph shows this improvement with non-negligible blocking building up only at higher levels of the required QoS.

In order to justify the triangular distribution being more realistic, we cite the traffic measure-

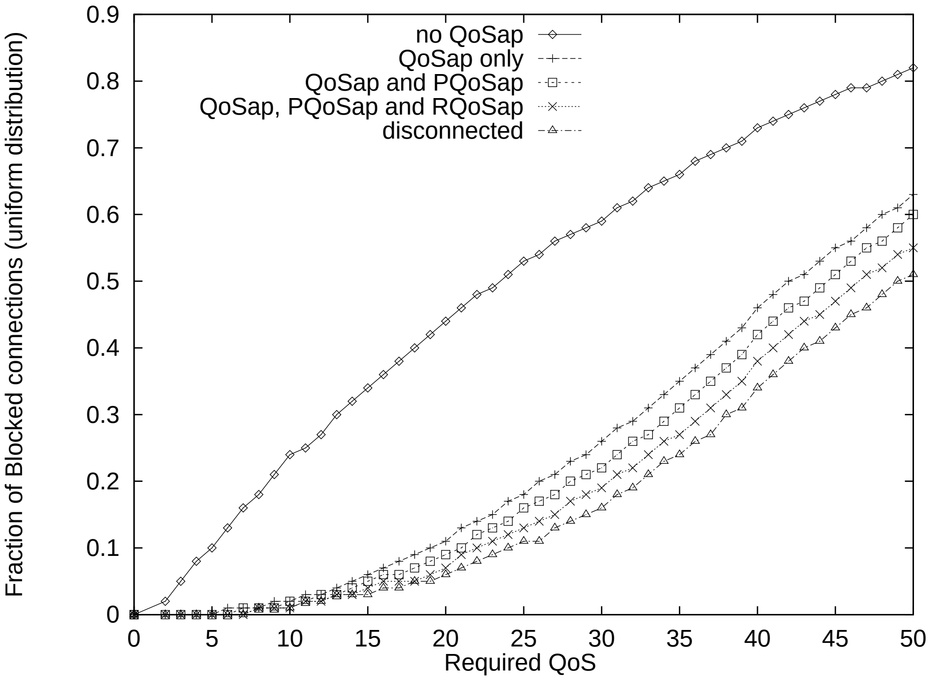

Figure 5: Fraction of Blocked Connections (uniform distribution)

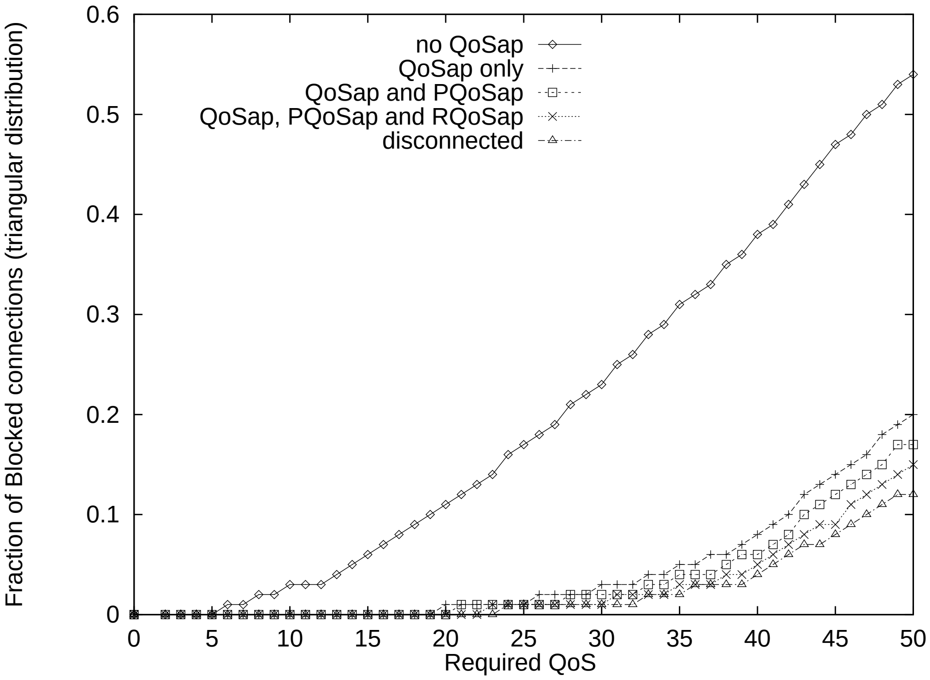

Figure 6: Fraction of Blocked Connections (triangular distribution)

ment studies carried out on the NSF's vBNS (very high speed Backbone Network Service) by Claffy et al. [16]. For example, Fig. 6 in [16] which shows the total traffic on a OC-3 backbone link implies that the instant traffic, as shown by the upper curve therein, is composed of a slowly varying mean value with a random variation that has a decaying tail distribution for the link utilization. This backs up our claim that the triangular distribution approximates reality.

7 Discussion & Conclusions

In this paper we have shown that an overlay network on the current Internet architecture can easily provide QoS-based routing. The major virtues of the solution are that of providing an evolutionary approach, whereby only a subset of the routers need to be enabled as *transit nodes* and that the additional algorithms are computationally simple without requiring state maintenance at every router. The performance of the algorithms as shown in Figures 5 and 6 implies that fairly simple alternative routing can yield almost all the benefits that any algorithm can provide. A significant reduction in blocking probability was achieved by having at most one *idest*, as allowed under the detour function $QoSap()$. Allowing for two *idest*'s, as allowed under the detour functions $PQoSap()$ and $RQoSap()$, almost achieves the lower bound in the reduction of blocking probability.

The detour functions we chose try to minimize the additional resources consumed in the network by alternative routing compared to shortest-path routing. However, there could be a long term impact on the network caused by the additional utilization of resources. We have not addressed these issues in this paper (and these are not likely to be real issues as long as the QoS traffic is a small fraction of the overall traffic). There is a rich literature exploring these topics in the circuit-switched and telephony networks. Of particular relevance is the work of Kelly (summarized in [17, 18]) and Ash [19], where the impact on the future performance of the network due to alternative routing is studied. We have to embark on a similar program to determine the impact of alternative routing on packet-switched connectionless networks.

One fact that we should point out is that our algorithms are easy to implement in the context of both the current generation (IPv4) and next generation (IPv6) Internet protocols. Without getting into the details, we point out that mechanisms are already in place that would allow for the easy deployment of our proposed algorithms. In the context of IPv4 one could use SDRP encapsulation [20] or loose source routing [4] for data packets once the *idest*s have been established. In the context of IPv6, the routing header [21] serves the purpose. In both cases, the control packets, e.g., the *SEEK* packet, can be normal IP packets with the payload containing the requisite information. Thus, routers not enabled for QoS routing can ignore these packets.

In summary, we proposed and evaluated alternative routing protocols in the context of the Internet. Our proposals can be incrementally deployed in an Intranet/Internet and offer significantly reduced blocking for sessions with QoS requirements.

A Unicast Algorithm Pseudocode

Protocol QOSPF

Messages

$SEEK(dest, idest, list, origin)$ - path-discovery message carrying:

- **dest** - the final destination
- **idest** - the intermediate destination
- **list** - accumulates the list of intermediate routing nodes
- **origin** - the origin node

Note:

- The origin node s acts as if it received $SEEK(dest, s, \emptyset, origin)$
- Data messages sent by the origin node will contain *list*, as received in the REPLY(list) message, in their routing header.

Algorithm for node i

```
A1    Upon receiving SEEK(dest, idest, list, origin)
A2    {  if (i = dest) then  QoS-AP route found, send REPLY(list) to origin;
A3       elseif (i ∉ list) then {
A4          if (i = idest) then {
A5             list == list ∪ {i} ;
A6             z == intermed(dest) ;
A7             if (direct(z) ≠ nil) then  send SEEK(dest, z, list, origin) on direct(z) ;
A8             else try–detour(z) ;
            }
A9          else {
A10            if (next–link(idest) is QoS-available) then {
A11               send SEEK(dest, idest, list, origin) on next–link(idest) ;
               }
A12            else {
A13               list == list ∪ {i} ;
A14               try–detour(idest) ;
               }
            }
         }
A15      else  discard packet;
      }
B1    try–detour(j)
B2    {  if (QoSap(j) ≠ nil) then
               send SEEK(dest, QoSap(j), list, origin) on next–link(QoSap(j)) ;
B3       elseif (PQoSap(j) ≠ nil) then
               send SEEK(dest, PQoSap(j), list, origin) on next–link(PQoSap(j));
B4       elseif (RQoSap(j) ≠ nil) then
               send SEEK(dest, RQoSap(j), list, origin) on next–link(RQoSap(j));
B5       else  discard packet ;
      }
```

C1 $QoSap(j)$
C2 $\{$ $D ==$ domain of i ;
C3 $\quad$ ***for*** $(k \in D)$ $\{$
C4 $\qquad$ $s(i,k) ==$ shortest-path from i to k ;
C5 $\qquad$ ***if*** $(s(i,k)$ contains only QoS-available links$)$ ***then*** $l(i,k) == length(s(i,k))$;
C6 $\qquad$ ***else*** $l(i,k) == \infty$;
C7 $\qquad$ $t(k,j) ==$ shortest-path from k to j ;
C8 $\qquad$ ***if*** $(t(k,j)$ contains only QoS-available links$)$ ***then*** $l(k,j) == length(t(k,j))$;
C9 $\qquad$ ***else*** $l(k,j) == \infty$;
$\quad$ $\}$
C10 $\quad$ ***if*** $(\min_{k \in D}(l(i,k) + l(k,j) = \infty)$ ***then*** $QoSap() == nil$;
C11 $\quad$ ***else*** $QoSap() == \arg\min_{k \in D}(l(i,k) + l(k,j))$;
$\}$
D1 $PQoSap(j)$
D2 $\{$ $D ==$ domain of i ;
D3 $\quad$ $D' == \emptyset$;
D4 $\quad$ $l(i,j) ==$ length of shortest-path from i to j ;
D5 $\quad$ ***for*** $(k \in D)$ $\{$
D6 $\qquad$ $s(i,k) ==$ shortest-path from i to k ;
D7 $\qquad$ $l(k,j) ==$ length of shortest-path from k to j ;
D8 $\qquad$ ***if*** $(s(i,k)$ contains only QoS-available links ***and*** $l(k,j) < l(i,j))$ ***then*** $\{$
D9 $\qquad\quad$ $D' == D' \cup \{k\}$;
$\qquad$ $\}$
$\quad$ $\}$
D10 $\quad$ ***if*** $(D' = \emptyset)$ ***then*** $PQoSap() == nil$;
D11 $\quad$ ***else*** $PQoSap() == \arg\min_{k \in D'} l(k,j)$;
$\}$
E1 $RQoSap(j)$
E2 $\{$ $D ==$ domain of i ;
E3 $\quad$ $D' == \emptyset$;
E4 $\quad$ $l(i,j) ==$ length of shortest-path from i to j ;
E5 $\quad$ ***for*** $(k \in D)$ $\{$
E6 $\qquad$ $s(i,k) ==$ shortest-path from i to k ;
E7 $\qquad$ $l(k,j) ==$ length of shortest-path from k to j ;
E8 $\qquad$ ***if*** $(s(i,k)$ contains only QoS-available links$)$ ***then*** $\{$
E9 $\qquad\quad$ $D' == D' \cup \{k\}$;
$\qquad$ $\}$
$\quad$ $\}$
E10 $\quad$ ***if*** $(D' = \emptyset)$ ***then*** $RQoSap() == nil$;
E11 $\quad$ ***else*** $RQoSap() == \arg\min_{k \in D'} l(k,j)$;
$\}$
F1 $intermed(dest)$
F2 $\{$ ***if*** $(i$ and $dest$ are in the same domain$)$ ***then*** $intermed() == dest$;
F3 $\quad$ ***else*** $intermed() ==$ next border node on route to $dest$;
$\}$
G1 $next\text{–}link(j)$
G2 $\{$ $next\text{–}link() ==$ next link on shortest path to j ;
$\}$
H1 $direct(j)$
H2 $\{$ $s(i,j) ==$ shortest-path from i to j ;
H3 $\quad$ ***if*** $(s(i,j)$ contains only QoS-available links$)$ ***then*** $direct() == next\text{–}link(j)$;
H4 $\quad$ ***else*** $direct() == nil$;
$\}$

B Multicast algorithm Pseudocode

Protocol Multi

Messages

$JOIN(core, idest, origin)$ - tree join message carrying:

core - the core of the QoS-CBT
idest - the intermediate destination
origin - the node that wants to join the tree

$JOIN\text{–}ACK$ - ack for the join process, carrying:

origin - the node that wants to join the tree
node-list - the nodes on the tree branch

$REJOIN(core, idest, origin)$ - tree rejoin message after a failure or CANCEL on parent link
$REJOIN - ACK(origin, node - list)$
$QUIT(core)$ - deattach from tree, sent to parent
$QUIT - ACK(core)$
$CANCEL(core)$ - deattach from tree, sent to all members of child-set

Variables

$child - set$
$parent$
$rejoin - source$
$state\ =$

pending - after JOIN or REJOIN, before ACK
active - after ACK
inactive - node not part of the algorithm

Note:

- The origin node s acts as if it received $JOIN(core, s, origin)$ from nil

Algorithm for node i

```
A1   Upon receiving JOIN(core, idest, origin) from k
A2   {  if (state = pending) then  hold message,
           until moving to active or inactive ;
A3      if ((i = core) or (parent(i) ≠ nil)) then  {                 /*state=active*/
A4         add k to child − set ;
A5         send JOIN − ACK(origin, {i}) to k ;
        }
A6      else {                                                      /*state=inactive*/
A7         if (i = idest ) then {
A8            z == intermed(core) ;
A9            if (direct(z) ≠ nil) then {
A10              parent == direct(z) ;
A11              next − idest == z ;
              }
A12           else try–detour(z) ;
           }
A13        else {
A14           if (next–link(idest) is QoS-available) then {
A15              parent == next–link(idest) ;
A16              next − idest == idest ;
              }
A17           else try–detour(idest) ;
           }
A18        if parent ≠ nil, then {
A19           state == pending ;
A20           current − child == k ;
A21           send JOIN(core, next − idest, origin) to parent;
           }
A22        else discard message ;
        }
     }
B1   Upon receiving JOIN − ACK(origin, node − list) from k
B2   {  if (k ≠ parent(i)) then exception;
B3      elseif (origin ≠ i) then {
B4         add k to node-list;
B5         send JOIN − ACK(origin, node − list) to current-child;
B6         child-set == {current − child};
B7         current-child == nil;
B8         state == active;
        }
B9      else tree-branch == node-list;
     }
C1   Failure on parent-link or CANCEL from parent
C2   {  if (i is not QoS-AP capable) then {
C3         send CANCEL to all members of child-set;
C4         child-set == ∅;
        }
C5      else {
C6         parent == nil ; state == pending ;
C7         send REJOIN(core, i, i) to yourself ;
        }
     }
```

```
D1      Upon receiving REJOIN(core, idest, origin) from k
D2      {   if  (state = pending)  then   hold message,
                until moving to active or inactive ;
D3          if  (i = core)  then   send REJOIN − ACK(origin, ∅) to k ;
D4          elseif  (state = active)  then   {                       /* parent ≠ nil*/
D5              if  (i ≠ origin)  then   {
D6                  current-child(origin) == k ;
D7                  send REJOIN(core, idest, origin) to parent ;
                }
D8              else   {
D9                  send QUIT to parent;
D10                 send CANCEL to all members of child-set;
D11                 child-set == ∅;
D12                 state == pending-inactive;
                }
            }
D13         else  {                                                  /*state=inactive*/
D14             if   (i = idest )   then   {
D15                 z == intermed(core) ;
D16                 if   (direct(z) ≠ nil)   then   {
D17                     parent == direct(z) ;
D18                     next − idest == z ;
                    }
D19                 else  try–detour(z) ;
                }
D20             else   {
D21                 if  (next–link(idest) is QoS-available)   then   {
D22                     parent == next–link(idest) ;
D23                     next − idest == idest ;
                    }
D24                 else   try–detour(idest) ;
                }
D25             if  parent ≠ nil,  then   {
D26                 state == pending ;
D27                 send REJOIN(core, next − idest, origin) to parent;
D28                 add k to child–set;
                }
D29             else   discard message ;
            }
        }
E1      Upon receiving REJOIN − ACK(origin, node − list) from k
E2      {   if  (k ≠ parent(i))  then   exception;
E3          elseif  (origin ≠ i)  then   {
E4              add k to node-list;
E5              if  (state = active)  then   {
E6                  send REJOIN − ACK(origin, node − list) to current-child(origin);
E7                  current-child(origin) == nil;
                }
E8              else   {                                         /* state = pending */
E9                  send REJOIN − ACK(origin, node − list) to current-child;
E10                 current-child == nil;
E11                 state == active;
                }
            }
E12         else   tree-branch == node-list;
        }
```

```
F1    Upon receiving QUIT(core) from k (state pending ????)
F2    {   if (k ∉ child − set) then   exception;
F3        else {
F4            remove k from child-set;
F5            send QUIT − ACK(core) to k;
F6            if (child-set==∅) then {
F7                send QUIT to parent;
F8                state == pending-inactive;
              }
          }
      }
G1    Upon receiving QUIT − ACK from k
G2    {   if (k ≠ parent) then   exception;
G3        else {
G4            parent == nil;
G5            state == inactive;
          }
      }
H1    try–detour(j)
H2    {   if (QoSap(j) ≠ nil) then {
H3            parent == next–link(QoSap(j)) ;
H4            next − idest == QoSap(j) ;
          }
H5        elseif (PQoSap(j) ≠ nil) then {
H6            parent == next–link(PQoSap(j)) ;
H7            next − idest == PQoSap(j) ;
          }
      }
I1    QoSap(j)
I2    {   D == domain of i ;
I3        for (k ∈ D) {
I4            s(i, k) == shortest-path from i to k ;
I5            if (s(i, k) contains only QoS-available links)   then  l(i, k) == length(s(i, k)) ;
I6            else  l(i, k) == ∞ ;
I7            t(k, j) == shortest-path from k to j ;
I8            if (t(k, j) contains only QoS-available links)   then  l(k, j) == length(t(k, j)) ;
I9            else  l(k, j) == ∞ ;
          }
I10       if (min_{k∈D}(l(i, k) + l(k, j) = ∞) then  QoSap() == nil ;
I11       else  QoSap() == arg min_{k∈D}(l(i, k) + l(k, j)) ;
      }
```

```
PQoSap(j)
{   D == domain of i ;
    D' == ∅ ;
    l(i,j) == length of shortest-path from i to j ;
    for (k ∈ D) {
        s(i,k) == shortest-path from i to k ;
        l(k,j) == length of shortest-path from k to j ;
        if (s(i,k) contains only QoS-available links and l(k,j) < l(i,j)) then {
            D' == D' ∪ {k} ;
        }
    }
    if (D' = ∅) then PQoSap() == nil ;
    else PQoSap() == arg min_{k∈D'} l(k,j) ;
}
intermed(core)
{   if (i and core are in the same domain) then intermed() == core ;
    else intermed() == next border node on route to core ;
}
next–link(j)
{   next–link() == next link on shortest path to j ;
}
direct(j)
{   s(i,j) == shortest-path from i to j ;
    if (s(i,j) contains only QoS-available links) then direct() == next–link(j) ;
    else direct() == nil ;
}
```

References

[1] H. Sandick & E. Crawley, Chair(s), "QoS Routing (qosr) working group," `http://www.ietf.org/html.charters/qosr-charter.html`, 1997.

[2] C. Huitema, *Routing in the Internet.* Englewood Cliffs, New Jersey: Prentice Hall PTR, 1995.

[3] D. Bertsekas and R. Gallager, *Data Networks.* Englewood Cliffs, NJ: Prentice-Hall, second ed., 1992.

[4] W. R. Stevens, *TCP/IP Illustrated Volume 1.* Reading, Massachusetts: Addison-Wesley, 1994.

[5] M. Hurwicz, "A virtual private affair," *BYTE Magazine*, pp. 79–87, July 1997.

[6] "Bay Networks IP Enhancements," `http://www.baynetworks.com /Products/ Routers/Protocols/Bridge/ip.html` , 1997.

[7] C. Partridge, *Gigabit Networking.* Reading, Massachusetts: Addison-Wesley, 1994.

[8] "Cisco IOS Software Features for Differentiated Class of Service for Internetworks," `http://www.cisco.com/warp/public/732/General/cos_wp.htm` , 1997.

[9] D. Newman, B. Melson, and S. Kumar, "DLSw Routers: Scaling to new heights," *Data Communications magazine*, July 1996.

[10] IBM Corp, "SNA APPN - High Performance Routing Architecture Reference." Version 2.0, SV40-1018, February 1997.

[11] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification." Internet Draft, March 1997.

[12] Eric Crawley and Raj Nair and Bala Rajagopalan and Hal Sandick , "A Framework for QoS-based Routing in the Internet." Internet Draft:draft-ietf-qosr-framework-01.txt, July 1997.

[13] R. Guerin and A. Orda, "QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms," in *Proceeding of IEEE INFOCOM*, May 1997.

[14] T. Bellardie, P. Francis, and J. Crowcroft, "Core Based Trees (CBT)," in *Proceeding of ACM SIGCOMM*, pp. 85–95, 1993.

[15] P. Baran, "On distributed communication networks," *IEEE Trans. Communication Systems*, vol. 12, pp. 1–9, 1964.

[16] J. Apisdorf, K. Claffy, K. Thompson, and R. Wilder, "OC3MON: flexible, affordable, high performance statistics collection," `http://www.nlanr.net/NA/Oc3mon`, 1997.

[17] F. P. Kelly, "Routing in circuit-switched networks: Optimization, shadow prices and decentralization," *Advances in Applied Probability*, vol. 20, pp. 112–144, 1988.

[18] F. P. Kelly, "Loss networks," *Annals of Applied Probability*, vol. 1, pp. 319–378, 1991.

[19] G. Ash, R. Cardwell, and R. Murray, "Design and optimization of networks with dynamic routing," *Bell Systems Technical Journal*, vol. 60, pp. 1787–1845, October 1981.

[20] D. Estrin, T. Li, Y. Rekhter, K. Varadhan, and D. Zappala, "Source Demand Routing: Packet Format and Forwarding Specification (Version 1)." Internet Draft, RFC 1940, available from `http://www.ietf.org`, May 1996.

[21] C. Huitema, *IPv6: The New Internet Protocol.* Englewood Cliffs, New Jersey: Prentice Hall PTR, 1996.