A Path Selection Method in ATM using Pre-Computation

Olivier Crochat[1], Jean-Yves Le Boudec[1], Tony Przygienda[2]

Abstract - We propose a method for path selection using pre-computation in an ATM network. The method is based on an algorithm that pre-computes a list of routes for each node in the network. We then study the lack of diversity in the paths chosen by our method, and the resulting risk of call blocking. Improvements to the method are proposed, and the decrease in call blocking is evaluated by simulation. We show that, with a load-balancing assumption, the method is able to provide paths equivalent to those of the On-Demand method.

Keywords - On-demand routing, pre-computed routing, path selection, path finding, resilience, blocking

1 Introduction

We consider the choice of a path when establishing a call in a communication network supporting *guaranteed bit rates*. Such connection-oriented services will be widely used to transport video transmission and other real-time applications. ATM, but also IP with some extensions (ST.II or RSVP) offer connection-oriented services [Top90, Zha93].

To respond to a connection request, a path satisfying the connection's constraints in bandwidth, delay, hops number has to be found. We call *routing* the act of finding such a path (also called a route in this article). The routing function can be seen as shown in figure 1.

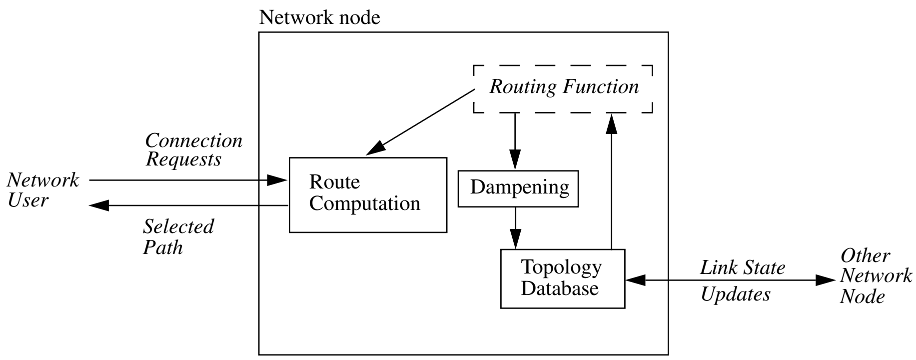

Fig. 1. Conceptual view of Routing Function

1. Laboratoire de Réseaux de Communication, DI - EPFL, CH - 1015 Lausanne, http://lrcwww.epfl.ch/
2. IBM Zurich Research Laboratory, CH - 8803 Rueschlikon,

Routing could be in the network nodes or in a central server. Its location does not affect the way the routing is done with our method. We assume that, for connection-oriented services, the routing is done at connection request time. This leads to the need for a global image of the network where the routing occurs. This image (which contains a description of all the links, their starting and ending points, their capacity) is stored in the topology database. The database maintains a correct view of the network through Link State Updates, which are sent by all nodes to give information about the status of their attached links. In order not to have too many of these updates, the *Dampening* function decides whether the changes in the database are important enough to require an update. A more documented discussion on the routing and its different parts can be found in [Leb94, Cyp71, OSI89].

A major choice in routing is whether it is done *On-Demand* or on a *Pre-Computed* basis. When On-Demand is used, the route computation is done at the time of the request, leading to a delay due to the calculation time. In contrast, when Pre-computation is used, connection setup time is faster [Gau95], but the lack of knowledge on the precise characteristics of the connection to be routed is a challenge. In section 2.3, we will see that our method is able to address that aspect without any a-priori classification of connection characteristics. Our method applies in particular to the selection of bi-directional, but asymmetric paths (namely, paths with different bandwidth requirements in the forward and return directions, but that require both directions to use the same links), as it is the case with ATM. Only peak bandwidth allocation is described in our examples, though the method applies to more general cases.

Pre-computation is attractive when the pre-computed routes can be used more than once without recalculating them. Such is the case when connection requests are frequent, but relatively small (in terms of required bandwidth), leading to high loads on real-time resources, but infrequent link state updates (thanks to the Dampening function).

In section 2 we define a path selection method that uses path pre-computation. We show that, under load-balancing assumptions, it finds paths with the same characteristics as On-Demand methods. Then, in section 3, we study the diversity of the paths selected with our method and point out a lack of diversity in some cases. In section 4 we propose modifications to the method that improve its performance.

2 Path Selection

2.1 Objectives

The route computation of figure 1 can be divided into two parts, *Paths list computation* and *Path selection*, as shown on figure 2.

Paths list computation is the algorithm which constructs, based on the topology database, the list of possible routes. This list has to be realized without any knowledge of future calls' characteristics. The paths list computation algorithm used in this article is described in [Prz95]. It analyses the network image and produces, for each destination, a list of optimal paths. This list is not dependent on any a-priori classification of the requests, but it guarantees that no better route can be found than those in the list, regardless of connection requirements at setup time.

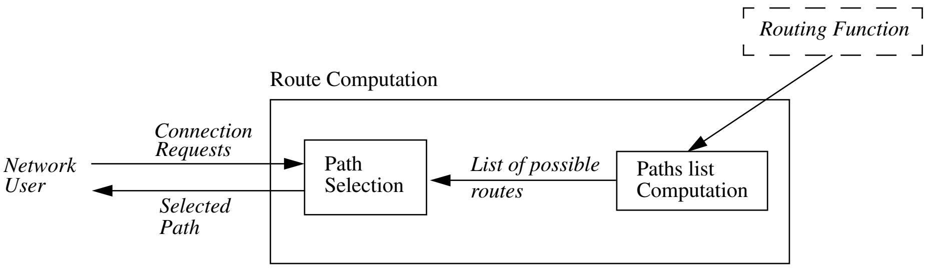

Fig. 2. Conceptual view of Route Computation Function

Path selection will then, on a per communication basis, choose a path among the ones proposed, depending on the particular requirements of each request.

2.2 Non-Dominated Routes

Before defining the method to select a route from a list, we have to see what kind of route is given by the paths list computation algorithm. We first explain the concept of *non-dominated* routes, and how it is possible to characterize and compare routes without having any knowledge of the future requests' characteristics.

Consider the sample network illustrated in figure 3. We assume that there exists

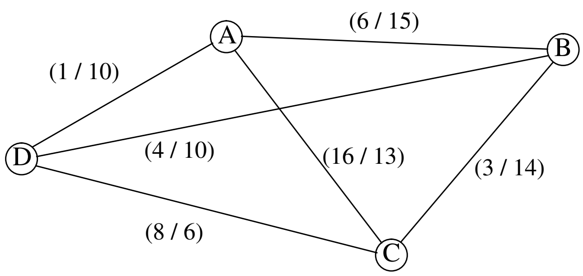

Fig. 3. Sample Network (additive cost / restrictive cost)

only one link between each node pair. Each link has a *cost vector*, containing two costs in this example. The first one is additive; the first cost of a path will be the addition of those of the links it goes through. It could be the number of VP hops a connection has to do in a network where nodes are ATM Virtual Channel Switches and links are Virtual Path (VP) Connections. The second one is restrictive; the second cost of a path going through a list of links is the minimum of their second cost. It may represent the available bandwidth of a link at a certain point in time, and can vary a lot.

All the possible loop-free routes between A and C and their costs are shown in Table 1.

A route is defined to be *strictly better* than another one if it offers an equal or a higher restrictive cost on its worst link, has a lesser or equal additive cost and one of them at least is not equal. In our example, routes C2 and C5 are strictly better than all

Route	Hops	Additive Cost	Restrictive Cost
C1	A,C	16	13
C2	A,B,C	9	14
C3	A,D,C	9	6
C4	A,B,D,C	18	6
C5	A,D,B,C	8	10

Table 1: Loop-free routes between A and C in the Sample Network

the other ones. However they cannot be compared since C5 has a smaller additive cost, while C2 is able to route larger calls, from a restrictive point of view. So, intuition would tell us to use C5 for calls having bandwidth requirements smaller than 10, and C2 for calls with requirements up to 14.

Those routes are said to be *non-dominated* (or extremal in [Prz95]), which means that there exists no other route that is strictly better. Generally, the structure of such routes is a lattice [Car79] and the set of non-dominated routes the *lower bound* of the lattice. Table 2 lists the non-dominated routes of the sample network shown in figure 3.

Route	Hops	Additive Cost	Restrictive Cost
C2	A,B,C	15	14
C5	A,D,B,C	8	10

Table 2: Non-dominated routes between A and C in the Sample Network

The algorithm presented in [Prz95] is usable in networks where links have two-dimensional cost vectors, but also for more costs, as long as only one of them is additive. We use it with three-dimensional cost vectors, having in mind ATM bi-directional asymmetric connections, which may have different bandwidth needs in each direction, but are required to follow the same path. The additive cost is the delay or number of hops, and the other two are the available bandwidth in each of the directions. With two restrictive costs, the method supports only peak-bandwidth asymmetric, bi-directional allocation. With more dimensions it is able to support allocation mechanisms that go beyond peak rate allocation.

This algorithm is used for the paths list computation. This is because it allows pre-computation in polynomial time of the list of all the non-dominated routes from a source to the other nodes in a network.

2.3 Path Selection with Load-Balancing

Based on the list given by the paths list computation algorithm, the path selection module chooses the best route for each different connection request. This is done in three steps:

1 - The first choice made by the policy is to eliminate all the paths that do not have enough bandwidth (in both directions) to fulfil the requirements of the request.

2- Then, among all the possible ones, the cheapest (in terms of additive cost) is chosen. This 'shortest path' is the one that will cause the least damage to future calls [Key90].

3 - In case of more than one path with minimal additive cost, the path preserving load-balancing is chosen.

This term, *load-balancing*, deserves an explanation. The aim is to try not to fill a certain path entirely, leaving the others empty. So, when more than one path is left after the two first steps, the path chosen is the one which has, after adding the pending connection, the largest capacity left for other connections. This is done by selecting the one which has the smallest of its restrictive costs larger than the one of every other possible routes. An example is shown in Table 3. Route X3 is chosen, since it has the most capacity left for future requests on its most loaded link.

X=(25 / 9 / 9) is to be routed			
Route	**Proposed routes (of minimal additive cost)**	**After adding the request**	**Minimum of Restrictive Costs**
X1	(19 / 30 / 10)	(19 / 21 / 1)	1
X2	(19 / 15 / 25)	(19 / 6 / 16)	6
X3	(19 / 20 / 20)	(19 / 11 / 11)	11
X4	(19 / 10 / 30)	(19 / 1 / 21)	1
(x / y / z) = (additive cost / first restrictive cost / second restrictive cost)			

Table 3: Example of the load-balancing method

Formally, the path selection module is defined as follows.

Path Selection Module

Let R be the connection request and P the path chosen at a certain iteration.
Let C(x), Cap1(x) and Cap2(x) be the additive, first and second restrictive costs of x, respectively

```
P={};
C(P)=Max;               /*Additive Cost */
Cap1(P)=Cap2(P)=0;      /* First & Second Restrictive Cost */
FOR each route I of the list DO
    IF C(I) < C(R) THEN
        IF C(I) < C(P) THEN
            P=I;
        ELSE IF C(I)==C(P) THEN
            IF Min[Cap1(I)-Cap1(R), Cap2(I)-Cap2(R)] >
                Min[Cap1(P)-Cap1(R), Cap2(P)-Cap2(R)] THEN
                    P=I;
            ENDIF;
        ENDIF;
    ENDIF;
ENDFOR;
```

We want to compare our path selection method with the Load-Balancing On-Demand method. On-Demand has the advantage of potentially finding paths that better suit the individual connection request. As we show below, this is not true in cases where a load-balancing option is taken. Let the request be (C_{req} / Cap_{req}) and the solutions found by On-Demand (C_{OD} / Cap_{OD}) and by Pre-Computed routing (C_{pre} / Cap_{pre}).

The method applied by the Load-Balancing On-Demand to find a route is the following:

Find (e.g. using Dijkstra) the shortest path
with the constraint of having $Cap_{OD} > Cap_{req}$.
If more than one path with minimal cost is found,
use the load-balancing policy described above.

The method applied by pre-computation is the following:

Using the path selection policy defined above, choose one path
from the list of non-dominated routes given by the paths list computation.

Proposition

The route proposed by our path selection method (shortest path with load-balancing on the list of non-dominated routes) is equivalent to the one which would be proposed by the Load-Balancing On-Demand Calculation.

Proof

The proof, done here for a two-dimensional cost vector, can be easily extended to more dimensions. Table 4 shows the possibilities of differences between the costs of the On-Demand and Pre-Computed solutions.

C_{OD} ? C_{pre}	Cap_{OD} ? Cap_{pre}	Is it possible?
<	<	No, (C_{OD} / Cap_{OD}) would have been chosen by pre-comp. too
<	=	No, (C_{OD} / Cap_{OD}) would be in non-dominated routes list
<	>	No, (C_{OD} / Cap_{OD}) would be in non-dominated routes list
=	<	No, to be discussed[a]
=	=	Possible, to be discussed[b]
=	>	No, (C_{OD} / Cap_{OD}) would be in non-dominated routes list
>	<	No, O.-D. would not have chosen a path that is not the shortest
>	=	No, O.-D. would not have chosen a path that is not the shortest
>	>	No, O.-D. would not have chosen a path that is not the shortest

Table 4: Loop-free routes between A and C in the Sample Network

So the only possibility is $C_{OD} = C_{pre}$ / $Cap_{OD} = Cap_{pre}$ showing that pre-computa-

a. If there is more than one path with the same cost and different capacities, then the load-balancing policy always chooses the largest capacity, so, as both use the same load-balancing method, it is not possible to have different capacities.
b. It is the same path, or an equivalent one. If two or more paths with the same cost vector are present in the graph, only one of them is stored in the non-dominated routes list.

tion with load-balancing gives equivalent results to On-Demand with load-balancing.

Load-balancing is not always optimal. In particular it may block large calls due to the fragmentation of large paths. However, we focus here mainly on cases where traffic consists of a lot of small connections. Large connections require a different treatment anyway, because they trigger the Dampening mechanism, and force a topology update and a new paths list computation.

3 Evaluation of Diversity for our Path Selection Method

3.1 Node-to-Node Diversity

The point we want to focus on is the *diversity* of the non-dominated routes. Two routes are said to have a *good diversity* if they have no link in common and a bad one if they share one or more. An example is shown on figure 4.

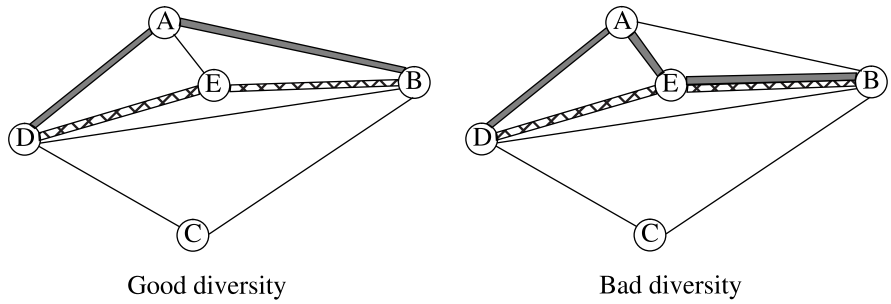

Fig. 4. Diversity between two routes

This is very important for two reasons. The first one is the resilience to component failures, for which it is good to have only a small amount of paths affected by a link/node failure (failure of link E-B in figure 4 will only cause one route to be out-of-order to the 'good diversity' case). The second is avoiding overload of only parts of a network. If many paths all want to go through a small number of links, these will soon become overloaded, and pending connections will be blocked until a new topology update.

The non-dominated routes have been tested to see if their diversity was good, or if they were all going through the same links. The results are presented in table 5. It shows the percentage of non-dominated routes going through the most used link (the link through which most of the routes go through) as a function of the number of nodes in the network, and their connectivity. The diversity comparison has been performed for non-dominated paths going from one node (the source) to one destination. The result is the mean over all the destinations.

Number of nodes in the network	Connectivity	Average number of optimal paths between two nodes	Percentage of paths having a bad diversity
15	0.2	2.7	81.3
15	0.3	4.5	80.0
20	0.2	5.2	60.7
20	0.3	7.2	63.1
25	0.2	7.3	70.0
25	0.3	18.3	53.1
30	0.2	11.6	53.0
30	0.3	11.9	50.5

Table 5: Percentage of paths going through the most used link

We can see that between 50% and 80% of the non-dominated routes all go through a certain link. This percentage can be explained by the fact that if a link is very “good” (high restrictive costs and very small additive one), it is normal that in the shortest path computation, routes will tend to go through it. When the number of nodes is small, this percentage greatly increases, due to the fact that there are not many possibilities to go from one source to a destination.

The construction of all the simulation networks used in this article has been done as follows. Given the number of nodes and their connectivity, links with independent and uniformly distributed integer random costs (0..Max) are put at random between two nodes. The computation of the non-dominated routes, the path selection policy and all the simulations done in this article were implemented in Mathematica 2.2, on a Sun Sparc20 workstation.

3.2 Global Diversity

The results obtained in 3.1 reveal the lack of diversity of the non-dominated routes calculated for a single node of the network. But in a real situation, each node is the source of only a small number of connections, and the interesting point is the diversity between connections originated in different nodes of the network. Note that, even in this case, each node has only local knowledge of what is going on, which means that, besides the topology database, it has no knowledge at all of the connections starting in the other nodes. Full information passing is essentially equivalent to a centralized solution of the problem [Key90] and is not the aim of this article, which is to find a good path selection method that uses only local knowledge (topology database), on a node per node basis.

To test the diversity on a global basis we did the following simulations.

We define a connection request to be a list containing the source node, the end node, the first restrictive component and the second one (S,D,C1,C2). Each node of the network is then the source for two requests. The destination, D, is chosen randomly. C1 and C2 are fixed to 1/20 of Max, so as to have small capacity requests in line with the assumption made in section 1. This traffic is then added to the network, one connection after the other, and the number of rejected requests is stored. It is done in this way:

```
FOR each node in the network DO
        Compute the list of its non-dominated routes; /* using paths list computation */
        Find the best route from the source to the destination,
                using the path selection policy;
        Remove the capacity used by this request from the links followed by it;
        IF capacity of one of the link < 0 THEN
                restore the capacity;
                reject the request;
                increment number of rejected routes;
        ENDIF;
ENDFOR
```

Table 6 shows the percentage of rejected connections as a function of the number of nodes, the connectivity, and the number of requests.

Number of nodes in the network	**Connectivity**	**Number of requests**	**Rejected requests (in %)**
15	0.2	30	35.0±6.3
15	0.3	30	39.4±12.2
20	0.2	40	38.3±6.5
20	0.3	40	24.0±12.6
25	0.2	50	32.4±4.1
25	0.3	50	31.6±5.7
30	0.2	60	38.8±16.0
30	0.3	60	25[a]

Table 6: Rejected requests due to blocking, with 90% confidence intervals

a. One run only, no confidence interval computed

The results are quite bad, with around a third of requests blocked. This proves that the diversity must also be poor at the network level. An overload of the most used links occurs rapidly, which leads to the blocking of many connections, and which would trigger a Link State Update. This would bring on a recomputation of all the routes, which has to be avoided if possible, because pre-computation routing is helpful only if the paths list computations are not too frequent.

It is interesting to note that the rest of the links are most surely not used a lot, and a solution could be to use them more, by having alternate paths. The next chapter will compare two methods that we propose to overcome these blocking problems.

4 Improvement of Path Diversity

4.1 Proposed Methods

The policy to choose a route given in 2.3 is not modified, because it always gives the shortest path and, when used in a lightly loaded network, it gives a route having no problem of blocking.

Below we propose two methods for improving resilience to blocking. The idea of both proposed methods is to add, for each non-dominated route given by the paths list computation, an *alternate* route, used after the rejection of the first try through the shortest possible non-dominated route [Ash90, Key90]. This 'second chance' solution is, as for the non-dominated routes, pre-computed.

Simple method

This solution consists of having another of the non-dominated routes as 'second choice', using the path selection policy shown in 2.3 to choose it, but after removing the 'first choice' route from the list. The implementation of this method is very simple and has very small computation needs, but the problems of diversity lack between non-dominated routes from the same source (shown in 3.1) could lead to a high probability of going through the same overloaded link.

Complex method

The second solution consists in computing, for each non-dominated route, a shortest path using Dijkstra on the additive cost, but on a modified topology database. This one is obtained by removing from the actual topology all the links with at least one restrictive cost smaller than the ones of the route we are trying to find an alternative to, and all the links being part of the 'first choice' route. The alternate route computed in this way is, if it exists, at least as good as the original one, and has no link in common with it.

Algorithm for the complex method

Let A be the alternate path of the route I in the list
Let C(x), Cap1(x) and Cap2(x) be the additive, first and second restrictive costs of x, respectively
Let T be the original topology database, and T_{tmp} the modified one.

FOR each route I of the list **DO**
    T_{tmp} = T - all links with Cap1 < Cap1(I) or Cap2 < Cap2(I);
    T_{tmp} = T_{tmp} - all links used by I;
    A = shortest path (using Dijkstra on T_{tmp}) to the same destination as I;
ENDFOR;

IF A exists **THEN** it is the alternate path with lowest C **ENDIF**;

Pre-computation is, of course, used for those alternate paths, but the complexity of this method, $O(N^4)$ (where N is equal to the number of nodes in the network) for three-dimensions cost vectors could be a problem. No optimization work has been done on it and improvements should be possible.

The simulation results for the two methods are shown in 4.2, along with a comparison between them.

4.2 Efficiency

In Table 7 we compare the results obtained by the three proposed methods:

The first one, with no protection against blocking, as shown in 2.3.

The second one, simple, with alternate route chosen among the other non-dominated routes.

And the third one, complex, with a completely disjoint alternate path for each non-dominated route (the last two methods being described in 4.1).

Number of nodes in the network	Connectivity	Number of requests	First method[ab]	Simple method[a]	Complex method[a]
15	0.2	30	35.0±6.3	23.3±9.0	29.4±9.5
15	0.3	30	39.4±12.2	21.7±8.6	22.8±7.7
20	0.2	40	38.3±6.5	22.9±7.6	22.5±7.5
20	0.3	40	24.0±12.6	12.5±9.0	7.0±4.1
25	0.2	50	32.4±4.1	17.2±6.9	13.2±5.9
25	0.3	50	31.6±5.7	16.8±6.4	6.8±5.3
30	0.2	60	38.8±16.0	20.0±37.0	10.4±4.0
30	0.3	60	25[c]	13.3[c]	5[c]

Table 7: Rejected requests due to blocking, with 90% confidence intervals

a. Percentage of rejected requests (with confidence interval in %)
b. Same results as shown in table 5
c. One run only, no confidence interval computed

The results obtained by the first method are quite bad, going up to 40% of requests rejected. The second and third ones show an improvement compared to the first one. It is worth noticing the following three points:

- The percentage of rejection decreases when the connectivity increases. If there are more links, there are also more paths, and the probability of overloading only certain links decreases.
- When the connectivity is high, there are a lot of non-optimal paths between each pair of nodes, and the complex method gives better results than the simple one.
- When the number of nodes is small, the complex method is not better than the simple one. This is because completely disjoint paths are often impossible in such small networks, and so, no alternate paths are found.

It is the difference between the results obtained by the second and the third method when the network is large that confirms the assumption that the overloaded links must be the ones through which most of the non-dominated routes go. Indeed, the fact of forbidding their use to find alternate paths (third method) leads to an evident improvement in the results, unfortunately to the detriment of the computation complexity.

Another way of considering this is to see when a Link State Update becomes necessary so to have a blocking probability lower than a certain threshold. The third method then becomes much more competitive, because its low blocking rate allows longer periods between Link State Update, requests being able to be satisfied even if the network is more loaded than with the other two methods. This could compensate its complexity, because it spaces the dates between paths list computation.

5 Conclusion

We have shown that it is possible to route communications in an ATM network using route pre-computation and that the results are the same as those that would be obtained by On-Demand computation if a load-balancing option is chosen for the path selection. Furthermore tests have revealed problems of request failures because of the lack of diversity of the routes given by paths computation. Modifications to the path selection policy proposed in chapter 3 have been tested, and an improvement to the blocking problem has been shown.

Further research focuses on finding the lower theoretical bound to the blocking of connections.

References

[Ash90] **Design and Control of Network with Dynamic Nonhierarchical Routing**, *G.R. Ash*, IEEE Communications Magazine, Oct. 1990

[Car79] **Graphs and Networks**, *B. Carré*, Clarendon Press - Oxford, 1979

[Cyp71] **Communications Architecture for Distributed Systems**, *R.J. Cypser*, ISBN 0-201-14458-1, 1971

[Gau95] **Scalability Enhancements for Connection-Oriented Networks**, *E. Gauthier, J.-Y. Le Boudec*, TR 95/127, DI-EPFL, 1995

[Key90] **Distributed Dynamic Routing Schemes**, *P.B. Key, G.A. Cope*, IEEE Communications Magazine, Oct. 1990

[Leb94] **Routing Metric for Connections with Reserved Bandwidth,** *J.-Y. Le Boudec, T. Przygienda*, EFOC-N, 1994

[OSI89] **ISO/IEC TR 9575, Information Technology - Telecommunications and Information Exchange Between Systems - OSI Routeing Framework**, *ISO*, 1989

[Prz95] **A Route Pre-Computation Algorithm for Integrated Services Networks**, *J.-Y. Le Boudec, T. Przygienda*, TR 95/113, DI-EPFL, 1995

[Top90] **Experimental Internet Stream Protocol, Version 2, (ST.II)**, *C. Topolcic,* IETF, 1990

[Zha93] **RSVP: A New Resource ReSerVation Protocol**, *L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zapalla,* IEEE Network, September 1993