

# TECHNICAL RESEARCH REPORT

## Call Rerouting in an ATM Environment

*by M.O. Ball, A. Vakhutinsky*

**CSHCN T.R. 95-22  
(ISR T.R. 95-58)**



*The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.*

**Web site <http://www.isr.umd.edu/CSHCN/>**

# Call Rerouting in an ATM Environment

Michael O. Ball\*, Andrew Vakhutinsky\*

College of Business and Management and Institute for Systems Research

University of Maryland

College Park, MD 20742

and

Phil Chimento, Levent Gun, Ted Tedijanto

IBM, Networking Systems Architecture

Research Triangle Park, NC 27709

June 1995

## Abstract

ATM networks must handle multiclass traffic with diverse quality of service requirements. We consider a multiclass routing model in which routes are calculated in a distributed fashion by the call origination nodes. Within this general context, we address the problem of rerouting a set of previously routed calls to avoid a failed link. Under the approach we propose, a single node executes an aggregate, global rerouting of all affected calls and then converts the set of aggregate routes into an allocation of bandwidth on each link to call origination nodes for the purpose of rerouting. The bandwidth allocation is distributed to each origination node, which in turn then calculates routes for the individual calls. The problem faced by each call origination node is a variant of the so-called bandwidth packing problem. We develop and analyze an approximate algorithm for solving this problem in the specific context that arises in our setting.

---

\*The research of these authors was supported by IBM Corporation and by NSF grant No. CDR-8803012.

## Introduction

In this paper we present mechanisms for rerouting around faults in multiclass traffic environments which will be encountered in most of the future ubiquitous broadband integrated service digital networks (B-ISDN). These networks should deliver service for the calls with widely varying bandwidth and quality of service (QOS) requirements. In order to provide high service availability, it is necessary for the networks to react in real-time to network faults, the topic of our research.

We assume a distributed routing environment in which the node at which the call originates calculates a route for the call. The node's calculations are based on link load information passed to it periodically by other nodes. Under normal conditions, the node selects a route for each call as each individual call request is received by the node. We associate a fixed bandwidth requirement with each call. Thus our model most directly applies to circuit switched networks where each call may have an arbitrary bandwidth requirement. However, the principal application intended for our model is ATM. ATM networks have a number of unique characteristics. Specifically, they should be designed to handle calls with widely varying characteristics and QOS requirements [11, 4]. Since varying traffic types must be handled, there can be wide variations in peak-to-average bandwidth requirements. For the purposes of routing we propose to make use of the existing research on the determination of a bandwidth allocation, or *effective bandwidth*, (see [11], pp 97 - 147) for each call based on its projected traffic characteristics. Thus, we would use the effective bandwidth as the call bandwidth value required by our model. Our model can be extended to allow for the possibility that the effective bandwidth of each call may vary from link to link. Thus the effective bandwidth can depend on the capacity of the link. This approach can be considered as a simplification of the "equivalent capacity" method proposed in [6, 7]. We recognize that most of the effective bandwidth theory to date has been in the context of a single link (and not a more general network). However, many feel that applying these ideas within a network context represents a reasonable approximation. Calls will also have an associated delay requirement varying from call to call. This is also included in our model. The requirement for handling multi-cast routes is one novel routing requirement which changes the underlying route selection structure from one of determining a simple path to determining a Steiner tree. In the analysis presented here we do not consider multi-cast routes, although we intend to do so in subsequent papers. A further complicating

factor especially in ATM networks is the two layer route structure consisting of call-level virtual circuits (VCs) and the higher level virtual paths (VPs).

The rerouting problems treated in this paper involve the simultaneous calculation of routes for several calls. Since calls with varying bandwidth requirements must be handled, the underlying optimization problem has a “packing” component. That is, calls with varying bandwidth must be packed into bandwidth limited links. The “bandwidth packing problem” has been studied in the literature [9, 12, 13]. Previous research has concentrated on finding an optimum or near-optimum solution by applying off-line algorithms. In this paper we develop a model and associated algorithms designed to operate in a distributed environment. Specifically, a two phase process is described. In the first phase, a centralized algorithm calculates a capacity allocation. This allocation is then distributed to each node. In the second phase, each node calculates paths, in parallel, for the calls it originated. The version of the problem we consider has certain unique characteristics including allowing the possibility of preempting existing calls.

The paper is structured as follows: section 1 gives a model description and general approach. The algorithms are described in section 2, computational experiments are presented in section 3 and section 4 deals with delay requirements and more general way of handling effective bandwidths.

## 1 Problem Formulation

### 1.1 Assumptions and Fault Scenario

The basic scenario we are presented with is the failure of a network link,  $e'$ , in an environment in which we may assume all network elements, including  $e'$ , are currently handling significant amounts of traffic. In general, at the time of its failure, several calls will be routed over  $e'$ . We denote the set of these calls by  $\mathcal{K}$ . The problem to be addressed is the determination of new routes for all calls in  $\mathcal{K}$ . Due to the dynamics of the network information used by the originating node when setting up the call, it is possible that when the call setup is actually carried out, it will be refused by an intermediate node due to the unavailability of sufficient resources on a link. We call this phenomenon a *call refusal*. Alternatively, it is possible that in order to set up the incoming call it is necessary to take down the route of an existing lower priority call.

We call this phenomenon *call preemption*. In such cases, the originating node of the preempted call would try to find a new route for that call. We note that each intermediate node along a call's path contains a limited amount of information concerning the call, including a call id, the originating node, call bandwidth and outgoing link. Certain rerouting strategies might require that this information be augmented.

We consider two failure scenarios for a network link,  $e'$ . Under the first scenario, the network's health monitoring system anticipates the failure of  $e'$  and the system has sufficient time to plan route adjustments accordingly. Under the second scenario, link  $e'$  experiences a sudden failure. In the first case, we could assume that the network management system has a "reasonable amount" of time to react to the failure and in the second, the network must react much more quickly. In either case, we assume that several calls are routed over  $e'$ .

For carrier networks (see [8, 2]) it appears that rerouting in the event of element failures can be restricted to the VP level. The strategy of setting up a backup VP with a 0 bandwidth allocation and then allocating bandwidth and switching to the backup VP at the time of a node or link failure appears to be a very effective one [8]. The effectiveness of this approach depends on the ability to reserve a certain amount of backup bandwidth. In lower speed (e.g. T1) private networks (our primary focus), we envision an environment in which backup bandwidth will not be reserved necessitating the preemption of lower priority calls in the event of element failures. To carry out the appropriate restoration, VC level rerouting together with call preemption will be required.

Our approach makes use of a certain capacity allocation scheme. Specifically, each call origination node is given its own capacity allocation for each link in the network. The capacity allocated to a particular call origination node will typically be substantially less than the total capacity available on the link. The path selection algorithms that the call origination node uses will penalize any violation of the capacity allocation. When a link fails a designated node will compute the capacity allocation for all call origination nodes that have routed calls over the failed link. This capacity allocation will be based on an aggregate rerouting of all affected traffic. The capacity allocation will then be transmitted to each origination node which will use the allocation when determining new routes for all calls previously routed over the failed link. The motivation for this approach is to reduce the incidence of call preemption and call refusal that

would normally take place when failures occurs. Specifically, it is anticipated that, when failures occur, since the origination nodes are trying to carry out “bulk” call setup, the information they would have on the traffic load on links would be inaccurate. Thus, it would be much more likely that call setups would be attempted on links that did not have sufficient capacity. The intent is to reduce the occurrence of this phenomenon by effectively allocating the link capacity ahead of time.

In this paper, we describe the underlying optimization problems, propose basic solution approaches for each and evaluate the merits of the two phase approach in determining high quality routes. We understand that several additional issues must be addressed in order to make this approach practical. These include:

- determination of what additional information must be stored by each node;
- thorough analysis of capacity allocation step, specifically – evaluation of alternative locations for computing the allocation, protocols for distributing the allocation information and determination of time required to carry out distribution;
- consideration of pre-calculation and storage-for-fast-retrieval of the bandwidth allocation;
- the development of a hybrid approach which reroutes as many calls as possible by switching to backup VPs, and then executes the algorithms presented here to carry out VC level rerouting.

In section 1.2 and 1.3 we present formulations of the rerouting problems. The purpose of these is to describe the fundamental problem structure. These formulations can not be used directly, since for the target environment distributed, real-time algorithms are required. In Section 2, when specific algorithms are presented, the telecommunications environment is taken into account.

## 1.2 Notation

Let  $\mathcal{G} = (\mathcal{N}, \mathcal{L})$  be the undirected graph representing a telecommunication network;  $\mathcal{N}$  is the set of nodes,  $\mathcal{L}$  is the set of links. Since each link carries different calls in each of its two directions, we consider each link  $[i, j] \in \mathcal{L}$  consisting of two directed arcs:  $(i, j)$  and  $(j, i)$ . The set of all arcs is denoted by  $\mathcal{A}$ ,  $\mathcal{A} \subset \mathcal{N} \times \mathcal{N}$ . Residual (free) capacity of the arc is defined as the difference between the total link capacity and the total bandwidth of the calls carried by the arc. In general, the arcs  $(i, j)$  and  $(j, i)$  have different residual

capacities  $B_{ij}$  and  $B_{ji}$  since different calls are routed over these arcs. We will also introduce the following notation for the network:  $IN(i)$  = set of arcs directed into node  $i$ ;  $OUT(i)$  = set of arcs directed out of node  $i$ .

As indicated previously, we let  $\mathcal{K}$  denote the set of calls to be rerouted. For each call  $k \in \mathcal{K}$ , the following set of parameters is defined: origin  $O(k) \in \mathcal{N}$ , destination  $D(k) \in \mathcal{N}$ , bandwidth  $b_k$  and priority class  $h(k) \in H$ . Additionally, we associate a revenue  $\hat{c}_{h(k)}b_k$  with call  $k$ . Thus  $\hat{c}_\ell$  can be considered as a revenue obtained from routing each unit bandwidth of class  $\ell$  (We note that the requirement that revenue is a linear function of bandwidth is *not* essential to the overall approach). Notice, that the revenue is received only after routing the “entire” call.

For the purposes of the exact problem formulation, we assume the set of all active calls  $\mathcal{M}$  is known explicitly. This assumption will be replaced with a reasonable, practical approximation in section 2. Let

$$\text{for all } m \in \mathcal{M}, e \in \mathcal{A}: \quad \delta_{em} = \begin{cases} 1 & \text{if call } m \text{ is carried by arc } e \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

and let  $\tilde{c}_{h(m)}b_m$  be the cost of preempting active call  $m$  with priority class  $h(m)$  and bandwidth  $b_m$ . Similarly to routing revenue,  $\tilde{c}_\ell$  can be considered as a cost of bumping each unit bandwidth of class  $\ell$ .

A path through the network for the call  $k$  is defined as a sequence of arcs  $(i_0, i_1), (i_1, i_2), \dots, (i_{m-1}, i_m)$  where  $i_0 = O(k)$  and  $i_m = D(k)$ . We will denote the path by  $p_k$ . The set of all feasible paths for call  $k \in \mathcal{K}$  will be denoted as  $\mathcal{P}(k)$ .

### 1.3 Problem Formulation

In this section, we present an integer programming (IP) formulation of the problem. This formulation is probably the most natural. It includes embedded “flow” problems for each call to be routed. The formulation uses the following 0/1 and continuous variables:

$$\text{for all } k \in \mathcal{K}: \quad y_k = \begin{cases} 1 & \text{if call } k \text{ is routed} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{for all } k \in \mathcal{K} \text{ and } e \in \mathcal{A}: \quad x_e^k = \begin{cases} 1 & \text{if call } k \text{ is routed over the arc } e \\ 0 & \text{otherwise} \end{cases}$$

$$\text{for all } m \in \mathcal{M} : z_m = \begin{cases} 1 & \text{if call } m \text{ is preempted} \\ 0 & \text{otherwise} \end{cases}$$

The exact IP formulation is given by:

**(IPR):**

$$\text{Maximize } \sum_{k \in \mathcal{K}} \hat{c}_{h(k)} b_k y_k - \sum_{m \in \mathcal{M}} \tilde{c}_{h(m)} b_m z_m - \sum_{k \in \mathcal{K}} \sum_{e \in \mathcal{A}} s_e^k x_e^k \quad (2)$$

subject to

$$\sum_{e \in OUT(i)} x_e^k - \sum_{e \in IN(i)} x_e^k = \begin{cases} -y_k & \text{if } D(k) = i \\ y_k & \text{if } O(k) = i \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } i \in \mathcal{N} \text{ and } k \in \mathcal{K}, \quad (3)$$

$$\sum_{k \in \mathcal{K}} b_k x_e^k \leq B_e + \sum_{m \in \mathcal{M}} \delta_{em} b_m z_m \quad \text{for all } e \in \mathcal{A}, \quad (4)$$

$$0 \leq y_k \leq 1 \quad \text{and integer} \quad \text{for all } k \in \mathcal{K}, \quad (5)$$

$$0 \leq z_m \leq 1 \quad \text{and integer} \quad \text{for all } m \in \mathcal{M}, \quad (6)$$

$$x_e^k \geq 0 \quad \text{and integer} \quad \text{for all } e \in \mathcal{A} \text{ and } k \in \mathcal{K}. \quad (7)$$

Objective function (2) maximizes the revenue received by routing the calls (the first term), minimizes the capacity violation (the second term) and a cost which encourages shorter routes to be chosen (the last term). Coefficients  $s_e^k$  are chosen to be relatively small in comparison to cost/revenue coefficients. Constraints (3) are imposed upon each call  $k$  are the usual network flow conservation equations. Since variables  $x_e^k$  are integral and the flow value for each call is 0 or 1 (the  $y$  variable restriction), the call can be routed over at most one, non-split path. Constraints (4) enforce the capacity restriction on each arc where the upper bound on the capacity is a “soft” constraint – it may be violated by preempting the calls.

We now present an alternate formulation, which uses a 0/1 variable for each feasible path. Thus, conceptually it potentially requires a very large number of variables/data. In general, this should be not deter one from considering it since formulations of this type can be quite practical *if one generates variables and their associated columns dynamically*. Specifically, for problems of this type “column generation” approaches are used, which generate columns of the constraint matrix only as they are required. It is typical that the number of columns generated is of a very manageable size and is much less than the total

number which appear in the problem statement. Although, initially we do not intend to use formal column generation approaches, our approximate algorithms will be similar in spirit to column generation and will use many of the same concepts.

The new variables for this formulation are path variables defined as follows:

$$f_{kp} = \begin{cases} 1 & \text{if call } k \text{ is routed over the path } p \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } k \in \mathcal{K} \text{ and } p \in \mathcal{P}(k)$$

**Flow-Path Formulation with Aggregate Call Preemption Costs:**

**(FPAB):**

$$\text{Maximize: } \sum_{k \in \mathcal{K}_r} \hat{c}_{h(k)} b_k y_k - \sum_{m \in \mathcal{M}} \tilde{c}_{h(m)} b_m z_m - \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}(k)} s_p^k f_{kp} \quad (8)$$

subject to

$$\sum_{p \in \mathcal{P}(k)} f_{kp} = y_k \quad \text{for all } k \in \mathcal{K}, \quad (9)$$

$$\sum_{k \in \mathcal{K}} b_k \sum_{p: e \in p \in \mathcal{P}(k)} f_{kp} \leq B_e + \sum_{m \in \mathcal{M}} \delta_{em} b_m z_m \quad \text{for all } e \in \mathcal{A} \quad (10)$$

$$0 \leq y^k \leq 1 \quad \text{for all } k \in \mathcal{K}, \quad (11)$$

$$f_{kp} \geq 0 \quad \text{and integer} \quad \text{for all } k \in \mathcal{K} \text{ and } p \in \mathcal{P}(k) \quad (12)$$

$$0 \leq z_m \leq 1 \quad \text{and integer} \quad \text{for all } m \in \mathcal{M}, \quad (13)$$

Constraints (9) state there is exactly one path for the call if it is routed and no paths if it is not. Constraints (10) are similar to those of (4).

The path formulation and associated column generation approaches provide a general framework for the development of a class of approximate and exact algorithms. Specifically, the process can be viewed as starting with a set of calls to be routed. A list of paths is built up dynamically over time. At any given time there exists:

- a list of calls
- a list of paths
- a “matrix” of path/call compatibilities and assignment costs (e.g.  $s_k^p$ )

- a (possibly partial) assignment of calls to paths

At any given iteration the algorithm could improve the assignment of calls to paths or alternatively, generate one or more new paths. Of course, these two options are clearly linked. If it did not seem possible to generate a good assignment of calls to paths using the current set of paths, then this would signal to need to generate additional paths. Furthermore, paths could be generated specifically to try to accommodate certain unassigned or poorly assigned calls.

## 2 Algorithms

In the previous section, we presented exact IP formulations of the problem. Theoretically, we could have plugged those formulations into an IP solver and obtained an optimal solution. Such an approach would not be appropriate in a real-time distributed setting. We propose a two-phase approach in which the final decision on, and calculation of, a call's path remains with the call origination node.

In this section we outline the design of the algorithms and overall problem structure and derive some theoretical running time bounds for the algorithm. The next section gives an experimental evaluation of the quality of these algorithms.

Subsection 2.1 gives a formulation of the Phase I capacity allocation problem. This problem is a linear program which can be solved using standard commercial software. The solution provides a global capacity allocation. In calculating this allocation, the total bandwidth allocated to all call origination nodes on a given link might include both bandwidth currently available and bandwidth that must be freed by preempting existing calls. In the second phase, each call origination node sets up calls "assuming" all allocated bandwidth is available. When each individual call is actually set up the nodes along the call's path will preempt existing calls if necessary. The Phase I problem must be solved once by a single central node. Possible candidate locations for solving this problem include one of the two nodes adjacent to the failed link and the network control/management center.

Subsection 2.2 presents an approximate algorithm for solving the bulk path selection problem that must be solved, in parallel, by each call origination node. The problem is variant of the bandwidth packing

problem. The algorithm presented involves the iterative solution of shortest path problems and is motivated by first-fit-decreasing bin packing heuristics

## 2.1 Phase I: Capacity Allocation

Below, we give a linear programming formulation of the capacity allocation problem. We introduce the following variables:

for all  $(i, j) \in \mathcal{N} \times \mathcal{N}$ ,  $\ell \in H$ :  $\tilde{y}_{ij}^\ell$  is total amount of priority  $\ell$  capacity assigned from node  $i$  to node  $j$ ;

for all  $e \in \mathcal{A}$ ,  $i \in \mathcal{N}$ :  $\tilde{x}_e^i$  is total amount of capacity assigned to origination node  $i$  on arc  $e$ .

The  $\tilde{x}_e^i$  variable values are the capacity allocations that are passed from Phase I to Phase II.

The LP problem is obtained from problem **(IPR)** by aggregating  $x$  and  $y$  variables and relaxing the integrality constraints.

**(CA)**:

$$\text{Maximize} \quad \sum_{\ell \in H} \hat{c}_\ell \sum_{(i,j) \in \mathcal{N} \times \mathcal{N}} \tilde{y}_{ij}^\ell - \sum_{m \in \mathcal{M}} \tilde{c}_{h(m)} b_m z_m - \sum_{i \in \mathcal{N}} \sum_{e \in \mathcal{A}} s_e^i \tilde{x}_e^i \quad (14)$$

subject to

$$\sum_{e \in \text{OUT}(j)} \tilde{x}_e^i - \sum_{e \in \text{IN}(j)} \tilde{x}_e^i = \begin{cases} -\sum_{\ell \in H} \tilde{y}_{ij}^\ell & \text{if } j \neq i \\ \sum_{m \in \mathcal{N}} \sum_{\ell \in H} \tilde{y}_{im}^\ell & \text{if } j = i \end{cases} \quad \text{for all } i, j \in \mathcal{N}, \quad (15)$$

$$\sum_{i \in \mathcal{N}} \tilde{x}_e^i \leq B_e + \sum_{m \in \mathcal{M}} \delta_{em} b_m z_m \quad \text{for all } e \in \mathcal{A}, \quad (16)$$

$$\tilde{y}_{ij}^\ell \leq \sum_{k: O(k)=i, D(k)=j, h(k)=\ell} b_k \quad \text{for all } i, j \in \mathcal{N}, \ell \in H, \quad (17)$$

$$\tilde{x}_e^i, \tilde{y}_{ij}^\ell \geq 0 \quad \text{for all } i, j \in \mathcal{N}, e \in \mathcal{A}, \ell \in H. \quad (18)$$

$$0 \leq z_m \leq 1 \quad \text{for all } m \in \mathcal{M}, \quad (19)$$

It can be easily seen that the formulation **(CA)** immediately follows from **(IPR)** if we perform the substitutions:

$$\tilde{x}_e^i = \sum_{k: O(k)=i} b_k x_e^k, \quad (20)$$

$$\tilde{y}_{ij}^\ell = \sum_{k:O(k)=i,D(k)=j,h(k)=\ell} b_k y_k \quad (21)$$

and relax the integrality constraints.

Because of the discrete nature of calls (they are not allowed to be split), it is not likely the calls can be “packed” within the capacity allocated if the amount of the capacity is exactly equal to the total bandwidth of the calls. To override this problem at least partially, we introduce an additional, “virtual” priority class  $\ell'$ , update the set of priority classes:  $H' = H \cup \{\ell'\}$ , and define  $\hat{c}_{\ell'} \ll \min\{\hat{c}_\ell : \ell \in H\}$ . Then the objective function (14) becomes:

$$\text{Maximize } \sum_{\ell \in H'} \hat{c}_\ell \sum_{(i,j) \in \mathcal{N} \times \mathcal{N}} \tilde{y}_{ij}^\ell - \sum_{m \in \mathcal{M}} \tilde{c}_{h(m)} b_m z_m - \sum_{i \in \mathcal{N}} \sum_{e \in \mathcal{A}} s_e^i \tilde{x}_e^i \quad (22)$$

A new upper bound for  $y$ -variables is added:

$$\tilde{y}_{ij}^{\ell'} \leq \varepsilon \sum_{k:O(k)=i,D(k)=j} b_k \quad \text{for all } i, j \in \mathcal{N}, \quad (23)$$

$\varepsilon = .1$  is a heuristically set parameter,

and the flow conservation constraints (15) are changed to:

$$\sum_{e \in OUT(j)} \tilde{x}_e^i - \sum_{e \in IN(j)} \tilde{x}_e^i = \begin{cases} -\sum_{\ell \in H'} \tilde{y}_{ij}^\ell & \text{if } j \neq i \\ \sum_{m \in \mathcal{N}} \sum_{\ell \in H'} \tilde{y}_{im}^\ell & \text{if } j = i \end{cases} \quad \text{for all } i, j \in \mathcal{N}, \quad (24)$$

(i.e.  $H$  is replaced by  $H'$ ) and we obtain a new formulation **(CA')** which is a slight modification of **(CA)**.

Till now, it was assumed the set of currently active calls to be known explicitly. It might not be the case in the real-life situations since

1. this set of calls is enormously large;
2. the algorithm will be executed by each individual source node and these nodes generally will not have global call information available.

Instead, we will assume that for each link, in addition to the residual capacity, the amount of bandwidth allocated to each priority class of calls is known. Specifically, we have for each link,  $e \in \mathcal{A}$ , and priority class  $\ell \in H$ ,

$$B_e^\ell = \text{the amount of bandwidth along link } e \text{ occupied by calls of priority class } \ell.$$

We then define variables for each  $e \in \mathcal{A}$  and  $\ell \in H$ ,

$$\tilde{z}_e^\ell = \text{the amount of bandwidth preempted from priority class } \ell \text{ on link } e,$$

Note that we have made a rather significant simplifying assumption that the cost of preempted traffic is a *linear function of the total bandwidth freed on each link*. This assumption represents an approximation for two reasons:

1. bandwidth is freed in discrete increments corresponding to the bandwidth associated with the calls that must be preempted.
2. preempting of a single call will, in general, free bandwidth on several links.

While there are certainly circumstances under which this assumption clearly leads to significant cost distortions, we feel that this simplification (or another similar one) is necessary in order to derive a practical approach to routing.

Using definitions for  $z$  variables and  $\delta_{em}$ , one can express  $B_e^\ell$  and  $\tilde{z}_e^\ell$  as follows:

$$\begin{aligned} B_e^\ell &= \sum_{m: m \in \mathcal{M}, h(m)=\ell} \delta_{em} b_m \\ \tilde{z}_e^\ell &= \sum_{m: m \in \mathcal{M}, h(m)=\ell} \delta_{em} b_m z_m \end{aligned}$$

Next, we use  $\tilde{z}$  variables to formulate the LP which can be used in practical computations.

**(CAP):**

$$\text{Maximize} \quad \sum_{\ell \in H} \hat{c}_\ell \sum_{(i,j) \in \mathcal{N} \times \mathcal{N}} \tilde{y}_{ij}^\ell - \rho \sum_{e \in \mathcal{A}} \sum_{\ell \in H} \tilde{c}_\ell \tilde{z}_e^\ell - \sum_{i \in \mathcal{N}} \sum_{e \in \mathcal{A}} s_e^i \tilde{x}_e^i \quad (25)$$

subject to

$$\sum_{e \in \text{OUT}(j)} \tilde{x}_e^i - \sum_{e \in \text{IN}(j)} \tilde{x}_e^i = \begin{cases} -\sum_{\ell \in H} \tilde{y}_{ij}^\ell & \text{if } j \neq i \\ \sum_{m \in \mathcal{N}} \sum_{\ell \in H} \tilde{y}_{im}^\ell & \text{if } j = i \end{cases} \quad \text{for all } i, j \in \mathcal{N}, \quad (26)$$

$$\sum_{i \in \mathcal{N}} \tilde{x}_e^i \leq B_e + \sum_{\ell \in H} \tilde{z}_e^\ell \quad \text{for all } e \in \mathcal{A}, \quad (27)$$

$$\tilde{y}_{ij}^\ell \leq \sum_{k: O(k)=i, D(k)=j, h(k)=\ell} b_k \quad \text{for all } i, j \in \mathcal{N}, \ell \in H, \quad (28)$$

$$\tilde{z}_e^\ell \leq B_e^\ell \quad \text{for all } e \in \mathcal{A}, \ell \in H, \quad (29)$$

$$\tilde{x}_e^i, \tilde{y}_{ij}^\ell, \tilde{z}_e^\ell \geq 0 \quad \text{for all } i, j \in \mathcal{N}, e \in \mathcal{A}, \ell \in H. \quad (30)$$

Notice that the cost of call preemption in this formulation may be overcounted since it is summed up over several links the call's bandwidth was freed. To compensate this effect, a corrective coefficient  $\rho$  was applied. Theoretically,

$$\frac{1}{\text{number of hops in the max hop route of the existing call}} < \rho < 1.$$

After some experimentation, we choose  $\rho = 0.5$ .

Similar to defining problem (CA'), substitution of priority set  $H$  with  $H'$  can be performed and then we obtain formulation (CAP')

Problems (CA) (or (CA')) and (CAP) (or (CAP')) were solved with a linear programming solver for the variety of instances. The solutions obtained indicated that the approximation was sufficiently accurate.

Observe that, since problems formulated in this section are essentially multicommodity flow problems, specialized algorithms could have been used. Exploration of such approaches lies beyond the scope of our current study.

As a result of solving the problem (CAP'), we assign arc capacities to each origination node. Those capacities are input into the algorithm described in the next section.

## 2.2 Phase II: Bulk Routing

In this section, we consider an algorithm which works independently for all origination nodes. Each node must receive a capacity allocation from the Phase I algorithm. That capacity allocation for call origination node  $i_0$ , for each link  $e$ , is denoted by  $B_e(i_0)$ . This link capacity is set equal to the value of variable  $\hat{x}_e^i$  computed in Phase I. The problem that must be solved in Phase II is a variant of FPAB (given in Section 1.3) in which the  $z_e^\ell$  (call preemption) variables are not present. That is, each call origination must carry out path selection using only the capacity allocated in Phase I. The paths calculated by each call origination node will automatically lead to appropriate call preemption since this was planned in determining the Phase I allocations.

Two considerations are critical in designing the routing algorithm: first, calls should be routed over shortest paths and, second, care should be taken in "packing" the calls into the constructed routes as tightly

as possible. In addition, the running time should be small enough to allow for real-time use. To address these issues, the algorithm designed makes use of shortest path algorithms (see e.g. [1]) and handles the calls in the spirit of the so-called *first fit decreasing* (FFD) heuristic which is known to be quite effective for the bin packing problem (see e.g. [10]).

We arrange the calls within the same priority class in descending order of their bandwidth. As the algorithm proceeds, we keep a pointer to the first unrouted call,  $\hat{k}$ . Focusing on  $\hat{k}$ , we execute a special shortest path algorithm which constructs a shortest path tree (SPT) rooted at node  $i_0$  using only those arcs which have enough capacity to handle  $\hat{k}$ . Afterwards, we try to route as many calls as possible over that SPT taking them in the order they appear in the list (descending by bandwidth). The process iterates using the new first unrouted call. When we can not route anymore calls within the same priority class, we proceed to the next priority class. Clearly, this approach is similar to the FFD algorithm for the bin packing problem. Indeed, the computational results presented in the next section indicate that it is quite effective in practice.

Input parameters of the algorithm are network  $(\mathcal{N}, \mathcal{A})$ , the origination node  $i_0$ , allocated capacities on the arcs  $B_e(i_0)$ ,  $e \in \mathcal{A}$  and set of calls  $\mathcal{K}_{i_0} = \{k \in \mathcal{K} | O(k) = i_0\}$ , output is the set of routes. We suppose the priority levels are ordered in the descending order:  $l_1 \succ l_2 \succ \dots \succ l_{|H|}$  and denote  $\mathcal{A}(w) = \{e \in \mathcal{A}, B_e \geq w\}$ . To better access the largest bandwidth calls, all calls are grouped by their destinations and stored at corresponding binary trees. Another set of internal data consists of capacity availability labels at each node which show the maximum amount of capacity that can be used to route a call to the node.

1. **for** priority level  $\ell = \ell_1, \ell_2, \dots, \ell_{|H|}$  **do**
2.     Initialize binary trees  $\mathcal{T}_j, \forall j \in \mathcal{N}$ :
3.      $\mathcal{T}_j$  contains items  $\{k \in \mathcal{K}_{i_0} \mid h(k) = \ell, D(k) = j\}$  with keys  $b_k$ ;
4.     **while** there is a non-empty tree  $\mathcal{T}_j$  **do**
5.         find call  $\hat{k} : b_{\hat{k}} = \max_{j \in \mathcal{N}} \max_{k \in \mathcal{T}_j} b_k$ ;
6.         Construct an SPT rooted at  $i_0$  for the network  $(\mathcal{N}, \mathcal{A}(b_{\hat{k}}))$ ;
7.         **if** there is a path from  $i_0$  to  $D(\hat{k})$ , route the call  $\hat{k}$  and update the labels  $w_j$ ;

8. Remove the call  $\hat{k}$  from its tree:  $\mathcal{T}_{\hat{k}} \leftarrow \mathcal{T}_{D(\hat{k})} - \{\hat{k}\}$ ;
9. **do**
10. find call  $\tilde{k} : b_{\tilde{k}} = \max_{j \in \mathcal{N}} \max_{k \in \mathcal{T}_j} \{b_k \mid b_k \leq w_j\}$ ;
11. if  $\tilde{k}$  was found,
12. route the call  $\tilde{k}$  in the current SPT;
13. remove the call  $\tilde{k}$  from its tree:  $\mathcal{T}_{\tilde{k}} \leftarrow \mathcal{T}_{D(\tilde{k})} - \{\tilde{k}\}$ ;
14. update the labels  $w_j \forall j \in \mathcal{N}$ ;
15. **while** there is a call  $\tilde{k}$ ;

The next issue we want to discuss here is the running time of the algorithm. All operations we perform with binary trees require time logarithmic on the size of the tree. These operations are: removing the items (lines 8 and 13) and finding the max key item (lines 5 and 10). Clearly, each call is considered only once. Indeed, if there is enough bandwidth in the existing SPT for the call (line 11), it is routed and removed (lines 12–13); if there are no calls which can be routed over the existing SPT (line 15), a shortest path algorithm is run to route the largest bandwidth unrouted call (line 6). If there is a path for that call, it is routed and removed, otherwise, it is removed anyway, since there is no chance for the call to be routed later. Capacity availability labels  $w_j$  are updated in time  $O(|\mathcal{N}|)$  (lines 7 and 14). The time required to extract the largest bandwidth call is  $O(|\mathcal{N}| \log |\mathcal{K}|)$ . If we denote the time required to construct an SPT by  $T_{SPT}$ , the worst case running time of the algorithm is  $O(|\mathcal{K}|(|\mathcal{N}| \log |\mathcal{K}| + T_{SPT}))$ . The average should be significantly better since, in general, several calls will be routed over each SPT constructed. The running time for the shortest path algorithm can vary depending on the approach used and the exact nature of the cost function. In the specific context we considered, since the objective was a pure “min-hop”, a breadth first search algorithm could be used which runs in time  $O(|\mathcal{A}|)$ .

### 3 Experimental Results

The purposes of our experiments were:

1. estimate the quality of our solution

Call Type	Bandwidth	Priority	Percentage		
	Distribution	Distribution	Mix 1	Mix 2	Mix 3
voice	all are .064 Mb/s	10% are 1, 90% are 2	54%	47%	41%
data	50% are .0096 Mb/s, 50% are uniformly from .01 to 1. Mb/s	uniformly, 1-4	43%	47%	53%
video	uniformly, 1 to 3 Mb/s	uniformly, 1-3	3%	6 %	6%

Table 1: Call Types

2. estimate the time required to perform the computations

First, we needed to obtain a reasonable distribution of calls in the network. That was done by running a basic simulation of call routing where calls were randomly generated according to the specified distribution and routed one by one through the network using the shortest path criteria. The generated calls are of three different types: voice, data and video. Table 1 gives bandwidth and priority characteristics for the calls. Each call of priority class 1, 2, 3 and 4 was assumed to have revenue 1.0, 0.1, 0.01 and 0.001 respectively. Table 1 also presents three different mixes of calls which were considered.

The network used for our simulation had a topology similar to ARPA net with 59 nodes, 71 undirected links (respectively, 142 directed arcs). Each link had a full duplex capacity of 154 Mb/s. The overall parameter setting was chosen so that after a link failure the total number of calls to be rerouted would be between 300 and 1,600.

We evaluate our approach in two general areas. First, there are the value/cost components reflected in the objective functions previously described, i.e. the value of all calls rerouted minus lost value of calls preempted. The second criterion to be considered involves call refusal. A call refusal occurs when the initial setup attempt fails. A call refusal causes a substantial cost in overall network overhead as well as delay associated with reestablishing the call. To accurately evaluate these concerns a detailed simulation is required. In lieu of such an analysis, we carried out certain experiments which indicate how alternate approaches would perform under "extreme" conditions. These were sufficient to derive definitive conclusions.

The experimental evaluation consisted of two parts. In the first part the quality of the call routing algorithm was estimated without considering call preemption. In the second part the evaluation of call preempting capabilities was performed. These parts of the experiment are described below in sections 3.1 and 3.2 respectively.

### 3.1 Evaluation of the Call Rerouting Algorithm

In this section, we consider two alternate approaches to routing as a basis for evaluating the call rerouting without preemption. In both approaches, each call is rerouted by an independent execution of a path selection (shortest path) algorithm. For the first approach, which can be considered a “worst case” scenario, we assume that each call origination node reroutes its calls but receives no updated information about link status during the rerouting process. Thus, all nodes assume that they have full access to all available bandwidth capacity. This is called “fast” rerouting. This approach will necessarily lead to high levels of call refusal in that competing call origination nodes will try to setup calls over the same sets of links. If a call is refused we do not attempt to reroute it – another very pessimistic assumption.

Under the second approach, we assume that a central node with perfect information reroutes all calls. That is, whenever a route is calculated the precise status of all links is known so that there will be no call refusal. The central node uses a first fit decreasing heuristic approach, by ordering calls in decreasing order of bandwidth. We call this method “slow” rerouting since time would be required to propagate the information about the state of the network and since the actual implementation of this approach would require the distribution of all routes from the central node. The two approaches constitute two extremes in that

- the fast approach requires the minimal amount of information exchange, but generally would achieve low quality routes;
- the slow approach should achieve the highest quality results (subject to the limitations of the first fit decreasing approach to packing) but is impractical due to the extreme information exchange requirements.

	Mix 1			Mix 2			Mix 3		
	30%	50%	65%	30%	50%	65%	30%	50%	65%
Total call value	32.46	79.50	115.28	49.79	97.65	124.99	38.29	114.40	139.38
Number of calls	357	897	1566	305	646	1414	287	670	1375

Table 2: Call Characteristics

Our approach can be viewed as a compromise which executes an approximate global rerouting with modest information exchange requirements while keeping final path selection in the hands of the call origination nodes.

Two sets of results are presented. The first is aimed at evaluating the effectiveness of the overall two phase approach and the second specifically addresses the Phase II packing algorithm. Table 2 gives the characteristics of the calls to be rerouted. For each utilization and call mix combination the number of calls to be rerouted and their value are given. The results given in Table 3 address the overall approach. The first four rows give the results of the fast and slow rerouting algorithms respectively. The fifth and sixth rows give the results of the two phase approach applied to the same problems. Note that it shows significant improvement over the slow approach and comes close to achieving the quality levels of the fast approach. Note that the results do degrade somewhat for the highest utilization level. This can be expected since for these levels packing becomes more difficult. In the next two rows results are given when preemption is allowed. This gives an indication of the rather significant potential improvements provided by preemption. However, since we do not have a base case to compare our algorithm against, we cannot evaluate its effectiveness with preemption. The final row gives the running times in seconds. We temper the discussion given below by noting that the experiments were run on a Sun Sparc 10 which may or may not be similar to the hardware environment anticipated in an actual telecommunications network. The running time of the first phase is always lies between 20 seconds and 1 minute. This would certainly not be fast enough for real time operation but is not unreasonable for the scenario where the network's health monitoring system had provided advance warning of a failure. In order to achieve real-time response, alternate approaches, including heuristics, for solving the LP could be considered. The running times for

Calls Rerouted		Mix 1			Mix 2			Mix 3		
		30%	50%	65%	30%	50%	65%	30%	50%	65%
"Fast" Scenario	value	13.76	33.73	13.67	32.37	42.44	10.66	16.67	52.78	15.61
	number	237	503	227	229	360	162	191	359	227
"Slow" Scenario	value	32.46	79.48	56.25	49.79	97.64	32.72	38.29	114.26	48.72
	number	357	869	148	305	630	171	287	636	179
2-Phase w/out preemption	value	32.18	78.03	47.76	49.56	95.83	32.07	38.03	109.25	40.54
	number	355	882	306	304	620	254	284	648	280
2-Phase with preemption	value	32.18	79.16	110.05	49.56	95.83	119.53	38.28	110.80	135.02
	number	355	882	1287	304	620	1118	286	653	1076
Running Time	Phase I	26.8	28.5	30.6	25.1	67.1	34.7	24.5	55.2	39.7
	Phase II	0.25	0.60	0.96	0.25	0.53	0.88	0.25	0.48	0.88

Table 3: Evaluation of the Two-Phase Approach

the second phase algorithm were always less than 1 second. The times reported consisted of the time used to execute *sequentially* the Phase II algorithm at each call origination node. Of course, in an actual implementation these calculations would be carried out in parallel leading to much smaller times. We feel these times are quite encouraging in that they would seem to allow for real-time operation.

Table 4 presents results concerning the quality of the Phase II algorithm. Three sets of experiments were carried out. In the first two, the fast and slow rerouting algorithms were each run. The capacity taken up on each link for calls associated with each call origination node were then allocated to that node. The Phase II algorithm was then given the job of routing calls using only the capacity allocated. This tested the ability of the phase II algorithm to choose a good set of calls and also to pack them into the bandwidth restricted links. The results for this experiment are given in the first 8 rows of Table 4. Note that the phase II algorithm generally did significantly better than the fast algorithm and generally came close to achieving the results of the phase II approach. In the 9th and 10th rows, the LP value is compared with the value achieved in Phase II (the Phase I value is an upper bound on the Phase II value). Note

Calls Rerouted		Mix 1			Mix 2			Mix 3		
		30%	50%	65%	30%	50%	65%	30%	50%	65%
“Fast” Scenario	value	13.76	33.73	13.67	32.37	42.44	10.66	16.67	52.78	15.61
	number	237	503	227	229	360	162	191	359	227
Phase II Algorithm	value	14.90	37.67	16.54	32.19	46.35	10.68	18.14	60.64	20.53
	number	228	496	234	220	353	167	172	388	226
“Slow” Scenario	value	32.46	79.48	56.25	49.79	97.64	32.72	38.29	114.26	48.72
	number	357	869	148	305	630	171	287	636	179
Phase II Algorithm	value	32.20	76.41	56.21	48.95	96.33	32.07	38.03	109.25	48.42
	number	312	802	161	271	582	176	252	597	183
Phase I LP value		32.46	79.50	56.29	49.79	97.65	32.77	38.29	114.40	50.27
Phase II value		32.18	78.03	47.76	49.56	95.83	26.21	37.35	106.80	40.54

Table 4: Effectiveness of the Phase II Packing Algorithm

that for the medium and low utilization cases, the Phase II algorithm came very close to achieving the LP value. For the high utilization it was not as close. This, again, can be explained by the fact that for these levels packing becomes more difficult. Of course, further investigations would be required to determine how much of the gap is due to the quality of the relaxation and how much is due to the quality of the heuristic.

### 3.2 Evaluation of the Call Preemption

In this section we present an evaluation of call preemption algorithm. Suppose after the work of the two-phase algorithm was completed a set of rerouted calls  $\mathcal{K}_r$  was obtained together with their routes. Clearly,  $\mathcal{K}_r \subset \mathcal{K}$ . The set of the new routes can be described by introducing notation similar to (1):

$$\text{for all } k \in \mathcal{K}_r, e \in \mathcal{A}: \quad \sigma_{em} = \begin{cases} 1 & \text{if call } k \text{ is carried by arc } e \\ 0 & \text{otherwise} \end{cases} \quad (31)$$

Calls Rerouted		Mix 1			Mix 2			Mix 3		
		30%	50%	65%	30%	50%	65%	30%	50%	65%
Upper Bound		32.46	79.50	113.24	49.79	97.65	121.64	38.29	114.40	136.27
Lower Bound	value	31.58	74.94	109.34	47.49	93.89	112.59	37.54	106.40	132.93
	number	318	835	1245	276	588	1084	254	604	1033
Gap		2.8%	6.1%	3.6%	4.8%	4.0%	8.0%	2.0%	7.5%	2.5%

Table 5: Preemption Evaluation

Now, consider the following IP formulation:

$$\text{Maximize} \quad \sum_{k \in \mathcal{K}_r} \hat{c}_{h(k)} b_k y_k - \sum_{m \in \mathcal{M}} \tilde{c}_{h(m)} b_m z_m \quad (32)$$

subject to

$$\sum_{k \in \mathcal{K}_r} \sigma_{ek} b_k y_k - \sum_{m \in \mathcal{M}} \delta_{em} b_m z_m \leq B_e \quad \text{for all } e \in \mathcal{A}, \quad (33)$$

$$0 \leq y_k \leq 1 \quad \text{and integer} \quad \text{for all } k \in \mathcal{K}_r, \quad (34)$$

$$0 \leq z_m \leq 1 \quad \text{and integer} \quad \text{for all } m \in \mathcal{M}, \quad (35)$$

It can be easily seen that solution to (32–35) is a feasible suboptimal solution to **(IPR)** corresponding to the call rerouting and preemption given by the two-phase algorithm. The output of this integer program (32–35) consists of decisions regarding call refusal (the  $y_k$  variables) and call preemption (the  $z_m$  variables). Thus, in a certain sense, it gives the optimal solution to the call setup process which is carried in a distributed fashion by the network nodes. Then the quality of the two-phase algorithm can be estimated by how close solution of (32–35) is to that of **(IPR)**. As it was noted above, the latter problem can not be solved explicitly but it is possible to estimate the value of its solution from above by solving its LP relaxation which is essentially **(CA)**. The results of the comparison for the previously described set of calls are presented in the table 5. We consider these results to be quite encouraging.

## 4 Extensions

In this section we consider two extensions of the basic model discussed in the paper. The first one deals with call delays in the network. This issue arises frequently when there are delay-sensitive calls and/or high-delay links such as satellite links.

The second extension deals with ATM networks. One characteristic feature of these networks is statistical multiplexing which, in the context of our model, means that the same call might require different effective bandwidth on different links.

### 4.1 Delay Requirements

In the preceding discussion of rerouting, call delay was not considered. Below we show how delay requirements can be incorporated into the call rerouting approach presented in the paper.

Suppose each call  $k$ ,  $k \in \mathcal{K}$  has a maximum delay limit  $d_k$  which cannot be exceeded and each arc  $e$ ,  $e \in \mathcal{A}$  has an associated delay  $L_e$ . Then, using variables  $x_e^k$  introduced in section 1.3, the maximum delay requirement over the path taken by the call  $k$  can be expressed as

$$\sum_{e \in \mathcal{A}} L_e x_e^k \leq d_k \quad \text{for all } k \in \mathcal{K}. \quad (36)$$

The constraint (36) can be then added to the **(IPR)** formulation to make it handle the delay requirements.

After defining aggregate variables  $\tilde{x}_e^i$  as in (20), a new aggregate constraint can be derived from (36) and (20):

$$\sum_{e \in \mathcal{A}} L_e \tilde{x}_e^i \leq \sum_{k: O(k)=i} b_k d_k \quad \text{for all } i \in \mathcal{N}. \quad (37)$$

Equation (37) can be interpreted as the upper bound on the total “volume” occupied by all calls originated at node  $i$ . This constraint is added to the **(CAP)** (or **(CAP')**) formulation which is solved to obtain the capacity allocation on Phase I.

During Phase II the specific path chosen for the call must be delay feasible. It makes sense to consider both the call’s bandwidth required and delay limitations when choosing the order in which to process the calls. As with the existing algorithm, among the calls with the same delay, the FFD criterion indicates that calls with larger bandwidth should be processed first. On the other hand, among the calls with the

same bandwidth, those with the smallest delay limitations should be routed first, since it is more difficult to find a path for those calls in the network. This leads to the idea of designing a heuristic, where calls are processed in order by decreasing value of a special weight function. This weight should increase with the call's bandwidth and decrease with its delay limit. Possible examples are a weighted sum of bandwidth and delay (delay coefficient should be negative) or bandwidth/delay ratio. Additional experiments are necessary to test the heuristics.

After the calls are ordered, the shortest path tree is constructed for the first call in the sorted list to find the min hop path to the destination from the origin satisfying the delay requirement. It can be done by an appropriate implementation of the Bellman–Ford algorithm. Other calls are routed over the same tree (according to their order in the sorted list) if the path from the root (origin) to the destination is feasible for both delay and bandwidth.

## 4.2 More General Handling of Effective Bandwidths

In this section, we consider more specific properties of effective bandwidths. In general it is possible that the effective bandwidth of the call can depend on which link it is routed over (specifically on the link capacity and the number of calls carried by the link). For each call  $k$ , instead of bandwidth  $b_k$  which is the same for all links, we consider bandwidth  $b_k^e$  dependent on a link  $e$  carrying the call. Thus our starting formulation (*IP*) can be easily changed to accommodate this more generic model.

The two-phase approximation algorithm cannot directly handle this case. However, with certain modifications/approximations it can be treated. Since phase I assumes a call's bandwidth is the same for all links, we must use a single surrogate bandwidth for all links to allocate the capacity. This bandwidth could be computed as an average of all real effective bandwidths. The phase II bulk routing algorithm could use the surrogate bandwidth to order the calls. Since the bandwidth labels at the destination nodes can not be used anymore, the algorithm should be modified to recognize whether the call can be routed to its destination. This could result in the increase in the theoretical running time.

## References

- [1] Ahuja, R., T. Magnanti and J. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ (1993).
- [2] Anderson, J., B. Doshi, S. Dravida and P. Harshavardhana, "Fast Restoration of ATM Networks", *IEEE Journal on Selected Areas in Communications*, **12**, No.1, 128-138 (1993).
- [3] Bertsekas, D. and R. Gallager, *Data Networks*, Prentice-Hall, Englewood Cliffs, NJ (1987).
- [4] LeBoudec, J.Y., "The Asynchronous Transfer Mode: a tutorial", *Computer Networks and ISDN Systems*, **24**, 279-309 (1992).
- [5] L.A. Cox, L. Davis, and Y.Qiu, 1991. Dynamic Anticipatory Routing in Circuit- Switched Telecommunication Networks, in *Handbook of Genetic Algorithms*, L.Davis ed., Van Nostrand Reinhold, New York, 1991.
- [6] Gun, L., and R. Guerin. "A Unified Approach to Bandwidth Allocation and Access Control in Fast Packet Switching Networks," INFOCOM '92, 1-12, Italy, 1992.
- [7] Gun, L., and R. Guerin. "A Framework for Bandwidth Management and Congestion Control in High-Speed Networks," to appear in *Computer Networks and ISDN*.
- [8] Kawamura, R., K. Sato, and I. Tokizawa, "Self-Healing ATM Networks Based on Virtual Path Concept", *IEEE Journal on Selected Areas in Communications*, **12**, No.1, 120-127 (1993).
- [9] M. Laguna, F. Glover, 1993. Bandwidth Packing: A Tabu Search Approach, *Management Science* **39**, 492-500.
- [10] S. Martello, P. Toth, 1990. Knapsack Problems, John Wiley & Sons, 1990.
- [11] R.O. Onvural, *Asynchronous Transfer Mode Networks: Performance Issues*, Artech House, Inc., 1994.
- [12] Park, K., S. Kang, S. Park, "An Integer Programming Approach to the Bandwidth Packing Problem", *manuscript*, (1994).
- [13] Parker, M. and J. Ryan, "A Column Generation Algorithm for Bandwidth Packing", *Telecommunication Systems*, **2**, 185-195 (1994).