

# Topology Design and Bandwidth Allocation in ATM Nets

MARIO GERLA, MEMBER, IEEE, JOSÉ AUGUSTO SURUAGY MONTEIRO, STUDENT MEMBER, IEEE,  
AND RODOLFO PAZOS, MEMBER, IEEE

**Abstract**—In emerging network technologies designed to support a variety of services, it is common to find that the packet switching service is implemented on top of a facility network. For example, in typical narrowband ISDN architectures, the packet switches installed at some of the central offices (CO's) are interconnected with trunks derived from an underlying channelized facility network. Likewise, in future broadband ISDN's, the ATM switches will be connected with trunks obtained from an underlying "pool" of fiber facilities interconnected by digital cross connect systems (DCS).

In this paper, we address the design of a P/S network embedded into a backbone facility network. We formulate the problem as a network optimization problem where a congestion measure based on the average packet delay is minimized, subject to capacity constraints posed by the underlying facility trunks. The variables in this problem are the routing on the "express" pipes (i.e., the channels that interconnect the P/S modes), and the allocation of bandwidth to such pipes. We present an efficient algorithm for the solution of the above problem and apply it to some representative examples. We show that for some test cases, the congestion measure is substantially reduced with respect to the values obtained when the embedded topology is kept identical to the backbone topology. We also discuss dynamic reconfiguration schemes where the embedded topology is periodically adjusted to track the fluctuations in traffic requirements.

## I. INTRODUCTION

**I**N emerging network technologies designed to support a variety of services, it is common to find that the packet switching service is implemented on top of a facility network. For example, in typical narrowband ISDN architectures, the packet switches installed at some of the CO's are interconnected with trunks derived from an underlying channelized facility network. The same trend is observed in some private networks (e.g., Netrix) which combine transparent (i.e., C/S) and packet services on T1 facilities: here, the packet channels are multiplexed on the T1 trunks together with the other services [1]. Finally, in future broadband ISDN's, the ATM switches will be connected with multiples of DS3 trunks. These trunks will be obtained from an underlying "pool" of fiber facilities interconnected by digital cross connect systems (DCS) [2].

The ability to reconfigure a customer network dynam-

cally is a well-known advantage of digital cross connect systems and has been reported extensively in the published literature [3], [4]. Most of the previous studies, however, have been based on transparent networks providing circuit-switched channels of various rates (from voice grade to DS3) between pairs of user sites. The main goals in the design of such systems were the dynamic network reconfiguration following trunk failures, and the reassignment of trunks to applications following a predefined time schedule, or on a reservation basis, or in response to sudden traffic changes.

In this study, we are not concerned with the configuration of the transparent, circuit-switched type network. Rather, we are interested in the packet-switched network built on top of the facility network. We want to exploit the flexibility of DCS in order to obtain a more efficient design and operation of the packet network.

Starting with the design, one notices that channel assignment and topology optimization in "embedded" packet-switched networks differs from that of traditional P/S networks. In traditional designs, we minimize total trunk cost subject to delay constraints. Here, the facility network is given, and therefore trunk cost is fixed (at least for the short term). The problem thus becomes one of optimally configuring the P/S network topology and capacities, within the constraints set by the underlying facility network. Furthermore, this reconfiguration can be carried out dynamically, and can be "tuned" to traffic fluctuations.

To illustrate the point, consider the network shown in Fig. 1. From the original (backbone) topology, several embedded topologies can be derived. The embedded topology of Fig. 2(a) is identical to the backbone topology, whereas the topology of Fig. 2(b) has introduced a number of "express pipes" between remote nodes. Express pipes reduce the number of intermediate hops along the path, and thus reduce store and forward delay and nodal processing overhead.

Express pipes also reduce the number of packet switch terminations. Note that topology 2(a) requires 192 packet switch terminations, while topology 2(b) (which has more express pipes) requires 180 terminations. Using a fully piped network with a direct pipe (of capacity 4, say) also between nodes *B* and *D* will further reduce the number of terminations to 172. This is a very important point since current trends in transmission and processing costs indi-

Manuscript received January 18, 1989; revised May 23, 1989. This work was supported by the State of California and Pacific Bell under a MICRO Grant, by NSF under Grants INT 85-16798, by DARPA under Grant MDA 903-87-0063, and by CAPES under Grant 6886/84-12.

M. Gerla and J. A. Suruagy Monteiro are with the Department of Computer Science, University of California, Los Angeles, CA 90024.

R. Pazos is with the Departamento Sistemas de Información, Instituto de Investigaciones Eléctricas, Cuernavaca, Morelos, Mexico.

IEEE Log Number 8929811.

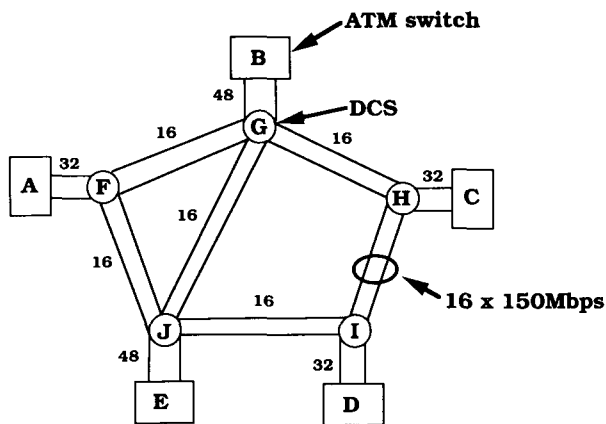


Fig. 1. Backbone topology.

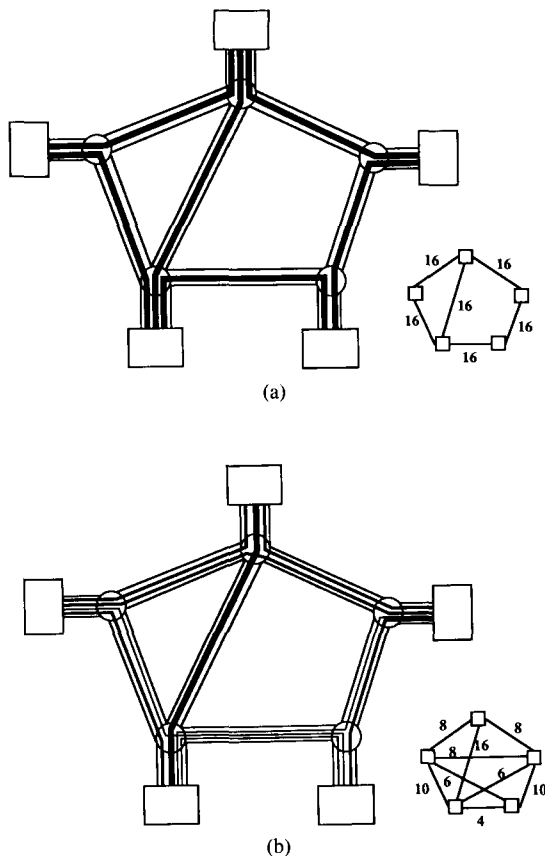


Fig. 2. Embedded topologies.

cate that terminations costs will soon dominate the cost of fiber trunks. Thus, adding more express pipes will imply reducing overall costs.

Another important advantage of express pipes is that of simplifying the congestion control problem. In fact, when a pipe becomes congested, the offending source(s) can be immediately identified and slowed down. In contrast, in a purely meshed packet network, it is often very difficult to trace the sources that cause internal congestion, let alone control them.

There are also drawbacks in the configuration of fully piped (i.e., fully connected) embedded topologies. For example, in networks with a large number of nodes, the

150 Mbit/s pipes may become underutilized when dedicated to individual node pairs, and the advantages of statistically multiplexing several sessions on the same trunk (or trunk group) may be lost. On the other hand, the efficient use of groups of 150 Mbit/s channels in parallel requires the development of complex, multilink protocols [5]. Thus, a good balance must be sought between express pipes and large trunks.

From the network management point of view, the DCS provides added flexibility in that it permits us to dynamically tailor the topology to traffic demands. This "topology tuning" is of particular interest in broadband packet networks implemented using ATM techniques. In fact, ATM nets are stripped of most of the congestion and flow control procedures found in conventional networks in order to improve switch throughput. If there is a mismatch between offered traffic pattern and network topology, congestion would be inevitable. The problem can be alleviated by dynamically tuning the topology to traffic pattern.

In this paper, we address the design of a P/S network embedded into a facility network structure. We formulate the problem as a network optimization problem where a congestion measure based on the average packet delay is minimized, subject to capacity constraints posed by the underlying fiber trunks. The variables in this problem are the routing on the express pipes (i.e., the channels that interconnect the P/S nodes) and the allocation of bandwidth to such pipes.

We present an efficient algorithm for the solution of the above problem and apply it to some representative examples. We show that for some test cases, the performance measure is substantially reduced with respect to the values obtained when the embedded topology is kept identical to the backbone topology.

We also discuss dynamic reconfiguration schemes where the embedded topology is periodically adjusted to track the fluctuations in traffic requirements.

## II. STATIC OPTIMIZATION

We assume that the backbone facility network has already been defined (i.e., number and location of DCS switches, interoffice fiber trunks, etc.). Generally, this facility will be partitioned among many services, public and private, operational and experimental (e.g., telephony, video distribution, private P/S nets, ISDN, B-ISDN, etc.). In particular, one may envision the presence of several ATM nets (public and private) sharing these facilities.

In our study, we will assume that for the ATM net under consideration, a number of interoffice trunk facilities have been reserved *a priori*. Also, the number and location of the ATM switches has been defined (the ATM switches may or may not be colocated with DCS switches). We refer to the reserved set of interoffice fiber trunks and associated DCS's as the *backbone* topology, as opposed to the *embedded* topology which is derived from the backbone one using DCS switching (see Figs. 1 and 2 for an example).

The first step in the design of an ATM net is the design of the backbone topology. This will be carried out at network installation. It will take into account current and future traffic demands, availability of fiber facilities, charges for such facilities, fairness in the allocation of fiber resources among various services and user subnets, etc. This design problem is very complex since it must account for many factors, some of which are not known very accurately *a priori* (e.g., traffic demands are difficult to forecast before network implementation). Thus, this design cannot be expected to be very accurate and should be periodically reviewed. However, corrections to the backbone topology design require reallocation of facilities from one service to another and, possibly, installation of new fiber facilities, all of which involve substantial lead times. Thus, we may refer to this design phase as *long-term planning*.

In this paper, we assume that the backbone topology is given, and thus a set of trunks is reserved to a particular ATM net. We concern ourselves with the problem of mapping the embedded topology into the backbone topology. This mapping may need to be revised very frequently, in part to correct the imprecisions inherent in the long-term plan, and in part to overcome traffic fluctuations. As pointed out before, the embedded topology design is very different from the conventional P/S net topology design in that we have a new set of capacity constraints stating that the sum of the capacities of the embedded links multiplexed on a backbone trunk cannot exceed the capacity of the trunk itself. This new feature makes it necessary to develop new network design tools, as reported in this paper.

We will distinguish between two types of reconfiguration procedures: *medium-term reconfiguration* and *topology tuning*. We envision that medium-term reconfigurations may involve a major change in embedded network topology, and may not be transparent to users (i.e., some connections may get interrupted and later resumed). Medium-term reconfiguration may take place with a frequency of months or weeks.

Topology tuning, on the other hand, should involve only minor perturbations in the embedded topology, and should be user transparent. It could be carried out on an hourly basis, or even more frequently, to overcome short-term traffic imbalances and in general render the network more robust to congestion.

The basic methodology that we propose for embedded network design is the same for both medium-term configuration and topology tuning. The latter, in fact, can be viewed as an incremental implementation of the basic algorithm.

In the sequel, we first introduce the notation and terminology and define the delay model. Then, we formulate the static (i.e., medium-term) design problem, define the optimality conditions, and propose an algorithm for its solution. We illustrate the static design algorithm with some simple examples. Finally, we discuss the topology tuning problem.

### III. NOTATION AND MODELS

#### A. Model for the Backbone Network

The backbone network will be modeled using a graph  $G = (V, A)$  where  $V = \{v_1, v_2, \dots, v_N\}$  is the set of vertices (nodes) and  $A = \{a_1, a_2, \dots, a_M\}$  is the set of arcs (edges). Each DCS or ATM node is represented by one vertex, and each fiber trunk is represented by one arc (for each direction of transmission).

If arc  $a_m$  is incident out of  $v_i$  and into  $v_j$ , it is denoted by  $a_m(i, j)$  or simply  $a(i, j)$ . Each arc  $a_m$  has a capacity  $C_m$  (bits/second). Similarly, each vertex  $v_i$  has a capacity  $\bar{K}_i$  (bits/second) equal to the switching capacity of the switch node represented by  $v_i$ . Typically, in an ATM switch, the capacity is equal to the sum of the input trunk speeds. DCS nodes are assumed to have infinite capacity.

The graph  $G$  associated with the backbone net will be called *first-order graph* (or simply, *graph*), and each of its arcs will be called *first-order arc* (or simply, *arc*).  $G$ , the first-order arc capacities, and the node capacities fully describe the original backbone network. Paths can be defined on  $G$  in the usual way [6, pp. 2-3] and will be called *first-order paths* (or simply, *paths*).

#### B. Model for the Embedded Network

Recall that an embedded link is a concatenation of channels on several trunks. This fact suggests the use of paths for representing embedded links. Therefore, let us consider all the possible paths from  $v_i$  to  $v_j$ :  $\pi(i, j; 1)$ ,  $\pi(i, j; 2)$ ,  $\dots$ ,  $\pi(i, j; M_{ij})$ , where  $\pi(i, j; n)$  denotes the  $n$ th path from  $i$  to  $j$ .

Each path  $\pi(i, j; n)$  defines an arc on another graph  $\bar{G} = (V, \bar{A})$ ; this arc will be denoted by  $\bar{a}(i, j; n)$ . Notice that  $\bar{G}$  is a fully connected graph and  $\bar{a}(i, j; 1)$ ,  $\bar{a}(i, j; 2)$ ,  $\dots$ ,  $\bar{a}(i, j; M_{ij})$  define multiple arcs from  $i$  to  $j$ . If  $\bar{a}(i, j; n)$  is the  $p$ th element of  $\bar{A}$ , it may be denoted simply by  $\bar{a}_p$ ; and  $\pi(i, j; n)$  may be denoted by  $\pi_p$ .  $\bar{M}$  will denote the number of elements of  $\bar{A}$  (i.e., the number of links). Each arc  $\bar{a}_p$  has a capacity, which will be denoted by  $\bar{C}_p$ .

$\bar{G}$  will be called *second-order graph*, and each arc  $\bar{a}_n$  will be called *second-order arc*. When there is no possibility for confusion, they will be called simply graph and arc.  $\bar{G}$  and the second-order arc capacities fully describe the embedded network.

#### C. Delay Analysis

In conventional packet-switched networks, average packet delay is the key performance measure (to be minimized) when the total budget is fixed. In ATM networks, average packet delay is not a very meaningful optimization variable since packet delays (transmission, queueing, switching) are extremely small. For example, for a 69 byte packet size (the proposed B-ISDN standard) and for 150 Mbit/s trunk speeds, the sum of transmission, queueing, and switching delays at an ATM node will be on the order of 10-20  $\mu$ s, much less than the propagation delay (about 20 ms on a cross-country connection). If anything, propagation delays should be minimized; unfortunately, prop-

agation delays depend on the geographical distribution of user sites, and are only marginally affected by network topology layout.

More important than delay in ATM nets is the buffer overflow probability at switching nodes. Very strict requirements on packet loss (on the order of  $10^{-6}$ – $10^{-9}$ ) are often imposed since some applications (e.g., compressed video) are extremely sensitive to packet loss and cannot rely on end-to-end retransmissions. Accurate models for buffer overflow in typical switching fabrics (buffered and/or unbuffered) under general traffic patterns are very complex and do not yield closed form results.

Given the complexity of the switch models on the one hand, and the need for a simple formulation to use in our network optimization model on the other, we have resorted to approximations. Since buffer overflow probability is related to average trunk queue lengths, and the latter are related to average delay, we have chosen average delay as the indirect measure of buffer overflow probability (to be minimized). To further simplify things, we have used an  $M/M/1$  model for trunk queueing delays (although packet length is fixed rather than exponentially distributed). The validity of these approximations is supported by experimental evidence (in the design and optimization of conventional packet-switched networks) that the optimal routing and topology solution is rather insensitive to the particular shape of the delay versus trunk load curve, and is only affected by its asymptotic value, i.e., the trunk capacity (which is generally the same for all models) [7].

As for the ATM switches, these have finite capacity (e.g., for a banyan fabric, the capacity is proportional to the number of input trunks). Thus, switch capacity constraints must be introduced in the model. Furthermore, some switches (e.g., buffered banyan fabrics) are subject to internal buffer overflow (in addition to output trunk buffer overflow). This internal buffer overflow probability is a function of the aggregate traffic transiting through the switch. In order to account for internal buffer overflow, we have chosen to model the switch as a single  $M/M/1$  queue. Note that this model also incorporates the finite switch capacity constraint as a penalty function.

In carrying out the overall average delay analysis, the usual assumptions for P/S networks are made [8]. These assumptions permit us to model ATM nets as a network of independent  $M/M/1$  queues. One should mention that the proposed approach could actually handle more complex models for both trunks and nodes (e.g., the node could be represented by a multiserver queue or by a network of queues). The additional complexity of the model, however, would make the overall solution approach more time consuming.

In order to complete the picture, we also need to describe the offered traffic. For each source  $i$  and each sink  $j$ , there is a flow of packets offered to  $i$  and destined for  $j$ . This flow defines a commodity  $com(i, j)$ , which is considered distinguishable from the flow of any other source-sink pair. Usually, the commodities have an order; if

$com(i, j)$  is the  $k$ th element in such an ordering,  $com_k$  will be equivalent to  $com(i, j)$ .

The average arrival rate of packets offered to  $i$  and destined for  $j$  will be denoted by  $\gamma_k$ . The average packet length (in bits) will be denoted by  $1/\mu$ , and it is assumed to be the same for all the source-sink pairs. Thus, the average offered flow of commodity  $k$  is  $r_k = \lambda_k/\mu$ . (Its units are bits/second.)

Using the results from [9], the average packet delay excluding propagation delay is given by

$$T = \frac{1}{\lambda} \left[ \sum_{m=1}^{\bar{M}} \frac{\bar{f}_m}{\bar{C}_m - \bar{f}_m} + \sum_{n=1}^S \frac{\mu \bar{f}_n}{\bar{K}_n - \mu \bar{f}_n} \right]$$

where  $\bar{M}$  is the number of embedded links,  $\bar{C}_m$  is the capacity of link  $m$ ,  $\bar{f}_m$  is the aggregate flow on link  $m$ ,  $Q$  is the number of commodities,  $\lambda = \mu \sum_{k=1}^Q r_k$ ,  $S$  is the number of ATM switches,  $\bar{K}_n$  is the throughput capacity of switch  $n$ , and  $\bar{f}_n$  is the aggregate flow through switch  $n$ . (The units of  $T$  are seconds, the units of  $\lambda$  and  $\bar{K}_n$  are packets/second, and the units of  $\bar{C}_m$  and  $\bar{f}_m$  are bits/second.)

Without loss of generality, we can simplify the above delay expression by redefining the nodal capacity as  $\bar{C}_n = \bar{K}_n/\mu$ . We can also model each ATM switch as two nodes of infinite capacity joined by a link of capacity  $\bar{C}_n$ . The delay expression then becomes

$$T = \frac{1}{\lambda} \sum_{m=1}^{\bar{M}} \frac{\bar{f}_m}{\bar{C}_m - \bar{f}_m}$$

where  $\bar{M}$  now includes also the equivalent intranode links.

The capacity  $\bar{C}_m$  of an embedded link is generally a discrete variable (e.g., multiples of 150 Mbit/s). To simplify the optimization problem, we assume that  $\bar{C}_m$  is a continuous variable. This implies that at the end, the continuous solution must be "discretized." In practice, this approximation does not seem to be too restrictive, however, since each embedded link will typically consist of several channels in parallel (i.e., fine granularity).

#### IV. PROBLEM DEFINITION AND FORMULATION

##### A. Definition

Let us consider an embedded network like the one in Fig. 2(b), which is defined on the backbone network also reported in Fig. 1. Let us assume that the topology and the trunk capacities of the backbone network are known. Suppose also that an initial embedded topology is defined. This is clearly a major assumption which will be later relaxed. The external traffic offered to the embedded network is known and will be routed according to some routing policy, which defines the distribution of the flow among the existing pipes.

Since the initial embedded topology has been already defined, the key design variables are bandwidth allocation and routing. More precisely, in the bandwidth allocation and routing problem, we want to choose embedded link bandwidths and packet routing such that

- the average packet delay is minimized,
- the capacity of each trunk (of the backbone network) is not exceeded by the aggregate bandwidth of all pipes using the trunk, and
- all the traffic requirements are satisfied without exceeding pipe bandwidths.

### B. Formulation

Let us consider a first-order path  $\pi_u$  in the backbone network; its arc-path vector is defined as  $\mathbf{p}_u = (p_{u1}, p_{u2}, \dots, p_{uM})^T$  where

$$p_{um} = \begin{cases} 1, & \text{if } a_m \in \pi_u \\ 0, & \text{otherwise.} \end{cases}$$

Let us now consider the graph  $\bar{G}$  associated with the embedded network. Although  $\bar{G}$  is not a graph in a strict sense (since it contains multiple arcs), we can still define paths on it. These paths (called second-order paths) are defined in a way similar to that of the first-order paths.

For each commodity  $k$ , there are, in general, several second-order paths from its source  $s_k$  to its sink  $t_k$ . Let  $\bar{\pi}(k; 1), \bar{\pi}(k; 2), \dots, \bar{\pi}(k; N_k)$  denote these paths. These paths can be described using arc-path vectors. For example, second-order path  $\bar{\pi}(k; n)$  corresponds to the following arc-path vector expression:  $\bar{\mathbf{p}}(k; n) = (\bar{p}(k; n)_1, \bar{p}(k; n)_2, \dots, \bar{p}(k; n)_{\bar{M}})^T$  where

$$\bar{p}(k; n)_u = \begin{cases} 1, & \text{if } \bar{a}_u \in \bar{\pi}(k; n) \\ 0, & \text{otherwise} \end{cases}$$

and  $\bar{M}$  is the number of second-order arcs.

If our network is to move commodities correctly from sources to sinks, for each commodity  $k$  there must be some flow of this commodity on the paths  $\bar{\pi}(k; 1), \bar{\pi}(k; 2), \dots, \bar{\pi}(k; N_k)$ . Then let  $\bar{x}(k; n)$  denote the flow of commodity  $k$  on  $\bar{\pi}(k; n)$ .

With the preceding definitions, we can state concisely the bandwidth allocation and routing problem as follows.

*Given:* Topology and arc capacities of first-order graph, topology of second-order graph, and offered traffic.

*Objective:* Minimize average packet delay, i.e.,

$$\min z = \frac{1}{\lambda} \sum_{u=1}^{\bar{M}} \frac{\bar{f}_u}{\bar{C}_u - \bar{f}_u}. \quad (1)$$

*Variables:* Capacities of second-order arcs and routing on the second-order graph.

*Constraints:*

$$\sum_{u=1}^{\bar{M}} \bar{C}_u \mathbf{p}_u \leq \mathbf{C} \quad (2)$$

$$\sum_{n=1}^{N_k} \bar{x}(k; n) = r_k \quad \forall k \quad (3)$$

$$\sum_{k=1}^Q \sum_{n=1}^{N_k} \bar{x}(k; n) \bar{\mathbf{p}}(k; n) = \bar{\mathbf{f}} \quad (4)$$

$$\bar{\mathbf{f}} \leq \bar{\mathbf{C}} \quad (5)$$

$$\bar{\mathbf{C}} \geq \mathbf{0}, \bar{\mathbf{f}} \geq \mathbf{0}, \bar{\mathbf{x}} \geq \mathbf{0} \quad (6)$$

where  $\mathbf{C} = (C_1, C_2, \dots, C_M)^T$  is the vector of first-order capacities,  $\bar{\mathbf{C}} = (\bar{C}_1, \bar{C}_2, \dots, \bar{C}_{\bar{M}})^T$  is the vector of second-order arc capacities,  $\bar{\mathbf{f}} = (\bar{f}_1, \bar{f}_2, \dots, \bar{f}_{\bar{M}})^T$  is the vector of second-order arc flows, and  $\bar{\mathbf{x}} = (\bar{x}(1; 1), \bar{x}(1; 2), \dots, \bar{x}(Q; N_Q))^T$  is the vector of second-order path flows.

Constraint (2) expresses the condition that the capacity of each first-order arc must not be exceeded by the aggregate capacity of all the second-order arcs that use the first-order arc. Equation (3) expresses the condition that the flow of each commodity  $k$  carried on all the paths  $\bar{\pi}(k; 1), \bar{\pi}(k; 2), \dots, \bar{\pi}(k; N_k)$  must be equal to the flow  $r_k$  of commodity  $k$  offered to its source.

Equation (4) states that the aggregate flow on each second-order arc must be equal to the sum of all commodity flows on all the second-order paths that use the arc. Constraint (5) states that the aggregate flow on each second-order arc must not exceed its capacity.

Notice that  $z \rightarrow \infty$  when  $\bar{C}_u - \bar{f}_u \rightarrow 0^+$ . This means that if we are in the region where  $\bar{\mathbf{f}} \leq \bar{\mathbf{C}}$ , the objective function will prevent us from getting out of this region. In this case, we say that constraint (5) is included in the objective function as a penalty function, and therefore, it can be removed from the formulation.

### V. OPTIMALITY CONDITIONS

Constraints (2)–(4) and (6) define a convex set, but  $z$  is not a convex function with respect to both  $\bar{\mathbf{C}}$  and  $\bar{\mathbf{f}}$ . (This can be shown easily by observing that the quadratic form of  $z$  is not positive semidefinite [10, p. 157].) Since this problem does not satisfy the requirements for a unique minimum [10, p. 150], it may have more than one local minimum. In fact, experimental results show that the problem has several local minima [11, pp. 195–202].

$(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*)$  is a local minimum if and only if the following conditions are satisfied.

- 1)  $\bar{\mathbf{C}}^*$  and  $\bar{\mathbf{f}}^*$  are feasible, i.e., they satisfy (2)–(6).
- 2) There are no feasible directions of descent at  $(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*)$ , i.e.,
  - a) there do not exist  $\bar{\mathbf{C}}$  and  $\bar{\mathbf{f}}$  such that

$$\nabla z(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*) \cdot [(\bar{\mathbf{C}}, \bar{\mathbf{f}}) - (\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*)] < 0 \quad (7)$$

subject to (2)–(4) and (6)

where  $\nabla z(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*)$  is the gradient of  $z$  with respect to  $(\bar{\mathbf{C}}, \bar{\mathbf{f}})$  evaluated at  $(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*)$ , and  $(\bar{\mathbf{C}}, \bar{\mathbf{f}})$  denotes the column vector

$$\begin{bmatrix} \bar{\mathbf{C}} \\ \bar{\mathbf{f}} \end{bmatrix},$$

- b) for every nonzero vector  $\mathbf{y}$

$$\mathbf{y}^T \nabla^2 z(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*) \mathbf{y} > 0$$

where  $\nabla^2 z(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*)$  is the Hessian matrix of  $z$  evaluated at  $(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*)$  [10, p. 10], and  $\mathbf{y}$  is a column vector whose number of elements equals the number of elements of  $(\bar{\mathbf{C}}, \bar{\mathbf{f}})$ . (In presenting this condition, mathematical formality

was sacrificed in favor of simplicity. The formal description can be found in [10, pp. 28–51].)

Conditions 1) and 2a) are first-order conditions (i.e., they guarantee that the solution is feasible and corresponds to a stationary point). 2b) is a second-order condition, and guarantees that the solution is indeed a local minimum (rather than a maximum or a saddle point).

## VI. ALGORITHM

Because of the reason mentioned in the first paragraph of Section V, there is no efficient algorithm that guarantees convergence to global minimum. Therefore, we propose the following algorithm, which finds good local minima.

First, a *random* initial solution is found, whence a local minimum is obtained (using Algorithm 1, which will be described later). This process is repeated several times with different random initial feasible solutions. Finally, the local minimum with the lowest objective value is chosen as the final solution.

The local minima required by this algorithm can be obtained using the first-order conditions 1) and 2a). Condition 2b) was not included in Algorithm 1 because it is extremely time consuming for large-scale networks.

The first-order conditions suggest the use of Frank-Wolfe's steepest descent method [12] for finding a local minimum for problem (1)–(6). In this method, given a feasible solution  $(\bar{C}^{K-1}, \bar{f}^{K-1})$ , we can find a feasible direction of descent by solving the following problem:

$$\min \nabla_z(\bar{C}^{K-1}, \bar{f}^{K-1}) \cdot (\bar{C}, \bar{f}) \quad (8)$$

subject to (2)–(4) and (6).

Observe that this problem can be separated into the following two problems:

$$\min \nabla_{\bar{C}} z(\bar{C}^{K-1}, \bar{f}^{K-1}) \cdot \bar{C} \quad (9)$$

$$\text{s. t. } \sum_{u=1}^{\bar{M}} \bar{C}_u p_u \leq C \quad (10)$$

$$\bar{C} \geq \mathbf{0} \quad (11)$$

and

$$\min y = \nabla_{\bar{f}} z(\bar{C}^{K-1}, \bar{f}^{K-1}) \cdot \bar{f} \quad (12)$$

$$\text{s. t. } \sum_{n=1}^{N_k} \bar{x}(k; n) = r_k \quad \forall k \quad (13)$$

$$\sum_{k=1}^Q \sum_{n=1}^{N_k} \bar{x}(k; n) \bar{p}(k; n) = \bar{f} \quad (14)$$

$$\bar{f} \geq \mathbf{0}, \bar{x} \geq \mathbf{0}. \quad (15)$$

If  $\bar{C}^\#$  minimizes problem (9)–(11) and  $\bar{f}^\#$  minimizes (12)–(15), then  $(\bar{C}^\#, \bar{f}^\#) - (\bar{C}^{K-1}, \bar{f}^{K-1})$  is a feasible direction of descent.

It can be shown [13] that problem (12)–(15) can be transformed into

$$\min y = \sum_{k,n} \bar{l}(k; n) \bar{x}(k; n) \quad (16)$$

$$\text{s. t. } \sum_{n=1}^{N_k} \bar{x}(k; n) = r_k \quad \forall k \quad (17)$$

$$\bar{x} \geq \mathbf{0} \quad (18)$$

where

$$\bar{l}(k; n) = \sum_{u=1}^{\bar{M}} \bar{p}(k; n)_u \frac{\partial z(\bar{C}^{K-1}, \bar{f}^{K-1})}{\partial f_u}$$

is the cost of transporting one unit of commodity  $k$  on the second-order path  $\bar{\pi}(k; n)$ .

Finally, problem (16)–(18) can be decomposed into  $Q$  problems:

$$\min y = \sum_{n=1}^{N_k} \bar{l}(k; n) \bar{x}(k; n) \quad (19)$$

$$\text{s. t. } \sum_{n=1}^{N_k} \bar{x}(k; n) = r_k \quad (20)$$

$$\bar{x}(k; 1), \bar{x}(k; 2), \dots, \bar{x}(k; N_k) \geq 0 \quad (21)$$

(for  $k = 1, 2, \dots, Q$ ).

Problem (19)–(21) is a minimum path problem on a second-order graph. Dijkstra's minimum path algorithm ([14], [15], and [6, pp. 11–13]) cannot be applied directly to this problem because the graph contains multiple arcs. Thus, we must first reduce the second-order graph by replacing each multiple arc by a simple arc, and then apply Dijkstra's algorithm. (Details of the modified algorithm can be found in [11, pp. 146–147].)

Before presenting the algorithm, it is convenient to introduce the following.

*Definition:* A minimum path solution is one in which the requirement of each commodity is routed on the shortest path. More formally, for each commodity  $com_k$ , let  $\bar{\pi}(k; n - k)$  denote the shortest second-order path from the source to the sink of  $com_k$ . The minimum path solution is  $(\bar{f}^\#, \bar{x}^\#)$  where

$$\bar{x}(k; n)^\# = \begin{cases} r_k, & \text{if } n = n_k \\ 0, & \text{otherwise} \end{cases} \quad \forall k, n$$

and  $\bar{f}^\# = \sum_{k=1}^Q \bar{x}(k; n_k)^\# \bar{p}(k; n_k)$ .

**Algorithm 1** (Algorithm for Bandwidth Allocation and Routing in the Logical Network)

*Input:* Topology and arc capacities of first-order graph, arc-path vectors of second-order arcs, traffic requirements, initial feasible solution  $(\bar{C}^0, \bar{f}^0)$ , and tolerance  $t$ .

*Objective Function:* Equation (1).

*Step 0:* Set  $K = 0$  and  $z^{\text{old}} = \infty$ .

*Step 1:* Set  $K = K + 1$ . Find the vector  $C^\#$  that minimizes (9)–(11). (This problem can be solved using the revised simplex method [16].) And find the vector  $\bar{f}^\#$  that minimizes (12)–(15). (This problem can be solved by constructing a minimum path solution  $(\bar{f}^\#, \bar{x}^\#)$  for the second-order graph where the cost of each second-order arc  $\bar{a}_u$  is  $\bar{l}_u = \partial z(\bar{C}^{K-1}, \bar{f}^{K-1}) / \partial f_u$ .)

*Step 2:* Find the value  $\alpha^*$  that minimizes  $z[\alpha(\bar{\mathbf{C}}^\#, \bar{\mathbf{f}}^\#) + (1 - \alpha)(\bar{\mathbf{C}}^{K-1}, \bar{\mathbf{f}}^{K-1})]$ . This optimum may be obtained by any convenient line search method (such as the golden section technique [17]).

*Step 3:* Set  $(\bar{\mathbf{C}}^K, \bar{\mathbf{f}}^K) = \alpha^*(\bar{\mathbf{C}}^\#, \bar{\mathbf{f}}^\#) + (1 - \alpha^*)(\bar{\mathbf{C}}^{K-1}, \bar{\mathbf{f}}^{K-1})$ . If  $z(\bar{\mathbf{C}}^{K-1}, \bar{\mathbf{f}}^{K-1}) - z(\bar{\mathbf{C}}^K, \bar{\mathbf{f}}^K) \leq t$ , go to 4, a potential local minimum has been found; otherwise, go to 1.

*Step 4:* If  $z^{\text{old}} - z(\bar{\mathbf{C}}^K, \bar{\mathbf{f}}^K) \geq t$ , go to 5. Otherwise, set  $(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*) = (\bar{\mathbf{C}}^{\text{old}}, \bar{\mathbf{f}}^{\text{old}})$  if  $z^{\text{old}} \leq z(\bar{\mathbf{C}}^K, \bar{\mathbf{f}}^K)$ , set  $(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*) = (\bar{\mathbf{C}}^K, \bar{\mathbf{f}}^K)$  if  $z^{\text{old}} > z(\bar{\mathbf{C}}^K, \bar{\mathbf{f}}^K)$ , and stop; a local minimum has been obtained.

*Step 5:* Set  $(\bar{\mathbf{C}}^{\text{old}}, \bar{\mathbf{f}}^{\text{old}}) = (\bar{\mathbf{C}}^K, \bar{\mathbf{f}}^K)$  and  $z^{\text{old}} = z(\bar{\mathbf{C}}^{\text{old}}, \bar{\mathbf{f}}^{\text{old}})$ . Find the vector  $\bar{\mathbf{C}}^a$  that solves

$$\begin{aligned} \min z &= \frac{1}{\lambda} \sum_{u=1}^{\bar{M}} \frac{\bar{f}_u^{K-1}}{\bar{C}_u - \bar{f}_u^{K-1}} \\ \text{s. t. } &\sum_{u=1}^{\bar{M}} \bar{C}_u p_u \leq C \\ &\bar{\mathbf{C}} \leq \bar{\mathbf{f}}^{K-1}. \end{aligned}$$

$\bar{\mathbf{C}}^a$  can be found using Algorithm 1, described in [18], with  $\bar{\mathbf{C}}^{K-1}$  as initial feasible solution). Set  $(\bar{\mathbf{C}}^K, \bar{\mathbf{f}}^K) = (\bar{\mathbf{C}}^a, \bar{\mathbf{f}}^{K-1})$  and go to 1.

*Output:* The local minimum is  $(\bar{\mathbf{C}}^*, \bar{\mathbf{f}}^*, \bar{\mathbf{x}}^*)$  where  $\bar{\mathbf{x}}^*$  is such that  $(\bar{\mathbf{f}}^*, \bar{\mathbf{x}}^*)$  satisfies (3), (4), and (6).

## VII. RANDOM INITIAL SOLUTIONS

As mentioned before, the optimization problem is non-convex and, therefore, a downhill technique like the one proposed in this paper cannot guarantee that the global optimum is reached starting from an arbitrary initial solution. Our approach to finding a cost-effective, suboptimal solution consists of applying the algorithm to several, randomly chosen initial solutions, and generating several local minima. The overall best local minimum represents our suboptimal solution.

Two choices must be made to generate a starting point: the embedded topology and a feasible flow and bandwidth assignment.

For the initial embedded topology, an obvious choice would be to include all possible  $M(M - 1)$  links (fully connected graph), and then let the optimization procedure pick the most cost-effective ones. This approach, however, would be computationally too cumbersome. An alternate approach (which is easy to automate) consists of inserting direct links between node pairs with highest throughput requirements. Thus, traffic between such nodes will not burden intermediate store and forward processors. Traffic between minor node pairs, on the other hand, will be required to cover multiple store-and-forward hops.

As for the selection of a random, feasible  $(\bar{\mathbf{C}}, \bar{\mathbf{f}})$  solution, there exist many possibilities. One approach, described below, consists of using a few iterations of the algorithm itself, with randomly chosen cost parameters. More precisely, consider the following problem:

$$\begin{aligned} \min w &= \sum_{u=1}^{\bar{M}} (\bar{C}_u + \bar{f}_u) \\ &\text{subject to (2)-(6)}. \end{aligned} \quad (22)$$

Since the objective function and all the constraints are linear functions, this is a linear programming problem. Therefore, a feasible solution for this problem can be obtained using Phase I of the revised simplex method [16].

Let  $(\bar{C}_i, \bar{f}_i)$  denote a feasible solution to this problem. Notice that  $(\bar{C}_i, \bar{f}_i)$  is also a feasible solution for problem (1)-(6), and consequently, it can be used as an initial feasible solution for Algorithm 1.

So far, we have a method for obtaining one initial solution for problem (1)-(6), but we need several random feasible solutions. Starting with the feasible solution  $(\bar{C}_i, \bar{f}_i)$ , we can obtain a *random* feasible solution  $(\bar{C}_r, \bar{f}_r)$  by solving the following problem:

$$\begin{aligned} \min z &= \sum_{u=1}^{\bar{M}} \frac{\bar{\eta}_u}{\bar{C}_u - \bar{f}_u} \\ &\text{subject to (2)-(6)} \end{aligned}$$

where the  $\bar{\eta}$ 's are parameters whose values are selected randomly. This problem can be solved using Algorithm 1.

A more extensive discussion of random feasible solutions can be found in [11, pp. 155-157].

## VIII. TOPOLOGY TUNING

Topology tuning refers to the adjustment of bandwidth allocation and routes in the embedded network in order to overcome congestion caused by an unfavorable traffic pattern or by a failure. This activity is carried out on line and is closely related to network monitoring and traffic measurement functions. Since topology tuning may be carried out fairly frequently, user transparency is an important consideration.

Our proposed approach is to use an incremental version of the previous algorithm. To illustrate the procedure, assume that there is a sudden surge of traffic between nodes  $i$  and  $j$ , which creates congestion at some of the intermediate switches. Let us further assume that there is no current direct embedded link between  $i$  and  $j$ . Prompted by network congestion alarms, the topology tuning procedure is run at the NCC (Network Control Center). First, the NCC will examine the current topology and traffic to identify the "pressure points" ( $i$  and  $j$ ). Then, the topology tuning algorithm is run using as a starting solution the current traffic flows, and a topology augmented by one or more embedded links between pressure points. The algorithm will iterate and will redistribute bandwidth and traffic from the existing links to the newly introduced link(s) in order to reduce delays and thus eliminate congestion.

Ideally, one would like to immediately open the new links and reassign bandwidth from the old to the new links. However, for transparency, a gradual expansion is

required. This can be accomplished by opening initially a link of small capacity and adding new calls to this link. As calls are cleared on the old path, bandwidth is reallocated to the new path.

The entire procedure could be automated and in fact run as part of Network Management. It should be clear in fact that in a large network, the dynamic adjustment of bandwidth is far too complex to be carried out manually. The incremental application of the bandwidth and routing algorithm appears to be a very effective solution to the problem.

## IX. EXPERIMENTAL RESULTS

The bandwidth and routing algorithm was implemented and tested on several representative cases. The implementation can handle problems with 60 nodes, 100 trunks, 200 embedded links, and 1000 commodities.

The execution times (on a VAX 11/780) and the number of iterations for 5 different problems are listed in Table I. The number of iterations has the form  $N_1 + N_2$ , where  $N_1$  represents the number of iterations needed to obtain the initial feasible solution, and  $N_2$  represents the number of iterations for getting the optimal solution. The number of local minima requested for each problem was five. Thus, the time and number of iterations listed in Table I are the aggregate quantities for the five minima.

In Table II, we compare the delays of the embedded, optimized topology to those of the original topology (which is identical to the backbone topology). For this experiment, five different backbone topologies ( $i = 1, 2, \dots, 5$ ) each with 6 nodes and 16 links were generated. Each network had 18 commodities, and the traffic requirements of two commodities were  $\rho$  times larger than those of the other commodities. Several values of the requirements ratio ( $\rho = 1, 2, 5, \dots, 100$ ) were explored. The purpose of this experiment was to investigate the relationship between the requirement imbalance (which is reflected in the ratio  $\rho$ ) and the improvement that can be obtained with an optimized embedded topology. One intuitively expects that the larger the imbalance, the more cost-effective it will be to introduce direct pipes between the heavy producing node pairs. In this experiment, the packet processing power of the switches (i.e., the value  $\bar{K}_n$  in Subsection C) was set to infinity in order to focus on the input traffic imbalance effect.

At the beginning of the experiment, for each network  $i$ , the initial embedded topology was generated by adding eight carefully chosen pipes to the original topology. Most pipes were chosen so as to connect directly the sources and sinks of the two commodities with higher requirements. Theoretically, we could have added all the possible links (i.e., fully connected topology). The algorithm would then have kept the good ones and discarded the bad ones. However, in order to reduce processing time, only the most promising links were added. Finally, for each network  $i$  and each value of  $\rho$ , the delay  $\bar{T}_{i\rho}$  of a good local minimum (the best among five local minima) was generated.

TABLE I  
EXECUTION TIME VERSUS PROBLEM SIZE

Trunks	Embedded Links	Nodes	Commodities	Execution Time (sec.)	Number of Iterations
30	40	10	40	170.1	77+282
30	40	10	40	254.0	46+321
30	40	10	40	339.5	79+439
30	40	10	40	300.2	98+394
30	40	10	40	172.7	75+154

TABLE II  
DELAY REDUCTION OBTAINED WITH EMBEDDED TOPOLOGY OPTIMIZATION

Requirement Ratio	Embedded/Backbone Delays				
	Network				
	1	2	3	4	5
1	1.29	0.81	1.05	0.79	0.53
2	1.27	0.72	0.96	0.70	1.15
5	1.10	0.59	0.93	0.56	0.50
10	0.91	0.48	0.89	0.40	0.31
20	0.42	0.40	0.51	0.27	0.25
50	0.38	0.24	0.69	0.17	0.21
100	0.32	0.86	0.68	0.12	0.31

The values of  $\rho$  are listed on the first column of Table II, and the ratios  $\bar{T}_{i\rho}/T_{i\rho}$  are listed on the other columns. Networks 2-5 were generated randomly, and the sources and sinks of the commodities of all the five networks were generated randomly.

The experiment shows that when traffic requirements are unbalanced, packet delays can be drastically reduced by optimizing the embedded topology.

*Topology Tuning Example:* Now we want to illustrate the proposed topology tuning procedure applied to a simple example. Let us consider the backbone network of Fig. 1. The labels attached to each trunk or switch node correspond to their capacities. We can assume that these capacities are multiples of a basic data rate channel such as 150 Mbit/s. The DCS nodes are assumed to be able to connect the input channels to any permutation of output channels.

For this example, a total of 28 candidate pipes and 102 distinct paths that use these pipes were taken into account. For example, to directly connect switches A and C we considered 3 pipes described by the following paths: A-F-G-H-C, A-F-J-G-H-C, and A-F-J-I-H-C.

Fig. 3 shows the best pipe capacities and flows for the traffic pattern of Table III. This embedded network has a delay of 1.373 and was obtained from a set of 5 local minima, for an aggregated 30 min processing time on a SUN 3/280. Each pipe and switch is labeled with its capacity and flow (in parentheses), while pipe widths are proportional to their capacities. Note that while pipe flows are unidirectional, switch flows are bidirectional (i.e., incoming and outgoing flows).

Now, suppose that the traffic between nodes B and D is increased from 0.0 to 3.0 while the remaining traffic is kept fixed. Simply routing the resulting traffic through the existing pipes would increase the delay to 2.364 (or a 72 percent increase).

A complete optimization for the new traffic pattern produced the embedded network of Fig. 4 that has a 1.717

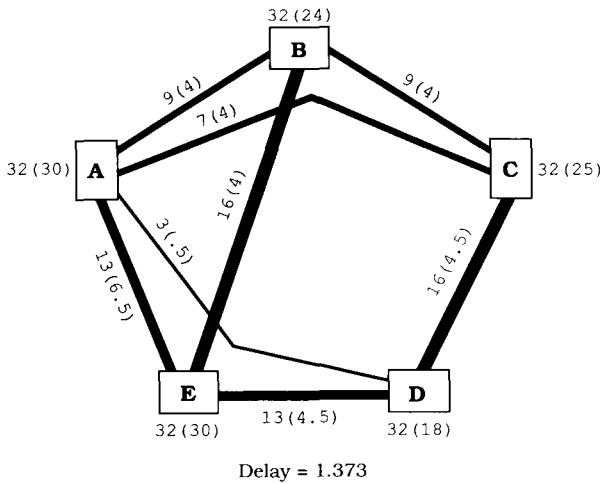


Fig. 3. Best network for original traffic.

TABLE III  
ORIGINAL TRAFFIC MATRIX

	A	B	C	D	E
A	-	4.0	4.0	0.5	6.5
B	4.0	-	4.0	0.0	4.0
C	4.0	4.0	-	4.0	0.5
D	0.5	0.0	4.0	-	4.0
E	6.5	4.0	0.5	4.0	-

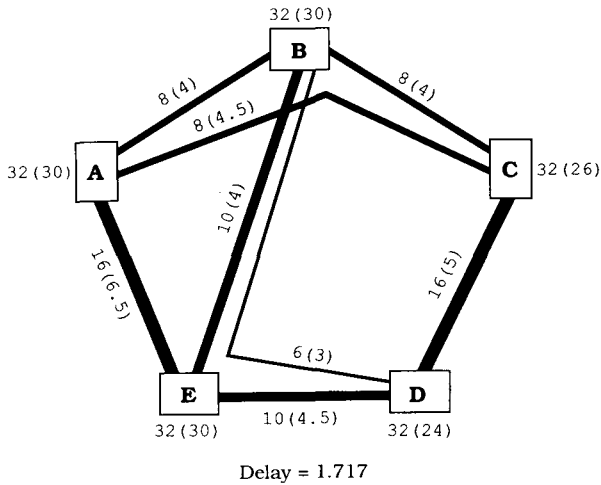
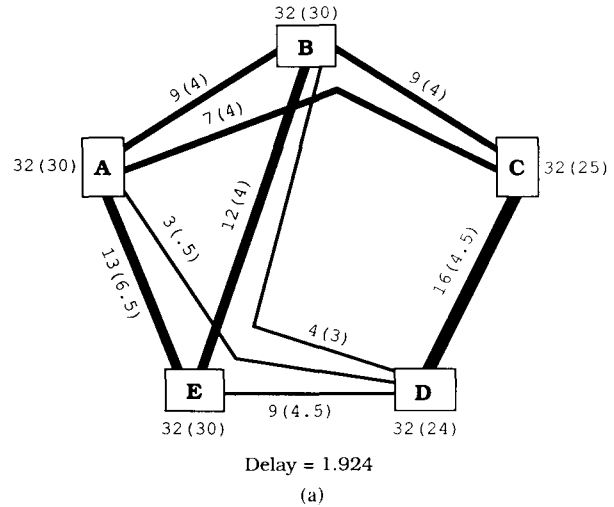


Fig. 4. Best network after traffic change.

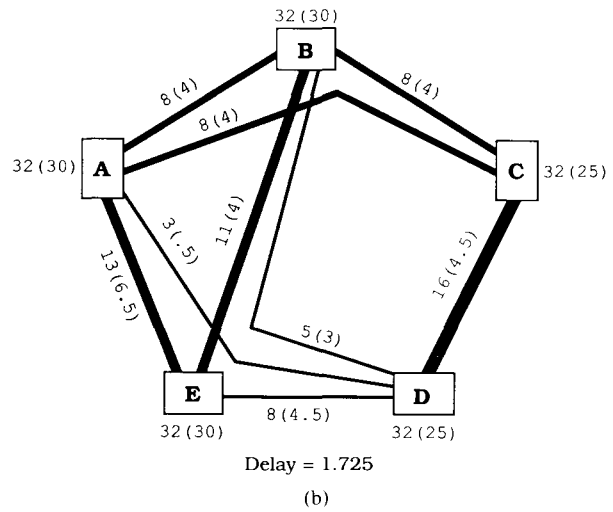
cost. This cost represents a 27 percent improvement over the nonreconfigured one, or only a 25 percent increase from the cost before the traffic change. Note that in this solution, the A-D pipe was eliminated and the A-D traffic was rerouted through switch C. In addition, a direct pipe between B-D was created and all the new B-D traffic was routed through it.

Finally, Fig. 5 illustrates the topology tuning procedure presented in Section VIII. Since there was no direct connection between nodes B and D, we introduced one, following path B-G-J-I-D (see Fig. 1) with capacity 4 [Fig. 5(a)]. In order to create this pipe, we removed 4 capacity units from each one of the B-E and D-E pipes.

Next, an initial routing for the new traffic is found, using the newly introduced pipe. The resulting configura-



(a)



(b)

Fig. 5. Topology tuning: (a) initial solution, (b) discrete-capacities solution.

ration has a 1.924 delay. After one iteration of the bandwidth and routing algorithm, followed by a pipe discretization, the network of Fig. 5(b) was obtained. It has a delay of 1.725 (or a 10 percent improvement over the initial solution) which is just 0.5 percent higher than the sub-optimal solution of Fig. 4.

Again, our experiment confirms our initial intuition that logical “pipes” are particularly cost effective in unbalanced traffic patterns.

### X. CONCLUSIONS

In future B-ISDN's, the network manager will have the ability to reconfigure the ATM network topology fairly dynamically and to “tune” it to time varying traffic patterns by using DCS switches and relying on an existing fiber trunk infrastructure. This reconfiguration is particularly important in ATM nets in order to compensate for the lack of efficient internal flow control mechanisms. Network reconfiguration implies the definition of an embedded topology, the assignment of bandwidth to embedded links, and the choice of routing. These decisions cannot be carried out manually in a large net; thus, appropriate network design tools are required.

In this paper, we have presented an efficient algorithm for network reconfiguration. This algorithm can be run off line (medium term planning); or it can be run on line to overcome short term traffic fluctuations (topology tuning). Since the underlying mathematical programming problem is very complex, yielding several minima, the algorithm incorporates several heuristics (generation of initial embedded topology; random generation of starting bandwidth and flow solutions, etc.) that render the solution numerically tractable. The experimental results show that the proposed technique can lead to substantial delay improvements over nonoptimized topologies.

There is opportunity to extend this work in many directions. First, since the solutions are suboptimal, it would be of interest to establish lower bounds on delay, so that we know how far (or how close) we are to the optimal solution, and therefore can decide when to stop exploring more local minima. Other optimization techniques such as *simulated annealing* [19] could also be considered. As for the objective function, we have used delay in our method. However, more accurate measures which reflect buffer overflow and account for the particular characteristics of the integrated traffic (voice, data, video) should be investigated. Regarding the "topology tuning" version of the algorithm, more work is required in the development of the interface between the network monitoring facility and the network optimization program (e.g., how to measure traffic requirements; how to measure node/trunk congestion, etc.). Also, more work is needed on the transparent implementation of the solution (i.e., bandwidth allocation and routing). Finally, if several ATM subnets (public and/or private) are embedded in the same fiber facility, the issue of how to fairly partition (and, perhaps, dynamically share) the common facility among subnets must be investigated.

In this paper, we have dealt only with bandwidth allocation and routing. It should be clear, however, that these problems are the first step toward more complex design problems involving the optimal sizing and location of ATM switches.

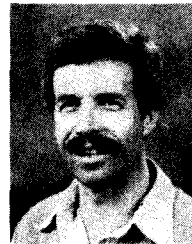
#### ACKNOWLEDGMENT

The authors wish to thank Dr. V. R. Saksena, AT&T Bell Laboratories, for his helpful comments and suggestions during the preparation of the manuscript.

#### REFERENCES

- [1] Netrix Corporation, "Netrix 1-ISS system overview," internal document, 1986.
- [2] R. G. Addie and R. W. Warfield, "Bandwidth switching and new network architectures," in *Proc. 12th Int. Teletraffic Congr.*, Torino, Italy, June 1988, pp. 2.3iiA.1-7.
- [3] C. H. Yang and S. Hasegawa, "FITNESS—Failure immunization technology for network service survivability," in *Proc. GLOBECOM '88*, Hollywood, FL, Nov. 1988, pp. 1549-1554.
- [4] P. J. Zanella, "Customer network reconfiguration applications utilizing digital cross-connect systems," in *Proc. GLOBECOM '88*, Hollywood, FL, Nov. 1988, pp. 1538-1543.
- [5] A. Pattavina, "Multichannel bandwidth allocation in a broadband packet switch," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1489-1499, Dec. 1988.
- [6] S. Even, *Graph Algorithms*. Potomac, MD: Computer Science Press, 1979.

- [7] M. Gerla, "The design of store-and-forward (S/F) networks for computer communications," Ph.D. dissertation, Comput. Sci. Dep., Univ. Calif., Los Angeles, 1973.
- [8] L. Kleinrock, *Communication Nets—Stochastic Message Flow and Delay*. New York: Dover, 1972.
- [9] —, *Queueing Systems, Vol. II: Computer Applications*. New York: Wiley-Interscience, 1976.
- [10] M. Avriel, *Nonlinear Programming Analysis and Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1976.
- [11] R. A. Pazos, "Evaluation and design of integrated packet switching and circuit switching computer networks," Ph.D. dissertation, Comput. Sci. Dep., Univ. Calif., Los Angeles, Dec. 1983.
- [12] B. Martos, *Nonlinear Programming Theory and Methods*. Amsterdam and Oxford: North-Holland, 1975, pp. 238-248.
- [13] R. A. Pazos and M. Gerla, "Express pipe network design," in *Proc. 1986 Int. Zurich Seminar Digital Commun.*, Zurich, Switzerland, Mar. 1986, pp. 61-68.
- [14] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.
- [15] E. M. Reingold et al., *Combinatorial Algorithms Theory and Practice*. Englewood Cliffs, NJ: Prentice-Hall, 1977, pp. 341-345.
- [16] K. Murty, *Linear and Combinatorial Programming*. New York: Wiley, 1976, pp. 187-189.
- [17] M. S. Bazaraa and C. M. Shetty, *Nonlinear Programming Theory and Algorithms*. New York: Wiley, 1979, pp. 257-259.
- [18] R. A. Pazos and M. Gerla, "Express pipe networks," in *Proc. Global Telecommun. Conf.*, Miami, FL, Dec. 1982, pp. B2.3.1-5.
- [19] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, May 1983.



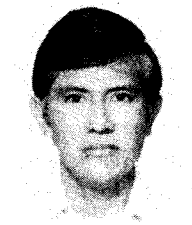
**Mario Gerla** (M'75) received the graduate degree in engineering from the Politecnico di Milano, Milano, Italy, in 1966, and the M.S. and Ph.D. degrees in engineering from the University of California, Los Angeles, in 1970 and 1973, respectively.

From 1973 to 1976 he was Network Planning Manager at Network Analysis Corporation, NY, and led several computer network design projects for both government and industry. From 1976 to 1977 he was with Tran Telecommunications, Los Angeles, CA, where he participated in the development of an integrated packet and circuit network. In 1977 he joined the University of California, Los Angeles, and is now a Professor in the Department of Computer Science. His research interests include the design and control of distributed computer communication systems and networks, and the development of protocols for high-speed local area networks.



**José Augusto Suruagy Monteiro** (S'87) was born in Recife, Brazil, in 1957. He received the B.S.E. degree (magna cum laude) in electrical engineering from Universidade Federal de Pernambuco, Recife, Brazil (UFPE), in 1979 and the Master's degree in electrical engineering (digital systems) from Universidade de São Paulo, São Paulo, Brazil (USP), in 1982.

In 1983 he joined the Faculty of the Computer Science Department at UFPE. Since 1985 he has been on leave of absence at U.C.L.A. where he is currently a Ph.D. candidate, specializing in the design of broadband ISDN's.



**Rodolfo Pazos** (S'81-M'88) was born in Tampico, Mexico, in 1951. He received the B.S.E.E. and M.S.E.E. degrees from the Instituto Politécnico Nacional, Mexico, in 1976 and 1978, and the Ph.D. degree in computer science from U.C.L.A. in 1983.

Currently he is Project Leader with the Instituto de Investigaciones Eléctricas, where he has been involved in the development of databases, information systems, and software for the electrical industry. He worked for the Mexican Ministry of Communications and Transportation on the development of routing algorithms for data transmission networks. His research interests are information systems, distributed databases, and computer communications networks.