Efficient Support for Client/Server Applications Over Heterogeneous ATM Networks

Ornan (Ori) Gerstel, *Associate Member, IEEE*, Israel Cidon, *Senior Member, IEEE*, and Shmuel Zaks

***Abstract*—We present a new network design problem that is applicable for designing virtual paths (VP's) in an asynchronous transfer mode (ATM) network to efficiently support client/server applications. We present several alternatives for the solution, compare their properties, and focus on a novel "greedy" solution, which we prove to optimize certain important criteria (namely, the network overhead for a request/response and the utilization of bandwidth and routing table resources). We also present simulation results that demonstrate the performance and scalability of our solution. In addition, we propose a new efficient bandwidth allocation scheme which is tailored for client/server applications over ATM networks.**

***Index Terms*—ATM network, client/server paradigm, network design, virtual path design, virtual path routing.**

I. Introduction

A. Asynchronous Transfer Mode Networks

THE *asynchronous transfer mode* (ATM) is the network architecture standard proposed for broadband Integrated Services Digital Networks (B-ISDN). This architecture is accepted by a large array of vendors and standard organizations (ITU and the ATM Forum), and is thoroughly described in the literature (e.g., [8], [10], [16], and [20]).

ATM is based on small fixed-size packets termed *cells*. Each cell is switched independently, based on two small routing fields at the cell's header, called the *virtual channel identifier* (VCI) and *virtual path identifier* (VPI). Since ATM is connection-oriented, the cells are routed on predetermined routes in the network that must be set up prior to their usage for data transfer.

Routing in ATM is hierarchical in the sense that the VCI of a cell is ignored by most of the network nodes in the path traversed by the cell; these nodes route the cell based on its VPI field alone. This scheme effectively creates two types of predetermined simple routes in the network: those based on VPI's [virtual paths (VP's)] and those based on VCI's [virtual channels (VC's)]. VP's may be viewed as virtual links, connecting (possibly remote) switches as neighboring nodes in a virtual network, while the route of VC's may be viewed as a concatenation of multiple VP's. We refer to the number of VP's that are used by a VC along its route as the *VP hops* of the VC. The reason for requiring a low number of VP hops per VC are discussed later. In ATM, VC's are typically used for connecting network users, while VP's are used for routing VC's and for simplifying network resource management.

Three types of ATM switching nodes have been considered and implemented.

CX Nodes: CX nodes are switching nodes that cannot switch the cell based on its VPI alone (since they either use both VPI and VCI for their switching decision or use the VCI alone, as is often the case in ATM local area network (LAN) switches). Clearly, these nodes must be involved in the setup of every VC that passes through them, and, thus, they are termed "VC switches" (CX's for short).

PX Nodes: PX nodes are nodes that use only the VPI field for switching cells. Such nodes do not allow switching of a VC from one VP to another (since they do not refer to the VCI field at all), and switch VP's exclusively; hence, they are termed "VP switches" (PX for short).

PCX Nodes: PCX nodes are nodes that route some cells by using the VPI alone and others by using both the VPI and VCI. An example of such a switch may be found in [7]. When a cell arrives at this switch, its VPI is first examined and used as an index into a VP routing table (as in PX's). The appropriate entry contains a new value for the cell's VPI and a port into which the cell is switched. If the entry contains a special "null" value, the VCI is used (together/without the VPI) as an index into a VC routing table in which a new VPI and VCI for the cell are found, and in which a port to send the cell to is specified (as in CX's). This mechanism enables some VP's to terminate at the node (by having "null" in the entry pointed by their VPI value) and some VP's to pass through the node. We term these nodes "PCX's" since they switch both VP's and VC's.

The VC and VP concepts received much attention in the communication literature, yet the problem of how VP's should be laid out in a given network topology was addressed only in a few works [1], [2], [6], [13], [14], [17]. These works assume only one type of node (usually PCX), a fact which—besides reducing the practical significance of the model—substantially simplified it and the obtained results. In this work we assume that the network is heterogeneous, i.e., that all three node types coexist in it, as is expected to be the case in realistic large-scale networks.

B. The Client/Server Paradigm

One of the major factors in the success of the penetration of ATM into data communication markets is its ability to adequately support existing network applications such as Internet

Manuscript received December 9, 1996; revised March 28, 1998; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Ammar.

O. Gerstel was with the Computer Science Department, Technion, Haifa, Israel. He is now with the Optical Networking Group, Tellabs Operations, Hawthorne, NY USA (e-mail: ori@tellabs.com).

I. Cidon is with the Electrical Engineering Department, Technion, Haifa, Israel.

S. Zaks is with the Computer Science Department, Technion, Haifa, Israel.

Publisher Item Identifier S 1063-6692(98)05644-1.

1063–6692/98$10.00 © 1998 IEEE

protocol (IP) and Web services, mail applications, file transfer, and distributed applications.

A very common paradigm for network applications is the client/server paradigm. Implemented explicitly in many network applications or implicitly—via remote procedure call (RPC) mechanisms [3]—in distributed operating systems (e.g., [4]), this paradigm is based on a "smart" server application that is accessed by many clients, whenever they need data from the server. The clients issue short request messages (typically less than 1 KB), that are handled by the server, which consequently sends a typically much longer (several megabytes) response [5]. The most prevalent example for this usage is the World Wide Web. Other examples are distributed file servers [e.g., network file server (NFS)], file repositories, databases (in which a query is sent as a request and the relevant data is sent as a response), and name/address servers (such as distributed name server (DNS) or X.500).

Due to the central role that the client/server paradigm plays in existing network applications, it is obvious that it must be properly accommodated by both private and public future ATM networks. There are two major alternatives for the implementation of the paradigm using ATM networks.

1) Use a native connectionless service [19] or an overlay IP network to send both requests and responses encapsulated in datagrams, without need for per-VC setup. However, such a service may not necessarily be available with every ATM network and incurs extra cost and management overheads. More importantly, most such services do not guarantee low data loss probability.
2) Use regular VC's for transferring data from the client to the server and vice versa. This method was implemented in some of the new transport protocols (e.g., versatile message transaction protocol (VMTP) [18]) and we focus on it in the rest of the paper. While most of this work applies for any quality of service (QoS) class, we assume constant bit rate (CBR) and variable bit rate (VBR) traffic for simplicity. Also, while available bit rate (ABR) and unspecified bit rate (UBR) classes may be more suitable for data applications, they do not provide hard service guarantees and entail complex buffer management.

C. Contribution of this Paper

In this paper we discuss methods for supporting client/server applications over an ATM network by setting up semipermanent VP's in the network that enable clients to efficiently set up short-lived VC's each time they wish to submit a request to the server. After the setup, these VC's are used by the client to send a request to the server and by the server to return a response, and are then terminated until the next request. We propose a method for designing the routes of VP's in a given network so as to support such VC's as quickly and as efficiently as possible. Our solution is optimal, both in the setup/teardown overhead for a new VC and its propagation and processing delay, while keeping its resource utilization as low as possible. We also suggest a novel bandwidth allocation scheme that further improves the resource utilization of the solution.

Besides its use for client/server applications, our VP layout design may be used as a building block for more complex general-purpose VP layouts, as demonstrated in [13] for simpler cases. Our greedy algorithm (see Section V-B) is based on a framework suggested in [14]; however, while [14] applies to networks which comprise of PCX's exclusively and clients are assumed to be connected to all network switches, the present solution applies to more realistic (and complex) cases, in which the network comprises all types of switches and clients are connected to a subset of the switches. A preliminary version of this paper appeared in [12].

Similar problems were considered in [1], [6], and [15]. While these formulations apply to a wider setting (not just client/server applications), they are less appropriate for such specific needs and have only heuristic solutions, whereas here we present a proven optimal solution.

The paper is structured as follows. In Section II we present the design problem which stems from supporting client/server applications over ATM and in Section III we explore the spectrum of possible solutions to the problem (including an overview of our approach). In Section IV we compare the possible solutions using numerical simulation results, while in Section V we focus on the formulation of the main problem solved in the paper (namely, the VP layout for client/server applications) and present an optimal algorithm for it (the proof of optimality may be found in the Appendix). We summarize the results in Section VI.

II. The Design Problem

The main focus of this paper is a design problem, in which it is requested to determine a set of VP's and a route for each of them in a given ATM network, so that a given set of clients will be able to set up VC connections to a given server, for the purpose of sending request/response messages. These VP's will enable setting up the VC's as quickly as possible, while not requiring too many network resources.

The design problem for a given client/server application[1] $\mathcal{X}$ is the following.

Input Data:

1) The topology of the ATM network (i.e., the set of switching nodes and the links connecting them) in the form of an undirected simple graph $G = (V, E)$;
2) the type of each switching node (i.e., which node is a PX, CX, or PCX);
3) a subset $CN \subseteq V$ of switches to which the clients of $\mathcal{X}$ are connected, and a node $s \in V$ to which the server is connected;
4) $\mathcal{T}_{\text{delay}}(e)$—the propagation delay of each physical link $e \in E$, including the processing delay of an attached node;
5) $\mathcal{BW}_{\text{avail}}(e)$—the amount of available bandwidth on each link that may be allocated for the purposes of $\mathcal{X}$;
6) $\mathcal{N}_{\text{req}}$—the maximum number of concurrent requests from clients to the server that is supported by $\mathcal{X}$;

[1] Recall that many such applications coexist in the network; however, the design problem focuses on each of them separately.

7) $\mathcal{BW}_{\text{req}}$—the required bandwidth for a single request from a client to the server of $\mathcal{X}$;
8) $\mathcal{BW}_{\text{resp}}$—the required bandwidth for a single response from the server to a client of $\mathcal{X}$;
9) $\mathcal{L}_{\max}$—a maximum quota on the number of entries in any VP routing table, that may be used for $\mathcal{X}$ (termed the *load* on the routing table). This is an important requirement, as the resource of routing entries in each table is limited to 2^{12} (see [8]), and the resource is shared by VP's which support other client/server applications, as well as many other services. In the sequel we show that this resource is easily exhausted, if not carefully allocated.

Feasible Solution: A set of VP's, a route in the network for each such VP and its bandwidth with the following characteristics.

1) The total bandwidth of VP's that share a link e does not exceed the link's bandwidth constraint $\mathcal{BW}_{\text{avail}}(e)$.
2) The VP's have enough bandwidth to support $\mathcal{N}_{\text{req}}$ simultaneous requests from client to server and a single response from server to client (assuming the server controls the bandwidth for responses on a link).
3) The VP's obey the constraints imposed by the different types of switching nodes (e.g., no VP terminates at a PX node).
4) The number of VP's that use any given physical link is bounded by $\mathcal{L}_{\max}$ (see Observation 1).

Optimization Goal: Find a feasible solution which reduces the overhead of the setup of VC's from client to server, by achieving the following two targets.

1) The maximum number of VP hops between any client node to the server's node is minimized. At the end of this section we shall reason why this improves the setup performance and bandwidth utilization in the network.
2) The physical route that each VC takes incurs minimal delay—both propagation, fixed node processing and queuing delays are taken into account. (Variable queuing delays are outside the scope of the paper and should be negligible for the QoS classes discussed herein.) Our results can be easily adapted to any other route related cost model—see discussion in Section III-B-2.

Observation 1: The number of used VP routing entries at a given port processor[2] is equal to the number of incoming VP's that use the attached link. This holds since the VPI of a cell that belongs to such a VP is used by the port processor as an index into its VP routing table.

It is more convenient to think of $\mathcal{L}_{\max}$ as the maximum number of VP's that share a link, rather than a limit on the routing table load.

Observation 2: An important fault tolerance issue that stems from Observation 1 is based on a VP rerouting protocol suggested in [7] for recovering the network from link failures: upon a link failure, the network reroutes VP's that use the faulty link to other paths in the network, thereby achieving a low overhead migration of all VC's that use the faulty link (and thus use the rerouted VP's) to alternative routes. The overhead of this recovery process is clearly proportional to the number of VP's that share the link and is thus bounded by $\mathcal{L}_{\max}$.

There are two reasons for keeping the VP hops per VC low.

1) As discussed below, a feasible solution for most networks relies on setting up VC's on demand, for a specific request/response interaction, after which they are removed from the network until the next request is submitted. As a result, the overhead of a lengthy connection setup due to high VP hop count for such "short-lived" VC's may exceed the necessary time for the data transfer itself.

   The setup overhead of a VC is directly proportional to the number of VP's that are used by the VC along its route (i.e., its VP hops). This stems from the fact that only at the end of each VP, must the network layer software be involved in setting up the VC routing tables in support of the new VC.
2) If the QoS class is CBR or VBR, the fact that a VC uses a VP reduces the available bandwidth on that VP. Therefore, the amount of reserved bandwidth for a VC is proportional to its VP hops.

A technique for reducing the effect of propagation delay on the setup protocol has been suggested [18] in which the setup protocol triggered by the client to set up a VC from the client to the server will also create the VC in the opposite direction (from the server to the client) to carry the response by the server. This technique reduces the setup overhead of two round-trip delays by half. This setup is termed *implicit setup* in [18] and requires that the routes taken by the VC from client to server and vice versa will be along the same route in opposite directions. Such pairs of VC's are termed *full-duplex* VC's; a full-duplex VP is similarly defined. The bandwidth of a full-duplex VC/VP will be denoted by a pair $\langle X, Y \rangle$ where X is the bandwidth from the client to the server and Y is the bandwidth in the opposite direction. Hereafter we discuss only such VC's/VP's, so the term "full-duplex" is omitted.

III. Solutions

A. Simple Approaches

Three straightforward solutions to this problem suit specific applications and small networks but do not scale well for other cases (see Fig. 1 for a pictorial demonstration).

Simple-1: Create a permanent VC from each client to the server (and vice versa)—whenever a client wishes to send a request, it may do so with no setup overhead. The main drawback of this solution is that these VC's must be maintained (e.g., when a physical link fails), a network-management overhead that is not justifiable if they are not frequently utilized. Another major drawback in the case of CBR and VBR traffic is its inefficient bandwidth allocation—each such VC requires allocated bandwidth that cannot be reused by any

[2] We assume here the widely accepted switch architecture of [7], in which the switch includes port processors that connect to the switches' ports. Each port processor has a VP and/or VC routing table, by which it determines how to handle incoming cells.

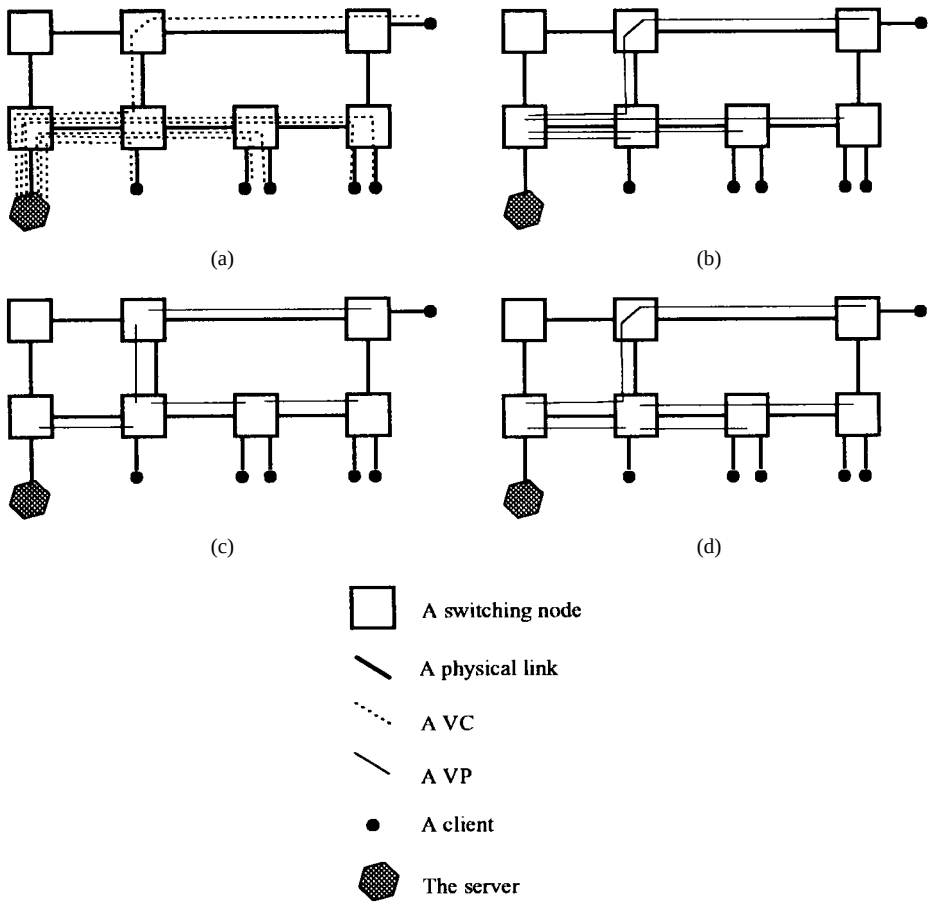

Fig. 1. Solutions to the VP layout problem. (a) SIMPLE-1. (b) SIMPLE-2. (c) SIMPLE-3. (d) Our solution.

other client. This is especially problematic from the server to the client, where the bandwidth requirement is significant.

Simple-2: Create a permanent VP from each client's switching node to the switching node connected to the server (and vice versa)—this solution has the advantage over SIMPLE-1 that the individual VC's from the clients to the server are only created upon request; hence, if many clients are connected to the same switching node, they are supported by the same VP. In this case a VC setup is necessary for every client's request, but this VC involves only a single VP,[3] thus its setup is very efficient. A main disadvantage of this solution is that the allocated bandwidth to each such "direct" VP may be reused only by clients that are connected to the same switching node. Another disadvantage is that VP routing tables at nodes that are close to the server tend to get overloaded; if a client/server application is large (or the servers of numerous such applications are in the same network vicinity), the VP routing tables close to the server are easily filled up entirely with entries of such VP's. As discussed earlier, this also reduces the efficiency of VP-based fault recovery.

Simple-3: Create only VP's that span a single physical link. This solution is optimal in its degree of reuse, since a VP may be shared by all clients for which the chosen physical route includes the appropriate link. This solution results in very long setup times and is thus impractical when the physical network is sparse and VC's traverse a large number of links. It is a viable solution for dense networks.

[3] Assuming there are no CX's en route from the client to the server—no VP can pass through a CX node without terminating at it. Hence, the number of VP hops of a given route is at least the number of CX's included in it.

TABLE I
CHARACTERISTICS OF THE DIFFERENT LAYOUTS. "ENTRIES" REFERS TO THE MAXIMUM NUMBER OF VP ENTRIES PER VP ROUTING TABLE, "BANDWIDTH" REFERS TO MAXIMUM BANDWIDTH PER LINK, AND "HOPS" REFERS TO THE MAXIMUM NUMBER OF VP HOPS PER VC

Solution	entries	bandwidth	hops
(a) SIMPLE-1	None	6	0
(b) SIMPLE-2	4	4	1
(c) SIMPLE-3	1	1	3
(d) Our solution	2	2	2

Example 1: Consider Fig. 1(a)–(c), which demonstrates the three above-mentioned solutions and one of the options enabled by our solution [Fig. 1(d)] on the same network. Table I includes the resources utilization of these solutions. In the table, bandwidth is measured in multiples of $\langle \mathcal{BW}_{\text{req}}, \mathcal{BW}_{\text{resp}} \rangle$ (i.e., 6 stands for a bandwidth of $\langle 6 \cdot \mathcal{BW}_{\text{req}}, 6 \cdot \mathcal{BW}_{\text{resp}} \rangle$). As shown in the next section, our solution may be tuned to produce many different VP layouts, and the solution depicted in the figure is only one option. In the figure we did not mark the delay of the various links; however, we assumed all delays to be equal, except for the delay on the link between the upper left and the lower left switches, which is higher (and accounts for the fact that none of the clients uses the upper left switch).

B. Our Approach

The General Framework: To overcome the shortcomings of the above SIMPLE-2 and SIMPLE-3 solutions, we have devised the following algorithm, which is essentially an intermediate solution between these two extremes. The solution can be tuned to become either of these solutions, but also any intermediate alternative, thereby displaying a tradeoff between the setup time and the amount of necessary resources at every network element (i.e., link, switch). Our solution is based on three main phases.

Phase 1: Given the topology of the network G and the amount of available bandwidth on each link $\mathcal{BW}_{\text{avail}}(e)$, we find a tree that spans all of the client' switching nodes, the server's switching node, and all necessary intermediate nodes. All of the routes of VC's used for the given application will be included in this tree; hence, it is termed the *client/server base tree* (CSBT for short). There are several different sets of properties that may be desirable for a CSBT. Each such set yields a different tree, and the possible options are discussed below. Between these options we have chosen to focus on a CSBT with the following properties.

- The route between any client to the server is as short as possible in terms of its propagation and processing delay.
- The allotted bandwidth ($\mathcal{BW}_{\text{avail}}(e)$) on each tree link e is large enough to accommodate the required bandwidth for all VP's that use e. Since there are at most $\mathcal{L}_{\max}$ such VP's, the total required bandwidth has to multiplied by a $\mathcal{L}_{\max}$ factor.

Phase 2: Given CSBT and the type of each node, we determine a set of VP's that enable connecting each client to the server using the path in CSBT, so that no more than $\mathcal{L}_{\max}$ VP's share any tree link, and so that the number of VP hops between any client to the server is minimal. This phase is the central focus of the paper. In Section V we formally define the problem in this phase and present an optimal greedy algorithm for it.

Phase 3: While the above two phases are performed during a design phase, this phase pertains to the run-time aspects of the service. These aspects include variations in the protocols for setting up (and taking down) VC's [18] and protocols for dynamically changing the design of PHASE 1 and PHASE 2, upon a request of a client to subscribe/unsubscribe from the service.

An overview of the three stages is given below. For sake of brevity, we do not extend the description of PHASE 1 and PHASE 3 beyond this overview (with the exception of the NP-completeness proofs in the Appendix), and focus in the rest of the paper on PHASE 2.

Overview of Phase 1: In this paper we assume that the base subgraph that spans the server and the clients is a tree. We do so for the following reasons.

1) Trees are easy to understand and manage, and are very suitable for such applications. This makes them a plausible choice for service providers. Current telecom access networks are predominantly trees for similar reasons.
2) It was proven in [14] that if the base is of general topology, then the problem of PHASE 2 is NP-complete. It is, therefore, challenging to find efficient solutions for it. No good approximation algorithms are known to this problem.
3) In Section III-C we present a novel efficient bandwidth allocation scheme tailored for client/server applications, for which the optimal spanning subgraph is clearly a tree.

There are a few different options for a CSBT, depending on the resource constraints and cost model.

CSBT-1—Minimize Delay and Cost C, Where $C \propto \sum_{\text{nodes: } a,\, b} BW(a, b)\text{Dist}(a, b)$: This is the CSBT mentioned earlier that is assumed in the rest of the paper due to its good characteristics, reasonable cost model, and algorithmic feasibility. To find such a tree, take the given graph G and remove from it links that do not have enough bandwidth to support $\mathcal{L}_{\max}$ VP's with $\langle \mathcal{BW}_{\text{req}}, \mathcal{BW}_{\text{resp}} \rangle$ bandwidth each. In the remaining graph find paths with shortest delay from the server to all of the clients (e.g., using the Bellman–Ford algorithm).

This algorithm clearly finds shortest delay paths (among the feasible ones). Also, since the bandwidth from the client nodes to the server does not depend on the tree, the cost is reduced if the sum of distances is minimized, and, hence, if each distance from a client node to the server is separately minimized.

CSBT-2—Minimize Delay and Cost C, Where $C \propto \sum_{\text{tree edge: } e} \text{Cost}(e)$: This tree is hard to find algorithmically. Its NP-hardness is proven in the Appendix subsection A. In addition, its cost model which ignored bandwidth considerations seems less reasonable.

CSBT-3—Minimize Cost C Only, Where $C \propto \sum_{\text{tree edge: } e} \text{Cost}(e)$: If the previous tree characterization is simplified by removing delay considerations, then the problem is identical to the Steiner tree problem, which is NP-hard, but many heuristics are known to solve it with an approximation factor of two [21].

CSBT-4—Any Tree for Which the Bandwidth Used Per Link Does Not Exceed the Available Bandwidth $\mathcal{BW}_{\text{avail}}(e)$: While CSBT-1 only considers links that can support $\mathcal{L}_{\max}$ VP's, it may be useful to also consider links with less free bandwidth ($\mathcal{BW}_{avail}(e) < \langle \mathcal{L}_{\max} \cdot \mathcal{BW}_{\text{req}}, \mathcal{L}_{\max} \cdot \mathcal{BW}_{\text{resp}} \rangle$). Such an approach may yield a feasible solution in places where CSBT-1 cannot find one. The main drawbacks of this approach are: 1) it yields solutions that tightly fit into the available bandwidth and may leave no room for expanding the CSBT to support more clients and 2) it is algorithmically hard to find; the NP-hardness of the algorithm is proven in subsection A of the Appendix.

Overview of Phase 2: This phase is based on a greedy procedure called `find_layout`, which is described in detail in Section V-B. `find_layout` takes as an input the tree CSBT, an upper bound on the number of allowable VP hops (h) for a client to the server, and an upper bound on the allowed load at any tree link ($\mathcal{L}_{\max}$); it returns as an output the layout of VP's, or announces that it cannot find a layout that meets the above bounds. In Section V-C it is proven that if there exists a layout that meets these bounds, `find_layout` will find it.

The procedure views CSBT as rooted at the server's node, and gradually extends VP's from the leaves toward the root. At

every node v that `find_layout` handles, if there are clients attached to v, `find_layout` creates a new VP from it (that includes at this point only the link from v to its parent) and strives to extend all of the VP's from v's subtree (including the new VP) toward the root. However, in most cases the VP's do not reach the root for several reasons.

- Whenever `find_layout` encounters a VC switch (i.e., v is a CX node), all of the VP's in its subtree must be stopped at v (by definition of a CX).
- If the load on the link from the current node v to its parent (i.e., toward the root) exceeds $\mathcal{L}_{\max}$, `find_layout` reduces the number of VP's that share this link by stopping some of them at v (if v is a PCX) or by stopping some of them at the closest descendants of v that are PCX's (if v is a PX).

Whatever the reason, if a VP is stopped before reaching the root, then the VP hops from clients that use the VP to access the server is increased. The crux of this algorithm is in the choice of VP's to be stopped, so that the maximum VP hops from client to server is minimally increased.

Overview of Phase 3: The run-time protocols for client/server connections over ATM can be divided into two categories: protocols that use the VP layout for request/response messages and protocols that change the layout.

The first category includes the VC implicit setup/teardown protocols. These protocols are essentially identical to regular VC setup/teardown protocols, with the exception that implicit setup must allocate entries in VC routing tables in both directions (i.e., from client to server and vice versa)—along the lines of [18]. In addition, bandwidth for a request and response $\langle \mathcal{BW}_{\text{req}}, \mathcal{BW}_{\text{resp}} \rangle$ is allocated in the appropriate directions. Teardown may be performed either explicitly or implicitly (see [18] for details).

The second category includes changes in the VP layout, caused by changes in the set of clients that may potentially access the server. A new signaling mechanism is necessary for informing the network that a client wishes to subscribe to (or unsubscribe from) the service provided by the server. If there are no subscribed clients previously connected to the client's switching node, then a new VP must be added (or an old VP must be removed) from the network, an operation that involves updating CSBT and changing the VP layout (of PHASE 2). As far as CSBT is concerned, it is easy to obtain a new optimal CSBT with only local changes in the network; as far as the VP layout is concerned, however, due to the properties of `find_layout`, there is no simple way to obtain a new optimal layout without global changes, which are impractical during the ongoing operation of the network. Therefore, a practical compromise would be to update the layout locally on every such event, reducing its optimality, and to occasionally reconstruct the whole layout in an optimal manner using PHASE 2 (whenever resource utilization of the current layout deteriorates severely).

C. Bandwidth Allocation Schemes

In the above discussion we did not take into account the effect of specific client/server properties on the scheme of bandwidth allocation. In this section we present a novel bandwidth allocation scheme that suits client/server applications based on CBR and VBR QoS classes. The scheme is considerably more efficient than the traditional scheme. The solution in this paper may be used in conjunction with both traditional and new schemes, as well as with ABR and UBR classes.

In what follows we assume that the server is capable of handling a single request at a time (since this is the common case). The results may be extended to the case where k such requests are handled in parallel, in a straightforward manner. In this context there are two possible schemes.

The Conservative Scheme: Use a traditional bandwidth allocation scheme for client/server applications—allocate the necessary bandwidth to accommodate up to $\mathcal{N}_{\text{req}}$ simultaneous bandwidth reservations[4] $\langle \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{req}}, \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{resp}} \rangle$. Each implicit VC setup procedure will allocate $\langle \mathcal{BW}_{\text{req}}, \mathcal{BW}_{\text{resp}} \rangle$ bandwidth for both of its directions. This scheme was assumed in the above discussion.

The Efficient Scheme: Rely on the server for policing the utilized bandwidth—allocate $\langle \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{req}}, \mathcal{BW}_{\text{resp}} \rangle$ bandwidth on every *link* that is part of CSBT; do *not* allocate bandwidth to each VP separately. During VC setup, do not allocate bandwidth to the VC. Since the server handles each request sequentially, it never transmits more than $\mathcal{BW}_{\text{resp}}$ bandwidth in the direction of all the clients together, hence keeping the utilized bandwidth on any link under the allocated $\mathcal{BW}_{\text{resp}}$.

The two schemes suit different cases. If the client/server application is supported by a general-purpose public network provider, it is more reasonable to use the conservative scheme since it does not require a specialized signaling mechanism and switch functions for such applications. On the other hand, if the application is supported by a more specialized network (e.g., a Web network), in which most of the connections are for client/server applications, an efficient scheme will significantly improve the utilization of the bandwidth resource.

Our solution can be adjusted to both schemes, the only difference being in the allocation of bandwidth for the CSBT (at PHASE 1), which is done along the lines of the appropriate scheme. In the conservative scheme each VP is allocated $\langle \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{req}}, \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{resp}} \rangle$ bandwidth, resulting in a maximum of $\langle \mathcal{L}_{\max} \cdot \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{req}}, \mathcal{L}_{\max} \cdot \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{resp}} \rangle$ bandwidth per link; in the efficient scheme each physical link in CSBT is allocated only $\langle \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{req}}, \mathcal{BW}_{\text{resp}} \rangle$ bandwidth, a decrease of a factor of nearly $\mathcal{N}_{\text{req}} \cdot \mathcal{L}_{\max}$.

Observation 3: If the efficient scheme is used, it is obvious that the base subgraph for PHASE 2 should indeed be a tree, since the total bandwidth required by the whole client/server application is provably minimal.

[4] Note that in order to support $\mathcal{N}_{\text{req}}$ concurrent requests, it does not suffice to allocate $\langle \mathcal{N}_{\text{req}} \cdot \mathcal{BW}_{\text{req}}, \mathcal{BW}_{\text{resp}} \rangle$ bandwidth for every VP, since if several clients that use the VP attempt to concurrently set up VC's, all attempts but one will be rejected by the network, since all of the $\mathcal{BW}_{\text{resp}}$ bandwidth in the direction of the clients will be allocated to a single client. $\mathcal{N}_{\text{req}}$ was assumed to be one hitherto.

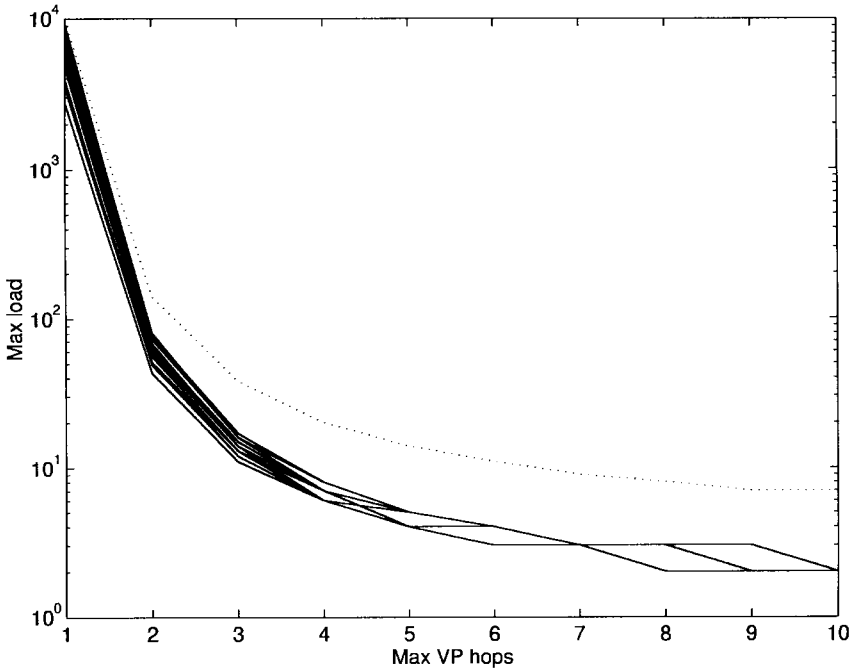

Fig. 2. The tradeoff between load and VP hops.

IV. Simulation Results

We have tested our solution on randomly chosen CSBT's. Our results are depicted in Figs. 2 and 3. The simulations were conducted on random CSBT's, since the CSBT is determined by both the topology of the network (which cannot be determined at this stage for large scale networks) and by the available bandwidth, which depends on the utilization pattern in the network. For the sake of simplicity, we have assumed that all switches are PCX's.

Fig. 2 shows an interesting tradeoff between the number of VP's that share a link (Max load), and the maximum number of VP hops. The graphs were obtained for 15 random CSBT's with 10 000 switches each (drawn in full lines), and a chain tree, with the server at one end. The latter represents a worst-case tree (drawn in a dotted line). It is interesting to note that while the solution of a single VP hop (i.e., SIMPLE-2) requires an average load of above 5000,[5] it suffices to allow two VP hops to reduce the average load to below 60 (and below 14 if three VP hops are allowed).

Fig. 3 shows the scalability of the solutions in terms of the average required multiple of bandwidth units (where $\langle \mathcal{BW}_{\rm req}, \mathcal{BW}_{\rm resp} \rangle$ is considered a single unit) in the conservative scheme and assuming $\mathcal{N}_{\rm req} = 1$. The results are based on an average of the maximum required bandwidth units in each CSBT, taken over ten CSBT's per each network size. The dotted line labeled `cpn=4` shows the behavior of SIMPLE-1, assuming there are four clients per node (hence "cpn"). The line labeled by `h=i` shows the bandwidth requirement when the number of VP hops is not larger than `i` (note that `h=1` represents the SIMPLE-2 solution and `h=9` approximates the SIMPLE-3 solution). From the figure, it is evident that three VP hops are sufficient to support a reasonable number of client/server applications on a large-size ATM network.

[5]Note that this load is unacceptable even if all of the routing resources of the switch were dedicated to the client/server application in question, since the maximum size of each such routing table is only 4096, since the VPI field is limited to 12 bits inside the network.

V. Optimal Solution for Phase 2

A. The Formal Model

We shall need the following definitions to allow a formal treatment of the problem (for basic terms and definitions, see [9]).

Definition 1: A *heterogeneous network* $\mathcal{N}$ is a tuple $\mathcal{N} = (T, s, CN, CX, PX, PCX)$, where:

$T = (V, E)$	physical tree CSBT produced by PHASE 1;
$CX \subseteq V$	set of CX nodes in T (in which VP's always terminate);
$PX \subseteq V$	set of PX nodes in T (in which VP's never terminate);
$PCX \subseteq V$	set of PCX nodes in CSBT (in which VP's may either terminate or not);
CX, PX, PCX	mutually disjoint sets, the union of which is equal to V;
$s \in CX \cup PCX$	*server node* of the current application;
$CN \subseteq CX \cup PCX$	set of *client nodes*.

We shall denote these entities by $T(\mathcal{N})$, $s(\mathcal{N})$, $CN(\mathcal{N})$, $CX(\mathcal{N})$, $PX(\mathcal{N})$, and $PCX(\mathcal{N})$, respectively. We also abbreviate $E(T(\mathcal{N}))$ to $E(\mathcal{N})$ and $V(T(\mathcal{N}))$ to $V(\mathcal{N})$.

Definition 2 (CSVPL): Let $\mathcal{N}$ be a network with tree topology $T(\mathcal{N})$ and let $\mathcal{P}(T(\mathcal{N}))$ be the set of all simple paths in

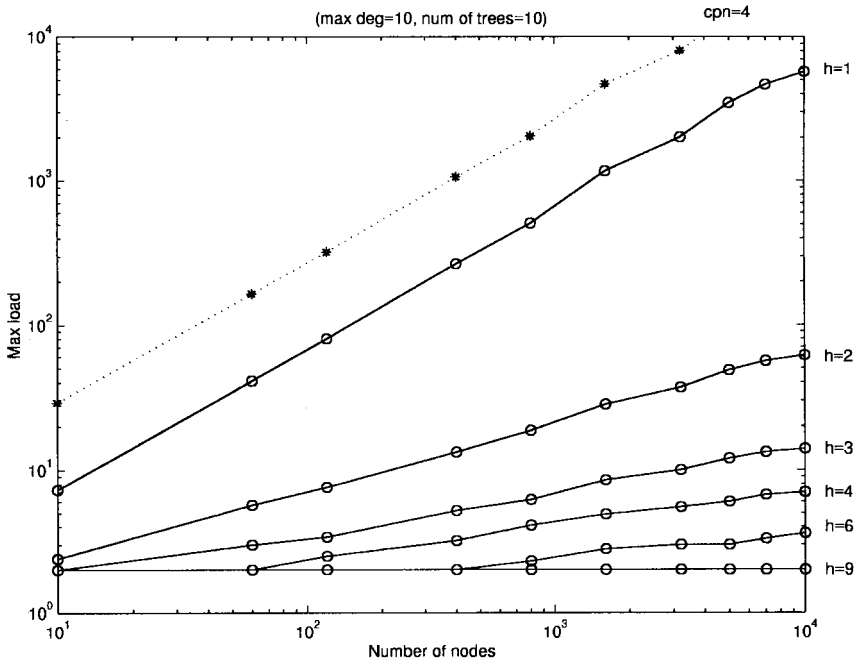

Fig. 3. Scalability of our solution.

$T(\mathcal{N})$. A *client/server virtual path layout (CSVPL for short)* $\Psi = (\mathcal{N}, G_\Psi, \mathcal{I})$ is represented by a graph $G_\Psi = (V, E_\Psi)$ rooted at $s(\mathcal{N})$ and a function $\mathcal{I}$: $E_\Psi \rightarrow \mathcal{P}(T(\mathcal{N}))$, where an edge $\psi = (a, b) \in E_\Psi$ corresponds to a VP between a and b. The function $\mathcal{I}$ maps each VP $\psi = (a, b)$ to its path $\mathcal{I}(\psi)$ in $T(\mathcal{N})$, so that a and b are the endpoints of $\mathcal{I}(\psi)$ as well. We term this path the *induced path* of the VP.

We extend the definition of $\mathcal{I}$ to simple paths in G_Ψ as follows.

Definition 3 (Induced Path): The *induced path* $\mathcal{I}(p)$ for a path $p \in \mathcal{P}(G_\Psi)$, where $p = (\psi_1, \psi_2, \cdots, \psi_k)$ and $\psi_i \in E_\Psi$ for every i, is the path obtained by concatenating the induced paths $\mathcal{I}(\psi_i)$ of all ψ_i's.

Definition 4 (VP Routing Table Load): The *load* $\mathcal{L}(e)$ on a link $e \in E(\mathcal{N})$ is the number of VP's $\psi \in E_\Psi$ that include e in their induced paths, namely, $\mathcal{L}(e) = |\{\psi \in E_\Psi | e \in \mathcal{I}(\psi)\}|$. The load definition is extended to a CSVPL Ψ by $\mathcal{L}(\Psi) = \max_{e \in E(\mathcal{N})} \mathcal{L}(e)$.

The next definition limits the discussion to layouts that satisfy the routing constraints of the various node types, and VP hops constraints.

Definition 5 (Feasibility): For a network $\mathcal{N}$ and $\mathcal{L} > 0$, a CSVPL Ψ is *$\mathcal{L}$-feasible* if the following conditions hold:

1) for every $v \in PX(\mathcal{N})$, there is no VP $(v, x) \in E_\Psi$;
2) for every $v \in CX(\mathcal{N})$, there is no VP $(x, y) \in E_\Psi$, such that $v \notin \{x, y\}$ and $v \in \mathcal{I}((x, y))$;
3) $\mathcal{L}(\Psi) \leq \mathcal{L}$.

Thus, in a feasible CSVPL, all of the VP's that include a CX node terminate at that node, and all VP's that include a PX node cannot terminate at it and are switched to an adjacent node.

We now define an optimal solution as a feasible solution which minimizes the VP hops between any client to the server. To this end we first define the VP hops $\mathcal{H}(v, s)$ to be the minimum number of VP's that may be used to form a VC between a client node v and the server s, such that the VC uses the simple path in CSBT. Note that the path has to be shortest possible in both the physical tree (CSBT) and the VP layout (CSVPL). A formal definition is given as follows.

Definition 6 (VP Hops): The *VP hops* $\mathcal{H}(v, w)$ for a pair of nodes $v, w \in CN(\mathcal{N})$ is the minimum k such that:

1) $\exists p = (\psi_1, \psi_2, \cdots, \psi_k) \in \mathcal{P}(G_\Psi)$, ($\psi_i \in E_\Psi$ for all i);
2) $\exists x, y \in V(\mathcal{N})$, $\psi_1 = (v, x)$, $\psi_k = (y, w)$;
3) the induced path $\mathcal{I}(p)$ is a simple path between v and w in $T(\mathcal{N})$.

If no such k exists, define $\mathcal{H}(v, w) = \infty$; also define $\mathcal{H}(\Psi) = \max_{v \in CN(\mathcal{N})} \mathcal{H}(v, s(\mathcal{N}))$.

Definition 7 (Rank): Let $\psi = (a, b)$ and let a be closer to the server in the CSBT. The *rank* of ψ is the maximum VP hops $\mathcal{H}(v, a)$ from any client node v which uses ψ in its shortest VP route to the server (again, shortest in both respects).

Definition 8: Given $\mathcal{L}$, a CSVPL Ψ is $\mathcal{L}$-optimal if it is $\mathcal{L}$-feasible and its worst-case VP hop count $\mathcal{H}(\Psi)$ is minimal amongst all other $\mathcal{L}$-feasible VP layouts.

Example 2: Consider the CSBT in Fig. 4 (depicted in bold lines) and the CSVPL (depicted in thin lines). It is easy to verify that the CSVPL obeys the routing rules of the various nodes, e.g., at node e, some VP's go through the node while others terminate, at node i no VP terminates, and at node j all VP's terminate. The VP hops of the CSVPL $\mathcal{H}(\Psi) = 5$ since for node a, $\mathcal{H}(a, s) = 5$. The shortest route from a to s, which is also simple in CSBT, is (a, c), (c, d), (d, e), (e, f), (f, s). Note that if the physical tree is ignored and routes need only be shortest in CSVPL, then the VP hop count from a to s is three— (a, c), (c, b), (b, s). The rank of VP (e, f) is 4 since it is the fourth hop toward the server for node a. The load on link (f, s) satisfies $\mathcal{L}((s, w)) = 4$ and this is also $\mathcal{L}(\Psi)$.

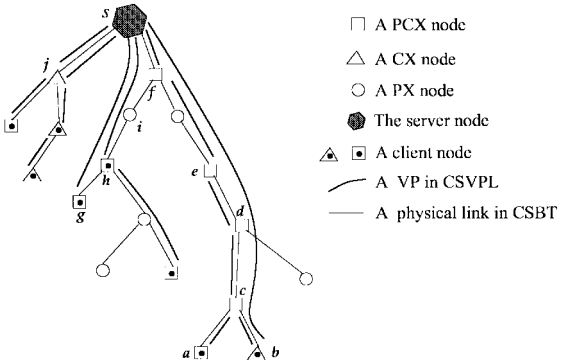

Fig. 4. A CSVPL on a CSBT.

B. The VP Design Algorithm

We now present a polynomial time greedy algorithm `find_layout` that produces an optimal CSVPL for any given load. The algorithm framework is based on the one presented at [14]; however, the present case is much more complex due to the various types of switches (in [14] all switches were assumed to be PCX's, a fact which substantially simplified the algorithm and its analysis). We first describe the algorithm informally, then present a more formal pseudocode, and finally show an example of how the CSVPL evolves during an execution of the algorithm.

Fix some constant integer h such that $h \geq \mathcal{H}(\Psi)$ (the depth of CSBT will do). The `find_layout` algorithm represents the VP's using vectors with h elements (i.e., vectors over $\mathbb{N}^h$) and, therefore, we shall need the following vector notations.

Notation: Let $X, Y \in \mathbb{N}^h$, $c \in \mathbb{N}$, $i, j \in \{1, \cdots, h\}$. We use the following notations:

$X[i]$	ith element in the vector X;
$\|\|X\|\|$	sum of all the elements $X[i]$;
$X[i:j]$	part of the vector from the ith to the jth position;
$\vec{c}[1:i]$	vector of i elements that are equal to c;
$X \bullet Y$	concatenation of the vectors X and Y;
$X + Y$	vector sum of the vectors X and Y.

`find_layout` starts from the leaves of the tree $T(\mathcal{N})$ and advances toward the root $s(\mathcal{N})$. For each node v, it maintains the number of VP's, say $\mathcal{L}$, that contribute to the load on the link from v to its parent in the tree. This number is split into h components represented in a vector $L_v \in \mathbb{N}^h$ such that $||L_v|| = \mathcal{L}$, where $L_v[i]$ represents the number of VP's with rank i. For example, in Fig. 4 $L_f = (3, 0, 0, 0, 1, 0)$, since three out of the four VP's traversing link (f, s) have a rank of 1 and the fourth VP has a rank of 5. Specifically, `find_layout` creates a VP with rank 1 for each leaf in $CN(\mathcal{N})$ (at line 4) and no VP for leaves that are not clients' nodes (at the same line). At an internal node v, the vector L_v is equal to the vector sum of all of the L_w vectors from its sons—reflecting an attempt to extend all VP's through v. An additional VP with rank 1 is added to this sum from v toward the server if v is a client node (at line 9). This VP will be used to carry VC's from the clients connected directly to v. If v is a CX node, then all VP's from the descendants are stopped at v (by definition of a CX node) and a single VP starts from v toward the server. The rank of this VP is k, assuming that the maximum rank of VP's that stopped at v is $k-1$. Consequently, $L_v = \vec{0}[1{:}\ k-1] \bullet 1 \bullet \vec{0}[1{:}\ h-k]$.

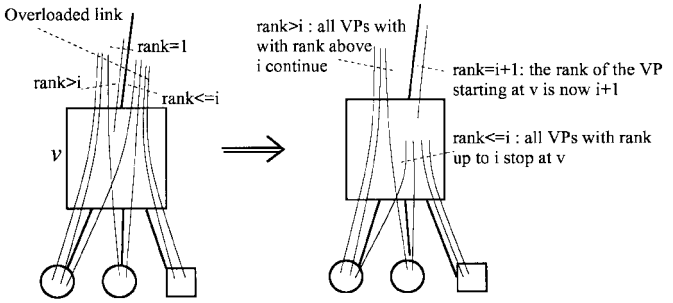

Fig. 5. Transformation of a node $v \in PCX(\mathcal{N})$—$\mathrm{Transform}(L_v, i)$.

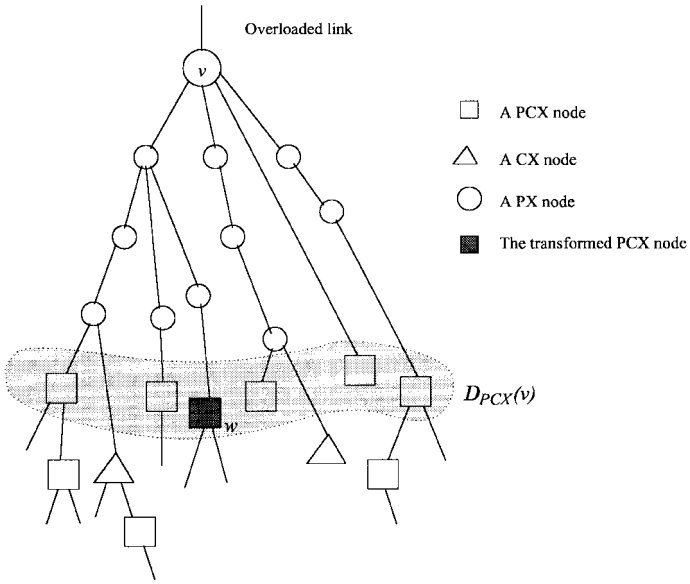

Fig. 6. Transformation of a PCX node due to an overloaded vector L_v of a PX node v.

At this stage, if the load on the link from v to its parent is not too high (i.e., $||L_v||$ does not exceed $\mathcal{L}_{\max}$), `find_layout` proceeds to another node. However, if the load is too high, it is reduced by transforming VP's at some chosen node x (using the $\mathrm{Transform}$ procedure defined below). The transformation involves stopping some VP's at x, thereby changing the load vector L_x.

The node that is transformed depends on the type of node v—if v is a PCX node, then it is transformed ($x = v$) so as to not increase the VP hops significantly (at lines 13–15, see Fig. 5). If v is a PX node, it cannot be transformed (since VP's are not allowed to stop at v) and x is chosen to be a descendent of v, which is a PCX node and which has the largest number of VP's that can be stopped without increasing the VP hops significantly (at lines 18–21, see Fig. 6). In this case, the algorithm may iterate on the descendants of v and repeatedly transform them until the load at v is decreased below the required level or until it determines that there is no way to decrease the load as required.

The idea behind the transformation in Fig. 5 is that it is better to stop VP's with as low of a rank as possible (i.e., VP's with rank j, which is lower than some value i in the

```
0   function find_layout( tree 𝒩, integer ℒ_max) return {SUCCESS, FAILURE}
1      var L: set of vectors {L_v ∈ ℕ^h | v ∈ V(𝒩)} )
2   begin
3      for all w ∈ V(𝒩):
4         if w is a leaf then L_w ← Extra(w) otherwise L_w ← UNDEFINED
5      loop forever
6         choose a node v ∈ V(𝒩) such that:
             (a) L_v = UNDEFINED,
             (b) for all x ∈ SONS(v), L_x ≠ UNDEFINED.
7         if v = s(𝒩) then return SUCCESS

8         loop forever
9            L_v ← Σ_{s∈Sons(v)} L_s + Extra(v)
10           if v ∈ CX(𝒩) then Transform(L_v, H(L_v))
11           if ||L_v|| ≤ ℒ_max then exit internal loop

12           if v ∈ PCX(𝒩) then begin
13              find i ∈ {1, ..., h − 1} such that
                   (a) ||L_v[i + 1 : h]|| < ℒ_max,
                   (b) ||L_v[i : h]|| ≥ ℒ_max.
14              if no such i exists then return FAILURE
15              Transform(L_v, i)
16           end if

17           if v ∈ PX(𝒩) then begin
18              find a node w ∈ 𝒟_PCX(v) and an integer i ∈ {1, ..., h − 1} such that:
                   (a) ||L_w[1 : i]|| > 1,
                   (b) For every w' ∈ 𝒟_PCX(v) \ {w}, L_w[1 : i] ≥ L_w'[1 : i],
                   (c) For every w' ∈ 𝒟_PCX(v), ||L_w'[1 : i − 1]|| ≤ 1.
19              if no such i, w exist then return FAILURE
20              Transform(L_w, i)
21              Update L_x for every x in the path from w to v
22           end if
23        end loop
24     end loop
25  end
```

Fig. 7. The `find_layout` algorithm.

figure). This causes the rank of the VP that starts at v to be $i+1$. All of the other VP's that go through v, continue toward the parent of v.

Definition 9: Define the following:

- a transformation on a vector L at location i is (see Fig. 5 for a visual demonstration)

$$\text{Transform}(L, i) \equiv L \leftarrow \vec{0}[1:i] \bullet (L[i+1]+1) \bullet L[i+2:h]$$

- for a node v, define the addition in L_v that is needed if v is a client node by

$$\text{Extra}(v) = \begin{cases} 1 \bullet \vec{0}[1:h-1], & \text{if } v \in CN(\mathcal{N}) \\ \vec{0}[1:h], & \text{otherwise} \end{cases}$$

- for a vector $V \in \mathbb{N}^h$, define $H(V) = \max\{i \in \{1, \cdots, h\} | V[i] > 0\}$;
- for a node $v \in PX$; define $\mathcal{D}_{PCX}(v)$ to be the set of descendants of v which are PCX nodes, and for which the root to v consists of PX nodes only (see Fig. 6).

A formal description of the algorithm follows in Fig. 7.

Example 3: An example of the execution of the algorithm appears in Fig. 8. Links in which the load exceeds $\mathcal{L}_{\max} = 3$ are pointed to by arrows. Circled in grey are the nodes that `find_layout` chooses to transform. For sake of brevity, the steps in the figure correspond to multiple steps of `find_layout` as long as these steps are independent of each other. We shall refer to the subtrees of this CSBT as the left, middle, and right subtrees. Note the following:

- The first transformation takes place in the left subtree. Before it, the ranks of VP's from the children of the PCX node had ranks 1, 2, and 3. `find_layout` only stops the VP with rank 1. Thus, the node's load vector changes from (2, 1, 1, 0) to (0, 2, 1, 0).
- The second transformation involves the left and middle subtrees. Thus, $\mathcal{D}_{PCX}(v)$ includes both PCX nodes from these subtrees. The PCX node of the middle subtree is chosen since it has VP's with rank 1 while the PCX of the left subtree has only ranks 2 and 3.
- Since the second transformation does not reduce the load enough, a third transformation takes place for the same subtree, this time involving the PCX of the left subtree again and stopping its VP with rank 2. Its load vector thus changes from (0, 2, 1, 0) to (0, 0, 2, 0).
- The next transformation involves the entire tree, and $\mathcal{D}_{PCX}(v)$ for the root v contains three PCX nodes, one

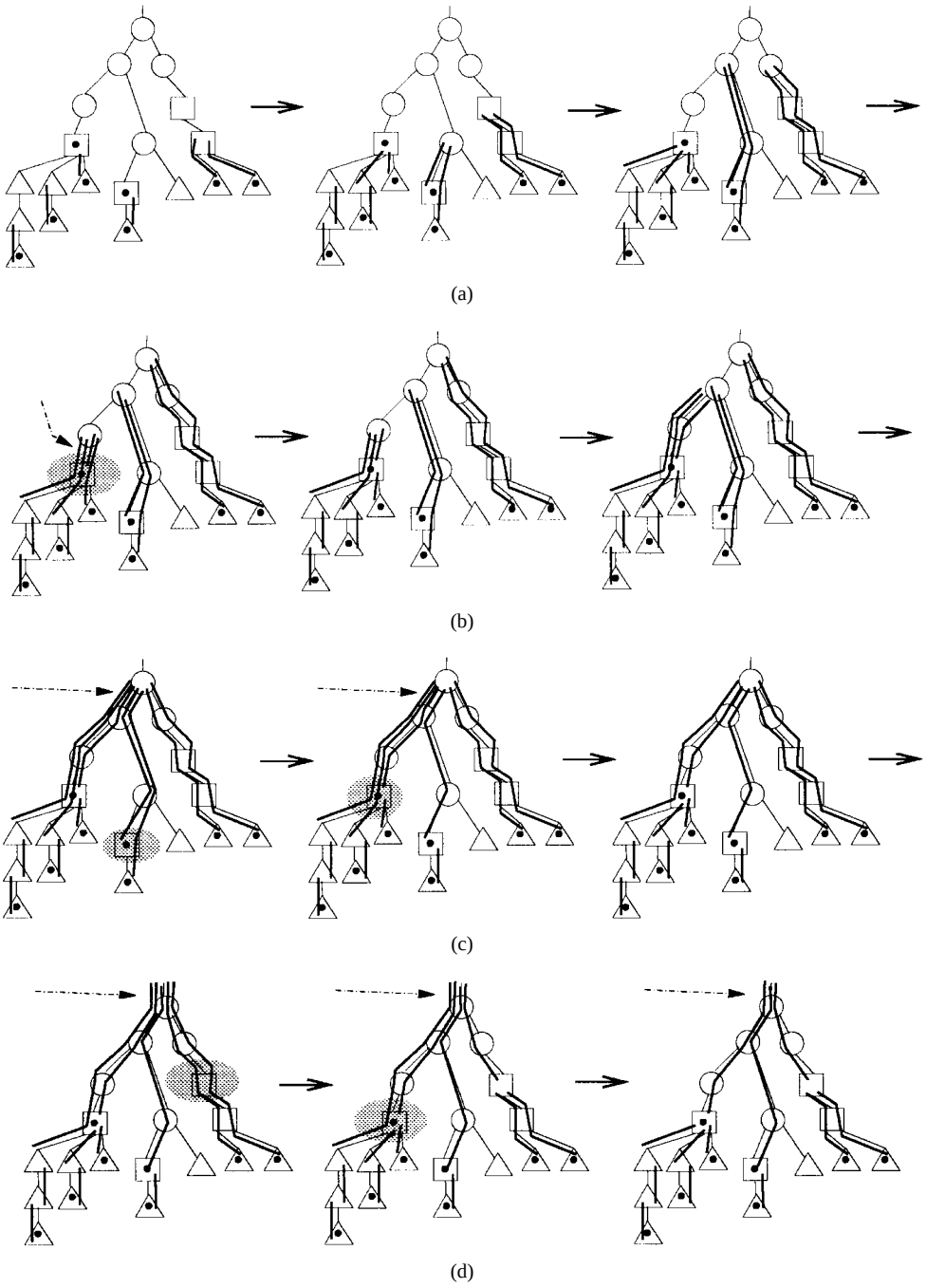

Fig. 8. Execution of `find_layout` for $\mathcal{L}_{\max} = 3$.

per subtree. The chosen PCX is of the rightmost subtree as it has VP's with rank 1.

- Finally, the last transformation involves the VPX of the left subtree again, now stopping the rank 3 VP at it. As a result, its load vector changes from (0, 0, 2, 0) to (0, 0, 0, 1).

C. Analysis

The proof of correctness of the above algorithm (i.e., proof that the resulting CSVPL is $\mathcal{L}_{\max}$-feasible with respect to $\mathcal{N}$) is straightforward and is omitted for the sake of brevity. We prove the solution that `find_layout` produces is optimal, for every given network $\mathcal{N}$ and $\mathcal{L}_{\max}$, by the following theorem (for a detailed proof, see subsection B in the Appendix).

Theorem 1: The `find_layout` procedure finds an $\mathcal{L}$-optimal CSVPL for any given tree network $\mathcal{N}$.

Sketch of Proof: The proof is based on the claim that if there exists a solution for a given h and a maximum load constraint $\mathcal{L}_{\max}$, the `find_layout` will find such a solution. This is done by inductively comparing the given "optimal" CSVPL to the CSVPL produced by `find_layout`, starting at the leaves of the tree.

We first show how every CSVPL may be expressed by load vectors $\{M_v | v \in V\}$ (termed the *vector representation* of the CSVPL). Next, we prove that if $\{L_v | v \in V\}$ is the vector

representation of the CSVPL found by `find_layout`, then at every node v, $||L_v|| \leq ||M_v||$. This proves that the solution of `find_layout` is optimal since its load on each link does not exceed that of any other solution. To this end, we strengthen the claim that we wish to prove and we add the condition that $L_v \leq M_v$ in a lexicographic order [in what follows, all comparisons between vectors in $\mathbb{N}^h$ are lexicographic, using an order in which $M[1]$ is the least significant component, e.g., (9, 11, 1) < (2, 12, 1) < (0, 0, 2)]. In fact, the transformations in `find_layout` were chosen so as to support this additional condition and aid the analysis.

It is important to note that these two conditions (namely, $||L_v|| \leq ||M_v||$ and $L_v \leq M_v$) do *not* hold for the final set of vectors L that `find_layout` produces; they do hold, however, for each node v immediately after v has been treated by the algorithm and before it is revisited for subsequent transformations (i.e., before a new node is chosen at line 6). Hence, the condition holds for the children of a node v when v is first chosen at this line. □

The time complexity of the above algorithm is bounded by the following lemma. This bound is quite loose, but still exhibits a relatively low complexity.

Lemma 1: The time complexity of `find_layout` on a network with N nodes is $O(h \cdot N^2)$.

Proof: In `find_layout` each node may be considered for transformation $O(N)$ times (in a straightforward implementation of the procedure) if it belongs to $\mathcal{D}_{PCX}(v)$ for $O(N)$ nodes in the network. At each such event, $O(h)$ time is needed to determine if it is indeed chosen for transformation. □

VI. Summary

In this paper we studied the problem of supporting a client/server application in an ATM network. We first listed the parameters that seem important for such applications on one hand and for ATM networks on the other hand. We presented simple solutions to the problem and explained why they do not scale well to large networks that are likely to be constructed in the future. We then presented our solution, which can be tuned to adjust various requirements, expressing a tradeoff between the setup time of a VP layout to its resource utilization, and demonstrated numerically its characteristics. We also presented a new bandwidth allocation scheme that is optimal for such applications.

Our solution is based on three main phases, amongst which the second phase (that deals with laying out the VP's from the clients to the server) is the most complex, and in the second half of the paper we focused on it. We devised a formalism for the problem and presented a fairly fast algorithm for solving it. Finally, we have proven that our algorithm indeed finds an optimal solution for any given set of parameters.

The main conclusion from this work is the role of nodes that switch both VP's and VC's (PCX's) in the VP design—these nodes enable the flexibility of choosing VP's to tune the design according to the required balance between the utilization of various resources in the network. Without such nodes, once the CSBT is chosen, the VP routes are fixed.

Appendix

A. NP-Hardness of Finding the CSBT

In this appendix we prove that two of the options for a CSBT, spelled out in Section III-B-2, are NP-complete.

A simplified version of the CSBT-2 option is the following one, for the case when all links have unit delays.

Problem: Shortest distance Steiner tree (SDST).
Instance: An undirected graph $G = (V, E)$, a subset of nodes $S \subset V$, a node $r \in S$, and an integer B.
Question: Does there exist a tree T in G that spans all of the nodes in S such that the shortest distance between each node in S and r is the same in G and in T, and such that the number of edges in T does not exceed B?

Lemma 2: SDST is NP-complete.

Proof: By reduction to the set cover problem ([11, Problem SP4])—given a set $S = \{s_i\}_{i=1}^{k}$, a set of subsets $C = \{c_i\}_{i=1}^{|C|} \subseteq 2^S$, and an integer B, does there exist a subset of C, $C' \subseteq C$, and $|C'| \leq B$ that covers all of the elements in S ($S = \cup_{c \in C'} c$).

Build a three-layer graph with a node r' in the first layer, connected to nodes $C' = \{c'_i\}_{i=1}^{|C|}$, representing the nodes in C in the second layer. The third layer contains nodes $S' = \{s'_i\}_{i=1}^{k}$, representing the nodes in S, where each node c'_i is connected to nodes in S', which represent the elements that the set c_i contains.

Now, let the set S of SDST contain the nodes in S' and r', let r of SDST be r', and let B of SDST be $B + k$. It is easy to see that all shortest path trees rooted at r' that span the nodes in S' include a single edge connected to each s'_i and some of the edges connected to nodes in C'. A minimal tree contains a minimum number of the latter edges. Therefore, if SDST can be solved with $B + k$ edges, then the set S can be covered by B sets out of C—the sets represented by nodes in C' connected by a tree edge to r'. On the other hand, if S can be covered by up to B sets, then the solution implies a SDST with up to $B + k$ edges. □

A simplified version of the CSBT-4 option is the following one, for the case when the available bandwidth for each link is B.

Problem: Bounded bandwidth spanning tree (BBWSP).
Instance: An undirected graph $G = (V, E)$, a subset of nodes $S \subset V$, a node $r \in S$, and an integer B.
Question: Does there exist a tree T in G that spans all the nodes in S such that if r and its adjacent edges are removed from T, each remaining subtree contains no more than B nodes from S?

More explanation is needed here as to why the problem represents CSBT-4 at all. Since the bandwidth required on a CSBT link from a subtree toward the server is proportional to the number of client nodes in the subtree, it grows on links that are closer to the root. If the available bandwidth for each link e is $\mathcal{BW}_{\text{avail}}(e) = \mathcal{L}_{\max} B$, then there exists such a CSBT iff the number of client nodes (i.e., nodes in S) in each of the largest possible subtrees is less than B.

Lemma 3: BBWSP is NP-complete.

Proof: By reduction to the node disjoint path problem ([11, Problem ND40])—given a graph G and a set of node disjoint pairs $\{(s_i, t_i)\}_{i=1}^{k}$, are there node disjoint paths from each s_i to t_i?

Add a new node r to the graph and connect it to each s_i. For each i, add nodes $X_i = \{x_i^j\}_{j=1}^{i}$ and connect them to s_i. Also add nodes $Y_i = \{y_i^j\}_{j=i}^{k-i+1}$ and connect them to t_i. Define $S = \{r\} \cup \cup_{i=1}^{k} X_i \cup \cup_{i=1}^{k} Y_i$ and let $B = k+1$.

First, note that all the edges between s_i and r must be part of the tree, or else the removal of r from the tree will split it into less than k subtrees and, by the pigeon-hole principle, one part will contain more than $k(k+1)/k = B$ nodes from S. Next, it is easy to see that the only way for the Y_i nodes to be connected to r via a tree is to connect each t_i to some s_j using node disjoint paths (if the paths share a node in addition to r, a cycle is formed). The value of B and the different number of X_i and Y_i nodes for each i ensure that the paths in the tree are from t_i to s_i (and not some other s_j), since only this way can each subtree contain no more than B nodes from S. Thus, if disjoint paths exist, then there is a solution to BBWSP and if they do not exist, no solution to BBWSP can be found. □

B. The Optimality Proof

In this appendix we present a detailed optimality proof of Theorem 1. For this purpose, we need the following definitions and lemmas.

Definition 10: A CSVPL Ψ is $(\mathcal{L}, h)$-minimal if it is $\mathcal{L}$-feasible, $\mathcal{H}(\Psi) \le h$, and the deletion of a VP $\psi \in E_\Psi$ yields a CSVPL Ψ' with $\mathcal{H}(\Psi') > h$.

Lemma 4: Let $\mathcal{N}$ be a CSBT tree. Every $(\mathcal{L}, h)$-minimal CSVPL can be represented by a set of vectors $\{M_v \in \mathbb{N}^h | v \in V(\mathcal{N}),\ v \ne s(\mathcal{N})\}$, such that for every node $v \ne s(\mathcal{N})$ and $M'_v = \sum_{s \in \mathrm{Sons}(v)} M_s + \mathrm{Extra}(v)$, the following conditions hold:

1) $\mathcal{L}((v, \mathrm{parent}(v)) = \|M_v\|$;
2) if v is a leaf, then $M_v = \mathrm{Extra}(v)$;
3) if $v \in CX(\mathcal{N})$, then $M_v = \vec{0}[1{:}H(M'_v)] \bullet 1 \bullet \vec{0}[1{:}h - H(M'_v) - 1]$;
4) if $v \in PX(\mathcal{N})$, then $M_v = M'_v$;
5) if $v \in PCX(\mathcal{N})$, then $M_v \ge M'_v$.

Before proving the main lemma in the optimality proof (Lemma 5), we cite several properties of the above vector representation.

Definition 11 ([14]): A vector L is called *k-nontrivial* iff $\|L[1{:}k-1]\| \le 1$ and $\|L[1 : k]\| > 1$; in particular, if $L[1] > 1$ it is 1-nontrivial, and if $\|L\| \le 1$, then L is ∞-nontrivial.

The following lemma is an extension of the one proposed in [14]. Despite the usage of a similar technique, this lemma is substantially harder to prove, the difficult case being the transformation of a PX. For this case, assume that the current node v is a PX and that its load vector L_v exceeds the maximum bound. Consider the list of load vectors for all nodes in $\mathcal{D}_{PCX}(v)$ in the optimal solution M and in the solution of `find_layout` L (assume they are numbered $1, \cdots, s$). Before transforming a node in $\mathcal{D}_{PCX}(v)$, the inequalities of

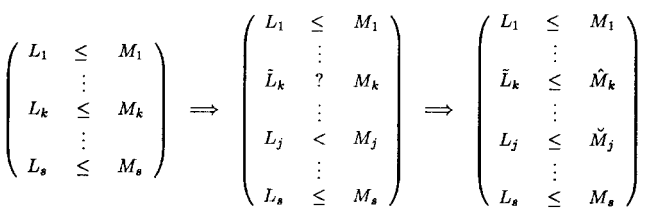

Fig. 9. Load vectors of $\mathcal{D}_{PCX}(v)$.

Fig. 9(a) hold by the induction hypothesis. After transforming some node k at line `20`, the inequality for k may be violated [as in Fig. 9(b)], in which case we show there exists some node j for which the inequality is strict. Using this, we "borrow" some of the load of M_j to fix the inequality for node k [see Fig. 9(c)]. A more formal proof follows.

Lemma 5: Let $\ell \in \mathbb{N}$. If there exists an (ℓ, h)-minimal CSVPL Ψ with vector representation $\{M_v | v \in V(\mathcal{N})\}$, then for every node $v \ne s(\mathcal{N})$, the following holds:

1) if `find_layout` is called with $\mathcal{L}_{\max} = \ell$, then it does not return `FAILURE` while handling v;[6]
2) the vector L_v produced by `find_layout` satisfies $L_v \le M_v$ when `find_layout` finishes handling v (before starting to handle a new node at line `6`).

Proof: By induction on the structure of the tree $T(\mathcal{N})$. At leaf nodes `find_layout` cannot fail (and condition 1 holds), and $L_v = \mathrm{Extra}(v)$ by the algorithm and $M_v = \mathrm{Extra}(v)$ by Lemma 4, thus condition 2 holds as well. At an internal node v, by induction $L_s \le M_s$ for every son s of v. Let $L'_v = \sum_{s \in \mathrm{Sons}(v)} L_s + \mathrm{Extra}(v)$, and let $M'_v = \sum_{s \in \mathrm{Sons}(v)} M_s + \mathrm{Extra}(v)$. Clearly $L'_v \le M'_v$.

The proof continues according to the three node types.

Case 1—$v \in CX(\mathcal{N})$: . It is clear that `find_layout` cannot fail while handling such a node. Also it is clear that $H(L'_v) \le H(M'_v)$. It follows from line `10` that $L_v = \vec{0}[1 : H(L'_v)] \bullet 1 \bullet \vec{0}[1 : h - H(L'_v) - 1]$. It follows from Lemma 4 that $M_v = \vec{0}[1 : H(M'_v)] \bullet 1 \bullet \vec{0}[1 : h - H(M'_v) - 1]$ and, thus, $L_v \le M_v$. □

Case 2—$v \in PX(\mathcal{N})$: If $\|L'_v\| \le \ell$, then `find_layout` does not fail and it is clear that $L_v = L'_v \le M'_v = M_v$.

If $\|L'_v\| > \ell$, then `find_layout` repeatedly transforms nodes in $\mathcal{D}_{PCX}(v)$. After a transformation of node x, we shall show that the transformed L_x still obeys $L_x \le M_x$ or change the vector M_x to $\hat{M}_x$ and the vector M_y to $\check{M}_y$ for some other $y \in \mathcal{D}_{PCX}(v)$, so that $L_x \le \hat{M}_x$, $L_y \le \check{M}_y$. The transformation will also obey the condition $M_x + M_y = \hat{M}_x + \check{M}_y$. By this we achieve the following:

$$\begin{aligned} L_v &= \sum_{w \in \mathcal{D}_{PCX}(v)} L_w \\ &\le \sum_{w \in \mathcal{D}_{PCX}(v)} M_w - (M_x + M_y) + (\hat{M}_x + \check{M}_y) \\ &\le \sum_{w \in \mathcal{D}_{PCX}(v)} M_w = M_v. \end{aligned}$$

[6] Note that this is a weaker condition than the one specified above (i.e., $\|L_v\| \le \|M_v\|$); however, the stronger condition was presented for sake of clarity, while the precise condition is the weaker one.

Formally, let x be the descendent chosen at line 18 during some attempt to reduce the value of $||L_v||$; let i be the index chosen at the same line. By definition, L_x is i-nontrivial. Split $\mathcal{D}_{PCX}(v)$ into the following two subsets:

$$\ell^+ = \{y \in \mathcal{D}_{PCX}(v) | y \neq x \text{ and } y \text{ is } j\text{-nontrivial for } j \geq i\}$$
$$\ell^- = \{y \in \mathcal{D}_{PCX}(v) | y \text{ is } j\text{-nontrivial for } j < i\}.$$

Let $\tilde{L}_x$ be the value of L_x after applying $\text{Transform}(L_x, i)$. Clearly $||\tilde{L}_x|| < ||L_x||$. If $L_x[i+1:h] < M_x[i+1:h]$, then $\tilde{L}_x \leq M_x$ and all nodes $w \in \mathcal{D}_{PCX}(v)$ obey $L_w \leq M_w$; if $L_x[i+1:h] = M_x[i+1:h]$, then two cases are possible.

1) For every $y \in \ell^+$, $L_y[i+1:h] = M_y[i+1:h]$; then since L_y is $i^{\geq}$-nontrivial, it can be shown that $||L_y|| \leq ||M_y||$. However, since $||L_y|| \leq ||M_y||$ for every $y \in \ell^-$, we have $\sum_{y \in \mathcal{D}_{PCX}(v)} ||L_y|| \leq \sum_{y \in \ell^-} ||M_y|| + ||M_x|| + \sum_{y \in \ell^+} ||M_y|| = ||M||$ and condition 2 is satisfied as well.
2) On the other hand, if there exists $y \in \ell^+$ such that $L_y[i+1:h] < M_y[i+1:h]$, we transform the vectors as follows:

$$\tilde{L}_x \longleftarrow \vec{0}[1:i] \bullet (L_x[i+1]+1) \bullet L_x[i+2:h]$$
$$\hat{M}_x \longleftarrow M_y[1:i] \bullet (M_x[i+1]+1) \bullet M_x[i+2:h]$$
$$\check{M}_y \longleftarrow M_x[1:i] \bullet (M_y[i+1]-1) \bullet M_y[i+2:h].$$

Clearly $\tilde{L}_x \leq \hat{M}_x$ and $M_x + M_y = \hat{M}_x + \check{M}_y$. Also, the inequality for node y still holds (i.e., $L_y \leq \check{M}_y$) because $L_y[i+1:h] \leq \check{M}_y[i+1:h]$ and $L_y[1:i] \leq L_x[1:i] \leq M_x[1:i] = \check{M}_y[1:i]$. □

Case 3—$v \in PCX(\mathcal{N})$: Assuming $||L'_v|| \leq \ell$, then find_layout finishes handling v at line 11 without failing and

$$L_v = \sum_{s \in \text{Sons}(v)} L_s + \text{Extra}(v) \leq \sum_{s \in \text{Sons}(v)} M_s + \text{Extra}(v) \leq M_v.$$

The left inequality is due to the induction hypothesis, and the right one is due to Lemma 4.

Assuming $||L'_v|| > \ell$, then if there exists no i that satisfies the condition at line 13, then

$$\ell \leq L'_v[h] = \sum_{s \in \text{Sons}(v)} L_s[h] \leq \sum_{s \in \text{Sons}(v)} M_s[h] < ||M_v||.$$

The first inequality stems from the unsatisfied condition, while the second inequality stems from the induction hypothesis, and the third one from Lemma 4 (it is strict since $||M_v|| \geq \sum_{s \in \text{Sons}(v)} ||M_s|| + 1$). This inequality contradicts the assumption that M is a feasible solution.

If, on the other hand, such an i exists, it is easy to see that find_layout does not return FAILURE and it remains to show that $L_v \leq M_v$. Clearly $M_v \geq L'_v$ (since $M_v \geq \sum_{s \in \text{Sons}(v)} M_s + \text{Extra}(v) \geq \sum_{s \in \text{Sons}(v)} L_s + \text{Extra}(v) = L'_v$); it is also clear that $M_v > L'_v$ (otherwise $\ell < ||L'_v|| = ||M_v||$).

Now, if $M_v[i+1:h] > L'_v[i+1:h]$, then $M_v[i+1:h] \geq L'_v[i+1:h] + (1) \bullet \vec{0}[1:h-i-1 = L[i+1:h]$, and since $M_v[1:i] \geq \vec{0}[1:i] = L[1:i]$, we get $M_v \geq L_v$.

On the other hand, if $M_v[i+1:h] = L'_v[i+1:h]$, then there exists $j \leq i$ such that $M_v[j] > L'_v[j]$ (and $M_v[j+1:h] = L_v[j+1:h]$) and $||M_v|| \geq ||M_v[j:h]|| = ||M_v[j+1:h]|| + M_v[i] > ||L'_v[j+1:h]|| + L'_v[j] \geq \ell$—a contradiction. □

The above lemma concludes the proof of Theorem 1, since if we take an optimal CSVPL as Ψ (in the lemma), then find_layout is guaranteed to find a layout that obeys the load constraint, with vectors L_v that do not exceed lexicographically those of Ψ. In particular, the VP hops implied by L_v do not exceed those of the optimal solution Ψ.

REFERENCES

[1] S. Ahn, R. P. Tsang, S. R. Tong, and D. H. C. Du, "Virtual path layout design on ATM networks," in *IEEE INFOCOM'94*, Toronto, Ont., Canada, 1994, pp. 192–200.

[2] M. H. Ammar, S. Y. Cheung, and C. M. Scoglio, "Routing multiple connections using virtual paths in an ATM network," in *IEEE INFOCOM'93*, San Francisco, CA, 1993, pp. 98–105.

[3] A. Birrell and B. Nelson, "Implementing remote procedure calls," *ACM Trans. Comput. Syst.*, vol. 2, no. 1, Feb. 1984.

[4] D. R. Cheriton, "The V kernel: A software base for distributed systems," *IEEE Software*, vol. 1, Apr. 1984.

[5] D. R. Cheriton and R. L. Williamson, "Network measurement of the VMTP request-response protocol in the V distributed system," in *ACM SiGMETRICS Conf. Measurement and Modeling of Computer Systems*, May 1987.

[6] I. Chlamtac, A. Ganz, and G. Karmi, "Transport optimization in broadband networks," in *IEEE INFOCOM'91*, Bal Harbour, FL, 1991, pp. 49–58.

[7] R. Cohen and A. Segall, "Connection management and rerouting in ATM networks," in *IEEE INFOCOM'94*, Toronto, Ont., Canada, 1994, pp. 184–191.

[8] D. Dykeman, Ed., *PNNI Draft Specification*, Contribution 94-0471, 1995.

[9] S. Even, *Graph Algorithms*. Rockville, MD: Computer Science, 1979.

[10] The ATM Forum, *ATM, User-Network Interface Specification*, version 3.0. Englewood CLiffs, NJ: Prentice-Hall, 1993.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: Freeman, 1979.

[12] O. Gerstel, I. Cidon, and S. Zaks, "Efficient support for the client-server paradigm over ATM networks," in *IEEE INFOCOM'96*, San Francisco, CA, 1996, pp. 1294–1301.

[13] ——, "The layout of virtual paths in ATM networks," *IEEE/ACM Trans. Networking*, vol. 4, pp. 873–884, Dec. 1996.

[14] ——. (1996, Oct.). "Optimal virtual path layout in ATM networks with shared routing table switches." [Online], *Chicago J. Theoret. Comput. Sci.* vol. 3, special issue of *ACM PODC'94*. Available: http://www.cs.uchicago.edu/publications/cjtcs/articles/1996/3/contents.html

[15] O. Gerstel and A. Segall, "Dynamic maintenance of the virtual path layout," in *IEEE INFOCOM'95*, Boston, MA, Apr. 1995, pp. 330–337.

[16] R. Händler and M. N. Huber, *Integrated Broadband Networks: An Introduction to ATM-Based Networks.* Reading, MA: Addison-Wesley, 1991.

[17] E. Kranakis, D. Krizanc, and A. Pelc, "Hop-congestion tradeoffs for high-speed networks," in *7th IEEE Symp. Parallel Distributed Processing*, 1995, pp. 662–668.

[18] T. F. La Porta and M. Schwarz, "Architectures, features, and implementation of high-speed transport protocols," *IEEE Commun. Mag.*, pp. 14–22, May 1991.

[19] J. Y. Le Boudec, A. Meier, R. Oechsle, and H. L. Truong, "Connectionless data service in an ATM-based customer premises network," *Comput.*

Networks ISDN Syst., vol. 26, pp. 1409–1424, 1994.
[20] C. Partridge, *Gigabit Networking*. Reading, MA: Addison-Wesley, 1994.
[21] P. Winter, "Steiner problem in networks: A survey," *Networks*, vol. 17, pp. 129–167, 1987.

Ornan (Ori) Gerstel (M'95–A'96) received the B.A., M.Sc., and D.Sc. degrees from the Technion—Israel Institute of Technology, Haifa, Israel.

After recieving the D.Sc. degree, he joined the Optical Network Systems Group, IBM T.J. Watson Research Center, Hawthorne, NY, and has moved with the group to develop optical networking products with Tellabs Operations. He is currently the System Architect for the Optical Networking Group, Tellabs Operations, Hawthorne, NY. His research interests include routing and related network design problems in ATM and WDM networks.

Israel Cidon (M'85–SM'90) received the B.Sc. (summa cum laude) and the D.Sc. degrees from the Technion—Israel Institute of Technology, Haifa, Israel, in 1980 and 1984, respectively, both in electrical engineering.

From 1984 to 1985 he was with the Faculty of the Electrical Engineering Department at the Technion. In 1985 he joined the IBM. T. J. Watson Research Center, Hawthorne, NY, where he was a Research Staff Member and the Manager of the Network Architectures and Algorithms group, involved in various broadband networking projects such as the PARIS/PLANET Gigabit networking testbeds, the Metaring/Orbit Gigabit LAN, and the IBM BroadBand Networking architecture. In 1994 and 1995 he was with Sun Microsystems Laboratories, Mountain View, CA, where he was the Manager of High-Speed Networking, founding various ATM projects including Openet, an open and efficient ATM network control platform. Since 1990 he has been with the Department of Electrical Engineering at the Technion—Israel Institute of Technology, Haifa, Israel. He is a Founding Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING. He has served as the Editor for Network Algorithms for the IEEE TRANSACTIONS ON COMMUNICATIONS and as a Guest Editor for *Algorithmica*. His research interests are in high-speed networks, distributed network applications and algorithms, and mohile networks.

Dr. Cidon received the IBM Outstanding Innovation Award in 1989 and 1993 for his work on the PARIS high-speed network and topology update algorithms, respectively.

Shmuel Zaks, photograph and biography not available at the time of publication.