Efficient Support for the Client/Server Paradigm over Heterogeneous ATM Networks

Ornan Gerstel *
T.J. Watson Research Center, Hawthorne, NY 10532
E-mail: ori@watson.ibm.com (Corresponding Author)

Israel Cidon
Electrical Engineering Department, Technion, Haifa 32000, Israel
E-mail: cidon@ee.technion.ac.il

Shmuel Zaks
Computer Science Department, Technion, Haifa 32000, Israel
E-mail: zaks@cs.technion.ac.il

Abstract

We present a new network design problem that is applicable when designing virtual paths in an ATM network to efficiently support client/server applications. We present several alternatives for the solution, discuss their pros and cons, 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.

Keywords: Virtual Path design, Virtual Path routing, ATM network, Client/Server paradigm, Network design.

1 Introduction

1.1 ATM networks

The *Asynchronous Transfer Mode* (ATM) is the network architecture standard proposed for 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, 14, 18]).

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 *virtual channel identifier* (VCI) and *virtual path identifier* (VPI). Since ATM is connection-oriented, the cells are routed on predetermined routes in the network, which must be set up prior to their usage for data transfer.

*Part of this authors work was done while being with the Computer Science Dept., Technion, Haifa 32000, Israel.

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 VPIs (VPs) and those based on VCIs (VCs). VPs may be viewed as virtual links, connecting (possibly remote) switches as neighboring nodes in a virtual network, while the route of VCs may be viewed as a concatenation of multiple VPs. We term the number of VPs that are used by a VC along its route, the *VP hops* of the VC.

In ATM, VCs are typically used for connecting network users, while VPs are used for routing VCs and for simplifying network resource management.

During the setup of a new VC, the network layer software of switches at the end of each VP along the VC's route, must be involved to set the appropriate routing data in the VC routing tables — a process which is responsible for a large part of the setup overhead; thus this overhead is proportional to the VP hops of the VC. In the communication literature, there are three potential types of ATM switching nodes:

CX nodes: 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 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" (CXs for short).

PX nodes: 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 VPs exclusively, hence they are termed "VP switches" (PX for short).

PCX nodes: 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 PXs). 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 a port into which the cell is sent (as in CXs). This mechanism enables some VPs to terminate at the node (by having "null" in the entry pointed by their VPI value), and some VPs to pass through the node. We term these nodes "PCXs" since they switch both VPs and VCs.

The VC and VP concepts received much attention in the communication literature, yet the problem of how VPs should be laid out in a given network topology was addressed only in a few works [2, 1, 13, 6, 15]. In these works, only one type of nodes was assumed (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.

1.2 The client/server paradigm

One of the major factors in the success of the penetration of ATM into computer communication markets is its ability to adequately support existing computer services and applications such as IP services, mail applications, file transfer, and various distributed applications.

A very common paradigm for distributed applications is the client/server paradigm. Implemented explicitly in many distributed 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, which is accessed by many clients, whenever they need data from the server. The clients issue short request messages (typically less than 1Kbyte), that are handled by the server, which consequently sends a (typically much longer — up to several Mbytes in current applications) response [5]. Examples for such applications are distributed file servers (e.g. NFS), file repositories (in which a file is requested by its name, keywords etc., and is transmitted by the server as a response), databases (in which a query is sent as a request, and the relevant data is sent as a response), name servers, and address directories (such as X.500).

Due to the central role that the client/server paradigm plays in existing distributed applications, it is obvious that it must be properly accomodated 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 connectionless data service [17], to send both requests and responses encapsulated in datagrams. This alternative assumes a special connectionless service either within or on top of the ATM network, which enables sending short messages, without prior setup of a VC for this purpose. However, such a service will not necessarily be available in every ATM network, since it requires special mechanisms at the switch level, or additional overlay switching. In addition, most such services do not guarantee low data loss probability, since they cannot rely on bandwidth allocation.

2. Use regular VCs 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. XTP and VMTP [16]) and we focus on it in the rest of the paper.

1.3 Contribution of this paper

In this paper we discuss methods for supporting such client/server applications over an ATM network by setting up semi-permanent VPs in the network, that enable clients to quickly set up short-lived VCs each time they wish to submit a request to the server. After the setup, these VCs are used by the client to send a request to the server, by the server to return a response, and are then terminated until the next request. We propose a method for designing the routes of VPs in a given network so as to accomodate such VCs as quickly and as efficiently as possible. Our solution is optimal, both in the setup/teardown overhead for a new VC and its propagation delay, while keeping its resource utilization as low as possible. We also suggest a novel bandwidth allocation scheme which 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 [6] for simpler cases. Our greedy algorithm (see Section 5.2) is based on a framework suggested in [13]; however, while [13] applies to networks which are composed of PCXs exclusively and clients are assumed to be connected to each network switch, the present solution applies to more realistic (and complex) cases, in which the network is composed of all types of switches and clients are connected to a subset of the switches.

A preliminary version of this paper appeared in [12].

The paper is structured as follows: In Section 2 we present the design problem which stems from supporting client/server applications over ATM, and in Section 3 we explore the spectrum of possible solutions to the problem (including an overview of our approach). In Section 4 we compare the possible solutions using numerical simulation results, while in Section 5 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 6.

2 The design problem

The main focus of this paper is a design problem, is which it is requested to determine a set of VPs 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 VPs will enable setting up the VCs as quickly as possible, while not requiring too many network resources.

The design problem for a given client/server application[1] $\mathcal{X}$ is based on the following 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. $\mathcal{T}_{propag}(e)$ — The propagation delay of each physical link $e \in E$,
5. $\mathcal{BW}_{avail}(e)$ — The amount of available bandwidth on each link which may be allocated for the purposes of $\mathcal{X}$.
6. $\mathcal{N}_{req}$ — The maximum number of concurrent requests from clients to the server, that is supported by $\mathcal{X}$.
7. $\mathcal{BW}_{req}$ — The required bandwidth for a single request from a client to the server of $\mathcal{X}$,
8. $\mathcal{BW}_{resp}$ — The required bandwidth for a single response from the server to a client of $\mathcal{X}$,
9. $\mathcal{L}_{max}$ — The maximum allowed number of entries in any VP routing table, that are used for purposes of $\mathcal{X}$ (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 VPs 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.

The design goal is to determine a set of VPs, a route in the network for each such VP and its bandwidth with the following characteristics:

1. The total bandwidth of VPs that share a link e does not exceed the link's bandwidth constraint $\mathcal{BW}_{avail}(e)$.
2. The VPs have enough bandwidth to support $\mathcal{N}_{req}$ simultaneous requests from client to server and a single response from server to client.
3. The VPs obey the constraints imposed by the different types of switching nodes (e.g. no VP terminates at a PX node).

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

4. The number of VPs that use any given physical link is bounded by $\mathcal{L}_{max}$ (see Observation 1 below).

While the above characteristics determine a feasible solution, an optimal solution reduces the overhead of the setup of VCs from client to server, by achieving the following targets:

1. The maximum number of VP hops between any client node to the server's node is minimized.
2. The physical route that each VC takes is the shortest possible in the network in terms of its propagation delay.

Observation 1. *The number of non-empty VP routing entries at a given port processor* [2] *is equal to the number of incoming VPs that use the link connected to this port. 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 will be more convenient to relate to $\mathcal{L}_{max}$ *as the maximum number of VPs that share a link, rather than a limit on the 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 VPs that use the faulty link to other paths in the network by this achieving a low overhead migration of all VCs that use the faulty link (and thus use the rerouted VPs) to alternative routes. The overhead of this recovery process is clearly proportional to the number of VPs that share the link and is thus bounded by* $\mathcal{L}_{max}$.

3 Possible solutions

3.1 Simple approaches

Three straightforward solutions to this problem suit small networks but do not scale well for large networks (see Figure 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 in this solution is its inefficient bandwidth allocation: each such VC requires allocated bandwidth — especially from the server to the client (where the bandwidth requirement is not trivial) — which cannot be reused by any other client. Another major drawback is that these VCs must be maintained (e.g. when a physical link fails), an overhead that is not justifiable if they are not frequently utilized.

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 VCs 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

[2] We assume here the switch architecture of [7], in which the switch is composed of 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. This architecture is a widely accepted alternative for switch implementation.

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

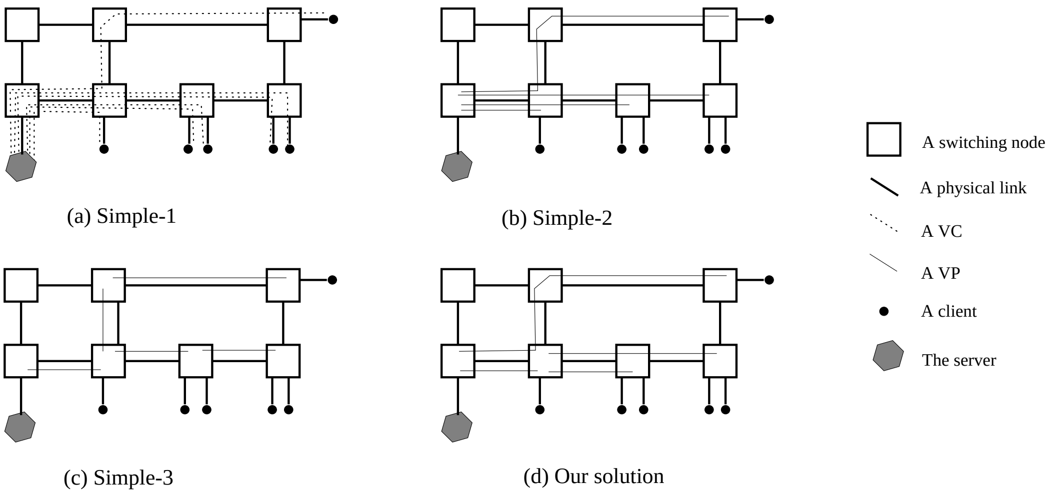

Figure 1: Solutions to the VP layout problem

setup is very quick. A main disadvantage of this solution is that the allocated bandwidth to each such "direct" VP may be reused only by clients which are connected to the same switching node. Another disadvantage is that VP routing tables at nodes which are close to the server tend to get overloaded; if a client/server application is large (or the servers of numerous such applications are geographically close), the VP routing tables in the vicinity of the server are easily filled up entirely with entries of such VPs.

Simple-3: Create only VPs that span a single physical link. This solution is optimal in the degree of bandwidth reuse, since a VP may be used by all clients for which the chosen physical route includes the appropriate link. The disadvantage of this solution lies in the large number of VPs through which the VC is routed, and hence the very long (and thus impractical) setup time.

Example 1. Consider Figure 1, which demonstrates the three above-mentioned solutions (Parts (a)-(c)), and one of the options, enabled by our solution (Part (d)) on the same network. The following table includes the resources utilization of these solutions. In the table, bandwidth is measured in multiples of $(\mathcal{BW}_{req}, \mathcal{BW}_{resp})$ (i.e. 6 stands for a bandwidth of $(6 \cdot \mathcal{BW}_{req}, 6 \cdot \mathcal{BW}_{resp})$). 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 propagation delay of the various links, however we assumed all propagation 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).

Solution	Max VP entries per table	Max BW per link	Max VP 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

From the drawbacks of the above-mentioned SIMPLE-1 solution (based on permanent VCs), it is evident that any reasonable solution will be based on VCs that are set up on-demand, for a specific request-response interaction, after which they are removed from the network until the next request is submitted. It is therefore evident that an overhead of a lengthly connection setup procedure for such "short-lived" VCs may exceed the necessary time for the data transfer itself. Thus, it is most important to keep this overhead very low, using the following techniques:

- The setup overhead of a VC is directly proportional to the number of VPs which are used by the VC along its route (i.e. the VP hops of the VC). 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. Therefore, VPs that cater to such applications must be designed so as to keep the VP hops from each client to the server low.

- In order to reduce the effect of propagation delay on the setup protocol it has been suggested (e.g. [16]) that the setup protocol triggered by the client will also create the reverse VC (i.e., from the server to the client), to carry the response by the server, in addition to setting up a VC from the client to the server. This technique reduces the setup overhead from two round trip delays to just one such delay. This setup is termed *implicit setup* in [16].

The latter technique essentially requires that the routes taken by the VC from client to server and vice versa will be along the same switches and links, but in opposite directions. Such pairs of VCs are termed a *symmetric* VC; a symmetric VP is similarly defined. The bandwidth of a symmetric VC/VP will be denoted by a pair (X, Y) 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 VCs/VPs, so the term "symmetric" is omitted for the sake of clarity.

3.2 Our approach

3.2.1 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, by this 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 $\mathcal{BW}_{avail}(e)$, we find a tree that spans all the client' switching nodes, the server's switching node, and all necessary intermediate nodes. All the routes of VCs 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 which may be desirable for a CSBT. Each such set yields a different spanning tree, and the possible options are discussed below. From 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 delay,
- The free bandwidth ($\mathcal{BW}_{avail}(e)$) of each tree link e is large enough to accommodate the required bandwidth $(\mathcal{BW}_{req}, \mathcal{BW}_{resp})$ for all VPs that use e. Since there are at most $\mathcal{L}_{max}$ such VPs, The bandwidth is bounded by at least $(\mathcal{L}_{max} \cdot \mathcal{BW}_{req}, \mathcal{L}_{max} \cdot \mathcal{BW}_{resp})$.

Phase-2: Given CSBT and the type of each node, we determine a set of VPs that enable connecting each client to the server using the path in CSBT, so that no more than $\mathcal{L}_{max}$ VPs 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 5 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 contains the run-time aspects of the service. These aspects include variations in the protocols for setting up (and taking down) VCs [16], 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 the sake of brevity, we do not extend the description of PHASE-1 and PHASE-3 beyond this overview, and focus in the rest of the paper on PHASE-2.

3.2.2 Overview of Phase-1

A first question that arises from the above description is: why is the base subgraph which spans the server and the clients, taken to be a tree? The answer to this question is three-fold:

1. It was proven in [13] that if the base is a general topology network, then the problem of PHASE-2 is NP-complete, and thus it is not plausible to find efficient solutions for it.
2. Since the bandwidth of classic client/server applications is not expected to be high in comparison to ATM capabilities, we sacrifice absolute optimality for the sake of design simplicity.
3. In Section 3.3 we present a novel, efficient bandwidth allocation scheme tailored for client/server applications, for which the optimal spanning subgraph is clearly a tree.

$\Downarrow$

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:* $C \propto \sum_{nodes_{a,b}} BW(a,b)Dist(a,b)$*:* This is the CSBT mentioned earlier, which is assumed in the rest of the paper due to its good charachteristics, reasonable cost model and algorithmic feasibility. To find such a tree, take the given graph G and remove from it links which do not have enough bandwidth to support $\mathcal{L}_{max}$ VPs with $(\mathcal{BW}_{req}, \mathcal{BW}_{resp})$ bandwidth each. In the remaining graph, find paths with shortest propagation delay from the server to all 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:* $C \propto \sum_{tree\ edge_e} Cost(e)$*:* This tree is hard to find algorithmically. Its NP hardness is proven in Appendix A.1. In addition, its cost model which ignored bandwidth considerations seems less reasonable.

CSBT-3. *Minimize cost C only, where:* $C \propto \sum_{tree\ edge_e} Cost(e)$*:* If the previous tree characterization is simplified by removing propagation 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 2 [19].

CSBT-4. *Any tree such that the bandwidth used per link does not exceed the available bandwidth $\mathcal{BW}_{avail}(e)$:* While Option 3.2.2 only considers links which can support $\mathcal{L}_{max}$ VPs, it may be useful to also consider links with less free bandwidth ($\mathcal{BW}_{avail}(e) < \mathcal{L}_{max}$). Such an approach may yield a feasible solution in places where Option 3.2.2 cannot find one. The main drawbacks of this approach are: (1) It yields solutions which tightly fit into the available bandwidth and may leave no room for expanding the CSBT to support more clients, (2) It is algorithmically hard to find: the NP-hardness of the algorithm is proven Appendix A.1.

$\Uparrow$

3.2.3 Overview of Phase-2

This phase is based on a greedy procedure called `find_layout`, which is described in detail in Section 5.2. `find_layout` takes as an input the tree CSBT, an upper bound on the allowed VP hops (h) and an upper bound on the allowed load at any tree link ($\mathcal{L}_{max}$); it returns as an output the layout of VPs, or the fact that it failed to find a layout that meets the above bounds. In Section 5.3 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 VPs from the leaves towards 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 (which includes at this point only the link from v to its parent), and strives to extend all the VPs from v's subtree (including the new VP) towards the root. Despite this effort, some (if not most) of the VPs do not reach the root for several reasons:

- Whenever `find_layout` encounters a VC switch (i.e. v is a CX node), all the VPs 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., towards the root) exceeds $\mathcal{L}_{max}$, `find_layout` tries to reduce the number of VPs 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 which are PCXs (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 is increased. The crux of this algorithm is in the choice of VPs to be stopped, so that the maximum VP hops from client to server is minimally increased.

3.2.4 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) — as suggested in [16]. In addition, bandwidth for a request and response ($\mathcal{BW}_{req}, \mathcal{BW}_{resp}$) is allocated in the appropriate directions. Teardown may be performed either explicitly or implicitly (see [16] for details).

The second category includes changes in the VP layout, caused by changes in the set of clients which 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 [4], 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).

3.3 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, which suits client/server applications. The scheme is considerably more efficient than the standard scheme, and enables more concurrency. The solution in this paper may be used with both schemes.

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 standard bandwidth allocation scheme for client/server applications: Allocate the necessary bandwidth $(\mathcal{BW}_{req}, \mathcal{BW}_{resp})$ for a maximum size request (in one direction) and a maximum size response (in the other) for every VP. Each implicit VC setup procedure will allocate $(\mathcal{BW}_{req}, \mathcal{BW}_{resp})$ bandwidth for both of its directions. This scheme was assumed in the above discussion.

The efficient scheme: Rely on the server for keeping the utilized bandwidth under the allocated limit: Allocate $(\mathcal{N}_{req} \cdot \mathcal{BW}_{req}, \mathcal{BW}_{resp})$ 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 $\mathcal{BW}_{resp}$ bandwidth in the direction of all the clients together, hence keeping the utilized bandwidth on any link under the allocated $\mathcal{BW}_{resp}$.

Remark. Note that in the conservative scheme, bandwidth is allocated for a single request, while in the efficient scheme we allocate bandwidth for $\mathcal{N}_{req}$ concurrent requests (in both cases, the bandwidth from the server supports a single response). If one wishes to support $\mathcal{N}_{req}$ concurrent requests using the conservative scheme, it does not suffice to allocate $(\mathcal{N}_{req} \cdot \mathcal{BW}_{req}, \mathcal{BW}_{resp})$ bandwidth for every VP, since if several clients that use the VP attempt to concurrently set up VCs, all attempts but one will be rejected by the network, since all the $\mathcal{BW}_{resp}$ bandwidth in the direction of the clients will be allocated to a single client. Alternatively, one could allocate $(\mathcal{N}_{req} \cdot \mathcal{BW}_{req}, \mathcal{N}_{req} \cdot \mathcal{BW}_{resp})$ for each VP, but since $\mathcal{BW}_{resp}$ is considerably larger, the approach is quite inefficient.

The two schemes suit different cases: If the client/server application is supported by a general purpose, public network (e.g. a national communication supplier), it is more reasonable to use the conservative scheme, since it does not require a specialized signaling mechanism, and switch functions

[4] Generally speaking, when a new client subscribes, a new VP is constructed from it to the closest node in CSBT, when a client unsubscribes - its VP is deleted from the network, provided no other clients depend on it for connecting the server.

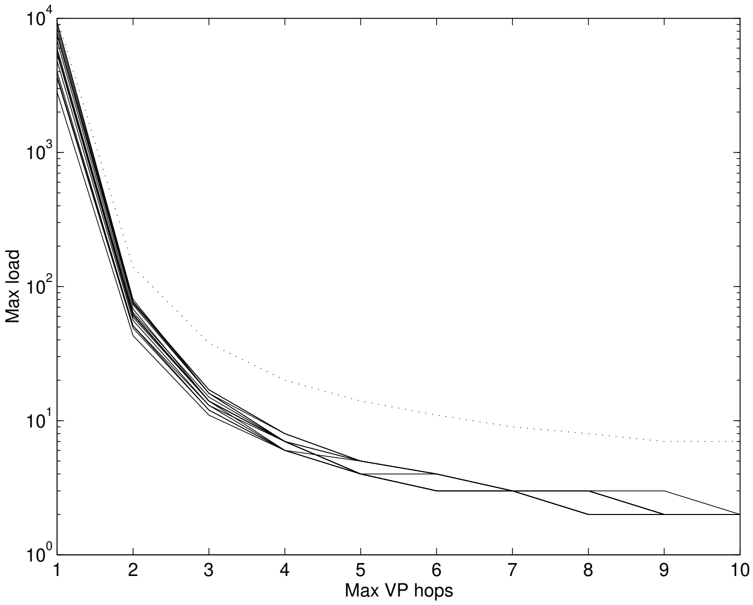

Figure 2: The tradeoff between load and VP hops

for such applications. On the other hand, if the application is supported by a computer oriented LAN/MAN, in which many of the connections are for client/server applications, an efficient scheme will lower the VC setup overhead (no need for costly bandwidth allocation procedures), and will 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 $(\mathcal{BW}_{req}, \mathcal{BW}_{resp})$ bandwidth, resulting in a maximum of $(\mathcal{L}_{max} \cdot \mathcal{BW}_{req}, \mathcal{L}_{max} \cdot \mathcal{BW}_{resp})$ bandwidth per link; in the efficient scheme each physical link in CSBT is allocated only $(\mathcal{N}_{req} \cdot \mathcal{BW}_{req}, \mathcal{BW}_{resp})$ bandwidth, a decrease of a factor of nearly $\mathcal{L}_{max}$ in the allocation, with a substantial increase in the level of concurrency.

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 Simulation results

We have tested our solution on randomly [5] chosen CSBTs. Our results are depicted in Figure ??.

Figure 2 shows an interesting tradeoff between the number of VPs that share a link (Max load), and the maximum number of VP hops. The graphs were obtained for 15 CSBTs with 10,000 switches each (drawn in full lines), and a chain tree, with the server at one end — which 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.

[5] The simulations were conducted on random CSBTs, 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 PCXs.

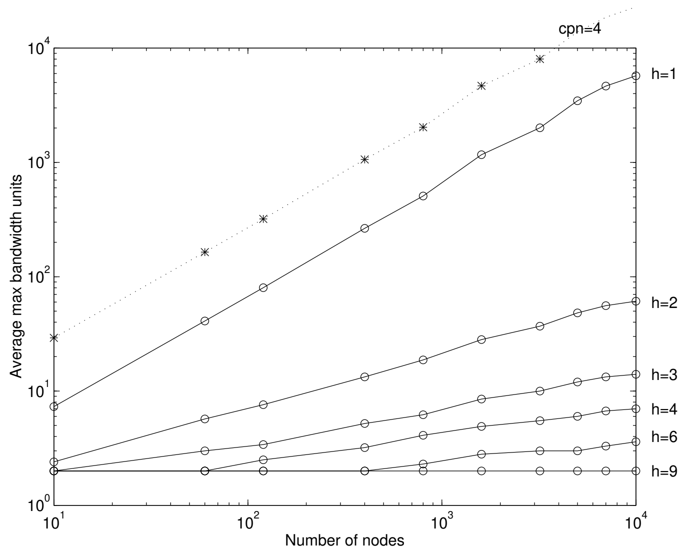

Figure 3: Scalability of our solution

SIMPLE-2) requires an average load of above 5000 [6], it suffices to allow 2 VP hops to reduce the average load to below 60 (and below 14 if 3 VP hops are allowed).

Figure 3 shows the scalability of the solutions in terms of the average required multiple of bandwidth units (were $(\mathcal{BW}_{req}, \mathcal{BW}_{resp})$ is considered a single unit), as networks grow larger. The results are based on an average of the maximum required bandwidth units in each CSBT, taken over 10 CSBTs per each network size. The dotted line labeled `cpn=4` shows the behavior of SIMPLE-1, assuming there are 4 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 3 VP hops are sufficient to support a reasonable number of client/server applications on a large size ATM network.

5 Optimal solution for PHASE-2

5.1 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 communication network* $\mathcal{N}$ is a tuple $\mathcal{N} = (T, s, CN, CX, PX, PCX)$ where:

1. $T = (V, E)$ is the CSBT produced by PHASE-1.
2. $CX \subseteq V$ is the set of CX nodes in CSBT (in which VPs always terminate).
3. $PX \subseteq V$ is the set of PX nodes in CSBT (in which VPs never terminate).

[6] Note that this load is unacceptable even if the whole 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

4. $PCX \subseteq V$ is the set of PCX nodes in CSBT (in which VPs may either terminate or not).

5. CX, PX, PCX are mutually disjoint sets, the union of which is equal to V.

6. $s \in CX \cup PCX$ is the *server node* of the current application,

7. $CN \subseteq CX \cup PCX$ is a set of *client nodes.*

We shall denote these entities by $T(\mathcal{N}), s(\mathcal{N}), CN(\mathcal{N}), CX(\mathcal{N}), PX(\mathcal{N}), PCX(\mathcal{N})$ 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 $T(\mathcal{N})$, let $\mathcal{P}(T(\mathcal{N}))$ be the set of all simple paths in $T(\mathcal{N})$. A *client/server virtual path layout (CSVPL for short)* $\Psi = (\mathcal{N}, T_\Psi, \mathcal{I})$ is represented by a tree $T_\Psi = (V, E_\Psi)$ rooted at $s(\mathcal{N})$ and a function $\mathcal{I} : E_\Psi \to \mathcal{P}(T(\mathcal{N}))$, where an edge $\psi = (a, b) \in E_\Psi$ corresponds to a VP between a and b. The function $\mathcal{I}(\psi)$ maps each VP $\psi = (a, b)$ to its path 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.

Definition 3 (VP routing table load). The *load* $\mathcal{L}(e)$ on a link $e \in E(\mathcal{N})$ is the number of VPs $\psi \in E_\Psi$ 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 $\mathcal{L}(\Psi) = \max_{e \in E(\mathcal{N})} \mathcal{L}(e)$.

We extend the definition of $\mathcal{I}$ to simple paths in T_Ψ, as follows:

Definition 4 (Induced path). The *induced path* $\mathcal{I}(p)$, for a path $p \in \mathcal{P}(T_\Psi)$, where $p = (\psi_1, \psi_2, ..., \psi_k)$ 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_is.

We now define $\mathcal{H}(v)$ to be the minimum number of VPs that may be used to form a VC between a client node v and the server $s(\mathcal{N})$, such that the VC uses the shortest path in CSBT. Formally:

Definition 5 (VP hops). The *VP hops* $\mathcal{H}(v)$ for a node $v \in CN(\mathcal{N})$ is the minimum k such that:

1. $\exists p = (\psi_1, \psi_2, ..., \psi_k) \in \mathcal{P}(T_\Psi)$, ($\psi_i \in E_\Psi$ for all i),

2. $\exists x, y \in V(\mathcal{N})$, $\psi_1 = (v, x), \psi_k = (y, s(\mathcal{N}))$,

3. The induced path $\mathcal{I}(p)$ is a shortest path between v and $s(\mathcal{N})$ in $T(\mathcal{N})$.

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

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

Definition 6 (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. $\mathcal{L}(\Psi) \leq \mathcal{L}$.

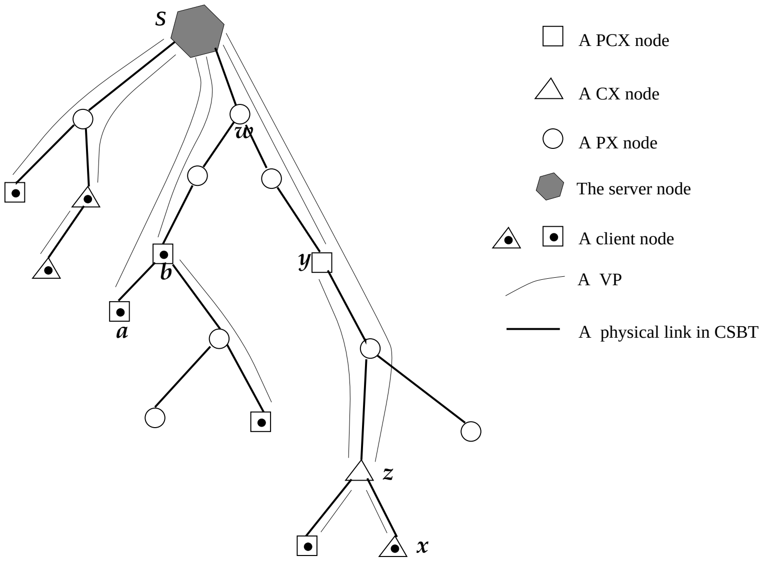

Figure 4: A CSVPL on a CSBT

Thus, in a feasible CSVPL, all the VPs that include a CX node, terminate at that node, and all VPs that include a PX node cannot terminate at it, and are switched to an adjacent node. Also, in such a CSVPL, every VC from a client node to the server consists of no more than h VPs, and uses the shortest physical path in $T(\mathcal{N})$.

We now define an optimal solution as a feasible solution that does not require much of the network resources:

Definition 7. 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 VPLs.

Example 2. Consider the CSBT in Figure 4 and the CSVPL (depicted in bold lines). It is easy to verify that the CSVPL obeys the routing rules of the various nodes, e.g. at node y some VPs go through the node while others terminate, at node w no VP terminates and at node z all VPs terminate. The VP hops for node x satisfies $\mathcal{H}(x) = 2$ since the route $(x, z), (z, s)$ is a shortest route in the tree from x to s. The load on link (w, s) satisfies $\mathcal{L}((s, w)) = 4$ and this is also $\mathcal{L}(\Psi)$.

5.2 The VP design algorithm

We now present a polynomial time greedy algorithm which produces an optimal CSVPL. The algorithm framework is based on the one presented at [13], however the present case is much more complex due to the various types of switches (in [13] all switches were assumed to PCXs, a fact which substantially simplified the algorithm and its analysis).

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

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

- $X[i]$ - the i^{th} element in the vector X,

- $||X||$ - the sum of all the elements $X[i]$,
- $X[^j_i]$ - the part of the vector from the i^{th} to the j^{th} position,
- $c[i]$ (where c is a constant) - a vector of i elements which are equal to c,
- $X \bullet Y$ - the concatenation of the vectors X and Y.

The `find_layout` procedure starts from the leaves of the tree $T(\mathcal{N})$ and advances towards the root $s(\mathcal{N})$. For each node v it maintains the number of VPs that go through v (and thus contribute to the load on the link from v to its parent in the tree). This number is kept in a vector $L_v \in \mathbb{N}^h$, where $L_v[i]$ holds the number of VPs that are used as the i^{th} hop for some client node x.

Specifically, `find_layout` creates a first-hop VP for each leaf in $CN(\mathcal{N})$ (at line 4) and no VP for leaves which 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 the VPs that arrive at it from all its sons - reflecting an attempt to continue VP through v without increasing the VP hops of the CSVPL. To this vector, an additional first-hop VP is added - from v towards the server, if v is a client node (at line 9). This VP will be used to carry VCs from the clients connected to v. If v is a CX node, then all VPs from the descendents are stopped at v (they cannot continue by definition of a CX node).

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. If, however, the load is too high, it is reduced by transforming VPs at some chosen node x (using the $Transform$ procedure defined below). The transformation involves stopping some VPs at x, by this 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 ($x = v$) in a way that does not increase the VP hops too much (at lines 13-15 - see Figure 5). If v is a PX node, it cannot be transformed (since VPs 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 VPs that can be stopped without increasing the VP hops too much (at lines 18-21 - see Figure 6). In this case, the algorithm may iterate on the descendents of v, and repeatedly transform them until the load 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 Figure 5, is that it is better to stop VPs with as little VP hops as possible (i.e. VPs with VP hops j which is lower than some value i in the figure). This causes the VP hops of the VP that starts at v to be $i+1$. All the other VPs that go through v, continue towards the parent of v.

Definition 8. Define the following:

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

$$Transform(L, i) \equiv L \leftarrow 0[i] \bullet (L[i+1]+1) \bullet L[^h_{i+2}]$$

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

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

- For a vector $V \in \mathbb{N}^h$ define $H(V) = \max\{i \in \{1, ..., h\} | V[i] > 0\}$

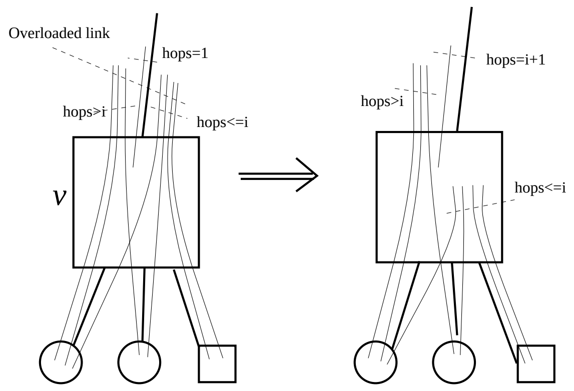

Figure 5: Transformation of a node $v \in PCX(\mathcal{N})$ - $Transform(L_v, i)$

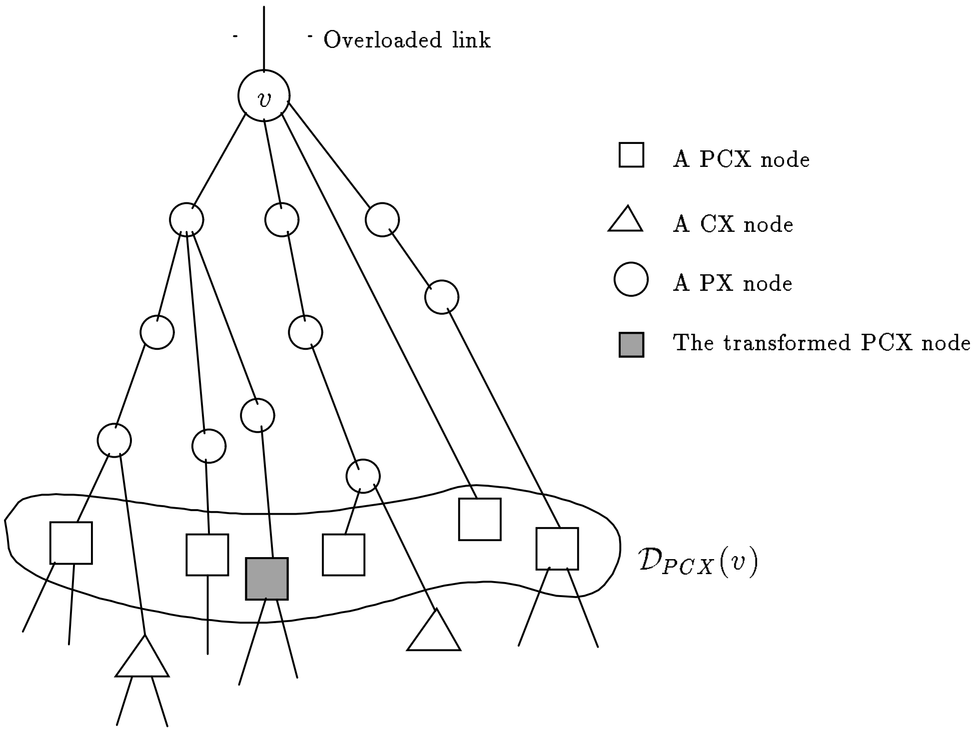

Figure 6: Transformation of a PCX node due to an overloaded vector L_v of a PX node v

- For a node $v \in PX$ define $\mathcal{D}_{PCX}(v)$ to be the set of descendents of v which are PCX nodes, and for which the root to v consists of PX nodes only (see Figure 6).

A formal description of the algorithm follows.

function find_layout(tree $\mathcal{N}$, integer $\mathcal{L}_{max}$) return {SUCCESS, FAILURE}
   var L: set of vectors $\{L_v \in \mathbb{N}^h | v \in V(\mathcal{N})\}$)
begin
   for all $w \in V(\mathcal{N})$:
      if w is a leaf then $L_w \leftarrow Extra(w)$ otherwise $L_w \leftarrow$UNDEFINED
   loop forever
      choose a node $v \in V(\mathcal{N})$ such that:
         (a) $L_v =$ UNDEFINED,
         (b) for all $x \in SONS(v)$, $L_x \neq$ UNDEFINED.
      if $v = s(\mathcal{N})$ then return SUCCESS

      loop forever
         $L_v \leftarrow \sum_{s \in Sons(v)} L_s + Extra(v)$
         if $v \in CX(\mathcal{N})$ then Transform$(L_v, H(L_v))$
         if $||L_v|| \leq \mathcal{L}_{max}$ then exit internal loop

         if $v \in PCX(\mathcal{N})$ then begin
            find $i \in \{1, ..., h-1\}$ such that
               (a) $||L_v[{}^{h}_{i+1}]|| < \mathcal{L}_{max}$,
               (b) $||L_v[{}^{h}_{i}]|| \geq \mathcal{L}_{max}$.
            if no such i exists then return FAILURE
            Transform(L_v, i)
         end if

         if $v \in PX(\mathcal{N})$ then begin
            find a node $w \in \mathcal{D}_{PCX}(v)$ and an integer $i \in \{1, ..., h-1\}$ such that:
               (a) $||L_w[{}^{i}_{1}]|| > 1$,
               (b) For every $w' \in \mathcal{D}_{PCX}(v) \setminus \{w\}$, $L_w[{}^{i}_{1}] \geq L_{w'}[{}^{i}_{1}]$,
               (c) For every $w' \in \mathcal{D}_{PCX}(v)$, $||L_{w'}[{}^{i-1}_{1}]|| \leq 1$.
            if no such i, w exist then return FAILURE
            Transform(L_w, i)
            Update L_x for every x in the path from w to v
         end if
      end loop
   end loop
end

5.3 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 Appendix A.2):

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 (although not necessarily the same 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 at every node v, $||L_v|| \leq ||M_v||$, by this proving that the solution of `find_layout` is optimal. To this end, we strengthen the claim we wish to prove, and 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 fit 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 (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 not necessarily tight, 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. □

6 Summary

In this paper we studied the problem of supporting a client/server application in an ATM network. We first listed the parameters which 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 solution to its resource utilization, and demonstrated numerically its characteristics. We also presented a new bandwidth allocation scheme, which is optimal for such applications.

Our solution is based on three main phases, amongst which the second phase (which deals with laying out the VPs 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 VPs and VCs (PCXs) in the VP design: these nodes enable the flexibility of choosing VPs 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.

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*, pages 192–200, 1994.

[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*, pages 98–105, 1993.

[3] A. Birrell and B. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1), February 1984.

[4] D.R. Cheriton. The V kernel: a software base for distributed systems. *IEEE Software*, 1(2), April 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. on Measurement and Modeling of Computer Systems*, May 1987.

[6] I. Cidon, O. Gerstel, and S. Zaks. A scalable approach to routing in ATM networks. In G. Tel and P.M.B. Vitányi, editors, *The 8th International Workshop on Distributed Algorithms (LNCS 857)*, pages 209–222, Terschelling, The Netherlands, October 1994. Springer Verlag. Accepted to IEEE/ACM Trans. on Networking.

[7] R. Cohen and A. Segall. Connection management and rerouting in ATM networks. In *IEEE Infocom'94*, pages 184–191, 1994.

[8] D. Dykeman (Editor). PNNI draft specification, 1995. Contribution 94-0471.

[9] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

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

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

[12] O. Gerstel, I. Cidon, and S. Zaks. Efficient support for the client-server paradigm over ATM networks. In *IEEE Infocom'96*, pages 1294–1301, 1996.

[13] O. Gerstel and S. Zaks. The virtual path layout problem in fast networks. In *The 13th ACM Symp. on Principles of Distributed Computing*, pages 235–243, Los Angeles, USA, August 1994. Accepted to a special issue of the Chicago Journal of Theoretical Computer Science.

[14] R. Händler and M.N. Huber. *Integrated Broadband Networks: an introduction to ATM-based networks*. Addison-Wesley, 1991.

[15] E. Kranakis, D. Krizanc, and A. Pelc. Hop-congestion tradeoffs for high-speed networks. In *7th IEEE Symp. on Parallel and Distributed Processing*, pages 662–668, 1995.

[16] T.F. La Porta and M. Schwarz. Architectures, features, and implementation of high-speed transport protocols. *IEEE Communications Magazine*, pages 14–22, May 1991.

[17] J.Y. Le Boudec, A. Meier, R. Oechsle, and H.L. Truong. Connectionless data service in an ATM-based customer premises network. *Computer Networks and ISDN Systems*, 26:1409–1424, 1994.

[18] C. Partridge. *Gigabit Networking*. Addison Wesley, 1994.

[19] P. Winter. Steiner problem in networks: A survey. *Networks*, 17:129–167, 1987.

A Appendix

⇓

A.1 NP hardness of finding the CSBT

In this appendix we prove that two of the options for a CSBT, layed out in Section 3.2.2 are NP-complete.

A simplified version of the CSBT-2 option is the following one, for the case when all links have propagation delay 1:

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 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]): Give a set $S = \{s_i\}_{i=1}^k$, a set of subsets $C \subseteq 2^S$, and an integer B, does there exist a subset of C, $C' \subseteq C$ which covers all 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' which 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'. □

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 is 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 towards the server is proportional to the number of client nodes in the subtree, it grows on links which are closer to the root. If the available bandwidth for each link e is $\mathcal{BW}_{avail}(e) = \mathcal{L}_{max} B$ 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 vertex disjoint path problem ([11, Problem ND40]): Given a graph G and a set of vertex dispoint pairs $\{(s_i, t_i)\}_{i=1}^k$, do there exist vertex 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 $B = k+1$.

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 other than r, a cycle is formed). The vaule of B and the different number of X_i and Y_i nodes for each i ensure that the paths are from t_i to s_i (and not some other s_j), since only this was can each subtree contain no more than B nodes from S. □

⇑

A.2 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 9. 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 $\{M_v \in \mathbb{N}^h | v \in V(\mathcal{N}),\ v \neq s(\mathcal{N})\}$, such that for every node $v \neq s(\mathcal{N})$ and $M'_v = \sum_{s \in Sons(v)} M_s + Extra(v)$, the following conditions hold:*

1. *$\mathcal{L}((v, parent(v)) = ||M_v||$,*
2. *If v is a leaf then $M_v = Extra(v)$, and*
3. *If $v \in CX(\mathcal{N})$ then $M_v = 0[H(M'_v)] \bullet 1 \bullet 0[h - H(M'_v) - 1]$*
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 10 ([13]). A vector L is called k-*non-trivial* iff $||L[_1^{k-1}]\,|| \le 1$ and $||L[_1^{k}]\,|| > 1$; in particular, if $L[1] > 1$ it is 1-non-trivial, and if $||L|| \le 1$ then L is ∞-non-trivial.

The following Lemma is an extension of the one proposed in [13]. 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 $1, ..., s$). Before transforming a node in $\mathcal{D}_{PCX}(v)$, the inequalities of Figure 7(a) hold by the induction hypothesis. After transforming some node k at line 20, the inequality for k may be violated (as in Figure 7(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 Figure 7(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 $v \neq s(\mathcal{N})$, the following holds:*

$$\begin{pmatrix} L_1 & \leq & M_1 \\ & \vdots & \\ L_k & \leq & M_k \\ & \vdots & \\ L_s & \leq & M_s \end{pmatrix} \Longrightarrow \begin{pmatrix} L_1 & \leq & M_1 \\ & \vdots & \\ \tilde{L}_k & ? & M_k \\ & \vdots & \\ L_j & < & M_j \\ & \vdots & \\ L_s & \leq & M_s \end{pmatrix} \Longrightarrow \begin{pmatrix} L_1 & \leq & M_1 \\ & \vdots & \\ \tilde{L}_k & \leq & \hat{M}_k \\ & \vdots & \\ L_j & \leq & \check{M}_j \\ & \vdots & \\ L_s & \leq & M_s \end{pmatrix}$$

(a) (b) (c)

Figure 7: Load vectors of $\mathcal{D}_{PCX}(v)$

1. *If* `find_layout` *is called with* $\mathcal{L}_{max} = \ell$ *then it does not return* `FAILURE` *while handling* v [7].

2. *The vector* L_v *produced by* `find_layout` *satisfies* $L_v \leq M_v$ *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 $L_v = Extra(v)$ by the algorithm and $M_v = Extra(v)$ by Lemma 4, thus condition 2 holds as well. At an internal node v, by induction $L_s \leq M_s$ for every son s of v. Let $L'_v = \sum_{s \in Sons(v)} L_s + Extra(v)$, and let $M'_v = \sum_{s \in Sons(v)} M_s + Extra(v)$. Clearly $L'_v \leq M'_v$.

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 $H(L'_v) \leq H(M'_v)$. It follows from line `10` that $L_v = 0[H(L'_v)] \bullet 1 \bullet 0[h - H(L'_v) - 1]$, It follows from Lemma 4 that $M_v = 0[H(M'_v)] \bullet 1 \bullet 0[h - H(M'_v) - 1]$, and thus $L_v \leq M_v$. □

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

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 $L_x \leq M_x$ 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 $L_x \leq \hat{M}_x$, $L_y \leq \check{M}_y$. The transformation will also obey the condition $M_x + M_y = \hat{M}_x + \check{M}_y$. By this we achieve the following

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

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-non-trivial. Split $\mathcal{D}_{PCX}(v)$ into the following two subsets:

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

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

$$\ell^- = \{y \in \mathcal{D}_{PCX}(v) | y \text{ is } j\text{-non-trivial for } j < i\}.$$

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

(1) For every $y \in \ell^+$, $L_y[^h_{i+1}] = M_y[^h_{i+1}]$: then since L_y is $i^{\geq}$-non-trivial, it can be shown that $||L_y|| \leq ||M_y||$. But 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 $L_y[^h_{i+1}] < M_y[^h_{i+1}]$, we transform the vectors in the following way.

$$\begin{cases} \tilde{L}_x & \longleftarrow 0[i] & \bullet(L_x[i+1]+1) & \bullet L_x[^h_{i+2}] \\ \hat{M}_x & \longleftarrow M_y[^i_1] & \bullet(M_x[i+1]+1) & \bullet M_x[^h_{i+2}] \\ \check{M}_y & \longleftarrow M_x[^i_1] & \bullet(M_y[i+1]-1) & \bullet M_y[^h_{i+2}] \end{cases}$$

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 $L_y[^h_{i+1}] \leq \check{M}_y[^h_{i+1}]$ and $L_y[^i_1] \leq L_x[^i_1] \leq M_x[^i_1] = \check{M}_y[^i_1]$. □

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

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

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

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

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 $||M_v|| \geq \sum_{s \in Sons(v)} ||M_s|| + 1$). 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 $M_v \geq \sum_{s \in Sons(v)} M_s + Extra(v) \geq \sum_{s \in Sons(v)} L_s + Extra(v) = L'_v$); it is also clear that $M_v > L'_v$ (otherwise $\ell < ||L'_v|| = ||M_v||$).

Now, if $M_v[^h_{i+1}] > L'_v[^h_{i+1}]$ then $M_v[^h_{i+1}] \geq L'_v[^h_{i+1}] + (1) \bullet 0[h-i-1] = L[^h_{i+1}]$ and since $M_v[^i_1] \geq 0[i] = L[^i_1]$, we get $M_v \geq L_v$.

On the other hand, if $M_v[^h_{i+1}] = L'_v[^h_{i+1}]$ then there exists $j \leq i$ such that $M_v[j] > L'_v[j]$ (and $M_v[^h_{j+1}] = L_v[^h_{j+1}]$) and $||M_v|| \geq ||M_v[^h_j]|| = ||M_v[^h_{j+1}]|| + M_v[i] > ||L'_v[^h_{j+1}]|| + L'_v[j] \geq \ell$ - 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 which 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.