The Layout of Virtual Paths on an ATM Network

Israel Cidon[1] , Ornan Gerstel[2] and Shmuel Zaks[3]

[1] Sun Microsystems Labs, 2550 Garcia Ave., Mountain View, CA 94043
E-mail: cidon@anchor.eng.sun.com

[2] Computer Science Dept., Technion, Haifa 32000, Israel
E-mail: orig@csc.cs.technion.ac.il

[3] Computer Science Dept., Technion, Haifa 32000, Israel
E-mail: zaks@cs.technion.ac.il

Abstract. In this paper we study the problem of designing a layout of virtual paths on a given ATM network. We first define a mathematical model that captures the characteristics of virtual paths. In this model we define the general virtual path layout problem, and a more restricted case; While the general case layout should cater connections between any pair of nodes in the network, the restricted case layout should only cater connections between a specific node to the other nodes.
For the latter case we present an algorithm that finds a layout by decomposing the network into sub-networks and operating on each sub-network recursively; This algorithm enables the formulation of a quantitative upper bound for the problem. We then present a greedy approach for the same problem, and prove the correctness and optimality of the resulting layout. Finally, we demonstrate how the algorithm for the restricted case is used as a building block in a solution to the general problem, and prove the asymptotic optimality of our result. The results exhibit a tradeoff between the efficiency of the call setup and both the utilization of the VP routing tables, and the overhead during recovery from link disconnections.

1 Introduction

1.1 Background

The Asynchronous Transfer Mode (ATM) [16, 10, 7] is the transmission and multiplexing technique which is the emerging industry standard for B-ISDN. ATM was chosen chosen by ITU (formerly CCITT), ANSI and a large group of companies which are members of the ATM forum. Due to the future importance of fast, broadband, integrated networks, ATM has been extensively discussed in recent years.

ATM is based on small fixed size packets, which are called *cells*. Due to the very high switching-rate requirements, the routing of cells must be carried at each network node by a dedicated hardware, implying simple routing algorithms. The routing scheme chosen in ATM is based on two fixed length fields in the header of each cell (VCI and VPI). These fields serve as indices into routing tables that reside at the nodes of the network, and they determine the route that a cell will take.

Routing in ATM is hierarchical in the sense that the VCI field of a cell is ignored in many nodes along the route, which perform the switching according to the VPI alone; Only at a small number of nodes, is the VCI considered for switching the cell and determining the next VPI and VCI. This scheme effectively creates two types of predetermined unidirectional routes in the network: those based on VPIs (called *virtual path connections* or VPs) and those based on VCIs (called *virtual channel connections* or VCs).

These two route types have different roles in the network: while VCs are used for creating a connection between two users of the network (e.g. a telephone call), VPs are used for bundling together several VCs that share part of their route, by this substantially reducing the magnitude of managed entities in the network. In particular, we are interested in two such management aspects:

1. Each VC requires a separate routing entry only at a small number of nodes, while in most of the nodes along its route, routing is performed in accordance to the VP in which it is contained and hence only a single routing entry is needed for that VP (rather than a separate VC entry at every node),
2. The time required for the setup of a new VC is proportional to the number of nodes in which routing tables must be updated. With VPs, the setup time depends on the number of VPs that are used by a VC (while without VPs, it depends on the total number of nodes in the path).

A broader discussion on B-ISDN and ATM may be found in [10, 7, 16].

1.2 Problem definition

The common view on the layout of VPs in an ATM network, is that VPs span through the entire network, connecting a pair of end nodes (or VP terminators) directly. In this view, a network of N nodes contains at least $N(N-1)$ VPs (probably much more, since multiple routes are desired between any pair of nodes, to overcome failures, and to enable better bandwidth allocation). This number requires an average of $N-1$ entries in the VP routing table of a node, with a high probability of $O(N^2)$ entries at certain, centrally located nodes. This layout is thus impractical for large networks, since the VPI is limited to 12 bits (and $O(N^2) \leq 2^{12}$ implies maximum network size of 64 nodes). Moreover, certain switch implementations further limit the number of bits actually used, because of hardware constrains and table size limitations. Such limitations are permitted by the UNI standard [10]. For example the Fujitsu ATM chip set [13] is limited to 10 bits.

A common solution to this problem, is a fragmentation of the network into domains, connected by VC switches (e.g. [26, 15]), however this solution causes a VC to be routed through a relatively large number of VC switches (if it passes through many domains), directly influencing the time required for a VC setup.

In this paper we propose a more integrated approach, in which the layout of VPs in a network is determined in a manner that allows to use small VP routing tables, and tune their size in accordance to the required call setup performance (thus exhibiting a tradeoff between the required time for call setup, and the size of the VP routing table).

Specifically, we study the problem of designing a graph over a given ATM network, the vertices of which are the nodes of the network, and the edges of which represent VPs — we term this graph the *virtual path pair layout* (VPPL for short). Since network design is a complex task [4], we separate the VPPL design from the design of the network itself; In other words, we design a VPPL for a given network, rather than changing the design of the network according to the layout considerations. In our layout design we have the following assumptions:

(A1) Linear connection structure: It is commonly assumed that VPs/VCs are coupled in pairs of unidirectional routes in opposite directions [5] since this structure improves connection management substantially [8]. This coupling defines *VP VC pairs* (VPPs/VCPs), and in the sequel we refer exclusively to them rather than to VPs/VCs; Thus, a VPP/VCP is a bidirectional, simple route in the network. In addition, a VCP may be viewed as composed of concatenated VPPs (refer to Figure 1 for a graphic demonstration of these definitions).

(A2) Full switching capability: We assume that each node can switch both VPs and VCs. This assumption is implied by an architecture in which VP and VC routing tables reside in every node/port-processor in the network. When a cell arrives at a node, its VPI is used to determine the next VPI (in the label swapping

[4] Network design typically involves multiple optimization criteria and a large number of input parameters, often resulting in a combination of automatic tools, heuristics, and human intervention during the process ([17]).

[5] Note, however, that this coupling is for routing purposes only, and other aspects of the unidirectional routes are managed separately, e.g. bandwidth allocation is not necessarily equal in both directions.

process), and the output port into which it is switched; During this process the cells' VCI is ignored as long as the new VPI is non-null. Only when the VPI is null, is the VCI considered, to "demultiplex" the VCs that used the VP. The VC table determines the new VCI label and the output port (similar to the VP routing table), and also a new VPI label which matches the VP into which the VC is multiplexed (see [8] for a full description).

In realistic implementations, it is plausible that many nodes will switch VPs exclusively; However, incorporating this fact into the model complicates it (and hence the proposed solutions) with details that may damage the insight into the problem, which motivated this work. At the summary of this paper we propose a method for dealing with such "heterogeneous" networks.

(A3) Pure routing: In this work we are not concerned with determining multiple routes between nodes, in accordance to bandwidth allocation or fault tolerance considerations. This attitude enables us to formulate the problem and solutions more clearly, and thus gain insight into them, while these additional parameters remain for future research. The proposed solutions are applicable with no modifications in certain scenarios, for instance a VPPL that caters relatively short messages between LANs (which have no bandwidth guarantee).

The following performance properties are affected by the design of VPPL. A "good" layout is characterized by achieving a good performance tradeoff among them.

(P1) VCP setup complexity: Low VCP setup complexity is important as it substantially reduces various overheads of connection management [6, 26, 27]. The setup complexity is proportional to the number of nodes in the VCP, in which the VCI is examined, since in these nodes the intervention of software is needed (to change the VC routing tables, to allocate bandwidth etc.). For this reason, the number of VPPs from which any VCP is composed (termed *hop count*) should be small.

(P2) Length of underlying physical route: The chosen route for a VCP must also be short in terms of the number of physical links it uses, to efficiently utilize the communication network. In this work we restrict the discussion to shortest physical routes only.

(P3) Utilization of VP routing tables: The number of occupied entries in the VP routing table (termed the *load* on the table) implied by the layout, should be low enough at any location in the network — see discussion above.

(P4) Recovery overhead: The resulting layout must overcome link disconnections with a low overhead. This is achieved by reducing the number of VPPs that share any link, so that if a link is disconnected, the number of VPPs that need to be rerouted, so that the faulty link is bypassed, will be small (see [8] for a description of this recovery and reroute procedure). In the sequel we show that this property is closely related to P3.

1.3 Related works and paper structure

Most of the works on the layout of VPs in a network have considered only the case in which each VP spans through the entire network [27, 15], an attitude which is suitable for relatively small networks. Several works have considered planning a VP layout so that a VC is routed through multiple VPs [1, 20]. In these works many parameters are taken into account, which cause the problem to be to hard to be handled analytically. Consequently, an experimental approach is taken, based on heuristic optimization techniques. In contrast with these works, this work considers only a few parameters, and is based on an analytical approach. Thus we hope to gain better insight into the problem, present simpler and more efficient algorithms, and base the performance analysis of the results on mathematical tools rather than empirically.

A problem which is related to ours is that of keeping small routing tables for routing in traditional datagram networks. This problem was widely studied [18, 19, 11, 12, 23, 2, 3] and yielded interesting graph decompositions and structures, but it differs from ours in some major aspects which deemed most of these solutions impractical

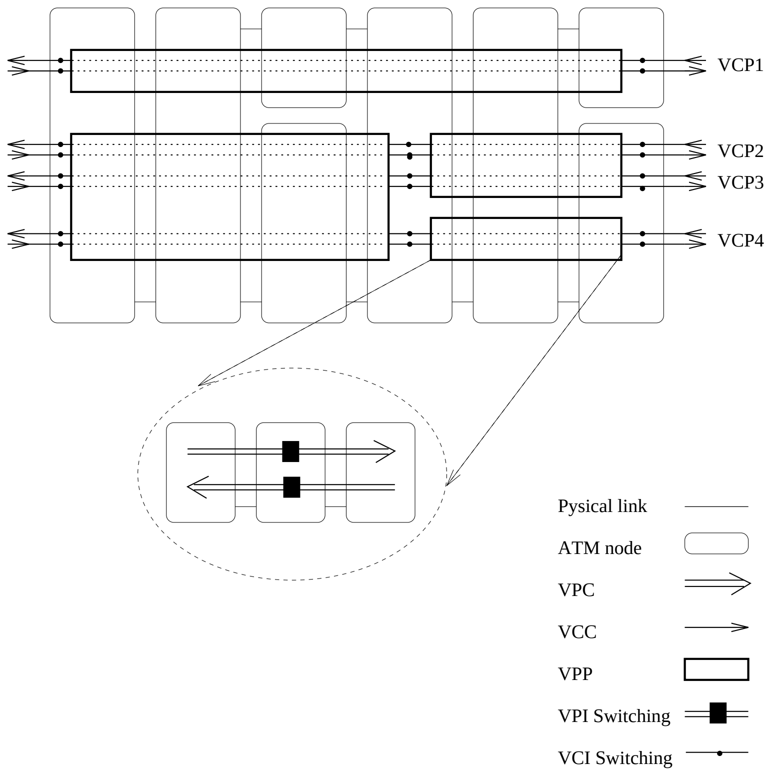

Fig. 1. The configuration of VPs and VCs in a simple network

for our purposes. Some of these differences stem from the fact that in our case there is no flexibility as to the nodal intermediate routing scheme itself since it is determined by the ATM standard [7], and by the requirement for very fast routing. For this reason we present a static structure in the network, while in the traditional model, the exact routing of a packet may be determined dynamically during its routing process (as in [3, 2]), or by information based on the name of the destination (as in [11, 12]).

Other differences stem from the ATM standard, in which the sizes of the routing tables at each node are fixed - implying a worst-case approach to the utilization of the tables, in contrast to the average-case approach, which is adopted by some of the solutions to traditional datagram routing (e.g. [11, 12, 23]). Many of the general solutions route packets in paths that are up to a multiplicative factor longer than the shortest path (this factor is termed *stretch factor*). These solutions are usually based on a large factor and are therefore not practical for our purposes.

The separator-based techniques for decomposing a graph that we have used in this work resemble those of [12, 11]. Another structure that bears some resemblance to our layout is the graph spanner [22].

The paper is structured as follows: we start (in Section 2) by formally defining our assumptions, the problem in its restricted and general form, and the essential characteristics of a "good" layout. In Section 3 we propose a solution to the restricted problem that is based on a structural decomposition of a network into small sub-networks, and a recursive solution for each sub-network. This solution is presented in stages, and is proven to yield a quantitative upper bound to the efficiency of the layout. In Section 4 we present a different approach, based on a greedy algorithm, which we prove to be correct. The crux of this section is the proof of optimality of the resulting layout (in fact, this proof influenced the non-trivial design of the algorithm). In Section 5, we show how the solution for the restricted problem is used for solving the general problem and prove the asymptotic optimality of this result. We conclude (Section 6) by summing up the results in this paper.

2 The mathematical model

In order to properly analyze the virtual path properties and layout, we first define a graph-theoretic model for it (for basic terms and definitions — see [9]). In our model we have an underlying communication network, which consists of nodes and links between them. This network is modelled by an undirected graph $G = (V, E)$, where V corresponds to the set of nodes and E to the set of physical links between them.

Definition 1. Let $\mathcal{P}(G)$ be the set of all simple paths in G. A virtual path layout $\Psi = (G, G_\Psi, \mathcal{I})$ is represented by a a graph $G_\Psi = (V, E_\Psi)$ and a function $\mathcal{I} : E_\Psi \to \mathcal{P}(G)$, where an edge $\psi = (a, b) \in E_\Psi$ corresponds to a VPP between the nodes a and b. The function $\mathcal{I}(\psi)$ maps each VPP $\psi = (a, b)$ to a path in G, so that a and b are the endpoints of $\mathcal{I}(\psi)$ as well; We term this path the *induced path* of the VPP.

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

Definition 2. The *induced path* $\mathcal{I}(p)$ for a path $p \in \mathcal{P}(G_\Psi)$, $p = (\psi_1, \psi_2, ..., \psi_k)$, ($\psi_i \in E_\Psi$ for all i) is the path obtained by concatenating the induced paths of all ψ_is.

For the sake of notational convenience we shall refer to a path $p \in \mathcal{P}(G)$ either as a set of edges or as a set of vertices. Also, when there is no risk of confusion, we shall mix the concept of a VPP $\psi \in E_\Psi$ and its induced path $\mathcal{I}(\psi)$. For example let $(x, y) \in \mathcal{I}(\psi)$. The following expressions are all true $(x, y) \in \psi$, $x \in \mathcal{I}(\psi)$, $y \in \psi$.

Definition 3. The *load* $\mathcal{L}(e)$ on an edge $e \in E$ is the number of VPPs $\psi \in E_\Psi$ that include e in their induced paths [6]. Namely, $\mathcal{L}(e) = |\{\psi \in E_\Psi | e \in \mathcal{I}(\psi)\}|$.
Also define the load $\mathcal{L}(\Psi)$ of a given VPPL, Ψ as $\mathcal{L}(\Psi) = \max_{e \in E} \mathcal{L}(e)$.

Definition 4. The *hop count* $\mathcal{H}(v, w)$ for $v, w \in V$ is the minimum number of VPPs that may be used to form a VCP between v and w, such that the VCP uses a shortest path in the physical network. Namely, it is the minimum k such that:

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

If no such k exists, define $\mathcal{H}(v, w) = \infty$.

We distinguish between two problems: One being the general layout problem in which it is required to cater for the setup of VCPs between every pair of nodes; We term this case "many-to-many" ($m-m$ for short). The second problem is a more restricted case, namely when the VPPL should cater for VCPs from a single node (called the *root*) to all the other nodes; We term this case "one-to-many" ($1 - m$). The VPPL for the $m - m$ case is denoted by VPPL^{m-m} , while a VPPL for the $1 - m$ case is denoted by VPPL^{1-m}.

Definition 5. Let $h > 0$. The following definitions correspond to the above-mentioned problems [7]:

– A VPPL^{m-m} is *h-feasible* if $\max_{v,w \in V} \mathcal{H}(v, w) \leq h$.

[6] Underlying this definition is the assumption that a VP routing table resides in the port processor, following the node architecture of [8]. We also assume that each VPP is a bidirectional route (see assumption A1). Hence, each VPP that goes through a physical link contributes one to the utilization of the VP routing table at the port processors that are connected via that link (property P3). This load definition is also suited to express the fault tolerance overhead (on the rerouting of VPPs) when a given link is disconnected (property P4).

[7] Note that for the VPPL^{1-m} case we are concerned with the hop count from all the nodes to the root only, while in the VPPL^{m-m} case we were concerned with the hop count between every pair of nodes.

– Let $r \in V$, a VPPL^{1-m} is (h, r)-*feasible* if $\max_{v \in V} \mathcal{H}(v, r) \leq h$.

Notation: When the scheme for producing a feasible VPPL^{m-m} from G, h is understood from the context, the VPPL will be denoted by $\Psi^{m-m}(G, h)$; For the VPPL^{1-m} case the notation will be $\Psi^{1-m}(G, h, r)$. For notational convenience, when G belongs to a given family and $|V| = N$ we shall replace G by N.

The feasibility of a VPPL captures the notion of a VPPL in which the worst-case VPP hop count is bounded by h (property P1 above), and the chosen routes are minimal in the underlying physical network (property P2). We now define an optimal solution as a solution that, in addition, minimizes the utilization of the VP routing tables (property P3), and enables efficient recovery from link disconnections (property P4).

Definition 6. Given h, a VPPL^{m-m} Ψ is *h-optimal* if it is h-feasible and its load $\mathcal{L}(\Psi)$ is minimal amongst all other h-feasible VPPLs. This definition is extended in a straightforward manner to the (h, r)-optimality of a VPPL^{1-m} (for a given root r).

Remark. Finding a VPPL^{1-m} is easier than finding a VPPL^{m-m} as hinted by the following facts:

1. Given a network G, every h-feasible VPPL^{m-m} is also a (h, r)-feasible VPPL^{1-m} for any $r \in V$, but the reverse is not necessarily true.
2. The load $\mathcal{L}(\Psi)$ of an h-optimal VPPL^{m-m} is never less than the load of a (h, r)-optimal VPPL^{1-m}. □

Besides its methodical value as an easier problem to be tackled first, VPPL^{1-m} has its own practical importance, as it may prove useful for server networks, where data is sent from a single source to multiple destinations and vice versa. An example for this is a video conferencing server — which has VCPs to all users who are currently engaged in a video conference [8] [21, 24]. Another example is a connectionless server (CLS), which enables a connectionless service (datagrams) using ordinary fixed VCPs [5].

3 A structural approach for the design of a VPPL^{1-m}

In this section we present results that concern the construction of a VPPL^{1-m}. The results are presented for increasingly complex cases. We first present a method for finding a VPPL^{1-m} for the case when the network is a linear array of nodes (an array of N nodes connected in a row) and then extend it to arbitrary tree networks. For the linear array case, we start with $h = 2$, and extend the results to an arbitrary h:
Given a linear array of N nodes and a root r at one of its ends [9], construct VPPL^{1-m} in the following way: First divide the linear array into $\sqrt{N}$ equal sections of size $\sqrt{N}$ each. Call the node that is closest to r in section i the "pivot of section i". Connect r to all pivots by VPPs; Connect the pivot of section i to all the nodes in its section (see Figure 2).
This result for $h = 2$ can be extended to any h by the following recursive scheme (see Figure 2):

Scheme:
1. Divide the linear array into $N^{1/h}$ sections of size $N^{1-1/h}$ each,
2. Connect each pivot to the root r by a direct VPP,

[8] This is not to be confused with a multicast service, where all destinations receive the same data from a given source, while here we discuss separate streams of data from a service center.
[9] It is easy to see that if the root is not at an end of the array, then the problem may be decomposed into two independent subproblems with r at the end of each sub-array.

3. Connect each pivot to its section by a VPPL^{1-m} with $h' = h - 1$ (i.e. let S_i be the subgraph of section i, p_i the pivot of the section then construct $\Psi^{1-m}(S_i, h-1, p_i)$).

□

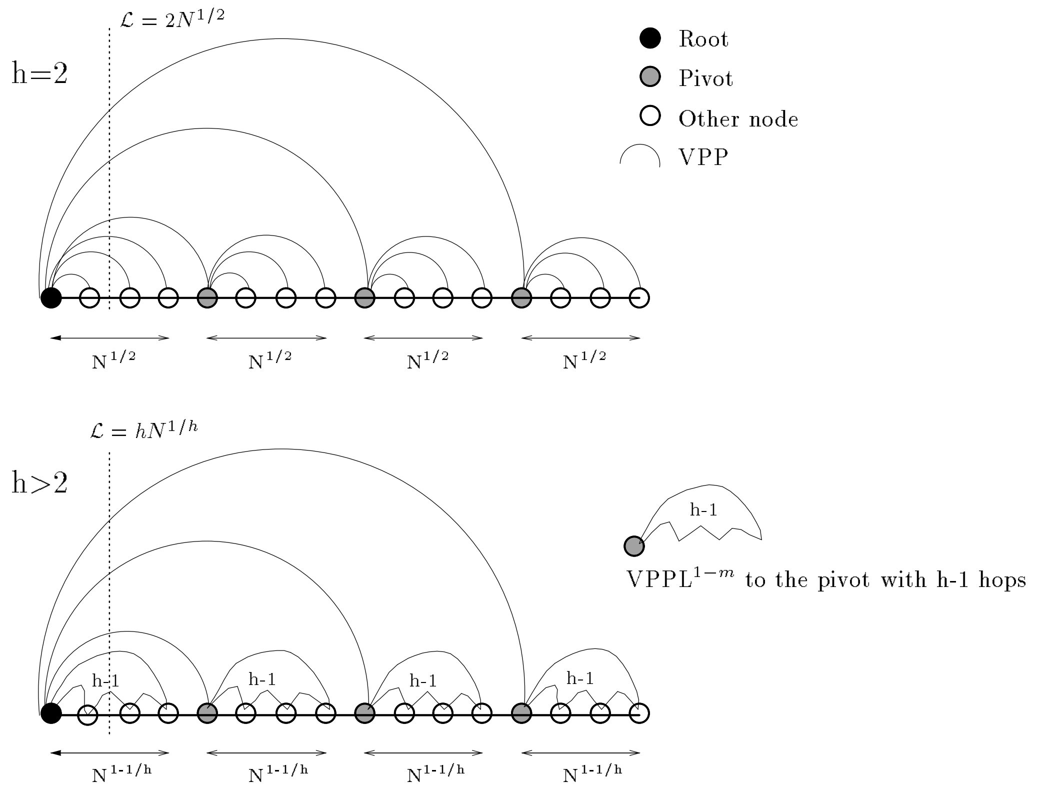

Fig. 2. VPPL^{1-m} on a chain of nodes

Correctness: The underlying route from any node to the root is the shortest possible, since the path advances only in the direction of the root. Also, each node can reach r using no more than h hops: $h-1$ hops to the pivot of its section and one more hop to r. □

Load analysis: Let P be the subgraph induced by the set of pivots, S the subgraph of a given section, whose pivot is p. Since the load on each edge is effected only by the VPPs that connect the pivots to r and by the VPPL^{1-m} in the section it belongs to, the load of the VPPL^{1-m} obeys the following recurrence formula: $\mathcal{L}(\Psi^{1-m}(G, h, r)) \leq \mathcal{L}(\Psi^{1-m}(P, 1, r)) + \mathcal{L}(\Psi^{1-m}(S, h-1, p))$
or

$$\mathcal{L}(\Psi^{1-m}(N, h)) \leq \mathcal{L}(\Psi^{1-m}(N^{1/h}, 1)) + \mathcal{L}(\Psi^{1-m}(N^{1-1/h}, h-1))$$

which implies that $\mathcal{L}(\Psi^{1-m}(N, h, r)) \leq hN^{1/h}$ since

$$hN^{1/h} \leq 1 \cdot (N^{1/h})^{1/1} + (h-1)(N^{1-1/h})^{1/(h-1)} =$$

$$= N^{1/h} + (h-1)N^{\frac{h-1}{h(h-1)}} = hN^{1/h}$$

□

We now extend the technique for an arbitrary tree network (with an arbitrary h). We shall need the following graph-theoretical result on trees:

Theorem 7 [28]. *Let G be a tree with N vertices. There exists at least one vertex v (called a* median*), whose removal divides the tree into subtrees, each containing at most $\frac{N}{2}$ vertices.*

Definition 8. Given a graph G, an (α, β)-*separator* is a set of vertices whose size does not exceed α and its removal divides the graph into subgraphs with size not greater than β (a median is a $(1, \frac{N}{2})$-separator of the tree).

Lemma 9. *Given a tree T with N vertices and an integer $k > 0$, there exists a $(2^{k+1}, \frac{N}{2^k})$-separator for T.*

Proof. We build a separator S in rounds as follows: take the median of the tree as the initial S; The resulting subtrees are not larger than $\frac{N}{2}$ (call this round 1). Now find a median in each subtree whose size is larger than $\frac{N}{4}$ (there are at most 3 such subtrees), and add these medians to S. The resulting subtrees does not exceed $\frac{N}{4}$.

If this is process repeated for k rounds we get a separator with size $|S| \leq 1 + 3 + \cdots + (2^k - 1) \leq 2^{k+1}$, where each remaining subtree contains at most $\frac{N}{2^k}$ vertices, as desired. □

Using the above result, we can now construct a layout for arbitrary trees by the following scheme (see Figure 3):

Scheme:

1. If $h = 1$ — connect each vertex to the root by a direct VPP.
2. Given $h > 1$, choose k such that $\frac{N}{2^k} \leq N^{1-1/h} \leq \frac{N}{2^{k-1}}$ (i.e. $k = \lceil \frac{1}{h} \log N \rceil$).
3. Find a $(2^{k+1}, \frac{N}{2^k})$-separator for T (as shown in Lemma 9) and define the vertices of the separator to be the pivots; Also, define the root to be a pivot.
4. Connect the root to all the pivots.
5. Connect each pivot to all the subtrees that are adjacent to it except for the subtree in the direction of the root, using a VPPL^{1-m} with $h' = h - 1$.

□

Correctness: To reach the root, each vertex v must first reach a pivot (by no more than $h - 1$ hops, using the shortest path — as guaranteed by the recursive application of the scheme), and then one additional hop to the root using a shortest path. If the root is not adjacent to the subtree in which v resides, then there exists a pivot which is on the shortest path from v to the root of the tree. □

Load analysis: Each edge may be used by no more than 2^{k+1} VPPs that connect the pivots to the root; It may also be used by a VPPL^{1-m} with $h' = h - 1$ which connects the subtree to which the edge belongs to the pivot. Since each edge belongs to a single subtree, and there is a single pivot in this subtree that is on the path to the root — there is only one such VPPL^{1-m}. The load function thus satisfies

$$\mathcal{L}(\Psi^{1-m}(N, h, r)) \leq \mathcal{L}(\Psi^{1-m}(2^{k+1}, 1, r)) + \mathcal{L}(\Psi^{1-m}(\frac{N}{2^k}, h - 1, pivot))$$

By the choice of k we get $\mathcal{L}(\Psi^{1-m}(2^{k+1}, 1, r)) = 2^{k+1} \leq 4 \cdot N^{1/h}$.
Combining these (and the fact that $\mathcal{L}(\Psi^{1-m}(x, y, z))$ increases as x increases), we get

$$\mathcal{L}(\Psi^{1-m}(N, h, r)) \leq 4 \cdot N^{1/h} + \mathcal{L}(\Psi^{1-m}(N^{1-1/h}, h - 1, pivot))$$

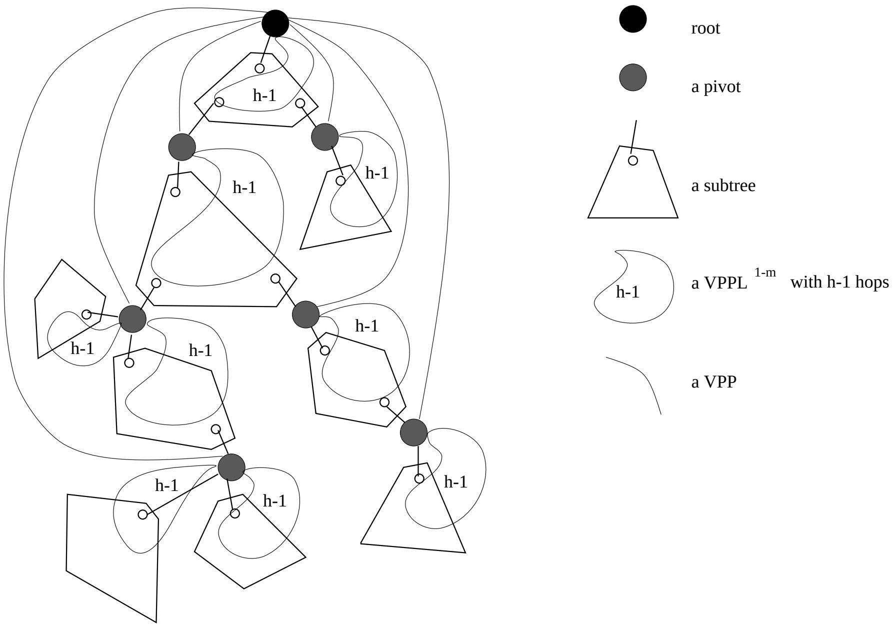

Fig. 3. VPPL^{1-m} on a tree

It is easy to verify that $\mathcal{L}(\Psi^{1-m}(N,h,r)) \leq 4h \cdot N^{1/h} = O(hN^{1/h})$. $\square$

4 A greedy approach for $\mathbf{VPPL}^{1-m}$

In this section we present a different technique for designing a VPPL^{1-m} on tree networks. We devise a greedy algorithm that produces an optimal VPPL^{1-m} for any tree (with a given root). By its optimality, it is clear that this algorithm always produces a better VPPL^{1-m} than the one produced by the structural approach of the previous section. However, the importance of the structural approach lies in an easy analysis of a quantitative upper bound on the load incurred by it, while the current algorithm does not imply such a bound easily.

The algorithm is an iterative process in which the minimal load is guessed by the `main` procedure, which calls the `find_layout` procedure with that load as a parameter. The `find_layout` procedure tries to construct a layout under the given load constraint and informs the `main` if the guess enabled the construction of the layout. Upon a positive response, `main` lowers the guess and retries the process, until the minimum is found. Finally, `main` creates the VPPL using the `construct_layout_from_loads` procedure. The pseudo-code of `main` follows,

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

- $||V||$ — the sum of all the elements $V[i]$,
- $V[^j_i]$ — the part of the vector from the i-th to the j-th position,
- $c[i]$ — a vector of i elements which are equal to c,
- $W \bullet V$ — the concatenation of the vectors W and V.

```
procedure main (a tree T, a vertex root ∈ V(T))
   var L : a set of vectors {L_v ∈ ℕ^h | v ∈ V(T)}
   begin
      choose ℒ_max ∈ {1, ..., |V(T)|} such that
```

```
        find_layout(T,root, 𝓛_max,L) = SUCCESS,
        find_layout(T,root, 𝓛_max-1 ,L) = FAILURE.

      Ψ←construct_layout_from_loads(T, root, L)
  end
```

We view the tree T as rooted at *root*. The `find_layout` procedure starts from the leaves of the tree and advances towards the root. For each vertex v it maintains the number of VPPs that go from v to its parent (i.e. the VPPs that load the edge between them). This number is kept in a vector $L_v \in \mathbb{N}^h$, where $L_v[1]$ holds the number of VPPs that serve only as a first hop from some descendant, and $L_v[i]$ holds the number of VPPs that are an i-th hop for some descendant x (e.g. the shortest path — in terms of VPPs — from x towards the root includes the VPP as the i-th VPP from x).

Specifically, the algorithm creates a first-hop VPP for each leaf (at line 7); At an internal vertex v, the vector L_v is equal to the vector-sum of all the VPPs that arrive at it from all its sons plus one additional first-hop VPP from v to its parent (at line 14). If this vector is too large (e.g. $||L_v||$ exceeds the current load constraint) then the VPPs on the overloaded edge $(v, parent(v))$ are reduced by stopping some of them at v (hence turning the first hop VPP that starts at v to an $(i+1)$-th hop VPP), by applying the transformation in Figure 4 (at lines 16-19).

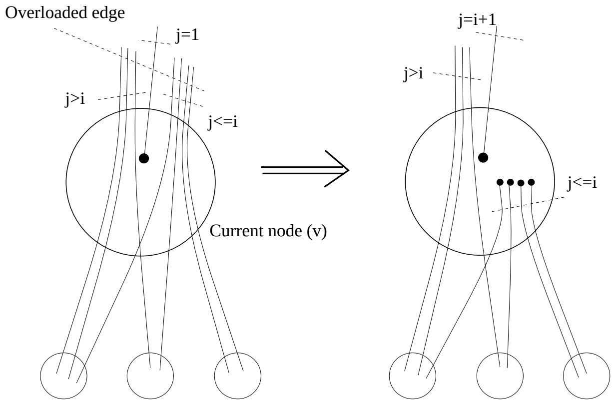

Fig. 4. The transformation on an overloaded edge

In this transformation, the minimum i is sought, such that if all VPPs which go through v and serve as a j-th hop are stopped at v, for $j \leq i$ - then the load on the edge $(v, parent(v))$ will not exceed the allowed limit (at lines 16-17). This criterion is crucial to the optimality proof of the resulting VPPL[10], and is based on the idea that if we keep this i low, then the resulting vector L_v will be lexicographically low.

```
0  function find_layout(a tree T, a root root ∈ V(T),
1                       an integer 𝓛_max,
2                       a set L of vectors {L_v ∈ ℕ^h | v ∈ V(T)} )
3                       return {SUCCESS, FAILURE}
4
5    begin
6      for all w ∈ V(T):
```

[10] In fact, it was shaped by the needs of the optimality proof.

```
7           if w is a leaf then L_w ← (1,0,...,0)
8           otherwise L_w ← UNDEFINED
9
10       loop forever
11          choose a vertex v ∈ V(T) such that: (1) L_v = UNDEFINED,
12                                              (2) for all x ∈ SONS(v), L_x ≠ UNDEFINED.
13          if v = root then return SUCCESS
14          L_v ← Σ_{s∈Sons(v)} L_s + (1,0,...,0)
15          if ||L_v|| ≤ 𝓛_max then skip until end of loop
16          find an integer i ∈ {1,...,h−1} such that: (1) ||L_v[^h_{i+1}]|| < 𝓛_max ,
17                                                     (2) ||L_v[^h_i]||     ≥ 𝓛_max .
18          if no such i exists then return FAILURE
19          Transform L_v as follows: L_v ← 0[i] • (L_v[i+1] + 1) • L_v[^h_{i+2}]
20       end of loop
21    end
```

main relies on the construct_layout_from_loads procedure for the actual creation of the VPPL^{1-m} from the set of vectors L. This procedure extends VPPs from leaves towards the root (at line 14), and stops the relevant VPPs from being further extended, at transformed vertices (at line 15). The pseudo-code of construct_layout_from_loads is as follows:

```
0    function construct_layout_from_loads(a tree T, a root root ∈ V(T),
1                                         a set L of vectors {L_v ∈ ℕ^h | v ∈ V(T)} )
2                                         return VPPL Ψ: a set {ψ_v | v ∈ V(T)}
3
4      begin
5        for all v ∈ V(T):
6           𝓘(ψ_v)←{v}
7           if v is a leaf then ℏ(ψ_v)←1 otherwise ℏ(ψ_v)←∞
8
9        loop forever
10          choose a vertex v ∈ V(T) such that: (1) ℏ(ψ_v) = ∞,
11                                              (2) for all x ∈ SONS(v), ℏ(ψ_v) ≠ ∞.
12          ℏ(ψ_v)←min_{i∈{1,...,h}} L_v[i] > 0
13          for every VPP ψ ∈ Ψ which includes a son x of v (i.e. x ∈ 𝓘(ψ)) do
14             if ℏ(ψ) > 0 then 𝓘(ψ)←𝓘(ψ) ∪ {v}
15             if ℏ(ψ) < ℏ(ψ_v) then ℏ(ψ)←0
16          end
17          if v = root then return Ψ
18       end of loop
19     end
```

Example 1. An example for the results of the execution of find_layout on a tree with $\mathcal{L}_{max} = 3$ and $h = 2$ can be found in Figure 5. In this figure, for each vertex v, L_v is displayed (e.g. $(3,0)$), and if v has been transformed then the new L_v appears as well (e.g. $(0,1) \leftarrow (5,0)$). In addition, the VPPs that would have been constructed by construct_layout_from_loads appear in a bold line. Note however, that since find_layout fails (in vertex x), construct_layout_from_loads is not called. Also note that if $\mathcal{L}_{max} = 4$ or $h = 3$, then find_layout does not fail.

□

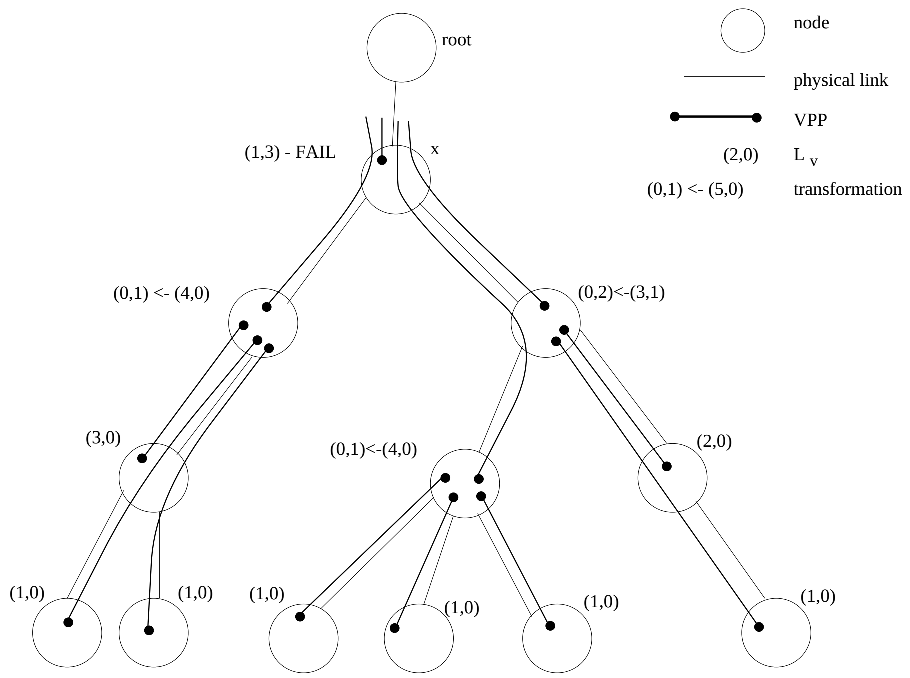

Fig. 5. Example for an execution of the algorithm

The correctness of the algorithm is proven by the following lemmata and theorem:

Lemma 10. *For every rooted tree T, the procedure* `main` *terminates.*

Proof. `main` calls `find_layout` at most `N` times and `construct_layout_from_loads` once. In `find_layout` the label `NEXT` is visited at most once for every vertex and the amount of work at each such iteration is bounded. The `construct_layout_from_loads` procedure visits each vertex once. □

Definition 11. An (h, r)-feasible VPPL^{1-m} is *minimal* if the deletion of a VPP $\psi \in E_{\Psi}$ yields a VPPL^{1-m} which is not (h, r)-feasible (clearly, there exists an optimal VPPL which is minimal).

Lemma 12. *Given a minimal $\Psi^{1-m}(T, h, r)$, for any vertex $v \neq r$ there exists a single VPP that starts at v towards r (i.e. $|\{\psi \in E_{\Psi} | v \in \mathcal{I}(\psi) \wedge \forall s \in Sons(v) \ : \ s \notin \mathcal{I}(\psi)\}| = 1$).*

Proof. Assume by contradiction that there exists more than one such VP for a vertex v. Find a (shortest) path $p \in \mathcal{P}(G_{\Psi})$ from r to v, whose length is $\mathcal{H}_1(r, v)$; Clearly that path contains one of the VPs that start at v. Now omit the other VPs that start at v towards r. It is easy to see that the VPPL^{1-m} is still $(1, h, V_{\Psi})$-feasible, hence the original VPPL^{1-m} is not minimal. □

Proposition 13. *Let $\hbar(\psi)$ be the value assigned to ψ at lines* `7,12` *in* `construct_layout_from_loads`. *Given a minimal VPPL^{1-m} Ψ, then $\max_{\psi \in E_{\Psi}} \hbar(\psi) = \max_{v \in V} \mathcal{H}(v, r)$.*

Lemma 14. *Let T be a tree, rooted at r, let $v \in V$, let $\{L_w | w \in V\}$ be the set of vectors produced by* `find_layout`, *let $\Psi = (T, G_{\Psi}, \mathcal{I})$ be the layout constructed by* `construct_layout_from_loads`; *Then for every $i \in \{1, ..., h\}$*

$$L_v[i] = |\{\psi \in E_{\Psi} | (v, parent(v)) \in \mathcal{I}(\psi) \wedge \hbar(\psi) = i\}|$$

Proof. The Lemma is proven by an induction on the height of the vertex v. It is clear that for every leaf `find_layout` produces the vector $L_v = (1, 0, \ldots, 0)$, and indeed a VPP VP will be created for that leaf. The inductive step follows from the extension of VPPs by `construct_layout_from_loads` according to the value of L_v (produced by `find_layout`). □

The correctness proof is concluded by the following theorem.

Theorem 15. *For every tree T rooted at r, the procedure* `main` *completes with a feasible $\Psi^{1-m}(T,h,r)$ with load $\mathcal{L}(\Psi^{1-m}(T,h,r)) \le \mathcal{L}_{max}$.*

Proof. When `find_layout` completes with `SUCCESS`, then $\|L_v\| \le \mathcal{L}_{max}$ for every vertex v, and by Lemma 14, the load on every edge in T is not more than $\mathcal{L}_{max}$ as well. It may also be shown that $\hbar(\psi)$ of a VPP ψ that reaches the root cannot exceed h. By combining this and Proposition 13 we get $\max_{v\in V} \mathcal{H}(v,r) \le h$, and hence the layout is (T,h,r)-feasible. □

We now turn to the proof of optimality of the above solution. To compare the solution produced by `find_layout` to an optimal solution, we must find a way to represent each optimal solution using the vector representation of `find_layout` (namely a vector $L_v \in \mathbb{N}^h$ for every $v \in V$). Comparison between vectors in $\mathbb{N}^h$ is done lexicographically [11].
The basis for this representation is prepared by the following lemma:

Lemma 16. *Let T be a tree rooted at r. Every minimal $\Psi^{1-m}(T,h,r)$ can be represented by a set of vectors $\{M_v \in \mathbb{N}^h | v \in V(T),\ v \ne r\}$, such that for every vertex $v \ne r$, the following conditions hold:*

1. *$\mathcal{L}(\ (v, parent(v))\ ) = \|M_v\|$,*
2. *If v is a leaf then $M_v = (1,0,\ldots,0)$, and*
3. *$M_v \ge \sum_{s\in Sons(v)} M_s + (1,0,\ldots,0)$.*

Proof. Let T_v denote the set of vertices in the subtree below v; Define M_v for a vertex v by

$$M_v[i] = |\{\psi \in E_{\Psi} | (v,f) \in \psi \wedge \hbar(\psi) = i\}|$$

Condition 2 is trivially satisfied. Condition 1 is satisfied since each VP that includes the edge (v,f) contributes one to a single $M_v[i]$ (according to $\hbar(\psi)$), hence $\mathcal{L}((v,f)) = \|M_v\|$. To prove condition 3, note that if $\hbar(\psi_v) = 1$ then no VP stops at v, and every VP that loaded an edge from a son of v, loads the edge to the father of v, which has an additional load of ψ_v, hence $M_v = \sum_{s\in Sons(v)} M_s + (1,0,\ldots,0)$. If $\hbar(\psi_v) = i > 1$ then a VP ψ that stops at v has $\hbar(\psi) \le i-1$, hence $M_v[{}^h_i] = \sum_{s\in Sons(v)} M_s[{}^h_i] + (1,0,\ldots,0)$ and $M_v > \sum_{s\in Sons(v)} M_s + (1,0,\ldots,0)$. □

We shall call the set of vectors $\{M_v | v \in V(T)\}$ which corresponds to a VPPL^{1-m} Ψ, the *vector representation of Ψ*.
The main lemma of the optimality proof is the following one. It is based on a comparison between a given optimal VPPL^{1-m} Ψ, and the one produced by `find_layout`, such that their load does not exceed $\mathcal{L}_{max}$. The lemma claims that at every vertex v, the vector representation of Ψ $\{M_v\}$ is not less than the vector representation of the solution produced by `find_layout` ($\{L_v\}$) with respect to the lexicographic order. This fact is used to show that if `find_layout` fails, the load of Ψ at v ($\|M_v\|$) exceeds $\mathcal{L}_{max}$ — contradicting the load constraint on Ψ.

Lemma 17. *If there exists a (h,r)-feasible VPPL^{1-m} Ψ with $\mathcal{L}(\Psi) = X$ for some $X > 0$ with a vector representation $\{M_v | v \in V(T)\}$, then for every vertex $v \ne r$ whose subtrees' height is at most H, the following holds:*

1. *If $\mathcal{L}_{max} = X$ then* `find_layout` *does not return* `FAILURE` *while handling v.*
2. *The vector L_v produced by* `find_layout` *satisfies $L_v \le M_v$*

[11] We use a lexicographic order in which $M[1]$ is the least significant component, e.g. $(9,11,1) < (2,12,1) < (0,0,2)$.

Proof. We shall prove by induction on the height of T. For a tree of height 0 (i.e. a leaf), it is clear that $L_v = M_v = (1, 0, \ldots, 0)$. In the induction step, assume that the claim holds for subtrees of height at most $H - 1$, and let v be a vertex such that T_v is of height H. By the induction hypothesis, `find_layout` does not return `FAILURE` for every son s of v and $L_s \leq M_s$. Let $L'_v = \sum_{s \in Sons(v)} L_s + (1, 0, \ldots, 0)$.

Case 1. $\|L'_v\| \leq X$: `find_layout` finishes handling v at without failing and

$$L_v = \sum_{s \in Sons(v)} L_s + (1, 0, \ldots, 0) \leq \sum_{s \in Sons(v)} M_s + (1, 0, \ldots, 0) \leq M_v$$

The left inequality is due to the induction hypothesis, and the right one is due to Lemma 16.

Case 2. $\|L'_v\| > X$: If there exists no i that satisfies the precondition for a transformation (see lines `16-17` in the pseudo-code of `find_layout`) then $X \leq L'_v[h] = \sum_{s \in Sons(v)} L_s[h] \leq \sum_{s \in Sons(v)} M_s[h] < \|M_v\|$. The first inequality stems from the unsatisfied condition, while the second inequality stems from the induction hypothesis, and the third one — from Lemma 16 (it is strict since $\|M_v\| \geq \sum_{s \in Sons(v)} \|M_s\| + 1$). This inequality contradicts the assumption that $\mathcal{L}(\Psi) = X$.

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 $M_v \geq \sum_{s \in Sons(v)} M_s + (1, 0, \ldots, 0) \geq \sum_{s \in Sons(v)} L_s + (1, 0, \ldots, 0) = L'_v$). It is also clear that $M_v > L'_v$ (otherwise $X < \|L'_v\| = \|M_v\|$).

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

On the other hand, if $M_v[{h \atop i+1}] = L'_v[{h \atop i+1}]$ then $M_v[i] \geq L'_v[i]$ (since $M_v[{h \atop i}] \geq L_v[{h \atop i}]$) and $\|M_v\| \geq \|M_v[{h \atop i}]\| = \|M_v[{h \atop i+1}]\| + M_v[i] \geq \|L'_v[{h \atop i+1}]\| + L'_v[i] \geq X$ — a contradiction. □

Theorem 18. *The* `main` *procedure finds an optimal $\Psi^{1-m}(T, h, r)$ for any given tree T, and any root r.*

Proof. By Lemma 17, `find_layout` successfully handles each son of r if $\mathcal{L}_{max}$ implies a feasible solution. Hence `find_layout` will return `SUCCESS` when reaching the root. The `construct_layout_from_loads` procedure returns a feasible layout by Theorem 15 for that $\mathcal{L}_{max}$. Since `main` finds the minimal value for $\mathcal{L}_{max}$ for which `find_layout` returns `SUCCESS`, it returns the optimal VPPL^{1-m}. □

5 The VPPL^{m-m} problem

5.1 Tree networks

The main motivation for studying the VPPL^{1-m} problem is that its solutions may be used as a building block in the solution of the general VPPL^{m-m} problem. We now demonstrate this by building a VPPL^{m-m} for arbitrary tree networks. This result may be extended with minor changes, to wider classes of network topologies as discussed in the next subsections. The recursive construction scheme follows (see Figure 6).

<u>Scheme:</u>

1. Choose a median m of T as a pivot,
2. Construct a VPPL^{1-m} with $h' = \frac{h}{2}$ in each subtree, with m as its root,
3. Recursively, build a VPPL^{m-m} with $h' = h$ in each subtree.

□

<u>Correctness:</u> Note that two nodes that reside in different subtrees may be connected using no more than h VPPs, by going from one node to the pivot (no more than $\frac{h}{2}$ hops), and from the pivot to the other node (again, no

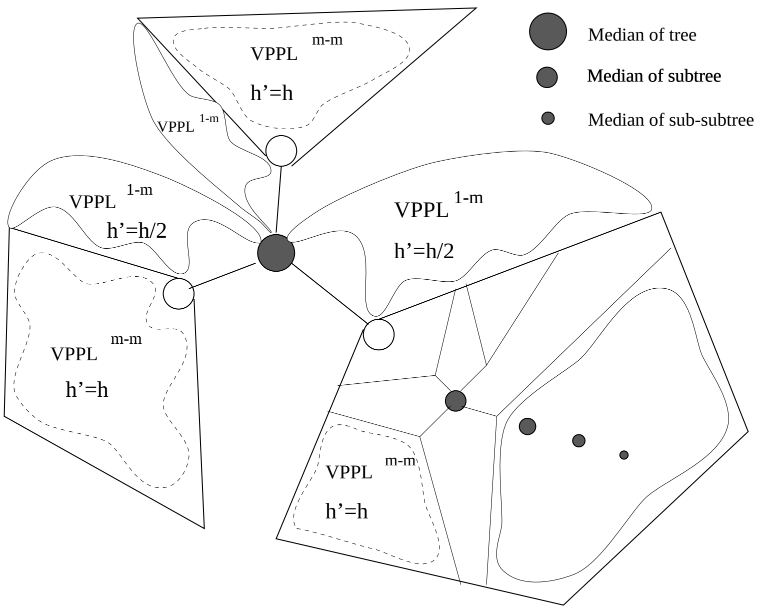

Fig. 6. VPPL^{m-m} on a tree

more than $\frac{h}{2}$ hops). Also note that this route is the shortest possible. If the nodes reside in the same subtree, then they are catered by the recursive application of the scheme in that subtree. □

Load analysis: The load on each edge is composed of the load of the VPPL^{1-m} to the pivot and the load of the recursive application of VPPL^{m-m} in the subtree. Since an edge participates in one such VPPL^{1-m} and one smaller VPPL^{m-m} (of its subtree), its load is bounded by the by the following recurrence equation.

$$\mathcal{L}(\Psi^{m-m}(h, N)) \le \mathcal{L}(\Psi^{m-m}(h, \frac{N}{2})) + \mathcal{L}(\Psi^{1-m}(\frac{h}{2}, \frac{N}{2}))$$

By using the load analysis result for VPPL^{1-m}, it can be shown by induction that the above inequality implies $\mathcal{L}(\Psi) \le 4hN^{\frac{2}{h}}$. □

This result is asymptotically optimal as proven by the following theorem:

Lemma 19. *Let L be the maximum load on any edge, h the maximum hop count; Then an edge $e \in E$ may be included in VPPs that connect no more than $(2L)^h$ pairs of nodes.*

Proof. Let $e \in E$ be an edge of the array, let $\psi = (a, b)$ be a VPP that includes e. At the node a there are no more than L VPPs that may be concatenated to ψ (since all of them must use a single edge (a, y), where y is adjacent to a further away from b). Therefore, the number of pairs that include ψ in a two-hop path is no more than $2L$ (L in each direction). By a similar argument, there are no more than $(2L)^2$ VPPs that include ψ in a three-hop path, and $(2L)^{h-1}$ VPPs that use it on a $(h-1)$-hop path, amounting to $2L + \cdots + (2L)^{h-1} < 2(2L)^{h-1}$ pairs that use it in a path with no more than h hops.

Clearly there are no more than L VPPs that include e (or its load would have been more than L), and hence there are no more than $(2L)^h$ VPPs that use the edge. □

Theorem 20. *Let T be a tree network with N nodes, let Δ be the maximum degree of a node, and $h > 1$. For every VPPL^{m-m} with h hops, there exists an edge $e \in E$ with load $\mathcal{L}(e) = \Omega(\frac{1}{\Delta^{\frac{1}{h}}} N^{\frac{2}{h}})$.*

Proof. By the definition of the median, there are many (i.e. $O(N^2)$) pairs of vertices, who are connected via the median. On the other hand, Lemma 19 restricts the number of pairs that use any edge which is adjacent to that median to $(2L)^h$, and there are no more than Δ such edges. From this we get $\Delta(2L)^h \geq O(N^2)$, which yields the desired bound. □

Corollary 21. *For realistic networks (with a small Δ, h), $\mathcal{L}(\Psi^{m-m}) = \Theta(N^{\frac{2}{h}})$.*

5.2 Beyond tree networks

So far, we have concentrated on ATM networks with a tree topology. The above technique can be easily extended for wider families of graphs as well — namely graphs with bounded treewidth. This family (see [4] for a survey on treewidth), includes many known families, amongst which are rings, chordal rings, interval graphs, circular arc graphs and cographs.

The extension scheme is based on the following theorem:

Theorem 22 [25]. *Let G be a graph with N vertices and treewidth at most K, then G contains a $(K, \frac{2}{3}N)$-separator.*

The construction of a VPPL^{m-m} for such graphs is the following one (note the similarity to the scheme for trees):

Scheme:
1. Find a $(K, \frac{2}{3}N)$-separator of G,
2. Construct a BFS spanning tree T_v for every vertex v of the separator,
3. Construct a VPPL^{1-m} with $h' = \frac{h}{2}$ on each T_v with v as its root,
4. Recursively build a VPPL^{m-m} with $h' = h$ in each separated part of G.

□

Correctness: The correctness is proven along the lines of the proof for trees. The only difference being that a shortest path between a pair of vertices in separated parts of G, must go through a vertex v of the separator, hence there is a path with h hops or less in VPPL, that is a shortest path in G (use the VPPL^{1-m} on T_v). □

Load analysis: Each edge participates in at most K VPPL^{1-m}s at the first stage of the recursion (namely in T_v for every v in the separator). Therefore, the load function of the scheme satisfies

$$\mathcal{L}(\Psi^{m-m}(N,h)) \leq K \cdot \mathcal{L}(\Psi^{1-m}(N, \frac{h}{2}, v)) + \mathcal{L}(\Psi^{m-m}(\frac{2}{3}N, h))$$

or

$$\mathcal{L}(\Psi^{m-m}(N,h)) = O(K \cdot h \cdot \log N \cdot N^{\frac{2}{h}})$$

□

We use a different technique for mesh networks, which exemplifies the construction scheme for composite networks. Recall that an $x \times y$ mesh network of size xy is comprised of horizontal chains of size x and vertical chains of size y. We construct a VPPL^{m-m} for these networks by building a VPPL^{m-m} with h_x hops for each horizontal chain, a VPPL^{m-m} with $h_y = h - h_x$ hops for every vertical chain, and taking the union of all these layouts. Note that the edges of different components are distinct, and hence the load on an edge is determined only by the layout of the component it belongs to.

Correctness: The hop count is restricted by h, since any switch can be reached by no more than h_x hops (to get to the correct vertical position) and no more than h_y hops in the vertical VPL. This adds up to no more than h hops. The stretch factor is preserved since one of the shortest routes in a mesh is composed of one horizontal segment and one vertical segment. □

Load analysis: To achieve a low $\mathcal{L}(\Psi)$, one has to take x and y into account, when determining the hop counts h_x, h_y for the horizontal/vertical components. The following choice for h_x, h_y is optimal (since it equates the load on edges of horizontal and vertical components):

$$h_x = \frac{h}{\frac{\log x}{\log y} + 1} \ ; \ h_y = h - h_x$$

The load of the scheme is $\mathcal{L}(\Psi) = x^{\frac{2}{h_x}} = y^{\frac{2}{h_y}}$. □

Note that for the simple case where $x = y = \sqrt{N}$ (i.e. a "square" mesh), $\mathcal{L}(\Psi) = hN^{\frac{2}{h}}$ — as for chain networks.

For general networks we have proven [14] that there exists no efficient algorithm which yields an optimal solution (i.e. that the problem is NP-hard). A feasible solution for such topologies is achieved by constructing a VPPL^{1-m} on a spanning BFS tree from each node to the rest of the network. This layout is easily proven to be h-feasible, and has a load of at most $hN^{1+\frac{1}{h}}$. Heuristic approaches may also be devised, in which a set of centrally located nodes are connected to their neighborhood by a VPPL^{1-m}, and interconnected by a smaller VPPL^{m-m} (see [14] for details).

6 Summary

In the paper we have studied the problem of the construction of a virtual path layout (VPPL) on a given ATM network. Our results exhibit a tradeoff between the hop count h (which effects the call-setup time), and the number of VPPs that use a physical link, $\mathcal{L}(\Psi)$ (which effects the utilization of the VP routing tables and the recovery overhead at link disconnection).

We have presented two main techniques for building a VPPL^{1-m} on tree networks (which yield $\mathcal{L}(\Psi) = O(N^{\frac{1}{h}})$), and proven their correctness and optimality. We have also demonstrated how the VPPL^{1-m} problem may be used in solving the VPPL^{m-m} problem efficiently (and proven its asymptotic optimality $\mathcal{L}(\Psi) = \Theta(N^{\frac{2}{h}})$).

While all these results were presented for tree networks, we have shown how to extend them for larger classes of networks , including rings, chordal rings, meshes. Finally, we discussed how to utilize these techniques for general topologies.

Throughout this paper we have assumed that every node has both VP and VC switching capabilities. This assumption may be eliminated by building a VPPL^{m-m} as described above, and allocating as such "dual function" nodes, only nodes in which VC switching is done in the VPPL. It is easy to see that if the recursive construction is not carried out for subnetworks which are small enough (in which direct VPPs are used), the number of "dual function" nodes need not be too large.

We believe that the approach in this paper improve the insight into the VP layout problem, and form a basis for extending the techniques to general topologies, to multiple routes between pairs of nodes, and to many other realistic extensions of the problem.

References

1. S. Ahn, R.P. Tsang, S.R. Tong, and D.H.C. Du. Virtual path layout design on ATM networks. In *INFOCOM'94*, pages 192–200, 1994.
2. B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive routing. In *21st Symp. on Theory of Computing*, pages 479–489, 1989.
3. B. Awerbuch and D. Peleg. Routing with polynomial communication-space tradeoff. *SIAM Journal on Discrete Math*, 5(2):151–162, May 1992.
4. H. L. Bodlaender. A tourist guide through treewidth. Technical report RUU-CS-92-12, Utrecht university, Dept. of Computer Science, Netherlands, March 1992.
5. J. Y. Le Boudec. The asynchronous transfer mode: a tutorial. *Computer Networks and ISDN Systems*, 24:279–309, 1992.
6. J. Burgin and D. Dorman. Broadband ISDN resource management: The role of virtual paths. *IEEE Communicatons Magazine*, 29, 1991.
7. ITU recommendation. I series (B-ISDN), Blue Book, November 1990.
8. R. Cohen and A. Segall. Connection management and rerouting in ATM networks. In *INFOCOM'94*, pages 184–191, 1994.
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. G.N. Frederickson and R. Janardan. Separator-based strategies for efficient message routing. In *27th Symp. on Foundations of Computer Science*, pages 428–437, 1986.
12. G.N. Frederickson and R. Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, 1988.
13. MB86689 address translation controller (ATC). Fujitsu data sheet, 1993. (Draft Edition 1.0).
14. O. Gerstel, I. Cidon, and S. Zaks. The layout of virtual paths in ATM networks. Technical Report (In preparation), Technion.
15. H. Hadama, R. Kawamura, T. Izaki, and I. Tokizawa. Direct virtual path configuration in large-scale ATM networks. In *INFOCOM'94*, pages 201–207, 1994.
16. R. Händler and M.N. Huber. *Integrated Broadband Networks: an introduction to ATM-based networks.* Addison-Wesley, 1991.
17. Aaron Kershenbaum. *Telecommunications Network Design Algorithms.* McGraw-Hill, 1993.
18. L. Kleinrock and F. Kamoun. Hierarchical routing for large networks; preformance evaluation and optimization. *Computer Networks*, 1:155–174, 1977.
19. L. Kleinrock and F. Kamoun. Optimal clustering structures for hierarchical topological design of large computer networks. *Networks*, 10:221–248, 1980.
20. F.Y.S. Lin and K.T. Cheng. Virtual path assignment and virtual circuit routing in ATM networks. In *GLOBCOM'93*, pages 436–441, 1993.
21. P. Lougher and D. Sheperd. The design of a storage server for continuous media. *The Computer Journal*, 36(1), 1993.
22. D. Peleg and A.A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.
23. D. Peleg and E. Upfal. A tradeoff between space and efficiency for routing tables. In *20th Symp. on Theory of Computing*, pages 43–52, 1988.
24. S. Ramanathan and P. Venkat Rangan. Feedback techniques for intra-media continuity and inter-media synchronization in distributed multimedia systems. *The Computer Journal*, 36(1):19–31, 1993.
25. B. A. Reed. Finding approximate separators and computing tree width quickly. In *24th Symp. on Theory of Computing*, pages 221–228, 1992.
26. K.I. Sato, S. Ohta, and I. Tokizawa. Broad-band ATM network architecture based on virtual paths. *IEEE Transactions on Communications*, 38(8):1212–1222, August 1990.
27. Y. Sato and K.I. Sato. Virtual path and link capacity design for ATM networks. *IEEE Journal of Selected Areas in Communications*, 9, 1991.
28. B. Zelinka. Medians and peripherians on trees. *Arch. Math. (Brno)*, pages 87–95, 1969.