Distributed hierarchical routing

P.J.A. Lentfert, A.H. Uittenbogaard, S.D. Swierstra
and G. Tel

RUU-CS-89-5
March 1989

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

Distributed hierarchical routing

P.J.A. Lentfert, A.H. Uittenbogaard, S.D. Swierstra and G. Tel

Technical Report RUU-CS-89-5
March 1989

Department of Computer Science
University of Utrecht
P.O.Box 80.089, 3508 TB Utrecht
The Netherlands

Distributed Hierarchical Routing

P.J.A. Lentfert, A.H. Uittenbogaard, S.D. Swierstra, G. Tel
Department of Computer Science, University of Utrecht
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands

Abstract

In this report a partitioning method and associated distributed routing algorithm on hierarchically divided networks are presented. This algorithm is based on a newly defined metric: the *hierarchical distance*. Important features of the algorithm are:

- *Absence of centralized control*: The algorithm is fully distributed, i.e., no node has special information and all nodes execute the same algorithm. The absence of essential components, whose malfunctioning would render the network useless, implies that the network is more reliable.
- *Compact routing tables*: As a result of hierarchically dividing the network, the size of routing tables remains small. The route, determined with these tables, along which messages travel will not always be the shortest path between source and destination. However, if the hierarchical division satisfies reasonable assumptions, the length of a chosen path does not differ by more than a constant factor from the shortest path.
- *Dynamical update of routing tables*: A distributed insertion algorithm is presented with which nodes recompute their routing tables after the coming up of a link. The insert algorithm does not require special nodes that possess global information.
- *Locality of changes*: As a result of the hierarchical division and of the hierarchical distance upon which information in routing tables is based, changes in the network topology only affect routing tables of nodes nearby. No node outside a specific area around a topology change participates in the update algorithm.
- *Extensibility*: As a result of the locality of updates, networks can become very large without excessive communication overhead for keeping routing tables up-to-date.

1 Introduction

In every network the routing problem arises: a computer on location X in the network that intends to communicate with a computer on location Y somewhere else

in the network, must determine a route to Y for its messages to proceed along. Many methods have been described, such as centralized-, distributed and hierarchical routing (see, for examples, [7, 8, 10]).

In a *centralized routing* there is a Network Routing Centre. All computers ask this specialized computer for information when they seek a route to some destination. The disadvantages of this method are obvious: success or failure of the routing depends on the NRC's functioning; since all the requests for routing information and answers from the NRC must travel over a limited number of links near the NRC, these links can become saturated; etc.

In a network using a *distributed routing method*, the computers have all information at their disposal necessary to determine a route themselves. A disadvantage of many distributed routing algorithms is that a computer must hold a lot of information to guarantee that every message sent, arrives within finite time at its destination. Another drawback of this method is the necessity to exchange much information by many computers after a change in the network topology (such as failure or coming-up of a computer or link). The exchange of this information and the actions taken by the computers, in response, (such as the determination of a new route to some destination) might take a lot of time. Moreover, distributed algorithms are hard to understand, and even harder to prove correct (cf., [3]). Theory about distributed algorithms and their correctness proofs is beginning to develop (cf., [1, 11]).

By using *hierarchical routing* the amount of information per node can be seriously reduced. A network in which routing is managed in a hierarchical way, is split up in different domains. These domains are connected through special computers, often called *gateways*. When a computer wants to send a message to some computer in a different domain, it must know through which gateway(s) the domain can be reached. By splitting the domains the same way the network is divided, the amount of information a computer must hold can be seriously reduced (see, [5]). A disadvantage of the method is the use of gateways. As with the use of a NRC in centralized routing, the use of gateways does have serious drawbacks. Another consequence of hierarchic routing is that because computers have no complete knowledge of the network topology, the routes along which messages are routed are usually not the shortest paths between source and destination.

In this report a new routing method is described: *distributed hierarchical routing.* This method is, as the name already suggests, a combination of distributed and hierarchical routing. This combination adopts advantages of previous methods, while some of the disadvantages are avoided. The result is a method in which all computers have an equal status, that is, they posses the same kind of information and execute the same routing algorithm. This is the distributed part. By dividing the network hierarchically in domains, sub-domains, sub-sub-domains, etc. it is not necessary to hold large (complete) routing tables.

Other distributed hierarchical routing algorithms have been described (cf., [2]) and analysed (cf., [4, 5]). What distinguishes our algorithm from these, is that it is based upon another distance metric. The newly defined *hierarchical distance*

between computers in the network used in this report takes into account the hierarchical division of the network. This means that details of parts of a path that are inside domains of which the first computer on the path has no internal knowledge do not count in the value of the hierarchical distance to the last computer on the path. This contrasts with the methods in [2] and [5] in which distances are expressed in terms of the least number of hops between two computers.

Using the hierarchical distance has the following advantage. When minor changes in the topology of the network occur, such as the coming up of a link between two computers in the same domain, this does not affect the value of path-lengths between computers outside the domain, even if the paths lead through the domain. Therefore, only nodes in a small area around the change have to update their routing tables. In other words, the effect of a topology change is only local. Again, we emphasize that although using the hierarchical distance has advantages, it does not result in optimal routes.

In Section 2 the routing algorithm is presented. In Section 3 the insertion algorithm is presented. In Section 4 the problems introduced when deletions occur are described. Section 5 contains some remarks about current research.

2 The Routing Method

2.1 Introduction

In this section, a distributed routing algorithm on hierarchically divided networks is presented. The algorithm results in loop-free paths from source to destination. The method described is based on a newly defined metric, which is called *hierarchical distance*. This metric is based on the well-known distance metric in "flat" (i.e., non-hierarchical) graphs. The hierarchical distance between two nodes in the network is a tuple of distances, for every level in the hierarchy the length of a path between domains on that level. For reasons to be explained the hierarchical distance is not symmetric, i.e., the hierarchical distance from node a to node b is not necessarily the same as that from node b to a. Correctness of the algorithm follows from the fact that each time a message is relayed to a next node, the hierarchical distance to its destination decreases.

In section 2.2 the model upon which the routing algorithm is defined is described. In section 2.3 the routing method is described. In section 2.4 it is described how the method can be applied to networks in which domains are allowed to overlap.

2.2 Model

This section introduces the kind of networks the routing algorithm is assumed to be used on. Furthermore, a structure on the networks, namely the hierarchical division, is defined. Based on this structure some basic definitions are presented, which will be used in the definition of hierarchical distance in the next section.

2.2.1 Network and Division

The routing algorithm is a distributed algorithm. This means that the networks it is intended for will have to satisfy some properties. In the first place, every computer is supposed to have local memory and computing capacity. Next, no computer has (direct) access to another computer's memory. Finally, computers cooperate by exchanging messages over links connecting them. These links are fault-free and have a first-in-first-out (FIFO) behaviour. The former (fault-free) means that any message sent by a computer will be correctly received by the computer at the other side within finite time. The latter (FIFO) means that messages are received in the order in which they are sent.

A network is identified with a bidirectional weighted graph, the *network graph*. The computers correspond to the nodes of this graph, the communication links between the computers correspond to the links in this graph. The set of links of the network graph is denoted by the identifier E. The weights of the links reflect some cost associated with the links, e.g., the delay over the link. For routing, links with low weights are preferred to links with high weights.

The networks just described are assumed to be hierarchically divided. This means that computers which, for some reason, belong together are grouped into domains. These domains, in turn, are grouped into superdomains, etc. Opposite to this bottom-up description, in this report a hierarchical division is interpreted in a top-down way: the network is a superdomain, divided into several subdomains, which in turn are divided into subsubdomains, etc. In this view, nodes are the smallest possible domains.

Domains are allowed to overlap, i.e., domains may share (parts of) subdomains. However, the following description assumes non-overlapping domains. Later on, it is described how the more general (overlapping) divisions can be made to fit this description.

The division of a network is modelled as a tree, called the *division tree*. The network domain corresponds to the root of this tree. The subdomains of a domain correspond to sons of the node corresponding to the domain. The leaves of the division tree correspond to the computers in the network.

On the leaves of the division tree the network graph is imposed: two leaves are connected by a link in this graph if and only if they represent computers that are directly connected in the network. Note that the links of this imposed network graph are not part of the division tree (which would no longer be a tree if this were the case). Note that every subtree of the division tree induces a subgraph of the network graph.

One more, rather strong, restriction to the division of networks is made in this report, namely that divisions are hierarchically connected. What is meant by this is formally defined in Section 2.2.3.

The following convention will be adopted throughout this report. Leaves in the division tree are denoted by small letters (a, b, ...). A link in E between the leaves a and b is denoted by $\{a, b\}$. Arbitrary nodes in the division tree (not necessarily

leaves) are denoted by capitals (A, B, ...).

2.2.2 Basic Definitions

Before embarking on defining new concepts related to the routing method, first the terminology and notations used for some well-known concepts with regard to trees are described. This section then concludes with a translation of these concepts in terms of the network.

Definition 2.1 (Son of, Father of) When node A is a *son of* B, and B is the *father of* A.
□

Note, that if A is a son of B, then $A \neq B$. The transitive closure of the son/father-of relation is defined next.

Definition 2.2 (Descendant of, Ancestor of) Node A is a *descendant of* B, and B is an *ancestor of* A, if A is a son of B or if A is a descendant of a son of B.
□

Definition 2.3 (Brother of) If A and B ($A \neq B$) are sons of the same node, they are called *brothers of* each other.
□

Definition 2.4 (Neighbour of) Node A is a *neighbour of* node B, if B is a brother of A, and if there are leaves a and b in the subtrees below A and B respectively (or if $A = a$ and $B = b$), such that $\{a, b\}$ is a link in E.
□

Nodes in the division tree are uniquely defined by their paths.

Definition 2.5 (Treepath) The *treepath* to a node in the division tree is the list of nodes on the path from the root to that node.
□

Note that the treepath to a node is a prefix of the treepath to all of its descendants.

Definition 2.6 (Level) The *level of a node* is the length of the treepath to the node minus one. The level of node A is denoted by $|A|$. The *level of the division tree* is the maximum of the levels of all leaves. We denote the level of the division tree by ν.
□

The root of the network tree has level 0, and the sons of a node with level i have level $i + 1$.

Everything defined above in terms of a tree, can be interpreted as modelling relations and concepts with regard to a division in a network. Because of the way in which the division tree is defined, it is clear that the counterpart of the son-of relation is the *subdomain-of* relation and that the father-of relation corresponds to the *superdomain-of* relation. The transitive closure of the subdomain-of relation is called the *contained-in* relation, and of superdomain-of: *contains*. Domains are called *brothers* if they are subdomains of the same domain. Domains are called *neighbours* if they contain nodes that are connected by a link. The counterpart of a treepath in a division tree is called the *address* of a domain. Domains, especially nodes, in the network graph, are uniquely determined by their address, and every domain has exactly one address. (This is no longer the case if overlapping of domains is allowed.)

The *level* of a domain indicates its depth in the division. The domain containing the whole network has level zero, whereas the individual nodes have higher orders. When a domain has level i, it will be refered to as a *sub^i-domain.*

2.2.3 Special Definitions

In this section the definitions of new concepts related to the routing algorithm are introduced.

Definition 2.7 (DIFF) For nodes A and B ($A \neq B$, A not a descendant of B), $DIFF(A, B)$ is the first node on the treepath to B that is not on the treepath to A. If A is a descendant of B, $DIFF(A, B)$ is B.
□

If A and B are not descendants of each other, e.g. if A is a node and B is a domain not containing A, several remarks can be made with regard to this definition. In the first place, $DIFF(A, B)$ is a brother of $DIFF(B, A)$, i.e., $DIFF$ is not symmetric in its arguments. Furthermore, every ancestor of $DIFF(A, B)$ is an ancestor of both A and B, whereas no descendant of $DIFF(A, B)$ is an ancestor of A. Finally, $|DIFF(A, B)| = |DIFF(B, A)|$.

Definition 2.8 (Visible to) Node A is *visible to* node B if A is a brother of B or if A is a brother of an ancestor of B.
□

Note that if A and B are not descendants of each other $DIFF(A, B)$ is visible to A (but not to B).

This visibility concept is very important for the rest of this report. Note that when a domain is visible to a node (in the network graph) none of the domains contained in it are. Furthermore, when a domain is visible to a node, the domain does not contain the node, but the superdomain of the domain does. The routing

algorithm is now based on the fact that every node keeps a table in which information is stored about how to reach its visible domains. When a message has to be forwarded to a node, it is determined (by inspecting the node's address) in which visible domain it is contained (there exists exactly one such domain!). The message will then be forwarded according to the routing information for that visible domain. Storing only one item of information for a visible domain and not for every single node in it, reduces the size of routing tables considerably. Kleinrock and Kamoun ([5]) showed that under certain assumptions the size of the tables for nodes in a hierarchically divided network containing N nodes can be reduced to $O(\log N)$ entries. This reduction is important when N is very large.

Routing deals with paths, so next paths are defined.

Definition 2.9 (Path) A *path* from leaf a to leaf b in the division tree is a list $(a_0, \ldots, a_l)$ of leaves in the division tree, where $a_0 = a$, $a_l = b$, $i \neq j \Rightarrow a_i \neq a_j$ and $\{a_i, a_{i+1}\} \in E$.
□

Since a hierarchical routing algorithm is being described, now a definition of a path that applies to any level of the division is given.

Definition 2.10 (Path^i, Length) A *path^i* from node A to node B in the division tree, where A and B have level i and are brothers, is a list $(A_0, \ldots, A_l)$ with $A_0 = A$, $A_l = B$, and A_j is a neighbour of A_{j+1} $(0 \leq j \leq l-1)$. The *length* of this path^i is defined as

$$\min\{\sum_{i=1}^{l-1} w(a_i, a_{i+1}) | a_i \in A_i, a_{i+1} \in A_{i+1}, \{a_i, a_{i+1}\} \in E\},$$

with $w(a_i, a_{i+1})$ the weight of the link $\{a_i, a_{i+1}\}$ and $\in$ the descendant-of relation.
□

Now it is possible to define when a division of a network is hierarchically connected.

Definition 2.11 (Hierarchically Connected) A division is *hierarchically connected* if between every pair of brothers on level i a path^i exists.
□

As mentioned before, all divisions are assumed to be hierarchically connected.

Just as the definition of paths was extended to that of path^{i}s, now the definition of link weights is extended. In a flat (i.e., not hierarchically divided) network, links are assigned costs reflecting, for example, the delay over the link. In addition to this value, a *hierarchical* link weight also reflects the level of the neighbouring domains it connects. If for leaves a and b, $|DIFF(a, b)| = i$ (i.e., a and b are in the same sub^{i-1}-domain, but in different sub^i-domains), from the hierarchical link weight of link $\{a, b\}$ this level i can be derived. A link between two leaves in one sub^i-domain, therefore, has another hierarchical weight than a link between two leaves in different

sub^i-domains (even if the costs of the links are equal). This enables us to ensure that paths used for routing between leaves within the same domain do not contain leaves outside the domain (which, by the assumption of hierarchically connectedness, is always possible). We will arrive at this later on.

The hierarchical link weight is defined as a $(\nu+1)$-tuple, with one non-zero element that reflects the cost of the link. The position of this non-zero element in the tuple reflects the level of the domains it connects. This is formalized in the following definition. (By $\overline{0}_i$ we mean $\underbrace{0,\ldots,0}_{i \text{ times}}$.)

Definition 2.12 (Hierarchical Link Weight) Let $\{a,b\} \in E$ have weight w. Let $\delta = |DIFF(a,b)|$. The *hierarchical link weight* of link $\{a,b\}$ is defined as the following $(\nu+1)$-tuple:

$$HW(\{a,b\}) = (\overline{0}_\delta, w, \overline{0}_{\nu-\delta}).$$

□

Hierarchical link weights are ordered lexicographically.

Since "internal details" of subtrees not containing a specific leaf are made "invisible" to that leaf, it is straightforward to also hide for a leaf irrelevant details of paths starting at it. We assume that parts of paths that lead through subtrees of which a leaf has no internal knowledge do not contribute to the hierarchical length. This hiding of details is established by defining the hierarchical length of a path as a $(\nu+1)$-tuple in which the i-th element is the length of a path^i between brothers on level i visible to the leaf. Moreover, these path^{i}s must all be in the direction of the destination, i.e., the end-point of the path.

Before the hierarchical path length is formally defined, first the operator "$\oplus_i$", used in that definition, is defined.

Definition 2.13 (Hierarchical Length Addition) Let the length of a path P from a to b be $(d_0,\ldots,d_\nu)$ and let the length of a path P' from b to c be $(d'_0,\ldots,d'_\nu)$. Then the length of the path $P.P'$ from a to c is

$$(d_0,\ldots,d_\nu)\oplus_i(d'_0,\ldots,d'_\nu) = (d'_0,\ldots,d'_{i-1},d'_i+d_i,d_{i+1},\ldots,d_\nu), \text{where } i = |DIFF(a,b)|.$$

□

Definition 2.14 (Hierarchical Path Length) Let $P = (a_0,\ldots,a_l)$ be a path in the division tree, and let $D_i = HW(\{a_i,a_{i+1}\})$ and $\delta_i = |DIFF(a_i,a_{i+1})|$ $(0 \le i \le l-1)$. The *hierarchical length* of P is defined as:

$$LEN(P) = (D_0\oplus_{\delta_0}(D_1\oplus_{\delta_1}(\ldots\oplus_{\delta_{l-2}}(D_{l-1}\oplus_{\delta_{l-1}}(\overline{0}_\nu))\ldots))).$$

□

Just as in a flat network, the hierarchical length of a path can be determined by (recursively) counting backwards from destination to source: the (hierarchical) length of a path from a node to itself is "zero", the (hierarchical) length of a path $(a_0, a_1, \ldots, a_l)$ is the (hierarchical) weight of the link $\{a_0, a_1\}$ "plus" the (hierarchical) length of the path $(a_1, \ldots, a_l)$.

As mentioned before, the hierarchical length of a path is a $(\nu + 1)$-tuple in which the i-th element corresponds to the length of a path^i. However, it has been argued that only the lengths of path^{i}s between visible nodes are relevant to the source. This is exactly what is realized by the special sum-operator $\oplus_i$ used in the definition. Suppose the hierarchical length of a path from node b to node c is $L = (d_0, \ldots, d_\nu)$ and suppose node a is connected to node b by a link with hierarchical weight $(\overline{0}_i, w, \overline{0}_{\nu-i})$. This implies that $|DIFF(a, b)| = i$, so the ancestor of b with level i (say B^i) is visible to a. Therefore, details about path^{j}s for $j > i$ refer to subtrees under B^i visible to b but not to a. Adding $(\overline{0}_i, w, \overline{0}_{\nu-i})$ to L renders the last $\nu - i$ elements of the tuple zero, thereby hiding these details for a.

Now that the hierarchical length of a specific path is defined, we can continue defining the hierarchical distance between two leaves in the division tree. As in the flat definition, the hierarchical distance between two nodes is the minimum of the hierarchical length of all possible paths. Note that the definition of hierarchical path length is not symmetric, i.e., the hierarchical length of a path from a to b is not necessarily the same as the hierarchical length of the reverse path from b to a. This is because the hierarchical lengths refer to details of domains in which the leaves are contained, whereas a and b are not contained in the same sub^i-domain for every i. Because the hierarchical length is not symmetric, the hierarchic distance will not be symmetric either. Therefore, the hierarchical distance will always be defined between a source node a and a destination node b; the hierarchical distance *from a to b.*

Definition 2.15 (Hierarchical Distance) Let a and b be two leaves in the division tree. The *hierarchical distance from a to b* is defined as:

$$DIST(a, b) = \min\{LEN(P) | P \text{ path from } a \text{ to } b\},$$

where "min" specifies the lexicographic minimum.
□

Theorem 2.1 *For all leaves a, b and c, with $i = |DIFF(a, b)|$:*

$$DIST(a, c) \leq DIST(a, b) \oplus_i DIST(b, c).$$

Proof: Direct from the definitions of $\oplus_i$ and *DIST*.
□

Theorem 2.2 *For all leaves a and c in the division tree:*

$$DIST(a, c) = \min\{HW(\{a, b\}) \oplus_i DIST(b, c) | \{a, b\} \in E \wedge i = |DIFF(a, b)|\}.$$

Proof: From Theorem 2.1 and the definitions of *DIST* and *LEN*.
□

Theorem 2.3 *If node D in the division tree is visible to leaf a, and if b and c are descendants of D, then:*

$$DIST(a,b) = DIST(a,c).$$

Proof: By hierarchically connectedness, $DIST(b,c) = (\overline{0}_{|D|+1}, \ldots)$, and $DIST(a,b) = (\overline{0}_{|D|}, l, \ldots), l > 0$ (because $DIFF(a,b) = D$). Therefore, $DIST(a,b) \oplus_{|D|} DIST(b,c) = DIST(a,b)$. Using Theorem 2.1:

$$DIST(a,c) \leq DIST(a,b) \oplus_{|D|} DIST(b,c) = DIST(a,b).$$

Equivalently,

$$DIST(a,b) \leq DIST(a,c).$$

The theorem follows from the above two inequalities.
□

From this theorem it is justified to refer to the *hierarchical distance from a leaf to a node.* (If node a is contained in domain D, the hierarchical distance of a to D can be defined to be $(\overline{0}_{\nu})$.) This will be used in the the routing method.

2.3 Routing Method

2.3.1 Introduction

The routing problem for node a in the network graph consists for any (reachable) destination c of the determination of an outgoing link $\{a,b\}$, such that the path, beginning with (a,b), that is recursively built (in b) ends up in c. A routing algorithm is a distributed algorithm, executed by every node in the network to obtain a solution of an instance of the routing problem. When a computer in a network has to send messages to a computer somewhere else in the network, it sends these messages over the link determined by the routing algorithm (for that specific destination). Assuming that the message contains the address of its destination, the receiving neighbour will execute its share of the routing algorithm to relay the message one step further.

It is essential that routes are loop-free. Furthermore, it is desirable that the paths along which messages are routed are optimal according to some qualitative criterion. Often used criteria include minimum number of hops and minimum delay. Our criterion is to minimize the hierarchical path length. It is obvious that this does not guarantee that the total weight of the path is minimal. This shortcoming, however, is inherent in the hierarchical nature of the algorithm. Consider, for example, a situation in which all links have weight 1, and consider a source node that has connections to several nodes in a domain that is visible to it. This source node has in its routing table only information related to the visible domain as a whole, i.e., no distinction is made between the nodes in it. Messages to be sent to any node in the

visible domain will be treated alike, so the link returned by the routing algorithm will, as long as routing information does not change, always be the same. Thus, the path a message travels along towards the node incident with another link will not be optimal. That is, the number of hops is not minimal, for the node could be reached in one hop, but will not be reached in less than two hops.

Using the hierarchical path length as a criterion upon which to base routing decisions thus cannot guarantee that shortest paths are found. However, we have the strong feeling that when the network is divided according to several reasonable assumptions (for example regarding the diameter of domains) the found paths will not differ by more than a constant factor from the optimal path. This issue is dealt with in a separate report.

2.3.2 Algorithm

Having already defined hierarchical distances, the description of the routing method is straightforward. Nodes hold routing tables in which there is an entry with information for every domain that is visible to them. This entry consists of two fields: the *LEN*-field, containing the hierarchical distance to that domain, and the *VIA*-field, containing the first link on one of the shortest path(s). This use of hierarchical distance from a node to a domain was justified by Theorem 2.2. The routing algorithm executed by a node consists simply of executing the following two actions: determine in which visible domain the destination is contained, and use the routing table to determine over which link to relay the message towards that domain.

An intuitive description which appeals more to the general ideas behind the hierarchical routing and to the hierarchical distance concept is presented after the formal definition of routing tables and algorithm.

Definition 2.16 (Routing Table) The *routing table* RT_s of node s in the network graph contains entries for all domains D that are visible to s:

$$RT_s(D) = \langle via, len \rangle,$$

where *len* is the hierarchical distance from s to D, and *via* is the first link on a path to D with hierarchical length *len*.

The first entry of this record is called the *VIA-field*, the second entry *LEN-field*.

□

The *LEN*-field is not used in the routing algorithm. The reason for storing it nevertheless, is that it is used when computing routing tables in the initialization phase. Also when, as a consequence of topology changes, routing tables have to be updated, the *LEN*-field is used. The issues of initialization and updating of routing tables are treated in Section 3. In the routing algorithm described below, the index s added to procedure names indicates that these procedures are executed by node s.

The function of procedure $MAIN_s$ is to interface between the part of a node that performs the receiving of messages and the actual routing algorithm.

Function *FIRSTDIFF* returns the ancestor of the destination that is visible to s. This is implemented by inspecting the destination's address and comparing it with the address of s. The shortest prefix of the destination's address that is not a prefix of the address of s is the address of that ancestor.

ROUTING ALGORITHM for node s.

$MAIN_s$ = **forever**
receive message m with destination d;
if $(s = d)$
then accept m
else send m over the link determined by $ROUTE_s(d)$
fi
reverof

$ROUTE_s(d) = RT_s(FIRSTDIFF(a_s, a_d)).VIA$
where a_s is the address of s **and**
a_d is the address of d

$FIRSTDIFF(a_s, a_d) = hd(a_d)$: **if** $(hd(a_s) = hd(a_d))$
then $FIRSTDIFF(tl(a_s), tl(a_d))$
else ( )
fi

□

It can be easily verified that the algorithm is correct, as is stated in the following theorem.

Theorem 2.4 *When routing is performed on a hierarchically divided network with routing tables as described and using the above algorithm, any message arrives at its destination.*

Proof: From Theorem 2.1 it follows, that every time the message is sent over a link, the hierarchical distance to the destination decreases. Since hierarchical distances are bounded from below (by $(\overline{0}_\nu)$), the theorem follows.
□

2.3.3 Informal Approach

In this section the routing method is informally described. The source node in which a message resides is denoted by s, and the destination node is denoted by d. The address of s is denoted by $(S_0, \ldots, S_\nu)$ and the address of d by $(D_0, \ldots, D_\nu)$. Furthermore, $|DIFF(s, d)|$ is abbreviated to δ, so $D_i = S_i$ for $0 \leq i \leq \delta - 1$, and D_δ

and S_δ are brothers (both are contained in $D_{\delta-1}$). After having executed the routing algorithm, node s sends the message over link l determined by $RT_s(D_\delta).VIA$.

By definition, there exists a path beginning with l and whose hierarchical weight equals $DIST(s,d)$. Recall the way this hierarchical distance is defined (Section 2.2.3). Every path from d to s is examined and a link on the path between two neighbouring sub^i-domains contributes to the hierarchical distance of it: its weight is added to the i-th element of the tuple and values of elements corresponding to higher hierarchical levels are removed (by making them zero).

Since the hierarchical length of a path that is completely inside $D_{\delta-1}$ has at least $(\delta-1)$ zeroes in front, it follows, that any path completely inside $D_{\delta-1}$ is (lexicographically) shorter than any path containing also nodes outside $D_{\delta-1}$. Thus, $DIST(s,d)$ is the hierarchical length of a path that is completely inside $D_{\delta-1}$. By hierarchically connectedness of the division, such a path exists.

Now, consider $DIST(s,d)$ more closely. As can be seen from the above description, the element on the δ-th position indicates the length of a shortest path$^\delta$ starting with S_δ and ending with D_δ. Let B_δ be the neighbour of S_δ on the path. On "adding" the link weight of the link on the path from s to d between S_δ and B_δ, all elements on positions $\geq \delta+1$ of the hierarchical length determined thus far are made zero. Therefore, the values on those positions in $DIST(s,d)$ are determined by lengths of path^{j}s inside S_δ. For example, the $(\delta+1)$-st element in $DIST(s,d)$ is the length of a path$^{\delta+1}$ from $S_{\delta+1}$ to $D'_{\delta+1}$, where $D'_{\delta+1}$ is the subdomain of S_δ containing a node connected to a node in B_δ.

Recursively we find that the j-th element of $DIST(s,d)$ is the length of a path^j within S_{j-1} from S_j towards B_j where B_j contains a node connected to a node in the neighbouring sub^j-domain on the shortest path^j to B_{j-1}. In the example of Figure 1 the following holds:

$$\begin{aligned} DIST(s,d) &= (0,1,2,3) \\ &= (0, \text{length shortest path from } S_\delta \text{ to } D_\delta, \\ &\quad \text{length shortest path from } S_{\delta+1} \text{ to } D'_{\delta+1}, \\ &\quad \text{length shortest path from } s \text{ to } D'_{\delta+2}). \end{aligned}$$

Having pointed this out, the routing decision s takes (i.e., route over link l) can be considered as a short-cut of the following chain of reasoning. To reach node d, the message has to first reach D_δ; once it has arrived in that domain, it will be directed further onwards to d. To reach D_δ, the message must hop from sub$^\delta$-domain S_δ to D_δ along a shortest path$^\delta$. Therefore, it should be directed to the neighbour of S_δ on this path$^\delta$. In order to reach this neighbour, the message must be routed along a shortest path$^{\delta+1}$ from $S_{\delta+1}$ towards a sub$^{\delta+1}$-domain of S_δ in which there is a connection to the neighbour. In order to reach this subdomain it must hop from sub$^{\delta+1}$-domain $S_{\delta+1}$ to that subdomain. Continuing this way, ultimately s finds that it must send the message to a neighbour node. The link to this node now, is the one stored in $RT_s(D_\delta)$.

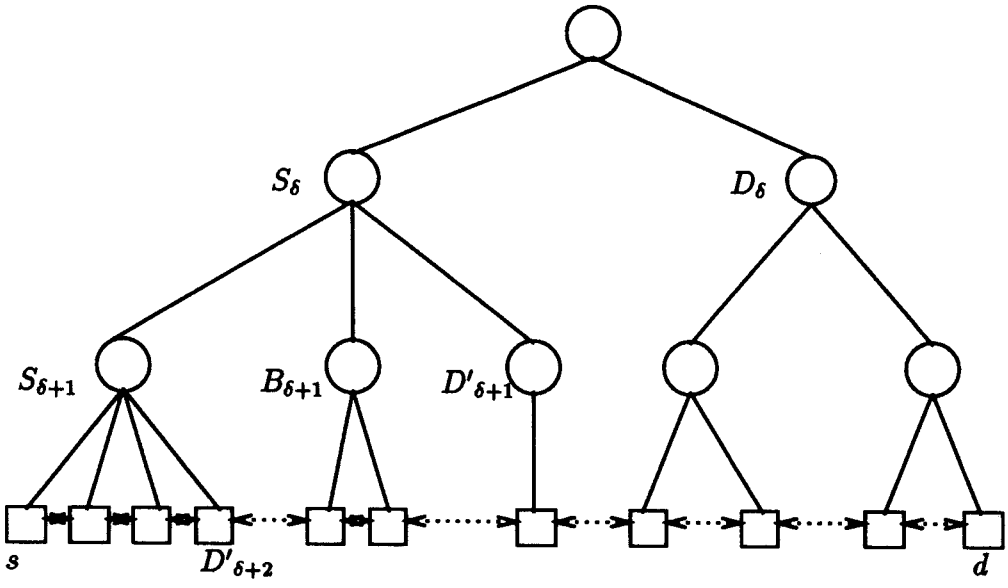

Figure 1: **Example division**

2.4 Unravelling

So far, it has been assumed that divisions are such that domains do not overlap. This allows us to model a division as a tree. Since in this division tree every node has exactly one treepath assigned to it, the set of domains that are visible to a node is uniquely determined. This property is used in the routing algorithm, since the information a node keeps in its routing table is defined with respect to this set of visible domains. In this section we describe the refinements to be made when overlapping domains are allowed.

We define overlapping of domains by a domain being a subdomain of several superdomains. This cannot be modelled by a tree, since a node in a tree cannot have several fathers. A division in which domains overlap can be modelled by a DAG, a directed acyclic graph, which is called the *division DAG*. As before, it is assumed that there is a network domain that contains all other domains. Therefore, every DAG in which there is exactly one "root" is the division DAG of a division.

Recall from the definition that a node is visible to another node if it is a brother of an ancestor of the node. Visibility, therefore, is very much determined by the path in the DAG to a node. If a node has two paths to it (which, in a DAG, is possible), it has two sets of ancestors, and thus two sets of visible nodes. We conclude that visibility depends on the "DAG-path" under consideration and the definition of visibility has to be refined to: visible *relative to a DAG-path.*

In terms of the network this translates to the following: depending on the address

of a node, the node has to keep different information (since the set of domains visible to it depends on this address). Consider a node s that is part of two domains S_1 an S_2. Than s has (at least) two addresses: a_{1_s} (indicating that s is in S_1) and a_{2_s} (indicating that s is in S_2). Every subdomain in S_1 is visible to every node in S_1, except the subdomain containing the node. The same holds for subdomains in S_2; these are only visible to nodes in S_2. Therefore, if s acts under the alias a_{1_s}, it has to keep information about subdomains of S_1. But this information it should not keep if it acts under the alias a_{2_s}. Equivalently, acting as a_{2_s}, it has to keep information about subdomains of S_2, which it does not keep acting as a_{1_s}.

As this example suggests, the information a node possesses, and the routing decisions it takes based upon this information, vary upon the address in use of the node. Once this is understood, it is but a small step to the following crucial remark. Instead of considering a node as an entity with different addresses whose actions are based upon the address in use, a node can be considered as being built up of several independent processes; one for each address. These processes can be considered as *virtual nodes* with only one address. Since virtual nodes in reality are part of one physical node it is reasonable to assume that they are connected to each other by means of *virtual links*. Two physical nodes connnected by a physical link, can thus be viewed as two sets of virtual nodes, interconnected by a set of virtual links (which, when implemented, are multiplexed over the physical link). This process of regarding nodes with multiple addresses as sets of virtual nodes, is called unravelling of the division.

Definition 2.17 (DAG-Unravelling) *Unravelling* DAG D, consists of repeatedly applying the following transformations to D until D is a tree:

1. Let A be a node in D with fathers $B_1, \ldots, B_n$, $n > 1$.

2. Let $A_1, \ldots, A_n$ be copies of A pointing to copies of the subDAGs under A.

3. Change every link in D from B_i to A into a link from B_i to A_i $(1 \leq i \leq n)$.

□

The implications of unravelling on the network graph have been described above, the following definition formalizes things a bit.

Definition 2.18 (Network-Unravelling) Suppose a network graph with edge set E is imposed on the leaves of a division DAG D. *Unravelling the network graph* is the result of unravelling D and, wherever in that process copies are made of a leaf a, extending E as follows:

1. For every $\{a, b\} \in E$:

- remove $\{a, b\}$ from E,
- for every copy a_i of a, add $\{a_i, b\}$ to E.

2. For every pair a_i, a_j of copies of a, add $\{a_i, a_j\}$ to E.

□

We conclude that overlapping domains do not pose any problems in the face of the routing method; if the network is considered to be unravelled, every node again has exactly one address.

3 The Insertion Algorithm

3.1 Introduction

In this section a distributed algorithm is presented for every node to compute correct routing tables after a link or domain has been added to the network. This algorithm is an extension of Tajibnapis' algorithm for flat networks (cf., [6, 9]) in case of insertions.

In Section 3.2 the insertion algorithm is described. In Section 3.3 the algorithm is proven to be correct, i.e., if the algorithm terminates, the resulting routing tables are correct, and within finite time after the last update the algorithm terminates.

3.2 Insertions

The insertion of a domain can be interpreted as the coming up of the nodes that are part of the inserted domain. The insertion of a node can be regarded as if the node is connected to the rest of the network by the coming up of the links incident with the node. Therefore, the insertion of domains can be considered as the coming up of a number of links, namely all links that join two nodes of which at least one is part of the domain. Since the network is also a domain, the initialization of the network can be considered as a special case of the insertion of a domain after the coming up of all links.

It is assumed that nodes incident with a link are able to detect the coming up of links. This is modelled by the nodes receiving notifications of these events.

> It is assumed that a notification is received before any other message sent after the coming up of a link.

Next the insertion algorithm is presented. From Theorems 2.2 and 2.3, it follows that the hierarchical distance from node a to a visible domain D, $DIST(a, D)$, satisfies the following relation:

$$DIST(a, D) = \min\{HW(\{a, b\}) \oplus_i DIST(b, DIFF(b, D)) \mid \{a, b\} \in E \wedge i = |DIFF(a, b)|\}. \quad (1)$$

If node a is part of domain D, then RT_a does not have an entry for D. In this case, by definition $RT_a(D).LEN = (\overline{0}_\nu)$.

When in a computation the value of a table entry, that does not (yet) exist is used, the value $(\overline{\infty}_\nu)$ is used instead, where $(\overline{\infty}_\nu) \oplus_i len = len \oplus_i (\overline{\infty}_\nu) = (\overline{\infty}_\nu)$, for every $0 \leq i \leq \nu$.

Function $dist_a(\{a,b\}, D)$ denotes the minimum, according to b, of the hierarchical lengths of the paths starting with $\{a,b\}$, towards D (which is visible to a). If b is not (yet) informed about how to reach $D' = DIFF(b,D)$, then $RT_b(D').LEN = (\overline{\infty}_\nu)$. In that case, D cannot be reached via link $\{a,b\}$, and we define $dist_a(\{a,b\}, D) = (\overline{\infty}_\nu)$. In other words:

$$dist_a(\{a,b\}, D) = HW(\{a,b\}) \oplus_i RT_b(DIFF(b,D)).LEN, \text{ where } i = |DIFF(a,b)|.$$

Relation 1 induces the following global algorithm for computing correct routing tables:

for as long as routing tables change **do:**
  for every node a **do:**
    for every domain D visible to a **do:**
      $RT_a(D).LEN := \min\{dist_a(\{a,b\}, D) | \{a,b\} \in E\}$
      $RT_a(D).VIA := \{a,b\}$, such that: $dist_a(\{a,b\}, D) = RT_a(D).LEN$
    rof
  rof
rof

This algorithm will be transformed into a distributed one. In order to keep RT_a up-to-date, a has to be informed about changes in b's routing table, if $\{a,b\} \in E$. To accomplish this, every time $RT_b(D').LEN$ changes, node b informs its neighbours about this change. Also, if b finds out that a is a new neighbour, then b informs its new neighbour about its routing table.

Neighbours of node b are informed about entries in RT_b by update-messages. An update-message has the form: $\ll$ UPDATE; b; D'; $len \gg$, where b is the address of the sender, D' is the key of the entry and len equals $RT_b(D').LEN$ at the moment b sent the message. The last two fields of an update-message need not be filled in. This is initially the case, because at that moment no table entries are available.

When link $\{a,b\}$ has come up, a and b send each other their routing table entries to be able to recompute their routing tables. If no entries are available in RT_a, a sends an update-message containing only its address.

When a receives an update-message from b, the information in it is processed in two phases. Both phases consist of the execution of function $PROCESSINFO_a$. This function has three formal parameters: $\{a,b\}$, D and len. Execution of it results in a assigning the minimum of $RT_a(D^\delta).LEN$ and len to $RT_a(D^\delta).LEN$, where $D^\delta = DIFF(a,D)$. If $RT_a(D^\delta).LEN = len$, then $RT_a(D^\delta).VIA := \{a,b\}$. Further, a determines whether $RT_a(D^\delta)$ has changed and update-messages need to be sent to all neighbours in result.

In the first phase $PROCESSINFO_a$ is used to determine whether a path (viz., $\{a,b\}$,) of hierarchical shorter length than the one currently in use towards $B^\delta = DIFF(a,b)$ has been revealed by the message. This, for example, is the case if the update-message is the first received over $\{a,b\}$.

In the second phase, a uses the information in the update-message (if available) to obtain possibly better routing information. From update-message $\ll$ UPDATE; b; D; $len \gg$ ($a \notin D$), received by a, it follows that there is a path to $D^i = DIFF(a,D)$ starting with link $\{a,b\}$ of hierarchical length $HW(\{a,b\}) \oplus_i len$ (with $i = |DIFF(a,b)|$). This path must be the hierarchical shortest of all paths to D^i a is informed about starting with link $\{a,b\}$. If $a \in D$, the update-message does not contain information about a domain that is visible to a and a does not have to change any table-entry as a result.

INSERTION ALGORITHM for node a

ON RECEIPT OF A NOTIFICATION OVER $\{a,b\}$:

if no entry in RT_a
then Send $\ll$ UPDATE; a; $-$; $-$ $\gg$ over $\{a,b\}$
else **for** every entry $RT_a(D)$ **do:**
Send $\ll$ UPDATE; a; D; $RT_a(D).LEN \gg$ over $\{a,b\}$
rof
fi

ON RECEIPT OF AN UPDATE-MESSAGE $\ll$ UPDATE; b; D; $len \gg$:

$PROCESSINFO_a(\{a,b\}, b, HW(\{a,b\}))$;
if the last two entries of the update-message are filled in
then if $a \notin D$
then $i := |DIFF(a,b)|$;
$PROCESSINFO_a(\{a,b\}, D, HW(\{a,b\}) \oplus_i len)$
fi
fi

$PROCESSINFO_a(\{a,b\}, D, len) =$

$D^\delta := DIFF(a,D)$;
if D^δ does not have an entry in RT_a
then *Make_Entry*$(RT_a(D))$
fi;
if $len < RT_a(D^\delta).LEN$
then $RT_a(D^\delta).LEN := len$;
$RT_a(D^\delta).VIA := \{a,b\}$;

Send $\ll$ UPDATE; a; D^δ; $RT_a(D^\delta).LEN \gg$ to every neighbour

fi

3.3 Correctness Proof

In this section a correctness proof of the insertion algorithm is given. The assertional proof method is used, in which one reasons about the program's state instead of its behavior. The matter of correctness proofs for distributed algorithms is extensively treated in [1, 11]. In [6] the method is demonstrated on Tajibnapis' routing method ([9]). The proof in this section is based on Lamport's proof.

The following property of the algorithm is proven:

> After a topology change (if no other updates take place), the nodes will eventually obtain correct routing tables.

In accordance with the assertional proof method this property is proven by showing that the following two properties hold:

> **PR1:** If the system is stable (i.e., no update-messages need be processed), then the computers have correct routing tables.
> **PR2:** After an update, the insertion algorithm will terminate.

Throughout this section a and b denote nodes executing the insertion algorithm, $B^\delta = DIFF(a,b)$, $\delta = |DIFF(a,b)|$, D is a domain that is visible to b and does not contain a, and $D^\delta = DIFF(a,D)$.

The proof of property PR1 uses the following predicate:

$\mathcal{P}(a, D^\delta) \equiv$

for every link $\{a,b\}$:
  if there is no notification underway on $\{a,b\}$
  then if there is an $\ll$ UPDATE; b; D; $len \gg$ underway on $\{a,b\}$ to a
    then (1) the last such message must follow a notification and must contain the current value of $RT_b(D).LEN$
    else (2) **if** $RT_a(D^\delta).VIA = \{a,b\}$
      then $RT_a(D^\delta).LEN = HW(\{a,b\}) \oplus_\delta RT_b(D).LEN$
      else $RT_a(D^\delta).LEN \leq HW(\{a,b\}) \oplus_\delta RT_b(D).LEN$
      fi and
      if $RT_a(B^\delta).VIA = \{a,b\}$
      then $RT_a(B^\delta).LEN = HW(\{a,b\})$
      else $RT_a(B^\delta).LEN \leq HW(\{a,b\})$
      fi and
      there are no notifications on $\{a,b\}$
    fi
  fi
rof

The conditions in the predicate are referred to by the given labels, for example, condition 1 states that there is no notification over link $\{a, b\}$ to b, there is an $\ll$ UPDATE; b; D; *len* $\gg$ underway on $\{a, b\}$ to a and the last such message must follow a notification and must contain the current value of $RT_b(D).LEN$.

In Lemma 3.1 predicate $\mathcal{P}(a, D^\delta)$ is proven to be an invariant of the algorithm. From this lemma it follows that in a stable state the routing table entries contain correct values. In Lemma 3.2 it is proven that when the algorithm has terminated every node has in its routing table information about all domains that are visible to it. From these two lemmata Theorem 3.1, which states property **PR1**, easily follows. Theorem 3.2 concludes the proof.

Lemma 3.1 *Predicate $\mathcal{P}(a, D^\delta)$ is an invariant of the algorithm.*

Proof: In the initial state no link exists. Therefore, initially $\mathcal{P}(a, D^\delta)$ is true.

From the definition of $\mathcal{P}(a, D^\delta)$, we see that the following actions affect the truth value of $\mathcal{P}(a, D^\delta)$.

Coming up of the link $\{a, b\}$: This immediately places a notification on $\{a, b\}$, making the predicate true. This notification will be received by a and b before any other message and no other notification will follow this notification. The receipt of the notification by b causes b to send to a update-messages, including $\ll$ UPDATE; b; D; $RT_b(D).LEN \gg$, making condition 1 true.

Receiving $\ll$ UPDATE; b; D; *len* $\gg$ by a: The only case that processing these fields could change the truth value of $\mathcal{P}(a, D^\delta)$ is if this is the last such update-message on that link and condition 1 was true before. From condition 1 it follows that a received a notification over $\{a, b\}$ earlier and there are no more notifications on that link. It is clear that after processing this message the following holds: $RT_a(B^\delta).LEN \leq HW(\{a, b\})$ and if $RT_a(B^\delta).VIA = \{a, b\}$ then $RT_a(B^\delta).LEN = HW(\{a, b\})$. Now, suppose the two last fields of the message are filled in. Condition 1 then implies that *len* contains the current value of $RT_b(D).LEN$. It is clear that after processing condition 2 will hold.

□

Lemma 3.2 *When the system is stable, the following holds: if domain D is visible to a, then $RT_a(D)$ exists.*

Proof: Because the division is hierarchically connected, there exists a path P between a and a node s such that:

- s has a neighbour $d \in D$, and
- D is visible to all nodes on the path (including s).

Consider the moment the link $\{s, d\}$ came up. Both s and d receive a notification over that link. This causes both nodes to send each other at least one update-message.

After s received the first such update-message, it recomputes $RT_s(D)$. Thus, $RT_s(D)$ exists. By assumption link $\{s, d\}$ remains working, and therefore, $RT_s(D)$ will not be removed.

Consider the last time s created $RT_s(D)$. This causes s to send each neighbour this entry in an update-message. Thus p, the following node on P will receive this message too (if $\{s, p\}$ has not come up yet, it will be sent when the link $\{s, p\}$ comes up). Receiving the message, p recomputes $RT_p(D)$. Again, $RT_p(D)$ remains existing, and p sends this entry in an update-message to all its neighbours, including the following node on the path.

Continuing the argument, it follows that node a receives an update-message containing information about D, which causes the creation of $RT_a(D)$ and after which the entry remains existing.

□

Theorem 3.1 *When the system is stable, all nodes have correct routing tables.*

Proof: From Lemma 3.2, it follows that the routing table of node a contains an entry for each domain visible to a.

From Lemma 3.1 predicate $\mathcal{P}(a, D^\delta)$ is an invariant. In a stable state, condition 1B holds. From this condition and Theorems 2.2 and 2.3, it follows that RT_a of node a contains correct values.

□

Theorem 3.2 *After the coming up of links has ceased, the system eventually becomes stable.*

Proof: It will be proven that the insert algorithm terminates, by giving a bound-function V. This is a bounded function of the state of the complete network which decreases in every step of the algorithm and is bounded from below.

This function V has as value the vector $(e, M + D, l)$, where

e = the total number of unprocessed notifications.

M = a vector containing elements $m(l_0, \ldots, l_\nu)$ for every combination (except $(\overline{0}_\nu)$) of values for the l_i in reversed lexicographic order, where each $(l_0, \ldots, l_\nu)$ is the hierarchical length of an existing path in the network or equals $(\overline{\infty}_\nu)$.

D = a vector containing elements $d(l_0, \ldots, l_\nu)$ for every combination (except $(\overline{0}_\nu)$) of values for the l_i in reversed lexicographic order, where each $(l_0, \ldots, l_\nu)$ is the hierarchical length of an existing path in the network or equals $(\overline{\infty}_\nu)$.

$m(\overline{l})$ = the number of update-messages in transit (i.e., already sent, but not yet received and processed) reporting the value $\overline{l}$.

$d(\overline{l})$ = the number of pairs (a, D) such that $RT_a(D).LEN = \overline{l}$.

It is obvious that V has a minimum value, since every constituent is a nonnegative integer. V has a bounded number of elements because the number of paths in the network is bounded. To show that V keeps decreasing, the actions performed by node a whilst processing notifications and update-messages, must be considered.

Processing a notification decreases the leftmost element e, of V, obviously decreasing V.

Suppose node a receives an $\ll$ UPDATE; b; D; len $\gg$-message. The processing of these messages by a involves the following steps:

1. Remove the message from the network in order to process it.
2. Recompute $RT_a(B^\delta)$.
3. If $RT_a(B^\delta)$ has changed, then send its new value, in an update-message, to every neighbour.
4. If D does not contain a, recompute $RT_a(D^\delta)$.
5. If $RT_a(D^\delta)$ has changed, then send its new value, in an update-message, to every neighbour.

Let dB_o be the original value of $RT_a(B^\delta).LEN$, and dB_n its new value. In the same way, dD_o is the original value of $RT_a(D^\delta).LEN$, and dD_n its new value.

The five actions have the following consequences for the value of V:

1. Decreases $m(len)$.
2. Decreases $d(dB_o)$, if $dB_o > dB_n$.
3. Increases $d(dB_n)$ and $m(dB_n)$, if $dB_o > dB_n$.
4. Decreases $d(dD_o)$, if $dD_o > dD_n$.
5. Increases $d(dD_n)$ and $m(dD_n)$, if $dD_o > dD_n$.

Consider the influence of actions 2 and 3 on V:

$dB_o = dB_n$: $m(len)$ decreases, thus V decreases.

$dB_o < dB_n$: It is obvious that this case cannot occur.

$dB_o > dB_n$: The dB_n-component that has increased lies to the right of the dB_o-component that is decreased. The overall result on V is that its value decreases.

Consider the influence of actions 4 and 5 on V:

$dD_o = dD_n$: From above it follows that V decreases.

$dD_o < dD_n$: It is obvious that this case cannot occur.

$dD_o > dD_n$: The dB_n-component that has increased lies to the right of the dB_o-component that is decreased. The overall result on V is that its value decreases.

□

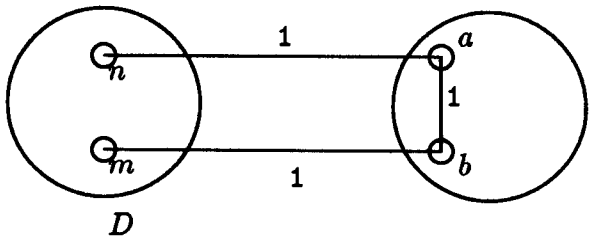

Figure 2: **Example Network**

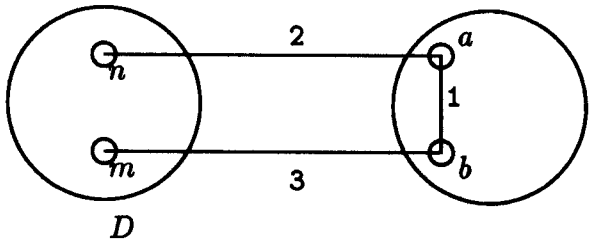

Figure 3: **Example Network**

4 Deletions

It is not straightforward to adapt the deletion algorithm described in [6, 9] for the case of hierarchical divisions. The method of Tajibnapis and Lamport in the face of deletions is based on every node keeping information about its neighbours' routing tables.

Whenever a node determines that the path over which it routed to some destination has its (hierarchical) length increased, the information about the neighbours' routing tables is used to determine whether an alternative path with a shorter (hierarchical) length is available.

As an example, consider the network of Figure 2. When a detects the failure of $\{a, n\}$, from its extra information it knows that via b, D can still be reached. This algorithm, however, may not terminate as is demonstrated by the next example.

Consider the network as depicted in Figure 3. The corresponding routing tables are given in Figure 4. Suppose link $\{a, n\}$ fails. Node a determines that a path via neighbour b to visible domain D is the hierarchical shortest, with hierarchical length $[0, 2, 2]$. Node a announces this new length to b. Node b then determines that the

[9] William D. Tajibnapis. A correctness proof of a topology information maintenance protocol for a distributed computer network. *Computer Systems*, 20(7):477–485, July 1977.

[10] Andrew S. Tanenbaum. *Computer Networks.* Prentice/Hall International, Inc., Englewood Cliffs, N.J., 1981.

[11] Gerard Tel. *The Structure of Distributed Algorithms.* PhD thesis, Rijksuniversiteit Utrecht, Utrecht, February 1989.

A possible solution is to split a domain that is no longer hierarchically connected into two domains that are. Splitting a domain may also be useful in case a domain, as a result of many insertions into it, has grown larger than is desirable.

Another open problem is how to join two domains into one.

5.5 Hierarchical Synchronization

During the execution of the update algorithm, the information nodes obtain converges to the correct routing information. Everytime a node updates a table-entry the result of this update is reported to its neighbours. It might be possible to reduce the amount of communication if nodes inside one domain reached agreement on what data they keep in their table-entries before this information is sent to nodes in other domains. It seems useful to investigate how such a synchronization can be obtained and what the advantages and disadvantages of it are.

References

[1] K. Mani Chandy and Jayadev Misra. *Parallel Program Design – A Foundation.* Addison-Wesley Publishing Company, Inc., 1988.

[2] J.J. Garcia-Luna-Aceves. Routing management in very large-scale networks. *Future Generations Computer Systems*, 4(2):81–93, September 1988.

[3] J.M. Jaffe, A.E. Baratz, and A. Segall. Subtle design issues in the implementation of distributed, dynamic routing algorithms. *Computer Networks and ISDN Systems*, (12):147–158, 1986.

[4] Farouk Kamoun and Leonard Kleinrock. Stochastic performance evaluation of hierarchical routing for large networks. *Computer Networks*, (3):337–353, 1979.

[5] Leonard Kleinrock and Farouk Kamoun. Hierarchical routing for large networks (performance evaluation and optimization). *Computer Networks*, (1):155–174, 1977.

[6] Leslie Lamport. An assertional correctness proof of a distributed algorithm. *Science of Computer Programming*, (2):175–206, 1982.

[7] Mischa Schwartz. Routing and flow control in data networks. IBM Research Report RC 8353 (No. 36329), IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA, October 1980.

[8] Mischa Schwartz and Thomas E. Stern. Routing techniques used in computer communication networks. *IEEE Transactions on Communications*, COM-28(4):265–278, April 1980.

functions (e.g., routing).

- The devices connected to the network, being *freed from the task of performing network functions*, can spend (almost) all their computing capacity to their specific task.
- If the devices connected to the network consist of Transputers, the network can be used for *communication within the devices*; the distinction between network and device vanishes. (Of course, network devices must be such that failure of one device does not affect the network nor any other device.)

5.2 Link Weights and Load Balancing

The routing algorithm described in this report is such that messages to be routed from one domain to a neighbouring domain are routed over the least weight link(s) between the domains. If all links have equal weight, all links will be used. If, on the other hand, only one has least weight, all traffic (under consideration) is routed along this single link, leaving all others idle. This may be intended: if link weights reflect, for example, the delay over links messages are routed over the shortest delay path. In many cases, however, it may be unwanted: if there exist many links, all (or most of them) should be used for routing.

Several interesting questions arise:

- How can the definition of hierarchical length be adapted to not count as the distance between two domains the minimum link weight, but some weighted average of all link weights?
- How can link weights be dynamically changed and how can routing information be kept up-to-date in this case? Is it possible to introduce thresholds: as long as link weights do not exceed a threshold, routing information is not adapted in response to minor changes.

5.3 Analyses

As mentioned in Section 2.3.1 the paths found by the routing algorithm usually are not minimum weight paths. Under what conditions can the method be expected to deliver short paths? Does there exist a class of divisions that is very well suited to use the routing method upon?

Further analysis also has to be performed to determine the complexity of the update algorithms.

5.4 Splitting and Joining Domains

An interesting question arises when, as a consequence of a deletion, a domain is no longer hierarchically connected. Since the assumption of hierarchically connectedness is essential in the routing algorithm, actions have to be undertaken in this case.

RT_a	LEN	VIA
⋮	⋮	⋮
D	$[0,2,0]$	$\{a,n\}$
⋮	⋮	⋮

RT_b	LEN	VIA
⋮	⋮	⋮
D	$[0,2,1]$	$\{a,b\}$
⋮	⋮	⋮

Figure 4: **Routing Tables**

$RT_a(D)$	
LEN	VIA
$[0,2,0]$	$\{a,n\}$
$[0,2,2]$	$\{a,b\}$
$[0,2,2]$	$\{a,b\}$
$[0,2,4]$	$\{a,b\}$
⋮	⋮
$[0,2,\infty]$	$\{a,b\}$

$RT_b(D)$	
LEN	VIA
$[0,2,1]$	$\{a,b\}$
$[0,2,1]$	$\{a,b\}$
$[0,2,3]$	$\{a,b\}$
$[0,2,3]$	$\{a,b\}$
⋮	⋮
$[0,2,\infty]$	$\{a,b\}$

Initially
After 1 announcement
After 2 announcements
After 3 announcements

Figure 5: **History**

minimum hierarchical length of the paths to D via a is $[0,2,3]$. Because $[0,2,3] < [0,3,0]$, the information that there is a path to domain D with length $[0,2,3]$ (via node a) is stored in b's routing table. Subsequent announcements produce the infinite history as shown in Figure 5.

In general, without sufficient precautions taken, when the weight of a link^i between two sub^i-domains increases, nodes within those domains may end up increasing the j^{th} element ($j \geq i$) of the hierarchical distance they keep in their routing tables for destinations reached over that link^i ad infinitum.

5 Further Research

In this section we briefly mention several topics that are subject of current research.

5.1 Implementation

The routing algorithm described in the previous sections has been developed for use in a network of *Transputers*. A Transputer is a fast processor with (four) communication links. As a result of these links, Transputers can easily be connected, thus creating a network. Several advantages of such a network are:

- It will be an *open* network. It is easy to connect devices to the network since it is not necessary to have the operating systems of the devices perform network