Revised Dec. 1977

September 1978

LIDS-P-851
Submitted to IEEE
Trans. on Communications

OPTIMAL DISTRIBUTED ROUTING FOR VIRTUAL LINE-SWITCHED DATA NETWORKS

Adrian Segall

Department of Electrical Engineering
Technion - Israel Institute of Technology
Haifa, Israel.

Abstract

An algorithm that provides minimum delay routing in a data communication network using virtual line-switching is presented. The algorithm uses distributed computation in the sense that the nodes of the network update their information in an orderly fashion based on messages received from their neighbors. Receipt of these messages also trigger the various steps of the update and rerouting, so that these operations are performed in appropriate sequencing. For stationary input requirements and fixed topology the algorithm reduces network delay at each step and provides loop-free routing in the network. The method also provides an algorithm for quasi-static routing, when the input flows are slowly changing.

This research was supported by the Advanced Research Project Agency of the US Department of Defence (monitored by ONR) under contract No. N00014-75-C-1183.

I. INTRODUCTION

In a recent paper [1], an algorithm for distributed adaptive routing that achieves minimum average delay in a message (or packet) switched data network was introduced. The essence of the algorithm is to dynamically change the entries of the routing tables, consisting of the fractions of incoming flow for each destination that a node sends on each outgoing line. The sequencing of the changes is such that the routing is always loop-free and converges to the minimum delay routing.

The main purpose of this paper is to extend the algorithm of [1] to networks using *virtual line-switching* namely store-and-forward networks where a user calling into a particular node of the network is assigned one or more virtual channels to that node. The capacity of the channels is not preassigned, but rather the nodes use some kind of statistical multiplexing to combine data belonging to the calls routed through each link. During conversations, if the situation in the network requires it, rerouting of virtual channels or portions thereof is possible, and finally the channels are cancelled when the conversation is over. Virtual line-switching is probably best suitable for networks where the basic message is composed of a small number of characters and is in fact already used in a number of terminal-oriented data networks, like TYMNET [9] and networks projected to use the CODEX 6000 Intelligent Network Processor [10].

Regarding analysis of distributed routing, there are two interrelated

main differences between message (or packet) switching and line-switching; in line-switching, the quantities to be controlled are the flows themselves and not the fractions, and also, if a node decides to initiate the rerouting of a channel passing through it, the entire portion of the old channel from that node to the destination will have to be cancelled and a new line established. The first issue above makes the analysis of line switching somewhat easier than for message switching, because under suitable assumptions the functions involved are convex in the flows, but not in the fractions; on the other hand, the second issue makes the implementation harder, since rerouting for line-switching requires a certain sequencing from the nodes to the destination, while in packet switching it suffices to update routing tables locally at each node and to perform the rerouting independently from node to node.

Finally, it is interesting to note that the algorithm of [1] as well as the version in this paper are actually a combination of the reduced gradient algorithm [7, p.262] and an algorithm proposed by McCormick [8] modified for the purpose of data network optimization. The resulting algorithms have the remarkable property of being implementable using *distributed computation*, when the nodes receive information in an orderly fashion from their neighbors, update their own information, and then perform the rerouting. The algorithms have the further properties that, for fixed topology and stationary traffic requirements, every single rerouting reduces the network delay and the routings provided by the steps of the algorithm converge to the optimal

routing in the sense of minimum average delay. Because of the above properties, the algorithms can be implemented on-line, while the network is operating and thus provide what they are really intended for, algorithms for quasi-static routing, when requirements are changing slowly compared to the speed of convergence of the algorithm and links or nodes occasionally fail or are added to the network. In the version of this paper, the rerouting must be performed in an appropriate sequencing, percolating from the initiating node down to the destination. As an important byproduct, the algorithm insures that the destination will know the time of completion of each update-rerouting cycle and therefore will start a new cycle only after the previous one is completed.

II THE MODEL

Consider a data-communication network consisting of N nodes $\{1,2,\ldots N\}$. The directed link connecting nodes i and k will be denoted by (i,k) and the collection of links by $\mathcal{L}$. We shall assume throughout the paper that all lines are byplex, namely if $(i,k) \in \mathcal{L}$, then $(k,i) \in \mathcal{L}$ and for each node i, denote by $Z(i)$ the collection of its neighbors.

Let $r_i(j) \geq 0$ be the average traffic entering the network at node i and destined for node j, $f_{ik}(j)$ be the flow in link (i,k) of messages destined for node j and C_{ik} be the capacity of link (i,k). Then the flows $f_{ik}(j)$ must satisfy

$$\sum_{k\in Z(i)} f_{ik}(j) - \sum_{\substack{\ell\in Z(i)\\ \ell\neq j}} f_{\ell i}(j) = r_i(j) \quad \text{all } i,j,\ i \neq j . \tag{1}$$

$$f_{ik}(j) \geq 0 \quad \text{all } i,j,k,\ i \neq j . \tag{2}$$

$$\sum_j f_{ik}(j) < C_{ik} \quad \text{all } (i,k) \in \mathcal{L} . \tag{3}$$

The objective of the routing is to minimize the average delay in the network. Let D_{ik} be the total delay per unit time of all traffic passing through link (i,k). Explicitly, D_{ik} is the average delay per unit of traffic multiplied by the amount of traffic per unit time transmitted over link (i,k). We shall assume here that D_{ik} is only a function of the total traffic flow $f_{ik} = \Sigma f_{ik}(j)$ passing through link (i,k). Some of the consequences of this assumption are indicated in [1]. Then the total delay

in the network per unit time is given by

$$D_T(f) = \sum_{(i,k)\in\mathcal{L}} D_{ik}(f_{ik}) \tag{4}$$

and since the total traffic in the network is independent of the routing procedure, we can minimize the average delay in the network by minimizing D_T. The main purpose of the paper is to indicate an iterative algorithm for performing this minimization.

Before proceeding, we should point out that the algorithm requires no explicit knowledge of the function $D_{ik}(f_{ik})$. Formulas for this function for various traffic models and assumptions have been previously obtained [11], [12], but here we shall need to assume only the following reasonable properties of the functions $D_{ik}(\cdot)$:

- D_{ik} is a non negative continuous increasing function of f_{ik}, with continuous first and second derivatives. (5a)
- D_{ik} is convex $\cup$. (5b)
- $\lim_{f_{ik}\uparrow C_{ik}} D_{ik}(f_{ik}) = \infty$. (5c)
- $D'_{ik}(f_{ik}) > 0$ for all f_{ik}, where D'_{ik} is the derivative of D_{ik}. (5d)

In addition, observe that the flows f are taken to be continuous variables. From a practical point of view this means that the flow for each connection is of small enough size, or that two or more subpipes can be opened for each pair of source-destination devices

(i.e. N-plexing is allowed) and a continuous amount of flow can be transferred from one subpipe to the other. This assumption will be further discussed in Section V.

Theorem 1 Assume that the set of flows satisfying (1), (2), (3) is nonempty. If the delay functions have properties given in (5), the flow $f^* = \{f^*_{ik}(j)\}$ minimizes D_T under constraints (1), (2), (3) if and only if there exists a set of numbers (Lagrange multipliers) $\lambda^* = \{\lambda^*_i(j)\}$ such that the Kuhn-Tucker conditions

$$D'_{ik}(f^*_{ik}) + \lambda^*_k(j) \begin{cases} = \lambda^*_i(j) & \text{if } f^*_{ik}(j) > 0 \\ \geqslant \lambda^*_i(j) & \text{if } f^*_{ik}(j) = 0 \end{cases} \qquad i \neq j, \quad k \in Z(i) \tag{6}$$

are satisfied. Here

$$\lambda^*_j(j) = 0 , \tag{6a}$$

and D'_{ik} is the derivative of $D_{ik}(f_{ik})$.

The proof of this type of theorems is well-known (see e.g. [13]) and therefore omitted. It is also well known [7, p. 231] that the Lagrange multipliers λ^* are the sensitivity coefficients of the optimal cost with respect to the level of the constraint. In our situation, if the flow $r_i(j)$ is increased by an incremental quantity $\delta r_i(j)$ and everything else is held fixed, then the incremental increase in minimum delay will be $\lambda_i^*(j) \cdot \delta r_i(j)$. Consequently, the optimality conditions (6) have an interesting interpretation: Consider a given destination j and an arbitrary node i in the network. Look at all neighbours k of i and calculate the sum of their incremental delay coefficient $\lambda_k^*(j)$ and the incremental delay coefficient D'_{ik} on the line connecting i to k. Optimality requires that for all neighbors to which i sends traffic destined for j, this sum will be the same and no larger than the sum corresponding to neighbors to which i sends no traffic with final destination j. If and only if this is the situation for all nodes and all destinations in the network, the corresponding routing f^* is optimal.

Another fact to be noted before proceeding is that in the optimality conditions (6), λ's corresponding to different destinations are not related. It is expected therefore that the rerouting algorithm will evolve independently from one destination to another.

III. THE REROUTING ALGORITHM

Similar to [1], the optimality conditions (6) show that generally speaking, the algorithm should be such that nodes will increase traffic on links with small incremental delay $D'_{ik} + \lambda_k(j)$ and decrease traffic on those with large incremental delay. In order to perform these actions, each node i will need the incremental delays D'_{ik} over each outgoing link (i,k) and the incremental delay $\lambda_k(j)$ of each neighbor k.

The quantity D'_{ik} can be obtained by node i by estimating f_{ik} and using one of the formulas for $D_{ik}(f_{ik})$. Alternatively, and probably preferably, node i can estimate D'_{ik} directly, thereby avoiding assumptions on the flow that are not always reasonable. Clearly both procedures will depend on the particular schemes for sending messages through the lines. An algorithm for estimating D'_{ik} for a virtual line-switched character multiplexing network was developed in [2, Eq.(30)-(33)].

The node incremental delays $\lambda_k(j)$ will have to be sent by the neighbors. This immediately brings up the question of a potential deadlock: in order to calculate $\lambda_i(j)$, node i needs the numbers $\lambda_k(j)$ from all the neighbors k, but to calculate its own $\lambda_k(j)$, a neighbor k needs the numbers from all its own neighbors, i included. It is therefore necessary to break this deadlock at the outset, and realize that in each step of the algorithm, each node will have to use only a subset of its neighbors to establish its number $\lambda_i(j)$. This is somewhat different from [1, Eq.(5)], where node i needs numbers only from a subset of its neighbors to calculate its $\partial D/\partial r$

We define a *step of the algorithm* to be a complete cycle consisting of updating of λ's and rerouting in the entire network. We shall see later that the destination can start a new step only after the previous one was completed, and although the algorithm is distributed, it has the property that the destination will indeed know when this completion occurs. In order to see how a step of the algorithm progresses through the network, we need several definitions. The discussion will refer to a given destination j. For a node i that has any flow passing through it destined for j, all neighbors k such that $f_{ik}(j) > 0$ are called its *real sons* and node i is called their *father* (a node can have more than one father). A node i such that $f_{ik}(j) = 0$ for all neighbors k has no real sons, but has exactly one *adopted son;* this is its prefered neighbor to which it would send any traffic destined for j if such traffic comes in. Observe that this is different from [1], where the concept of adopted son is nonexistent and not necessary. A node k is said to be a *son* of i, if it is either its real son or its adopted son. We denote by $S_i^n(j)$ the list of sons of node i for destination j at step n of the algorithm. The exact algorithm to choose adopted sons will be presented shortly. Again similar to [1], if there is a sequence of nodes $i_1, i_2, \ldots i_m$ such that i_{r+1} is the son (real or adopted, for destination j) of i_r for $r = 1,2,\ldots (m-1)$, then we say that i_1 is *upstream* from i_m (for destination j) and i_m is *downstream* from i_1. The network is said to be *loop-free* if there are no two nodes that are each upstream from each other, and is said to

have *loops* otherwise. If the network is loop-free for a given destination, then the downstream relationship forms a partial ordering of the nodes in the network.

A step of the algorithm will proceed such that the updating of λ's propagates from the destination upstream using the above mentioned partial ordering and the rerouting proper will propagate downstream from the peripheries towards the destination. We see therefore that maintaining loop-freedom in the network at each step of the algorithm is not only saving resources, but is also essential to provide a natural sequencing in the network.

Before indicating the algorithm, it will be useful to discuss several special points connected with updating, loop-freedom and rerouting. The discussion will hopefully help in understanding the various parts of the algorithm. We are still referring to a given destination j. Regarding updating, in order to be sure to prevent loops, we shall need the concept of *blocking* introduced in [1]. Briefly, if $f_{ik}(j) > 0$ and $\lambda_i(j) \leqslant \lambda_k(j)$, then there is danger of producing a loop in the next step. Therefore if, because of the constraints on the step-size, node i is not sure that in one step it can reroute all of $f_{ik}(j)$, it declares itself blocked and so do all nodes upstream from it. If a node k was not the son of a node i at stage n and node k is blocked, then k cannot become its son at stage $(n+1)$. The exact procedure and proof that blocking prevents looping appear in the algorithm and the subsequent theorems.

Another issue to be raised is connected with routing. As said before, since we are dealing with (virtual) line-switched networks, if a node decides to initiate the rerouting of a line, the entire portion of the old line from that node to the destination will have to be cancelled and a new line established. This procedure will be performed in a distributed fashion, but it requires that a node will do its own rerouting only after all of its fathers - and in fact all nodes upstream from it - have completed their rerouting. This is because a node must know what connections passing through it have been cancelled by nodes upstream, and only then it can adjust the remaining connections. In fact, at each stage, the routing procedure at each node will consist of three possible parts: cancel those outgoing lines corresponding to lines that were previously coming in, but have been cancelled by fathers, initiate rerouting, and finally establish outgoing lines corresponding to new incoming lines.

We are now ready to indicate the algorithm. It proceeds independently for each destination, so that we shall describe it for a given destination j. For the sake of clarity, we shall first describe an arbitrary updating-rerouting step and then show how to initialize the algorithm. A step of the algorithm is started by the destination that sends $\lambda_j(j) = 0$ to all its neighbors and it consists of taking the flows $f^n = \{f_{ik}^n(j)\}$ and obtaining a new flow $f^{n+1} = \{f_{ik}^{n+1}(j)\}$. Nodes that have at stage n any flow destined for j will proceed in a slightly different way than nodes that have none. The algorithm has a step-size η connected with it; this will be discussed later.

A. Algorithm for a node i with $f_{ik}^n(j) > 0$ for some k.

Updating

(i) Wait until receiving λ's from all sons. If any of the sons is blocked, node i declares itself blocked. Let $C_i^n(j)$ be the set (at node i) consisting of all sons and also those nodes that have sent their λ by now and are not blocked.

(ii) Let

$$\lambda_i^n(j) = \min_{k \in C_i^n(j)} \left[\lambda_k^n(j) + D'_{ik} \right] \tag{7}$$

(iii) If for any son k,

$$\lambda_k^n(j) \geqslant \lambda_i^n(j) \qquad \text{and} \tag{8a}$$

$$\eta \left[\lambda_k^n(j) + D'_{ik} - \lambda_i^n(j) \right] < f_{ik}^n(j) \tag{8b}$$

then node i declares itself blocked.

(iv) Send $\lambda_i^n(j)$ and a special tag indicating blocking status to all neighbors, *except sons*.

Routing

(v) Wait until λ's were received at node i from all neighbors, and let $B_i^n(j)$ be the set (at node i) of all neighbors k, except those that are both blocked and $f_{ik}^n(j) = 0$. For all neighbors k, let

$$a_{ik}^n(j) = \left[\lambda_k^n(j) + D'_{ik} \right] - \min_{m \in B_i^n(j)} \left[\lambda_m^n(j) + D'_{im} \right] \tag{9}$$

Let $k_o^n(i,j)$ be any neighbor that achieves the minimum in (9).

(vi) Cancel all outgoing lines corresponding to incoming pipes that have been cancelled by fathers. Let $f'^{n}_{ik}(j)$ be the remaining outgoing flows.

(vii) Let

$$\Delta^{n}_{ik}(j) = \min \left[f'^{n}_{ik}(j), \quad \eta a^{n}_{ik}(j) \right] \tag{10}$$

Reroute as follows

$$f''^{n}_{ik}(j) = \begin{cases} f'^{n}_{ik}(j) - \Delta^{n}_{ik}(j) & \text{for } k \neq k^{n}_{o}(i,j) \\ \\ f'^{n}_{ik}(j) + \sum\limits_{k \neq k^{n}_{o}} \Delta^{n}_{ik}(j) & \text{for } k = k^{n}_{o}(i,j). \end{cases} \tag{11}$$

(viii) Outgoing lines corresponding to new incoming lines are all opened on $(i, k^{n}_{o}(i,j))$, so that

$$f^{n+1}_{ik}(j) = \begin{cases} f''^{n}_{ik}(j) & k \neq k^{n}_{o}(i,j) \\ \\ f''_{ik}(j) + \text{any new flow} & k = k^{n}_{o}(i,j) \ . \end{cases} \tag{12}$$

(ix) The list $S^{n+1}_{i}(j)$ of sons will include $k^{n}_{o}(i,j)$ and all nodes k such that $f^{n+1}_{ik}(j) > 0$.

(x) Send $\lambda^{n}_{i}(j)$ and blocking status to all neighbors k for which $f^{n}_{ik}(j) > 0$.

Note that step (x) allows the rerouting to percolate down, from peripheries to the destination. Also observe that f^n does not uniquely define f^{n+1} both because the minimum in (9) may be achieved by more than one neighbor, and because the cancellation in (vi) depends on the paths of the individual lines and not only on f^n. However, we shall show in the next section that this is irrelevant insofar as the properties of the algorithm are concerned.

B. Algorithm for a node i with $f^n_{ik}(j) = 0$ for all k.

The algorithm insures that at each step after the initial one, each node has at least one son.

(i) Wait for λ from the adopted son. If the adopted son is blocked, node i declares itself blocked. Let $C^n_i(j)$ be the set (at node i) consisting of the adopted son and those nodes that have sent their λ by now and are not blocked.

(ii) Same as in A. (ii)

(iii) Send $\lambda^n_i(j)$ and a tag indicating blocking status to all neighbors except the adopted son.

(iv) Wait until λ's were received at node i from all neighbors and let $B^n_i(j)$ be the set consisting of the adopted son and all other non blocked neighbors. Let $k^n_o(i,j)$ be any node in $B^n_i(j)$ that

achieves

$$\min_{m \in B_i^n(j)} \left[\lambda_m^n(j) + D'_{im} \right] \quad . \tag{13}$$

(v) Send $\lambda_i^n(j)$ and blocked status to the adopted son, and let $S_i^{n+1}(j)$ consist of $k_o^n(i,j)$ (i.e., this is the new adopted son).

(vi) If any new lines are opened to node i, continue them over the link $(i,k_o^n(i,j))$. In this case, $k_o^n(i,j)$ becomes a real son.

We may note that although the concepts of real and adopted sons are useful for analysis and descriptional purposes, the nodes do not need to distinguish between them.

In order to complete the description of the algorithm, we need only to indicate how to initialize it. Observe that when the network starts operating, no blocking is present at the first stage and the list of sons is empty at each node.

C. Initialization

(i) Let $k_1(i,j)$ be the first node from which i receives a number λ. Let

$$\lambda_i^o(j) = \lambda_{k_1}^o(j) + D'_{ik_1} \quad . \tag{14}$$

(ii) Send $\lambda_i^o(j)$ to all neighbors except $k_1(i,j)$.

(iii) By the time λ's were received from all neighbors, let $B_i^o(j)$ be the set consisting of all neighbors. Let $k_o^o(i,j)$ be any neighbor that achieves

$$\min_{m \in B_i^o(j)} \left[\lambda_m^o(j) + D'_{im} \right] . \tag{15}$$

(iv) Send $\lambda_i^o(j)$ to $k_1(i,j)$ and let the list of sons $S_i^1(j)$ consist of $k_o^o(i,j)$.

(v) Same as B. (vi).

In words, since nodes have no lists of sons, the role of a son is played by the first node one hears from. This allows the nodes to establish numbers λ and later choose their adopted sons. The traffic is routed then to the adopted sons (at which stage they become real sons). Clearly this procedure may require flows that numerically exceed link capacity, but a good end to end flow control can regulate the inputs until enough routes to accommodate all requirements are found (clearly, such flow control is needed during the operation of the network as well).

IV PROPERTIES OF THE ALGORITHM

In this section we investigate the descent, convergence and loop-freedom properties of our routing algorithm. Specifically, we shall show that every step reduces the delay in the network, that the algorithm converges to the minimum delay routing and that at each step the network is loop-free. The latter also imples that, since the updating propagates upstream and the rerouting action propagates downstream, both operations are deadlock free, namely each node that is physically connected to the destination will update and reroute exactly once at each step.

As said before, another interesting property of the algorithm is the fact that at each stage n, a node that has received numbers λ from all neighbors knows that all updating and rerouting has been completed at nodes that were upstream from it at stage n (the only further action it may have to take after completing its own rerouting is to have to open new lines corresponding to new incoming lines). This property can be used by the destination to insure that it will not start a new updating before the previous one is completed[1].

We remind the reader that we are working throughout under the standing assumptions (5).

Definition A set of nodes $\ell_1, \ell_2, \ldots \ell_m, \ell_1$ form a *loop* if ℓ_{i+1} is a son of ℓ_i for $i = 1,2,\ldots (m-1)$ and if ℓ_1 is a son of ℓ_m.

Theorem 2 For arbitrary step size η, there are no loops in the

network at each step of the algorithm.

Proof. The proof is similar in spirit to the proof of [1, Thm.4] except that here we have to look separately at adopted and real sons and also, because of the special rerouting sequence, the intermediate flow values f' defined in A. (vi) in the Algorithm enter explicitly in the calculations.

We first show that if there are no loops at stage $n \geq 1$, then the network has no loops at stage $(n+1)$. Suppose there is a loop at stage $(n+1)$ for destination j. Then it must contain a link (ℓ,m) such that $\lambda_\ell^n(j) \leq \lambda_m^n(j)$. But m cannot be the adopted son of ℓ, since A. (ix) and B. (v) in the algorithm and the assumption (5d) shows that this would require $\lambda_\ell^n(j) > \lambda_m^n(j)$. Therefore m is a real son of ℓ at stage (n+1), i.e. $f_{\ell m}^{n+1}(j) > 0$. Since the algorithm increases flow only on links corresponding to neighbors with lower numbers, we also have $f_{\ell m}^n(j) > 0$.

This implies that :

$$\eta a_\ell^n(j) < f'^n_{\ell m}(j) \leq f_{\ell m}^n(j) \ . \tag{16}$$

But on the other hand from (13) and (11) we have

$$\eta a_{\ell m}^n(j) \geq \eta \left[\lambda_m^n(j) + D'_{\ell m} - \lambda_\ell^n(j)\right] \tag{17}$$

so that

$$\eta \left[\lambda_m^n(j) + D'_{\ell m} - \lambda_\ell^n(j)\right] < f_{\ell m}^n(j) \tag{18}$$

and therefore ℓ is blocked at stage n.

Now move backwards around the assumed loop at stage (n+1) to the *first* link (i,k) such that k is not a son of i for stage n. There must be such a link if the network was loop-free at stage n. But since ℓ is blocked at stage n, so is k, since on the portion from k to ℓ, of the assumed loop, each node is at stage n a son of the previous node. But this says that k was not a son of i at stage n and became one at stage (n+1), although it was blocked at stage n. The algorithm does not allow this to happen and we therefore have a contradiction.

The proof is completed now if we observe that at stage 0 we have only adopted sons, so that there cannot be loops at stage 1.

Theorem 3 Let $D_1 = D_T(f^1)$ and let M be an upper bound to all $D''_{ik}(f_{ik})$ over the set

$$F = \{f | D_T(f) \leqslant D_1\}. \tag{19}$$

If $\eta = (2MN^5)^{-1}$, any rerouting strictly decreases the total delay in the network and

$$D_T(f^{n+1}) \leqslant D_T(f^n) - (MN^5) \sum_{i,k,j} \left[\Delta^n_{ik}(j)\right]^2 . \tag{20}$$

The proof appears in the Appendix.

Note The asserted bound exists because the set (19) is compact and D'' is continuous.

The next theorem is interesting in its own and will also be used to prove convergence properties of the algorithm. It shows that if any flow is such that the algorithm requires no flow changes, then that particular flow has no blocking. In particular, we shall show later (see Appendix) that, as expected, no reroutings are produced if a flow is optimal, i.e. satisfies (6), so that no blocking is present in optimal flows. On the other hand it shows intuitively that blocking will not occur very often near optimality. This statement is addressed more precisely in the proof of Lemma 3 in the Appendix.

Theorem 4 If any blocking is present in an updating stage, then routing changes definitely occur in the network in that stage.

Proof. Let i be any node that declared itself blocked for a destination j in A. (iii) (rather than A. (i) or B. (i)) and let k be the son that caused the blocking. Suppose this happened during updating stage n. Then $f_{ik}^n(j) > 0$ from (8). Also $a_{ik}^n(j) > 0$ since

$$\lambda_k^n(j) + D'_{ik}(f_{ik}^n) > \lambda_k^n(j) \geqslant \lambda_i^n(j) \geqslant \min_{m \in B_i^n(j)} \left[\lambda_m^j + D'_{im}(f_{im})\right] \qquad (21)$$

(The last inequality follows from (7) and the fact that $B_i^n(j) \supset C_i^n(j)$.) Now if $f'^n_{ik}(j) \neq f_{ik}^n(j)$, then clearly there was a routing change upstream of i. If $f'^n_{ik}(j) = f_{ik}^n(j)$, then

$$\Delta_{ik}^n(j) = \min \left[f_{ik}^n(j), \ \eta a_{ik}^n(j)\right] > 0 \qquad (22)$$

so that a routing change occurs at i.

We are finally able to state the theorem describing the convergence properties of the algorithm.

Theorem 5 Let D_{min} be the value of the minimum delay in the network and let F_{min} be the set of optimal flows

$$F_{min} = \{f | D_T(f) = D_{min}\} \quad . \tag{23}$$

Let $d(f,F)$ be the Euclidean distance between a vector of flows f and a set F, defined by

$$d^2(f,F) = \min_{f^o \in F} \sum_{i,k,j} (f_{ik}(j) - f^o_{ik}(j))^2 \quad . \tag{24}$$

Then for each initial flow f^1, and for step size as in Theorem 3, we have as $n \to \infty$:

$$\text{(a)} \quad D_T(f^n) \searrow D_{min} \quad . \tag{25}$$

$$\text{(b)} \quad d(f^n, F_{min}) \to 0. \tag{26}$$

This implies that any limit point f^* of f^n is an optimal flow. It also implies that if $D_{ik}(\cdot)$ are all strictly convex, then F_{min} consists of a single point f^* and $f^n \to f^*$.

The proof of the theorem appears in the Appendix.

V. DISCUSSION

In Section I we have indicated the main differences between the algorithms for message (packet) switching (MS) of [1] and for virtual line-switching (LS). After indicating the algorithm and its properties, we are now in a better position to elaborate on these differences and on other points regarding the algorithms.

As already said, LS requires a particular sequencing for re-routing, from nodes with no fathers, using the downstream relationship for sequencing, down to the root (destination). This provides a natural update-rerouting cycle, which, although using distributed computation, allows the destination to know exactly the time of completion of each cycle, and therefore makes it possible that no two cycles will simultaneously run in the network. Clearly, it requires only a simple change in the algorithm of [1] to obtain a similar property for MS^1, but in the context of MS networks with fixed topology this property is not really essential. It becomes indispensable however when topological changes have to be taken into account (see [14] - [16]).

The above mentioned ordering through the network also requires that each node with real sons will use a careful sequencing for its routing, namely first cancel, then reroute and then continue new incoming lines. The concept of adopted son is necessary for nodes with no flow to the destination, to designate the *"best"* neighbor to which it would route traffic when it comes in. Also observe that in the LS algorithm, the number λ is defined by a minimizing operation (7), while

in the MS algorithm it is calculated by a weighted average [1, Eq. (5)]. While both are natural quantities for the corresponding algorithms, other possibilities probably exist.

Regarding implementation of the LS algorithm, its use for quasi-static routing should now be clear. Immediately or an arbitrary interval of time after its previous update-rerouting cycle was completed, each destination starts a new update that propagates upstream through the network; then the rerouting propagates downstream. The nodes estimate the incremental delay D'_{ik} and use them in the update. The destination node knows that the cycle is completed as soon as it receives numbers λ from all neighbors. Old connections that are terminated are cancelled together with the rerouting, while the algorithm propagates downstream. New connections can be established at any time, but it may be preferable to wait for the next rerouting stage. Observe that the algorithm creates situations in which the new circuit is only partially established during rerouting, while the old circuit is partially or totally destroyed, with no physical circuit connecting the source to the destination. This however does not mean that the call must be suspended during this time. When a node initiates the rerouting of a line, it stops sending data on the old line and at the same time it can start sending it over the new one. This of course assumes that the protocols for sending data over each link and for establishing new lines are such that either both are processed on a common first-come-first-served basis or such that establishing lines takes priority. Since in a network with fixed topology every newly initiated line will indeed reach the destination, there is no need to

hold the data until the end-to-end protocol for establishing a line is completed. Clearly, in actual implementation, more thought is necessary to decide the final protocols best suited for the particular application.

Another issue regarding implementation is the step-size η, and this is common to both line-switching and message switching. Our assumption just before Theorem 1 means that the atomic size of flow is much smaller than $\eta \cdot D'_{ik}$. On the other hand, η is taken in Theorem 3 to be very small, in order to prove convergence. It is of course important to know that a certain η insures convergence to the optimal routing, but practically speaking this may not be the best step size. First, because much larger η's will probably still insure convergence, while also allowing enough routing dynamics to follow slowly changing traffic requirements and second, even if convergence to the exact optimum does not occur, still we may be able to provide bounds on the performance. In fact, an interesting future research topic is to obtain such bounds for a given step size η. Other important future research topics are to study the dynamics of the networks using the algorithms of [1] and of this paper, as well as the stochastic behavior due to stochastic requirements and errors in the estimation of D'_{ik}. Finally, we may mention that although the algorithms have been shown to be deadlock-free when the topology is fixed, they are clearly not suitable in their present form to acommodate failures and recoveries of links and nodes. Using those algorithms in a simpler form as a basis,

a distributed update algorithm has been developed in [14] having the properties of i) loop freedom, and ii) deadlock freedom and recovery in finite time under arbitrary sequence, location and quantity of topological changes. The proofs of these properties appear in [15] . More recently, the algorithms of [1] and this paper have been completed to acommodate topological changes, and the resulting algorithms [16] provide optimal routing, while also retaining properties i) and ii) above.

APPENDIX

Proof of Theorem 3 First recall that when a node i decides to reroute a line passing through it and going to destination j, then the entire portion of the line from i to the destination will be rerouted. We call such rerouting of a single line an *elementary rerouting*. Now using assumptions (5), the change in delay from stage n to stage $(n+1)$ is :

$$D_T(f^{n+1}) - D_T(f^n) = \sum_{i,k} D'_{ik}(f^n_{ik})\left[f^{n+1}_{ik} - f^n_{ik}\right] + \sum_{i,k} D''_{ik}(f^\lambda_{ik})\left[f^{n+1}_{ik} - f^n_{ik}\right]^2 \qquad (27)$$

where f^λ is some point on the segment connecting f^n and f^{n+1}.

Lemma 1

$$\sum_{i,k} D'_{ik}(f^n_{ik})\left[f^{n+1}_{ik} - f^n_{ik}\right] \leqslant -\eta^{-1} \sum_{i,k,j}\left[\Delta^n_{ik}(j)\right]^2 \qquad (28)$$

Proof. Consider first an elementary rerouting of a line through which the flow is Δ, transferred from a path $(i,k,\ell_1,\ldots\ell_m,j)$ to a path $(i, k^n_o(i,j), r_1,\ldots r_p,j)$. Its contribution to the left hand side of (28) will be

$$\Delta\cdot\{- D'_{ik} + D'_{ik_o} - (D'_{k\ell_1} + D'_{\ell_1\ell_2} + \ldots + D'_{\ell_m j}) + (D'_{k_o r_1} + D'_{r_1 r_2} + \ldots + D'_{r_p j})\} \qquad (29)$$

Now observe that r_1 is the best node out of k_o in the sense that it achieves the minimization in (9) for k_o, node r_2 is the best node out of r_1 and so on down the path, so that using the fact that $C^n_i(j) \subset B^n_i(j)$ for any node i, we have :

$$\lambda^n_{r_p}(j) \geqslant \lambda^n_j(j) + D'_{r_p j} = D'_{r_p j}$$

$$\lambda^n_{r_{p-1}}(j) \geqslant \lambda^n_{r_p}(j) + D'_{r_{p-1} r_p} \geqslant D'_{r_{p-1} r_p} + D'_{r_p j}$$

and by induction

$$\lambda^n_{k_o}(j) \geqslant D'_{k_o r_1} + D'_{r_1 r_2} + \ldots + D'_{r_p j} \tag{30}$$

On the other hand, before the rerouting, the line in question passed through nodes $k_1, \ell_1, \ell_2, \ldots \ell_m, j$, so that each was a son of the previous one and hence was included in its set $C^n(j)$ appearing in (7). Therefore,

$$\lambda^n_{\ell_m}(j) \leqslant \lambda^n_j(j) + D'_{\ell_m j} = D'_{\ell_m j}$$

$$\lambda^n_{\ell_{m-1}}(j) \leqslant \lambda^n_{\ell_m}(j) + D'_{\ell_{m-1} \ell_m} \leqslant D'_{\ell_{m-1} \ell_m} + D'_{\ell_m j}$$

and by induction

$$\lambda^j_k \leqslant D'_{k \ell_1} + D'_{\ell_1 \ell_2} + \ldots + D'_{\ell_m j} \quad . \tag{31}$$

Equations (31) and (30) imply that expression (29) is

$$\leqslant \Delta \cdot \{ - D'_{ik} + D'_{ik_o} + \lambda^n_{k_o}(j) - \lambda^n_k(j) \} = -a^n_{ik}(j) \cdot \Delta \tag{32}$$

Since at step n a total flow of $\Delta^n_{ik}(j)$ was rerouted from paths starting

with link (i,k), their contribution to the left hand side of (28) is

$$\leqslant - a_{ik}^{n}(j) \cdot \Delta_{ik}^{n}(j) \leqslant - \frac{1}{\eta} \left[\Delta_{ik}^{n}(j)\right]^2 \quad . \tag{33}$$

Summing up over i,k,j gives (28).

Lemma 2 Let M be an upper bound on $D''_{ik}(f_{ik}^{\lambda})$ over all i,k, when f^{λ} ranges over the segment connecting f^{n} and f^{n+1}. Then

$$\sum_{i,k} D''_{ik}(f_{ik}^{\lambda})(f_{ik}^{n+1} - f_{ik}^{n})^2 \leqslant (MN^5) \sum_{i,k,j} \left[\Delta_{ik}^{n}(j)\right]^2 \tag{34}$$

Proof. We have

$$\Sigma\, D''_{ik}(f_{ik}^{\lambda})(f_{ik}^{n+1} - f_{ik}^{n})^2 \leqslant M \sum_{i,k} (f_{ik}^{n+1} - f_{ik}^{n})^2 \tag{35}$$

and since the largest change in flow in any given link cannot be larger than the sum of all changes occuring in the network

$$|f_{ik}^{n+1} - f_{ik}^{n}| \leqslant \sum_{\ell,m,j} \Delta_{\ell m}^{n}(j) \quad . \tag{36}$$

Now the sum in (36) has no more than N(N-1)(N-2) terms, so that applying the Cauchy-Schwartz inequality gives

$$(f_{ik}^{n+1} - f_{ik}^{n})^2 \leqslant N^3 \cdot \sum_{\ell,m,j} \left[\Delta_{\ell m}^{n}(j)\right]^2 \quad . \tag{37}$$

Summing over (i,k) and using (35) gives (34).

Now, combining the results of the lemmas 1,2 we have for M as in lemma 2

$$D_T(f^{n+1}) - D_T(f^{n}) \leqslant \left(- \frac{1}{\eta} + MN^5 \right) \cdot \Sigma \left[\Delta_{ik}^{n}(j)\right]^2 \quad . \tag{38}$$

If we choose $\eta = (2MN^5)^{-1}$, then $D_T(f^{n+1}) < D_T(f^n)$ and by induction $D_T(f^{n+1}) < D_T(f^n) < D_T(f^1)$ so that both f^n and f^{n+1}, as well as the entire segment connecting them is in the set F of (19). Consequently M can be taken as in the statement of Theorem 3. Also the above choice of η gives (20) which completes the proof of the theorem.

Proof of Theorem 5 Since $D_T(f^n)$ is bounded from below by zero and is a decreasing sequence, it has a limit D^*. Since $\{f^n\}$ belong to the compact set F of (19), it has a limit point f^* (i.e. a subsequence converging to f^*) and continuity of D_T implies that

$$D_T(f^n) \searrow D_T(f^*) = D^* \quad . \tag{39}$$

For simplicity, we denote this subsequence also by $\{f^n\}$.

We shall now continue the proof with a series of lemmas: Lemmas 3 and 4 show that f^* is an optimal flow, namely it satisfies the Kuhn-Tucker conditions. This implies that $D^* = D_{min}$, which proves part a) of the Theorem. Part b) will be covered in Lemma 5.

Lemma 3. No routing changes occur when applying the algorithm to f^*.

Proof. Suppose the contrary. In any flow pattern f^* that has routing changes (for destination j) there must exist a node i such that $\Delta^*_{ik}(j) > 0$ and $f'^*_{ik}(j) = f^*_{ik}(j)$ for some k. (If $f'^*_{ik}(j) \neq f^*_{ik}(j)$ at a node i with $\Delta^*_{ik}(j) > 0$ for some k, keep moving upstream in all paths until you find a node such that none of its fathers has a routing change or until a node i with $\Delta^*_{ik}(j) > 0$ for some k with no fathers). The triplet i,k,j will remain fixed for the rest of the proof of Lemma 3.

Now observe that all λ's as generated by the algorithm are positive and bounded by

$$\max_{f\in F} \sum_{\ell,m} D'_{\ell m}(f_{\ell m}) < \infty \tag{40}$$

so that both $\{f^n\}$ and $\{\lambda^n\}$ belong to compact sets. Also observe that the sets $C_i^n(j)$, $B_i^n(j)$ as generated by the algorithm are drawn from a finite set (the power of the set of all neighbors of i). Therefore, there must exist a subsequence, which we again denote by $\{n\}$ for simplicity, such that $n\to\infty$, $f^n \to f^*$, λ^n converges to some $\bar{\lambda}$, and the sets $C_i^n(j)$ and $B_i^n(j)$ are nonvarying along the mentioned subsequence and are identical to the corresponding sets generated by the algorithm when applied to f^*. Let us denote them by $C_i^*(j)$ and $B_i^*(j)$ respectively, and then we have :

$$\lambda_i^n(j) = \min_{\ell\in C_i^*(j)} (\lambda_\ell^n(j) + D'_{i\ell}(f_{i\ell}^n)) \tag{41}$$

$$a_{ik}^n(j) = \lambda_k^n(j) + D'_{ik}(f_{ik}^n) - \min_{\ell\in B_i^*(j)} (\lambda_\ell^n(j) + D'_{i\ell}(f_{i\ell}^n)) \tag{42}$$

$$\Delta_{ik}^n(j) = \min \left[\eta a_{ik}^n(j),\ f_{ik}^n(j)\right] \tag{43}$$

implying that

$$a_{ik}^n(j) \to a_{ik}^*(j) \tag{44}$$

$$\Delta_{ik}^n(j) \to \Delta_{ik}^*(j) \quad , \tag{45}$$

where $a_{ik}^*(j)$, $\Delta_{ik}^*(j)$ are generated by applying the algorithm to f^*, namely from (41)-(43) with f^* replacing f.

Now let f^{**} be the flow obtained after applying the algorithm to f^{*}. From Theorem 3 follows that

$$D_T(f^{**}) - D_T(f^{*}) \leq -(MN^5)\left[\Delta^{*}_{ik}(j)\right]^2 \tag{46a}$$

and for any n along the subsequence in consideration

$$D_T(f^{n+1}) - D_T(f^{n}) \leq -(MN^5)\left[\Delta^{n}_{ik}(j)\right]^2 . \tag{46b}$$

But (45) and the fact that $\Delta^{*}_{ik}(j)$ is assumed to be strictly positive (at the beginning of this proof), imply that there exists an N_1 such that, for $n > N_1$ we have

$$D_T(f^{n+1}) - D_T(f^{n}) \leq -(MN^5)\left[\Delta^{*}_{ik}(j)\right]^2/2 . \tag{47}$$

Moreover since $D_T(f^{n}) \searrow D_T(f^{*})$, we can choose N_2 such that for $n > N_2$ we have

$$D_T(f^{n}) - D_T(f^{*}) < (MN^5)\left[\Delta^{*}_{ik}(j)\right]^2/2 . \tag{48}$$

But (47) and (48) imply that for $n > \max(N_1,N_2)$ we have

$$D_T(f^{n+1}) < D_T(f^{*}) \tag{49}$$

which contradicts (39).

Note. Lemma 3 implies that if the algorithm is applied to f^{*} an arbitrary number of times, still f^{*} remains unchanged. Observe however that the Lagrange multiplies $\bar{\lambda}$ are not necessarily identical to λ^{*} of Theorem 1. The next lemma settles this question.

Lemma 4. The flow f^* satisfies the Kuhn-Tucker conditions (6) for some Lagrange multipliers. After application of the algorithm to f^* at most N times, the λ's generated by the algorithm which we call λ^*, are such that f^*, λ^* satisfy the Kuhn-Tucker conditions (6).

Proof. From Lemma 3 and Theorem 4 follows that no blocking occurs when the algorithm is applied to f^*. This implies that for every node i, the set $B_i^*(j)$ as generated by the algorithm applied to f^* is the set of all neighbors of i. In addition $f_{ik}^{*'}(j) = f_{ik}^*(j)$, so that

$$0 = \Delta_{ik}^*(j) = \min \left[f_{ik}^*(j),\ na_{ik}^*(j) \right] \quad . \tag{50}$$

Now, if a node i has nonzero flow destined for j, then all its neighbors k such that $f_{ik}^*(j) > 0$ have $a_{ik}^*(j) = 0$. This implies that for such neighbors k

$$\bar{\lambda}_k(j) + D'_{ik}(f_{ik}^*) = \min \left[\bar{\lambda}_m(j) + D'_{im}(f_{im}^*) \right] = \bar{\lambda}_i(j) \tag{51}$$

where $\bar{\lambda}$ are the λ's generated by the algorithm applied of f^* and the minimum is taken over *all* neighbors of i. In words, the first equality in (51) says that all neighbors k such that $f_{ik}^*(j) > 0$ have the same $\lambda_k + D'_{ik}$ and this number is less than or equal to this sum computed for neighbors with $f_{ik}^*(j) = 0$. The second equality in (51) follows if we observe that the set $C_i^*(j)$ of (7) contains at least one neighbor with $f_{ik}^*(j) > 0$. Therefore, for nodes i with nonzero flow, f^* satisfies the Kuhn-Tucker conditions with $\lambda^* = \bar{\lambda}$. It is also easy to see that at these nodes not only f^*, but also $\bar{\lambda}$ do not change, no matter how many times the algorithm is applied to f^*.

The situation is different at nodes i such that $f^*_{ik}(j) = 0$ for all k. There the λ's may still change, but the following shows that after at most N steps they will remain fixed and will satisfy the Kuhn-Tucker conditions. Consider a group of connected nodes with no flow to destination j. Consider the link with minimum $\Delta^*_k(j) + D'_{ik}(f^*_{ik})$, where the minimum is calculated over all links such that i is in this group and k is outside. Then in the next iteration i will choose k as its adopted son and in the iteration after that it will definitely define $\lambda_i(j)$ to be

$$\lambda^*_i(j) = \lambda^*_k(j) + D'_{ik}(f^*_{ik}) \ . \tag{52}$$

It will not change its decisions for all coming iterations. Clearly $\lambda^*_i(j)$ satisfies the Kuhn-Tucker conditions. Continuing in a similar manner for all nodes with no flow to j, we obtain that in not more than N iterations, all nodes will arrive to λ's that satisfy the Kuhn-Tucker conditions.

Lemmas 3 and 4 show that f^* is an optimal flow, so that $D_T(f^*) = D_{min}$. This completes the proof of part a) of Theorem 5 because of (39) and the sentences preceding it.

Lemma 5. The sequence of flows $\{f^n\}$ obtained by repetitive application of the algorithm converges to the set of minimum flows in the sense of (26).

Proof. We first note here that f^n denotes again the original sequence (and not all subsequences considered in Lemmas 3 and 4). Now suppose (26) is not true. Then there is an $\varepsilon > 0$ and a subsequence $\{f^{n_m}\}$ with

$n_m \to \infty$ such that

$$d(f^{n_m}, F_{min}) > \varepsilon \ . \tag{53}$$

Since $\{f^{n_m}\}$ belongs to a compact set, it has a converging subsequence, $\{f^{n_{m_k}}\}$ such that $n_{m_k} \to \infty$, and let $\hat{f}$ be the limit. Then (57) shows that

$$d(\hat{f}, F_{min}) \geqslant \varepsilon \ , \tag{54}$$

which implies that $D_T(\hat{f}) \neq D_{min}$. But $f^{n_{m_k}} \to \hat{f}$ and continuity of D_T imply

$$D_T(f^{n_{m_k}}) \to D_T(\hat{f}) \neq D_{min} \tag{55}$$

which contradicts part a) of the theorem.

ACKNOWLEDGEMENT

The author would like to thank R.G. Gallager for interesting discussions related to the first stages of this research and F.H. Moss for comments related to its later stages.

REFERENCES

[1] R.G. Gallager: "A minimum delay routing algorithm using distributed computation", IEEE Trans. on Comm., Vol. COM-25, pp. 73-85, January 1977.

[2] A. Segall: "The modelling of adaptive routing in data-communication networks", IEEE Trans. on Comm., Vol. COM-25, pp. 85-95, January 1977.

[3] J.M. McQuillan: Adaptive routing algorithms for distributed networks, Bolt, Beranek & Newman, Inc., Rep.2831, May 1974.

[4] H. Frank & W. Chou: "Routing in computer networks," Networks, Vol.1, pp. 99-122, 1971.

[5] G.L. Fultz: Adaptive routing techniques for message switching computer-communication networks, Report UCLA-ENG-7252, July 1972.

[6] D.G. Cantor & M. Gerla: "Optimal routing in a packet-switched computer network", IEEE Trans. Comput., Vol. C-23, pp. 1062-1069, October 1974.

[7] D.G. Luenberger: Introduction to linear and nonlinear programming, Addison Wesley, 1973.

[8] G.P. McCormick: "Anti-zig-zagging by bending", Management Science (Theory), Vol. 15, pp. 315-320, January 1969.

[9] M. Schwartz, R.R. Boorstyn and R.L. Pickholtz: "Terminal-oriented computer-communication networks", Proc. IEEE, Vol. 60, pp. 1408-1423, November 1972.

[10] G.D. Forney & J.E. Vander Mey: "The Codex 6000 series of Intelligent Network Processors", Computer Communication Review, Vol. 6 pp. 7-13, April 1976.

[11] L.Kleinrock: Communication Nets: Stochastic message flow and delay, McGraw Hill, 1964.

[12] L. Kleinrock: "Analytic and simulation methods in computer network design," in 1970 Spring Joint Computer Conference, AFIPS Conf. Proc., Vol.36, AFIPS Press, pp. 569-579, 1970.

[13] P.P. Varaiya: Notes on Optimization, Van Nostrand Reinhold, 1972.

[14] A. Segall, P.M. Merlin & R.G. Gallager: "A recoverable protocol for loop-free distributed routing", submitted to ICC 78, Toronto, Canada, June 1978.

[15] P.M. Merlin & A. Segall: "A failsafe algorithm for loop-free distributed routing in data-communication networks", submitted to IEEE Trans. on Comm.

[16] M. Sidi & A. Segall:"Optimal failsafe distributed routing in data networks", in preparation.

FOOTNOTES

1. Algorithms having this property were first indicated to the author by R.G. Gallager .