Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology

ELI M. GAFNI, STUDENT MEMBER, IEEE, AND DIMITRI P. BERTSEKAS, SENIOR MEMBER, IEEE

Abstract—**We consider the problem of maintaining communication between the nodes of a data network and a central station in the presence of frequent topological changes as, for example, in mobile packet radio networks. We argue that flooding schemes have significant drawbacks for such networks, and propose a general class of distributed algorithms for establishing new loop-free routes to the station for any node left without a route due to changes in the network topology. By virtue of built-in redundancy, the algorithms are typically activated very infrequently and, even when they are, they do not involve any communication within the portion of the network that has not been materially affected by a topological change.**

I. INTRODUCTION

THERE has been considerable interest recently in mobile packet radio (PR) networks (see, e.g., [1]). In such networks, it is often necessary to use intermediate PR's as repeaters in order to transfer a message from a source to its destination. This gives rise to the usual routing problem encountered in wire packet switched networks. Moreover, when the PR's are mobile, the limited broadcast range, multipath interference, changes in shielding factors, etc., induce rapid topological changes. Thus, communication links frequently go down while other links are established. In such an environment, it is a formidable task for a routing algorithm to keep communication going not to mention trying to optimize its transfer.

Most of the routing schemes considered for PR networks involve the use of a central station that collects information regarding network connectivity and sets up routes from PR's to itself. One possibility is to determine routes on the basis of a shortest path algorithm. We refer to such routes established by the station as the *primary routes* and for the sake of the following discussion we assume that each PR has a single primary route to the station which may be altered periodically.

A centralized routing algorithm of the type just described must by necessity deal with topological changes through the station. By this we mean that the station must be informed of the topological changes that have occured in order to maintain up-to-date connectivity information which is in turn the basis for altering primary routes when necessary. One possibility that comes to mind is to require that each PR that becomes aware of a topological change should immediately send this information to the station which in turn will take appropriate action if necessary.

One difficulty with a scheme of this type, when topological changes are frequent, is that a great deal of information is passed on to the station. Thus, the station may find itself swamped with messages even though many of these messages do not convey truly important information since many of the topological changes can be only temporary, lasting, for example, for only a few seconds. A possible remedy would be for a PR to take a time out before reporting a topological change. This will result in a reduction of topological change messages flowing to the station at the expense of making decisions based on information that may be incomplete.

A second and more irritating difficulty results from the fact that the primary routes over which a topological change is to be reported may have been themselves affected by the topological change. Thus, a PR that has lost its primary route to the station due to a topological change must find an alternate route to report this fact to the station. In wire networks, there is a failsafe method for doing this, namely by using a flooding scheme whereby the topological change message is broadcast to all neighbors who in turn broadcast it to all their neighbors and so on until the message reaches the station. For PR networks, however, a flooding scheme is quite unsuitable. The reason is that it triggers nearly simultaneous bursts of broadcast messages throughout the network. This results in collisions of the type arising when two messages arrive at a PR, simultaneously. Collisions necessitate retransmissions, more collisions occur and the network can be driven to instability and ultimate collapse.

We thus arrive at the conclusion that a scheme based on immediate reports of topological changes to the station coupled with some type of flooding scheme when a primary route fails is dubious for PR networks with frequent topological changes. It thus appears to us that it is necessary to have, in addition to the primary routing algorithm operated by the station, a *contingency routing algorithm* to cope effectively with topological changes affecting primary routes.

Desirable properties for such an algorithm are as follows.

1) It should provide some redundancy in the form of additional routes to the station which can be used when the primary route fails. This has the effect of drastically reducing the frequency with which any particular PR will lose all its routes to the station.

2) It should not rely on instructions from the station in establishing new routes when all the existing routes of any PR fail.

Paper approved by the Editor for Computer Communication of the IEEE Communications Society for publication without oral presentation. Manuscript received October 11, 1979; revised May 5, 1980. This work was supported by Grant ONR-N00014-75-C-1183.

The authors are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139.

0090-6778/81/0100-0011$ 00.75

3) It should not employ flooding or otherwise create serious problems due to collisions.

4) It must ensure that each route is loop-free at all times.

5) It must be capable of incorporating awakened links into existing routes with little communication overhead.

The purpose of this paper is to propose and investigate some algorithms that can form the basis for developing contingency routing algorithms with properties 1)-5) above. In the next section, we introduce a graph theoretic problem closely related to the contingency routing problem. In Section III, we provide two example algorithms for solving this problem. In Section IV, these algorithms are embedded within a general class of algorithms based on a generalized numbering system. We conclude the paper with a discussion of some implementation aspects.

II. A PROBLEM ON ACYCLIC DIRECTED GRAPHS

Following standard terminology, we say that a directed graph is acyclic if it contains no directed cycle. By this we mean that every link of the graph is assigned a direction and it is impossible to find a nontrivial directed path that originates and terminates at the same node. Assume that we are given an acyclic directed graph (ADG for short) with a special node which is referred to as the *destination*. We say that the ADG is *destination oriented* if for every node there exists a directed path originating at this node and terminating at the destination. Otherwise, we say that the ADG is *destination disoriented*. It is easy to see that a connected ADG is destination disoriented if and only if there exists a node other than the destination that has no outgoing link, i.e., it is not the head node of any link.

We consider the following problem (*P*): given a connected destination disoriented ADG transform it to a destination oriented ADG by reversing the directions of some of its links.

Problem (*P*) is closely related to the contingency routing problem described in the previous section. The destination can be associated with the station of a PR network while other nodes can be associated with PR's. Any communication link between any pair of PR's has a direction associated with it and can be used for sending messages to the station in this direction only. If the resulting directed graph has no directed cycles and it is destination oriented, then every link will be part of a route leading to the station when used in the assigned direction. This provides redundancy in that for each PR there may be several downstream neighbors along which there is a path leading to the station. If communication with one of these neighbors is disrupted (for example, the one on a primary route) then the PR can communicate with the station through one of its other downstream neighbors (i.e., through one of its secondary routes). It is possible, however, that due to topological changes a PR will be left without a downstream neighbor. This means that the corresponding ADG has become destination disoriented. The problem that we are then faced with is to reverse the direction of communication along several links so as to reestablish a directed path to the station for each PR. This is in fact problem (*P*).

Naturally, there is a large number of algorithms for solving problem (*P*). For example, one can assign positive weights on all links and use a shortest path algorithm. We are, however, interested in algorithms that not only solve problem (*P*), but also have other desirable properties in connection with the contingency routing problem.

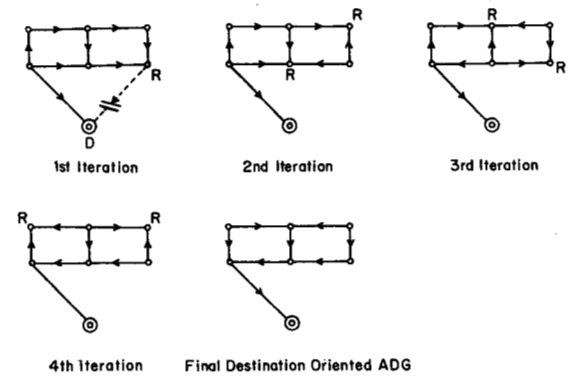

Fig. 1. Full reversal method.

Communication between nodes in a data network is subject to strict unambiguous rules (protocol) and the implementation of our algorithms in the context of a communication network will have to incorporate these rules. Since these rules are not yet universal and change from network to network, we have decided to detach the statement and analysis of the algorithms that follow from this context and use the more abstract graph theoretic framework.

III. TWO ALGORITHMS FOR SOLVING PROBLEM (*P*)

The two algorithms for problem (*P*) that follow will be stated loosely and described pictorially as they are executed on a linear machine. The proof of all our claims about these algorithms will be given in Section IV in the context of a more general algorithm.

Full Reversal Method: At each iteration each node other than the destination that has no outgoing link reverses the directions of all its incoming links.

Fig. 1 provides an example of the sequence of successive iterations of this algorithm. The nodes that reverse at each iteration are marked by *R*. The algorithm provides a sequence of ADG's and terminates when a destination oriented ADG is obtained.

Partial Reversal Method: Every node i other than the destination keeps a list of its neighboring nodes j that have reversed the direction of the corresponding links (i, j). At each iteration each node i that has no outgoing link reverses the directions of the links (i, j) for all j which do not appear on its list, and empties the list. If not such j exists (i.e., the list is full), node i reverses to the directions of all incoming links and empties the list.

Fig. 2 provides an example of the sequence of successive iterations of this algorithm (starting with empty lists). The examples of Figs. 1 and 2 are extreme in that they require a large number of reversals. The reader can convince himself through other examples that in most networks (particularly with relatively high connectivity) the reversal process will be initiated infrequently and will typically not require a long chain of iterations.

We claim the following for both algorithms.

1) If the graph is connected then the reversal process will

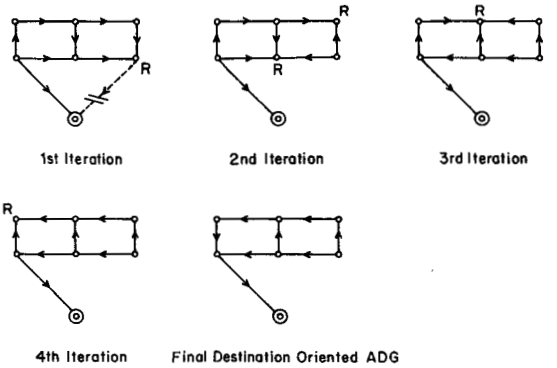

Fig. 2. Partial reversal method.

terminate after a finite number of iterations at a destination oriented ADG.

2) The directed graph generated at each iteration is acyclic.

3) The direction of any link between two nodes that have a direct path to the destination in the initial ADG will never be reversed.

We now provide an alternative statement of both algorithms which, within the context of the contingency routing problem, allows the addition of new directed links in an ADG without forming a cycle.

Full Reversal Method: At each stage of the algorithm, we associate with every node i a pair (α_i, i) where i is the unique identification number of the node and α_i is an integer. The set of pairs $\{(\alpha_i, i)\}$ is ordered lexicographically, i.e., $(\alpha_i, i) > (\alpha_j, j)$ if $\alpha_i > \alpha_j$ or if $\alpha_i = \alpha_j$ and $i > j$. Let N be the set of nodes and consider a set of integers $\{\alpha_i^0 \mid i \in N\}$ such that in the initial ADG in problem (P) the direction of any link (i, j) goes from i to j if and only if $(\alpha_i^0, i) > (\alpha_j^0, j)$. Such a set can be shown to exist (see [2, p. 29]). The kth iteration of the full reversal method $k = 0, 1, \cdots$, can be implemented as follows. At the beginning of the iteration each node i has an integer α_i^k associated with it. A node i other than the destination has no outgoing link if for every neighbor j of i we have $(\alpha_i^k, i) < (\alpha_j^k, j)$. At the kth iteration such a node i increases α_i^k to

$$\alpha_i^{k+1} = \max \{\alpha_j^k \mid j \text{ is a neighbor of } i\} + 1,$$

while all others nodes j maintain the same number, i.e., $\alpha_j^{k+1} = \alpha_j^k$. In addition, link directions are reversed according to the rule that link (i, j) should be directed from i to j if $(\alpha_i^{k+1}, i) > (\alpha_j^{k+1}, j)$. Thus, in this scheme the directions of any link (i, j) is determined by the ordering of the "numbers" (α_i^k, i), (α_j^k, j) and are always oriented from higher to lower "number." This precludes the formation of a cycle. Furthermore, within the context of contingency routing, it is trivial to assign a direction to a new link in the graph without forming a cycle by simply orienting the link from the node with a higher "number" to the node with lower "number."

Partial Reversal Method: At each stage of the algorithm, we associate with every node i a triple (α_i, β_i, i) where α_i and β_i are integers. The set of triples $\{(\alpha_i, \beta_i, i)\}$ is ordered lexicographically. The initial set of triples $\{(\alpha_i^0, \beta_i^0, i) \mid i \in N\}$ is such that $\alpha_i^0 = 0$, for all i and for any link (i, j) we have $(\alpha_i^0, \beta_i^0, i) > (\alpha_j^0, \beta_j^0, j)$ if and only if the direction of (i, j) in the initial ADG in problem (P) goes from i to j. The kth iteration is implemented as follows. A node i other than the destination for which $(\alpha_i^k, \beta_i^k, i) < (\alpha_j^k, \beta_j^k, j)$ for all neighbors j increases α_i^k to

$$\alpha_i^{k+1} = \min \{\alpha_j^k \mid j \text{ is a neighbor of } i\} + 1$$

and sets β_i^k to

$$\beta_i^{k+1} = \begin{cases} \min \{\beta_j^k \mid j \text{ is neighbor of } i \text{ with } \alpha_i^{k+1} = \alpha_j^k\} - 1 \\ \qquad \text{if there exists a neighbor } j \text{ with } \alpha_i^{k+1} = \alpha_j^k \\ \beta_i^k \quad \text{otherwise.} \end{cases}$$

All other nodes j maintain the same integers α_j and β_j, i.e., $\alpha_j^{k+1} = \alpha_j^k, \beta_j^{k+1} = \beta_j^k$.

The two algorithms described in this section are representative of a general class of algorithms for problem (P), which are based on a generalized numbering system. The next section is devoted to the development of this class.

IV. A GENERAL CLASS OF ALGORITHMS

We are given a connected graph G the nodes of which are denoted by $1, 2, \cdots, N+1$. Node $N+1$ is designated as the destination. The set of links is denoted by L.

Let A be a countably infinite set which is totally ordered by a relation $<$ in the sense that for any two distinct elements a_1 and a_2 of A there holds $a_1 < a_2$ or $a_2 < a_1$, but not both. Assume that A is partitioned into N disjoint subsets $A_1, \cdots, A_N$, each of which is countably infinite and unbounded in the sense that there exists no element $a^* \in A$ and index $i \in \{1, \cdots, N\}$ such that for all $a_i \in A_i$ we have $a_i < a^*$. Assume further that each subset A_i is equipped with an addition operation under which it is an Abelian group. This means that for each i there is a mapping $p: A_i \times A_i \to A_i$ such that 1) if $b \in A_i$, $c \in A_i$ then $p(b, c) \in A_i$, 2) there exists an element $z \in A_i$ such that $p(z, b) = b$ for all $b \in A_i$, 3) for each $b \in A_i$ there exists an $\bar{b} \in A_i$ such that $p(b, \bar{b}) = z$; 4) for each $b \in A_i$, $c \in A_i$, and $d \in A_i$ we have $p(b, c) = p(c, b)$ and $p(p(b,c), d) = p(b, p(c, d))$.

Where there is no danger of confusion we use the symbol "+" to denote the addition operation on each group A_i and the symbol "−" to denote its inverse, i.e., we denote $p(a, b) = a + b$, and $d = a - b$ if $p(d, b) = a$ for all $a, b \in A_i$.

Examples: For the full reversal method, each set $A_i, i = 1, \cdots, N$, consists of the set of pairs (α_i, i) where α_i is an integer. The addition operation on A_i is $p[(\alpha_i, i), (\alpha_i', i)] = (\alpha_i + \alpha_i', i)$. For the partial reversal method, each set $A_i, i = 1, \cdots, N$, consists of the set of triples (α_i, β_i, i) where α_i and β_i are integers. The addition operation on A_i is $p[(\alpha_i, \beta_i, i), (\alpha_i', \beta_i', i)] = (\alpha_i + \alpha_i', \beta_i + \beta_i', i)$. The order on $A = \cup_{i=1}^N A_i$ in both cases is specified lexicographically as described in the previous section.

Let V be the set of N-tuples $v = (a_1, a_2, \cdots, a_N)$ where $a_i \in A_i$ for each i. Each $v = (a_1, \cdots, a_N) \in V$ assigns a direction on each link $(i, j) \in L$ with $i, j \neq N+1$ according to the rule

$a_i > a_j \Rightarrow$ Link (i, j) is directed from i to j.

It assigns to each link $(i, N+1) \in L$ the direction from i to $N+1$. The resulting directed graph is clearly acyclic for each $v \in V$. A sequence $\{v_k\}$ corresponds to a sequence of acyclic graphs. We will describe a class of algorithms that generate sequences $\{v_k\} \subset V$ according to rules that guarantee that there exists an index $\bar{k}$ such that $v_k = v_{\bar{k}}$ for all $k \geq \bar{k}$ and the ADG corresponding to $v_{\bar{k}}$ is destination oriented. Clearly, any algorithm of this type solves problem (P).

For each $v = (a_1, \cdots, a_N) \in V$ denote by $S(v)$ the subset of $\{1, \cdots, N\}$ given by

$$S(v) = \{i \mid (i, N+1) \notin L, \text{ and } a_i < a_j \text{ for all } j \text{ with } (i, j) \in L,\ i = 1, \cdots, N\}. \tag{1}$$

Clearly, $S(v)$ is nonempty if and only if the ADG corresponding to v is destination disoriented.

We consider algorithms of the form

$$v_{k+1} \in M(v_k), \qquad v_0 \text{: given} \tag{2}$$

where M is a point-to-set mapping which assigns to each element $v \in V$ a nonempty subset $M(v)$ of V. The meaning of (2) is that v_{k+1} is selected arbitrarily among the elements of $M(v_k)$. Thus, the sequence $\{v_k\}$ generated by (2) need not be unique. As will become apparent shortly, this lack of uniqueness captures the distributed and asynchronous nature of the contingency routing algorithms we envision.

We will make the following assumptions regarding algorithm (2).

(A.1): There exist functions g_i: $V \to A_i$, $i = 1, \cdots, N$, such that, for each $v = (a_1, \cdots, a_N) \in V$, the set $M(v)$ is given by

$$M(v) = \{v\} \qquad \text{if } S(v) \text{ is empty} \tag{3a}$$

$$M(v) = \{\bar{v} = (\bar{a}_1, \cdots, \bar{a}_N) \mid \bar{v} \neq v \text{ and either } \bar{a}_i = a_i \text{ or } \bar{a}_i = g_i(v),\ \forall i = 1, \cdots, N\} \quad \text{if } S(v) \text{ is nonempty.} \tag{3b}$$

(A.2): For each $v = (a_1, \cdots, a_N) \in V$ and $i = 1, \cdots, N$ the functions g_i satisfy

$$g_i(v) > a_i \quad \text{if } i \in S(v) \tag{4}$$

$$g_i(v) = a_i \quad \text{if } i \notin S(v). \tag{5}$$

Furthermore, $g_i(v)$ depends only on a_i and those a_j for which $(i, j) \in L$, i.e., for each i, $v = (a_1, \cdots, a_N)$ and $v' = (a_1', \cdots, a_N')$ such that $a_i = a_i'$ and $a_j = a_j'$ for all j with $(i, j) \in L$, we have

$$g_i(v) = g_i(v').$$

(A.3): For each $i \in \{1, \cdots, N\}$, and each sequence $\{v_k\} \subset V$ for which $i \in S(v_k)$ for an infinite number of indices k, the sequence

$$\left\{a_i^0 + \sum_{r=0}^{k} [g_i(v_r) - a_i^r]\right\}$$

is unbounded in A_i, where a_i^r denote the coordinates of v_r.

Note that (A.1) implies that if $v_{k+1} \in M(v_k)$, $k = 0, 1, \cdots$, then $v_{k+1} \neq v_k$ iff $S(v_k)$ is nonempty, and if $S(v_{\bar{k}})$ is empty for some $\bar{k}$ then $v_k = v_{\bar{k}}$ for all $k \geq \bar{k}$.

Examples: In the full reversal method, the function g_i is defined for all $v = ((\alpha_1, 1), \cdots, (\alpha_N, N))$ by

$$g_i(v) = \begin{cases} (\max\{\alpha_j \mid (i, j) \in L\} + 1, i) & \text{if } i \in S(v) \\ (\alpha_i, i) & \text{if } i \notin S(v). \end{cases}$$

In the partial reversal method, the function g_i is defined for all $v = ((\alpha_1, \beta_1, 1), \cdots, (\alpha_N, \beta_N, N))$ by

$$g_i(v) = \begin{cases} (\bar{\alpha}_i, \bar{\beta}_i, i) & \text{if } i \in S(v) \\ (\alpha_i, \beta_i, i) & \text{if } i \notin S(v) \end{cases}$$

where

$$\bar{\alpha}_i = \min\{\alpha_j \mid (i, j) \in L\} + 1$$

$$\bar{\beta}_i = \begin{cases} \min\{\beta_j \mid (i, j) \in L, \bar{\alpha}_i = \alpha_j\} - 1 & \text{if there exists } j \text{ with } (i, j) \in L,\ \bar{\alpha}_i = \alpha_j \\ \beta_i & \text{otherwise.} \end{cases}$$

It is easy to see that the corresponding algorithms satisfy Assumptions (A.1)-(A.3). Within the context of these two algorithms and the discussion of the previous sections one can appreciate the value of introducing the general algorithm via a point-to-set mapping as in (2). Thus, (2) and (3) imply that if the ADG corresponding to v_k is destination disoriented and, hence, the set $S(v_k)$ of nodes with no outgoing link is nonempty, then any one of the nodes in $S(v_k)$ or several of these nodes, simultaneously, can initiate a reversal, i.e., can apply the mapping g_i to v. The timing and order in which reversals take place is unspecified. Different orders of reversals merely correspond to different sequences $\{v_k\}$ generated by algorithm (2).

The following propositions give our main results regarding algorithm (2). The proofs are relegated to the Appendix.

Proposition 1: Assume that the graph G is connected and that (A.1)-(A.3) hold. Then given any $v_0 \in V$ there exists a $v^* \in V$ (depending only on v_0) such that for every sequence $\{v_k\}$ generated by algorithm (2) there is an index $\bar{k}$ such that $v_k = v^*$ for all $k \geq \bar{k}$.

Proposition 2: Assume that the graph G is connected and that (A.1)-(A.3) hold. Let $\{v_k\} = \{(a_1^k, \cdots, a_N^k)\}$ be a sequence generated by algorithm (2). Then for every node

$i \in \{1, \cdots, N\}$ for which there exist links $(i, j_1), (j_1, j_2), \cdots (j_m, N+1)$ with $a_i{}^0 > a_{j_1}{}^0, a_{j_1}{}^0 > a_{j_2}{}^0, \cdots, a_{j_{m-1}}{}^0 > a_{j_m}{}^0$, we have

$$a_i{}^0 = a_i{}^k \quad \forall k = 0, 1, \cdots.$$

Proposition 1 shows that the algorithm essentially terminates in a finite number of iterations at an element v^* which corresponds to a destination oriented ADG. A rather remarkable fact is that v^* depends only on v_0 and does not depend on the particular sequence $\{v_k\}$ generated by the algorithm. In the context of the reversal methods, this means that *regardless of the timing and order of reversals the same final destination oriented ADG will be obtained within a finite number of iterations.* Proposition 2 shows an important stability property of the algorithm, namely, that *a node that lies on a directed path to the destination in the initial ADG corresponding to v_0 essentially will not participate in the algorithm* (will never undergo a reversal in the context of the reversal algorithm).

CONCLUSIONS

The algorithms proposed in this paper can form the basis for developing contingency routing algorithms for mobile PR networks with a central station. In such schemes, each PR will have a generalized number (i.e., an element of a suitable totally ordered set) associated with it. Link directions will always be oriented from higher to lower number. This precludes the formation of loops and provides reliable secondary routes that can be used for transmitting connectivity information and data to the station when primary routes fail. The station does not have to respond immediately to a topological change as long as secondary routes are available but can at any time intervene and reestablish primary and secondary routes on the basis of some criterion. This can be done by assigning new generalized numbers to selected nodes. When a PR loses all its routes to the station it undergoes a reversal process whereby on the basis of the numbers of its neighbors is selects a number according to the rules of one of the algorithms proposed. This number is broadcast to all neighbors who thus become implicitly informed of any reversals in the direction of communication that affect them.

A number of practical issues will have to be resolved before a scheme of the type described can be implemented in a real operating environment. For example, an error detection scheme will be required to ensure that each PR operates on the basis of correct numbers for all its neighbors. One possibility here is to require each PR to acknowledge a number change of a neighbor the first time it sends a packet to that neighbor after a change has been made as evidence that he has heard the correct number. Another difficulty has to do with the possibility of some numbers becoming too large during the algorithmic process. This will happen, for example, if the network becomes disconnected. One possibility for dealing with this is to require all nodes, the numbers of which have exceeded the allowable limit, to wait for the station to intervene and reset the number to lower values. This can be combined with other possibilities involving, for example, distributed schemes for reducing high numbers starting from the destination and proceeding in the direction of nodes with high numbers. It thus appears that it is possible to resolve satisfactorily the implementation aspects of the algorithms described.

It is of interest to compare the algorithms of this paper with distributed shortest path algorithms based on minimum number of hops to the station [3], [4]. The algorithms of this paper do not guarantee the generation of shortest paths and must rely on the station for periodic optimization of routing. On the other hand, they offer two substantial advantages over shortest path algorithms. Firstly, because of the multiplicity of available routes to the station the contingency algorithm will be activated only in the typically rare occasion where a PR will lose all of its available routes to the station. Even if this occurs it is unlikely that a long chain of reversals and message exchanges will be necessary before a destination oriented ADG is established. This is particularly true for networks with high connectivity. It is important to emphasize in this connection the result of Proposition 2 which essentially states that the nodes that have not lost all their routes to the station will not participate in the reversal process or communication exchange. Secondly, in order to awaken a new link no direction reversals or communication will be necessary other than the necessary exchange of end node numbers. This is particularly important in networks with PR's frequently moving into new areas and establishing new connections with other PR's. By contrast in distributed shortest path algorithms the failure of a link on the shortest path tree or the awakening of a link near the station can trigger message exchanges which can propagate through the entire network. Thus, both the frequency of activation and the communication overhead are much higher for shortest path algorithms. In addition, our algorithms guarantee loop freedom of generated routes at all times (although not necessarily loopfree communication). This is not the case for some of the distributed shortest path algorithms, e.g., the original ARPANET algorithm [4].

APPENDIX

In this Appendix, we prove Propositions 1 and 2. We assume that the graph G is connected and that (A.1)-(A.3) hold. Throughout the Appendix, we denote the coordinates of $v_k, \tilde{v}_k, \bar{v}_k$, etc., by $a_i{}^k, \tilde{a}_i{}^k, \bar{a}_i{}^k$, etc.

Proof of Proposition 1: We establish the proof through a sequence of lemmas.

Lemma 1: For every sequence $\{v_k\}$ generated by algorithm (2), there exists an index $\bar{k}$ such that $S(v_k)$ is empty for all $k \geqslant \bar{k}$.

Proof: Assume the contrary, i.e., there exists a sequence $\{v_k\}$ generated by algorithm (2) such that $S(v_k)$ is nonempty for an infinite number of indices k. Since $S(v_k)$ is a subset of the finite set $\{1, \cdots, N\}$, the assertion above and Assumption (A.1) imply that there exists an infinite index set K and some node $i \in \{1, \cdots, N\}$ such that

$$i \in S(v_k), \qquad \forall k \in K \tag{6}$$

$$a_i{}^{k+1} = \begin{cases} g_i(v_k) & \text{if } k \in K \\ a_i{}^k & \text{if } k \notin K. \end{cases} \tag{7}$$

We have for every k

$$a_i^{k+1} = a_i^0 + \sum_{r=0}^{k} [g_i(v_r) - a_i^r]. \tag{8}$$

From (6), (8), and Assumption (A.3) it follows that $\{a_i^k\}$ is unbounded in A_i and, hence, also unbounded in A. From (6), we obtain

$$a_i^k < a_j^k \quad \forall k \in K, \ j \text{ with } (i,j) \in L. \tag{9}$$

Since for all $j \in \{1, \cdots, N\}$, $\{a_j^k\}$ is "monotonically nondecreasing" in the sense of the total order of A_j, it follows from (9), that for all nodes j with $(i,j) \in L$, $\{a_j^k\}$ is unbounded in A and, hence, $j \in S(v_k)$ for an infinite number of indices k. Repeating for all nodes j with $(i,j) \in L$, the argument made above for node i and continuing in this manner we obtain that, for all nodes j that are connected to i by an undirected path in G, the sequences $\{a_j^k\}$ are unbounded in A. Since G is connected we conclude that $\{a_j^k\}$ is unbounded in A for all $j \in \{1, \cdots, N\}$. On the other hand, for every node j directly connected to the destination [i.e., with $(j, N+1) \in L$] we have $j \notin S(v_k)$ for all k, and, therefore, $a_j^k = a_j^0$ for all k. This contradicts the earlier conclusion that $\{a_j^k\}$ is unbounded for all $j \in \{1, \cdots, N\}$. Q.E.D.

Lemma 1 proves the portion of Proposition 1 that asserts that every sequence generated by algorithm (2) essentially terminates in a finite number of iterations at some v^*. It remains to show that v^* depends only on v_0 and not on the particular sequence generated by the algorithm. To show this, we need some preliminary lemmas and definitions.

Lemma 2: For any $v \in V$, if $i \in S(v)$ and $j \in S(v)$ then $(i,j) \notin L$.

Proof: Assume $(i, j) \in L$. Then $i \in S(v)$ implies $a_j > a_i$ while $j \in S(v)$ implies $a_j < a_i$, a contradiction. Q.E.D.

Definition 1: Two finite sequences $\{v_0, \cdots, v_m\}$ and $\{\bar{v}_0, \cdots, \bar{v}_n\}$ in V are said to be *equivalent* if $v_0 = \bar{v}_0$ and $v_m = \bar{v}_n$.

Consider, for $i = 1, \cdots, N$, the mapping $f_i: V \to V$ defined for each $v = (a_1, \cdots, a_N)$ by

$$f_i(v) = (a_1, \cdots, a_{i-1}, g_i(v), a_{i+1}, \cdots, a_N). \tag{10}$$

From Lemma 2 and Assumptions (A.1) and (A.2), it follows easily that, for every $v = (a_1, \cdots, a_N) \in V$, if $i \in S(v)$, $j \in S(v)$ and $i \neq j$ then

$$i \in S[f_j(v)], \quad j \in S[f_i(v)]$$

and

$$f_i[f_j(v)] = f_j[f_i(v)] = (a_1, \cdots, a_{i-1}, g_i(v), a_{i+1}, \cdots, a_{j-1}, g_j(v), a_{j+1}, \cdots, a_N).$$

A similar statement holds for more than two nodes in $S(v)$. Using this fact and the definitions (3) and (10) of M and f_i, respectively, we obtain the following lemma.

Lemma 3: For every finite sequence $\{v_0, \cdots, v_m\}$ for which $v_{k+1} \in M(v_k)$ for all $k = 0, 1, \cdots, m-1$, there exists an equivalent sequence $\{\bar{v}_0, \cdots, \bar{v}_n\}$ and nodes $i_0, \cdots, i_{n-1}$ such that

$$i_k \in S(\bar{v}_k) \quad \text{and } \bar{v}_{k+1} = f_{ik}(\bar{v}_k), \quad \forall k = 0, 1, \cdots, n-1.$$

Lemma 3 shows that for any finite sequence generated by algorithm (2) there exists an equivalent sequence generated by the algorithm

$$v_{k+1} = \bar{M}(v_k), \qquad v_0: \text{ given} \tag{11}$$

where $\bar{M}$ is the point-to-set mapping defined for all $v \in V$ by

$$\bar{M}(v) = \begin{cases} \{f_i(v) \mid i \in S(v)\} & \text{if } S(v) \text{ is nonempty} \\ \{v\} & \text{if } S(v) \text{ is empty.} \end{cases} \tag{12}$$

Note that every sequence generated by algorithms (11) and (12) can also be generated by algorithm (2). Thus, in a sense, algorithms (2), (11) and (12) are equivalent.

For any $v \in V$ for which $S(v)$ is nonempty we denote by $i(v)$ the numerically smallest node i in $S(v)$, i.e.,

$$i(v) = \min \{i \mid i \in S(v)\}. \tag{13}$$

Consider the point-to-point algorithm

$$v_{k+1} = \tilde{f}(v_k), \qquad v_0: \text{ given} \tag{14}$$

where the function $\tilde{f}: V \to V$ is given by

$$\tilde{f}(v) = \begin{cases} f_{i(v)}(v) & \text{if } S(v) \text{ is nonempty} \\ v & \text{if } S(v) \text{ is empty.} \end{cases} \tag{15}$$

For any $v_0 \in V$, algorithm (14) will generate a (unique) sequence $\{\tilde{v}_k\}$. Since, this sequence can also be generated by algorithm (2), it follows from Lemma 1 that there exists some $v^* \in V$ (depending only on v_0) and a smallest index $\bar{k}$ such that $\tilde{v}_k = v^*$ for all $k \geq \bar{k}$. The idea of the proof of the remainder of Proposition 1 is to show that for any sequence $\{v_k\}$ generated by algorithm (11) starting from the same initial element v_0, the sequences $\{\tilde{v}_0, \cdots, \tilde{v}_{\bar{k}}\}$ and $\{v_0, \cdots, v_{\bar{k}}\}$ are equivalent and, hence, $v_{\bar{k}} = \tilde{v}_{\bar{k}} = v^*$. We will need the following lemma as a first step in this argument.

Lemma 4: Let $\{v_k\}$ be a sequence generated by algorithm (11). Assume that for a node i and two indices p and q with $p < q$ we have $a_i^p = a_i^q$. Assume further that $i \in S(v_p)$. Then for all k with $p \leq k \leq q$, we have $i \in S(v_k)$, and $j \notin S(v_k)$ for all j with $(i,j) \in L$.

Proof: By Lemma 1, for all j with $(i, j) \in L$ we have $j \notin S(v_p)$ and, hence, by (A.2), $a_j^{p+1} = a_j^p$. The hypothesis

$a_i^p = a_i^q$ and Assumption (A.2) imply that $a_i^k = a_i^p$ for all k with $p \leq k \leq q$. Since $i \in S(v_p)$ and $a_i^{p+1} = a_i^p$, we obtain $i \in S(v_{p+1})$. We complete the proof by repeating this argument for $p+1, \cdots, q$. Q.E.D.

Any finite sequence $v_0, \cdots, v_{m+1}$ generated by algorithm (11) for which $S(v_k)$ is nonempty for all $k \in 0, \cdots, m$ can be specified by v_0 and the set of nodes $i_0, \cdots, i_m$ for which $v_{k+1} = f_{i_k}(v_k)$ for $k \in 0, \cdots, m$. This motivates the following definition.

Definition 2: Given a $v_0 \in V$ and a set of nodes $i_0, \cdots, i_m$ we say that the sequence $\{v_0; i_0, \cdots, i_m\}$ is *valid* if

$$i_k \in S(v_k), \qquad \forall k = 0, 1, \cdots, m$$

where $v_1, \cdots, v_{m+1}$ are defined recursively by

$$v_{k+1} = f_{ik}(v_k), \qquad \forall k = 0, 1, \cdots, m.$$

Clearly, there is a one-to-one correspondence between valid sequences and "nonterminating" finite sequences generated by algorithm (11). We can thus unambiguously talk about equivalences of valid sequences generated by algorithm (11) (cf. Definition 1). The following lemma follows easily from Definitions 1 and 2 and the remark following Definition 1.

Lemma 5: If $\{v_p; i_p, i_{p+1}, \cdots, i_m, i_{m+1}, \cdots, i_q\}$ is a valid sequence and $(i_m, i_{m+1}) \notin L$, then the sequence $\{v_p; i_p, i_{p+1}, \cdots, i_{m+1}, i_m, \cdots, i_q\}$ is valid and equivalent to $\{v_p; i_p, i_{p+1}, \cdots, i_m, i_{m+1}, \cdots, i_q\}$.

The following lemma provides the crucial construction for the remainder of the proof of Proposition 1.

Lemma 6: Let $\{v_k\}$ be a sequence generated by algorithm (11), and assume that, for some k_0, $v_{k_0+1} \neq \tilde{f}(v_{k_0})$. For each k for which $S(v_k)$ is nonempty let i_k be the node for which

$$v_{k+1} = f_{ik}(v_k). \tag{16}$$

Let $p \geq 0$ be the first index k_1 for which $i_{k_1} \neq i(v_{k_1})$. Then there exists an index $k_2 > p$ for which $i_{k_2} = i(v_p)$ and if q is the first such index, i.e.,

$$i_q = i(v_p), \qquad i_k \neq i(v_p), \qquad \forall k \in \{p, \cdots, q-1\} \tag{17}$$

then the sequence $\{v_p; i(v_p), i_p, \cdots, i_{q-1}\}$ is valid and equivalent to $\{v_p; i_p, i_{p+1}, \cdots, i_q\}$.

Proof: If there were no index $k_2 > p$ for which (17) holds, then by (A.1) and Lemma 4 $i(v_p) \in S(v_k)$ for all $k \geq p$ which contradicts the conclusion of Lemma 1.

If q satisfies (17), then $a^q_{i(v_p)} = a^p_{i(v_p)}$, It follows from Lemma 4 and the fact $i(v_p) \in S(v_p)$ that

$$i(v_p) \in S(v_k), \qquad \forall k \in \{p, \cdots, q\} \tag{18}$$

$$j \notin S(v_k), \qquad \forall k \in \{p, \cdots, q\}, \quad j \text{ with } (i(v_p), j) \in L. \tag{19}$$

and, therefore,

$$(i_k, i(v_p)) \notin L, \qquad \forall k = p, p+1, \cdots, q-1. \tag{20}$$

By using (20) and Lemma 5 it follows that the sequence $\{v_p; i_p, \cdots, i_{q-2}, i(v_p), i_{q-1}\}$ is valid and equivalent to $\{v_p; i_p, \cdots, i_{q-1}, i(v_p)\}$. Applying again (20) and Lemma 5 we have that the sequence $\{v_p; i_p, \cdots, i_{q-3}, i(v_p), i_{q-2}, i_{q-1}\}$ is valid and equivalent to $\{v_p; i_p, \cdots, i_{q-1}, i(v_p)\}$ and continuing in this manner the result follows. Q.E.D.

The proof of the remainder of Proposition 1 is now straightforward. Let $\{v_k\}$ be a sequence generated by algorithm (11) and let $\{\tilde{v}_k\}$ be the sequence generated by algorithm (14) with $\tilde{v}_0 = v_0$. It will suffice to show that there exists an index $\bar{k}$ and an element $v^* \in V$ such that

$$v_k = \tilde{v}_k = v^*, \qquad \forall k \geq \bar{k} \tag{21a}$$

$$v_k \neq v_{k-1}, \qquad \tilde{v}_k \neq \tilde{v}_{k-1}, \quad \forall k \leq \bar{k}. \tag{21b}$$

Let p be the first index k_0 for which $v_{k_0+1} \neq \tilde{v}_{k_0+1}$, i.e.,

$$v_k = \tilde{v}_k, \qquad \forall k \leq p \tag{22}$$

$$v_{p+1} \neq \tilde{v}_{p+1}. \tag{23}$$

Let q be the index satisfying (17). Applying Lemma 6, we obtain a sequence $\{\hat{v}_k\}$ that can be generated by algorithm (11) for which

$$\hat{v}_k = \tilde{v}_k, \qquad \forall k \leq p+1 \tag{24}$$

$$\hat{v}_k = v_k, \qquad \forall k \geq q. \tag{25}$$

Note from (22)-(24) that $\{\hat{v}_k\}$ has at least its first $(p+1)$ element identical with $\{\tilde{v}_k\}$ whereas $\{v_k\}$ has only its first p elements identical with $\{\tilde{v}_k\}$. Furthermore, from (25) it follows that $\{\hat{v}_k\}$ "terminates" at the same element as $\{v_k\}$. Apply now the same procedure to $\{\hat{v}_k\}$. This yields another sequence with its first $(p+2)$ elements identical with $\{\tilde{v}_k\}$ and the same terminating element as $\{v_k\}$. From Lemma 1, we have that there exists a smallest index $\bar{k}$ such that $\tilde{v}_k = v_{\bar{k}}$ for all $k \geq \bar{k}$. Hence, in a finite number of iterations this process will terminate at the point where the sequence $\{\bar{v}_k\}$ obtained is identical to $\{\tilde{v}_k\}$. Since $\{\bar{v}_k\}$ has the same terminating element as $\{v_k\}$ it follows that for some $\bar{k}$ and $v^* \in V$, (21) holds. Q.E.D.

Proof of Proposition 2: The definition of $S(v)$ and Assumptions (A.1) and (A.2) imply that $a_{j_m}{}^k = a_{j_m}{}^0$ for all k. Since $a_{j_{m-1}}{}^0 > a_{j_m}{}^0$ by hypothesis, we obtain $j_{m-1} \notin S(v_k)$ for all k which implies that $a_{j_{m-1}}{}^k = a_{j_{m-1}}{}^0$ for all k. Proceeding similarly, we show that $a_i{}^k = a_i{}^0$ for all k.

Q.E.D.

REFERENCES

[1] R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzelman, "Advances in packet radio technology," *Proc. IEEE*, vol. 66, pp. 1468–1496, 1978.

[2] E. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.

[3] P. Merlin and A. Segall, "A failsafe distributed routing protocol," *IEEE Trans. Commun.*, vol. COM-27, pp. 1280–1288, 1979.

[4] W. P. Tajibnapis, "A correctness proof of a topology information maintenance protocol for a distributed computer network," *Communication Assoc. Comput. Mach.*, vol. 20, no. 7, pp. 477–485, 1977.

★

Eli M. Gafni (S'79) was born in Tel-Aviv, Israel, on March 8, 1950. He received the engineering degree in electrical engineering from the Technion I.I.T., Haifa, Israel, in 1972, and the M.S. degree from the University of Illinois, Champaign-Urbana, in 1979, and is presently a doctoral candidate in the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, where he is working on optimization and routing in networks.

From 1972 to 1977 he served in the Israeli Defense Force as a technical officer, where he participated in projects involving estimation and control.

Dimitri P. Bertsekas (S'70–SM'77) received the diploma of mechanical and electrical engineering at the National Technical University of Athens, Athens, Greece, in 1965, the M.S.E.E. degree from George Washington University, Washington, DC, in 1969, and the Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1971.

He has served on the Faculty of Stanford University and the University of Illinois. He is currently an Associate Professor at M.I.T. where he teaches courses on optimization and communication networks. His research interests are in optimization theory and algorithmic aspects of communication networks. He consults regularly with private industry.

Dr. Bertsekas is the author of *Dynamic Programming and Stochastic Control* (New York: Academic, 1976) and *Constrained Optimization and Lagrange Multiplier Methods* (New York: Academic, to be published), and coauthor of *Stochastic Optimal Control: The Discrete-Time Case* (New York: Academic, 1978). He is also an Associate Editor of the *SIAM Journal on Control and Optimization*.

A Comparison of Four Methods for Analog Speech Privacy

NUGGEHALLY S. JAYANT, SENIOR MEMBER, IEEE, BARBARA J. MC DERMOTT, SUSAN W. CHRISTENSEN, AND ANN MARIE S. QUINN

***Abstract*—Four well-known procedures for analog speech privacy have been compared in terms of *residual intelligibility, bandwidth expansion*, and *encoding delay*. *Intelligibility* scores have been determined from a perceptual experiment where about 70 untrained listeners were given the task of recognizing each of 200 spoken digits that occurred in a balanced set of 50 encrypted four-digit utterances, and by averaging resulting probabilities of correct digit recognition. *Bandwidth expansion* has been expressed in terms of a new segmental measure that is more sensitive to short-time bandwidth manipulations than a conventional, long-time-averaged power spectrum measurement. *Encoding delay* is a straightforward function of analog scrambler parameters.**

The scrambling procedures that have been compared are *sample permutation* (S), *block permutation* (B), *frequency inversion* (F), and a *combination of methods B and F*, denoted by $[BF]$. *Sample permutations* involved a contiguous set of L_S (2 to 128) 8 kHz samples, while *block permutations* operated on a contiguous set of N_B (4 to 128) speech segments each of which was L_B (8 to 256) samples long. *Frequency inversion* is obtained by simply inverting the sign of every other Nyquist (8 kHz) sample. The parameters, L_S, N_B, and L_B, determine residual intelligibility as well as transmission properties such as encoding delay and bandwidth.

The comparisons in our study provide a quantitative justification for the popular approach $[BF]$. For example, with $N_B = 8$ and $L_B = 128$, although the encoding delay is as much as 128 ms, the bandwidth expansion is only about 100 Hz (using the new segmental measure), and the digit intelligibility I is 20 percent. Note that in the specific problem of recognizing ten digits, purely random (input-independent) listener responses correspond to $I = 10$ percent.

Paper approved by the Editor for Data Communication Systems of the IEEE Communications Society for publication without oral presentation. Manuscript received March 24, 1980; revised June 19, 1980.

The authors are with Bell Laboratories, Murray Hill, NJ 07974.

I. INTRODUCTION

THE purpose of this paper is to compare four well-known procedures for analog privacy from a comprehensive viewpoint that includes measures of residual intelligibility and transmission suitability. Although analog scramblers have been widely discussed, [1]-[10], there appears to be little documentation of quantitative comparisons such as those attempted in this paper. What is *not* attempted in this paper is a comparison of an exhaustive set of analog scramblers, which would include, for example, the rather complex procedure of analog sample masking [6]. The scramblers included in our study are relatively simple procedures (Section II) based on sample permutation (S), segment or block permutation (B), frequency inversion (F) and a combination of methods B and F, denoted by $[BF]$. The criteria for comparisons will include residual intelligibility, (I), encoding delay (D), and bandwidth expansion (expressed by parameters W_L and W_H to be defined in Section III). In general, it is desirable to realize smallest possible values of I, D and bandwidth expansion, although the last parameter may sometimes be uncritical (see Section III). Issues beyond the scope of this paper include cryptanalysis [1], [5] and effects of channel impairments on the quality of descrambled speech [3], [5], [6].

Analog scramblers in general, and the simple examples of this paper in particular, can only provide *privacy* in the context of casual eavesdropping. For *security* in the presence of formal cryptanalysis, digital encryption [11], [12] is certainly more appropriate. The problem with the digital approach, of course, is that the available transmission bandwidth may not

0090-6778/81/0100-0018$00.75