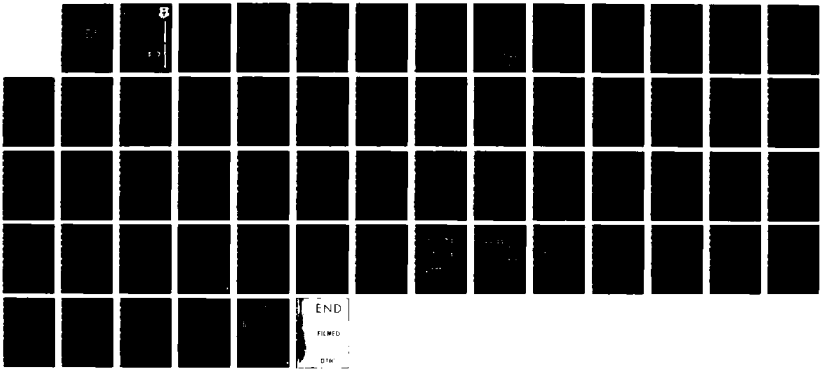
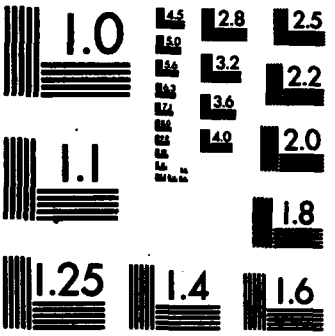
AD-A147 687 MESSAGE ROUTING METHODS FOR A TACTICAL AIR CONTROL SYSTEM COMMUNICATIONS NETWORK(U) ELECTRONIC SYSTEMS DIV HANSCOM AFB MA C S HOLT 10 SEP 84 ESD-TR-84-192 1/1

UNCLASSIFIED F/G 17/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ESD-TR-84-192

AD-A147 687

Message Routing Methods for a Tactical Air Control System Communications NETWORK

CRAIG S. HOLT

10 September 1984

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

DTIC FILE COPY

DTIC
ELECTE
NOV 19 1984
S D
B

Prepared for

ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
DEPUTY FOR DEVELOPMENT PLANS
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731

84 11 05 109

LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

THOMAS SELINKA, Capt. USAF
Project Officer

SILVIO V. D'ARCO, Lt Col, USAF
Deputy Director, Tactical C³I Systems Planning
Deputy for Development Plans

FOR THE COMMANDER

DONALD L. MILLER, Colonel, USAF
Assistant Deputy for Development Plans

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ESD-TR-84-192	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Hq Electronic Systems Division Deputy for Development Plans 6b. OFFICE SYMBOL (If applicable) ESD/XR	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Hanscom AFB Bedford, MA 01731	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) Message Routing Methods for a Tactical (Cont)	
12. PERSONAL AUTHOR(S) Craig S. Holt	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1984 September 10 15. PAGE COUNT 62
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR. and	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Amorphous network, Backup, Distributed network, ~~Distribution,~~ Graceful degradation, Network, Routing methods, (Cont)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

An investigation is made of possible routing methods for use in a communications network linking Tactical Air Control System Command and Control centers. Important qualities for the routing methods are assumed to be ability to work in a distributed, amorphous network, ability to adapt reliably to frequent changes in the network, and ability to take advantage of one-way communication links. Previous routing methods are reviewed and a new method is proposed. In the proposed method, each nodel unit reports the status of its inward links to all other units. Through these "updates", each unit is able to construct its own "model" of the system, and develop routing tables. A preliminary comparison of the methods is made, and areas for future research are recommended. Originator supplied Keywords include:

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☒ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Capt Thomas Selinka	22b. TELEPHONE NUMBER (Include Area Code) (617) 271-8400 22c. OFFICE SYMBOL ESD/XRTF

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

Block 11 Continued

Air Control System Communication Network.(Unclassified)

Block 18 Continued

Tactical Air Control System

TABLE OF CONTENTS

		PAGE
1.	Introduction	1
2.	Objectives of the Research Report	4
3.	Problem Statement and Definitions	4
4.	The Routing Problem	4
5.	Other Assumptions and Terms	5
6.	Routing Approaches	6
7.	Centralized vs Distributed	6
8.	Approaches for Distributed Routing	8
9.	Comparisons of Distributed Routing Approaches	9
10.	Selecting the Best Route	11
11.	Past Work	13
12.	Warner-Spragins Methods	13
13.	Wech Method	15
14.	Brayer Method	17
15.	Proposed Routing Method	18
16.	Addition or Deletion of Links	20
17.	Addition of Units	20
18.	Deleting Units From the Network	25
19.	Algorithm for Computing Routing From a System Model	26
20.	Comparison of Routing Methods	28
21.	Communications Requirements	28
22.	Storage Requirements	30
23.	Computation Requirements	30
24.	Performance in an Unchanging Network	30

TABLE OF CONTENTS

	PAGE
25. Response to Network Changes	31
26. Recommendations	34
27. References	37
28. Appendix A: Computer Simulation	39

LIST OF FIGURES

FIGURE		PAGE
1	Illustration of Case in Which Rules 2 and 3 Do Not Produce Sufficient Updates	24
A-1	Unit Structure	41
A-2	Link Structure	42

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
PER CALL JC
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

ACKNOWLEDGEMENTS

The author would like to thank the Air Force Systems Command, the Air Force Office of Scientific Research, and the Southeastern Center for Electrical Engineering Education for providing the opportunity for this summer project. He would also like to acknowledge the hospitality of the Electronic Systems Division/XR (MITRE location), especially the XRTF section. Numerous people in both the XR section and The MITRE Corporation were of great assistance in selecting and investigating a topic. Particular appreciation is due to Mr. Otto Wech and Mr. Murray Black for helpful discussions and key insights.

I. INTRODUCTION

In order for a deployable Tactical Air Control System (TACS) to perform its functions, reliable communications must be assured among the TACS command and control (C^2) centers in the face of enemy counter action.

Centers which must communicate with each other include:

- radar stations along the front edge of battle area (FEBA)
- army units in the field
- intelligence gathering stations
- logistics, supply, and backup centers in rear areas
- airfields which may be hundreds of kilometers from the FEBA
- aircraft on both sides of the FEBA
- a hierarchy of command and control centers

A communications network which spans the entire battlefield area is clearly needed.

Currently, many different communication systems serve different types of centers and carry different types of data. A unified system, which could be used by all centers for all types of data, could result in both a reduction in hardware and an improvement in service.

Some desired characteristics for a TACS C^2 communication network are:

- Survivability: The system must continue working in spite of significant damage and jamming.

- Mobility: Small mobile centers are less vulnerable than large fixed ones.

- Flexibility: The system must be easily expanded, contracted, or rearranged.

- Low visibility: Centers which are hard to identify will be more likely to survive.

- Security: Transmissions should be difficult to intercept and if intercepted, should not be useful to the enemy.

- Interoperability: The system will need to interface with existing and future communication systems.

- Ability to be implemented through evolutionary transition from the present system.

- Affordability.

This report considers schemes for message routing within a TACS communication network. "Routing" refers to the process of directing each message through the proper communication links so that it ultimately arrives at its destination. The "routing algorithm" of a network is the collection of procedures used by elements of the network to accomplish the routing task.

Most types of networks require some type of routing mechanism. In store-and-forward networks and some packet switching networks, routing is used to individually direct each message or packet. In circuit switching and virtual circuit networks, routing is used to direct control messages during the initial set-up of a "circuit" from source unit to destination unit. In the remainder of this report, a store-and-forward network will be assumed, with the understanding that the same routing ideas can be applied to other types of networks as well.

One type of network which requires no routing mechanism is the broadcast-bus type network, in which all messages are transmitted to all units. Broadcast-type systems are not considered in this report.

In general, one desires a routing algorithm which adapts reliably to network changes, which provides the greatest possible throughput with the least possible delay, and which minimizes use of network resources (bandwidth, storage, and computation time) for routing "overhead." Routing algorithms which attempt to achieve these goals have been widely studied (see [1] for a survey). There are, however, some distinguishing characteristics of the distributed TACS network that make its routing requirements different from the requirements in most studies.

(1) The network is amorphous (has no specific structure) and is subject to rapid changes in its links and units. Links and units may be added, deleted, or moved frequently. Multiple simultaneous failures are not unlikely.

(2) It is particularly important, in the TACS network, that when damage to a link or unit occurs there be as little disruption of service in the rest of the network as possible. The times when damage occurs are likely to be the times when proper operation of the network is most essential. (In some other networks, a temporary severe degradation of service, while the network recovers from failures, is acceptable.)

(3) Due to expected jamming effects, some directive communication links may work in one direction only. The requirement that the network maintain maximum service for as many units as possible in the face of jamming implies that the routing scheme should take advantage of one-way

links as well as two-way links. (In most other types of networks, any communiation path that works in only one direction is considered totally out of service.) Though a number of routing algorithms satisfy requirements (1) and (2) to varying degrees, very few satisfy requirement (3).

II. OBJECTIVES OF THE RESEARCH EFFORT

The main objectives of this project were:

(1) to investigate existing methods for routing in amorphous networks containing one-way links,

(2) to propose a routing method which meets the requirement of a TACS network to the greatest possible degree, and

(3) to compare the proposed method to other methods.

III. PROBLEM STATEMENT AND DEFINITIONS

A. The Routing Problem

We assume an abstract network which consists of a set of nodal "units" and a set of "links" connecting pairs of units. A unit represents any intersection of communication paths where the routing algorithm must determine which link to send a message out on. Links represent communication paths and may be either one-way ot two-way. Each unit refers to its own links by means of internal "port numbers" which are unknown to other units. It is assumed that units initially have no information about the network topology.

The routing algorithm must provide every unit with a "routing table" that tells the unit what port it shuld use to send a message destined for any other unit in the system.

Desirable qualities for the TACS routing algorithm are:

- reliable operation in spite of connection and disconnection of units and links.

- maximal utilization of the links available, even if some of them work in only one direction.

- maximal message throughput

- minimal delay

- minimal overhead (bandwidth, storage, and computation time)

- ability to work in large networks (networks of several hundred units)

B. Other Assumptions and Terms

The following notation will be used:

y is o(x) — (read "y is of order x") standard notation to indicate that the growth of quantity y is proportional to the growth of function x.

n — the number of units in the network.

e — the number of links in the network.

d — the maximum number of links attached to one unit.

Due to hardware limitations in the TACS network, it is expected that d will be a small number, independent of n. Thus in assessing the complexity and communication load of each algorithm, it will be assumed that d is $O(1)$, and e is $O(n)$.

"Flooding" refers to a special procedure which provides a fast and reliable way to get data to every reachable unit in a network. When a unit wants to flood a network, it begins by sending copies of its message

to all its neighbors. A unit receiving a flooded message first checks to see if it has already received another copy of the message. If no other copy has been received, the unit accepts the message, and sends copies out to all its neighbors (except possibly the one from which it was received). If a previous copy has been received, the new copy is discarded. Thus copies of the message spread rapidly outward from the originating unit, but the spreading stops wherever the message meets other copies of itself. Since each link is used at least once and no more than twice, the total network communication load involved in the flood is o(e x message length).

Finally, a "directed message" is a message which has a particular destination, as contrasted with a flooded message. A directed message moves from unit to unit, each unit using its routing table to determine the next move.

IV. ROUTING APPROACHES

A. Centralized vs. Distributed

Routing algorithms can be classified as either centralized or distributed. With a centralized algorithm, a master unit is designated to perform the routing task. This master gathers data about the system, analyzes the data, then distributes routing data out to all other units. The other units may then have to process the data further to produce actual routing tables. Since the master unit becomes essential to network operation, provision has to be made for reassigning the master function in case the master is disabled or disconnected from the rest of the system. Thus there are actually two modes of operation, normal mode

(with master) and provisional mode (used to reappoint a master).

With a distributed algorithm, all units perform the same algorithm to independently arrive at their own routing tables. Each unit uses information supplied by other units, but does not depend on the operation of any particular unit to successfully complete its own tables. There is no need for a special mode of operation when some units are disabled.

Neither centralized nor distributed algorithms have a clear advantage in terms of total computation or total communication overhead. The centralized approach, however, does have some major disadvantages:

- the need for a provisional operational mode for reassigning the master. This mode must be of the distributed type, since there is no master available. There must be a reliable method for switching to the provisional mode if the master is disabled. Switching to provisional mode to appoint a new master could be a major disruption of network service at a critical time, though it should very rarely happen.

- slow response. The report of a failure must always go all the way to the master and then back to all units. Thus when a unit which is far from the master fails, the surrounding units may experience a large delay before their routing tables are updated.

- bottleneck effects. If many changes are taking place in the network, communication and computation bottlenecks in the vicinity of the master unit may cause large delays.

Because of the disadvantages of the centralized approach, only distributed algorithms will be considered here.

B. Approaches for Distributed Routing

All the distributed routing algorithms of which the author is aware are based upon one or more of the following three methods:

Delay-Table-Passing: Each unit computes its delays to every other unit based on the delay tables passed to it by each of its neighbors. The neighbor showing the shortest delay to a given destination is used as a route to that destination. Changes in network conditions ripple through the network from neighbor to neighbor. Techniques have been developed to prevent certain "looping" instabilities that can occur with this type of algorithm [2,3]. A well-known example of routing by passing delay tables is the original ARPANET routing algorithm.

Flooded-Path-Searching: Flooded messages are used to discover the "best" paths from unit x to another unit y. When a message is flooded from x, the first copy to arrive at y is assumed to have taken the best path from x to y. If links are assumed to have the same delay in both directions, then y can record the port on which the message from x arrived and use the corresponding output port for sending messages back to x. If one-way links are involved, then y must somehow send the path information back to x so that x will know the best route to y. This return message may be accomplished by means of a second "flood." In effect, a round trip from x to y to x is used to inform x of its path to y.

Model-Building: Units distribute information to each other about the operating links and units of the network, and each unit uses the information it received to build its own "model" of the system, (i.e.,

its own [possibly incomplete] view of the system topology). Once the system model has been constructed, an analysis is done to generate the routing table. There are a variety of methods for distributing information about the network and for analyzing the model to produce the routing table.

C. Comparisons of Distributed Routing Approaches

Some preliminary comparative comments about the potentials and limitations of these three routing approaches can be made. Particular implementations, of course, may not achieve the potentials indicated here.

With Route-Table Passing, a unit needs to send its delay table to neighbors only when its table has changed. Thus it is possible to produce routing traffic only when needed, and these messages need propagate only over the portion of the network affected by the change. During certain types of network changes, however, it is necessary to introduce additional routing message and delays in order to avoid instabilities in the algorithm [2, 3]. Note that routing messages are of length O(n), and in any given message most of the information will usually be unchanged since the last time the message was sent.

There are some difficulties in applying Route-Table-Passing to systems with one-way links, since a routing table passed from some neighbor is of no value without additional information about paths back to that neighbor.

With Flooded-Path-Searching, all units must generate flooded search messages in order to be sure of fully adapting to a change anywhere in the network. Flooding by units near the change is not sufficient, since

even the routes of far away units can be affected by a small change. Searches could be triggered by special flooded messages from units detecting a change in the network, or could simply be generated by each unit periodically, regardless of network changes. The length of the routing messages varies depending on other aspects of the algorithm.

With Model-Building, it is only necessary to distribute information about the network when changes occur. Any unit that detects a change in the network can flood this information out to the rest of the units so that all units change their network models as soon as possible. Furthermore, each unit reporting changes need only report information about its own links, so messages can be short. Thus there is a potential for algorithms with low message overhead and rapid adaptation to network changes.

Model-Building algorithms are particularly versatile in that once the system model has been constructed, various types of routes can be generated, depending on the algorithm used to produce the routing table from the model. Simply by substituting different model analysis algorithms, one can implement fewest hops routes, shortest path routes, split-path routes, etc. (see Section IV.D).

From a command and control viewpoint, another advantage of Model-Building is that an up-to-date graph of the working network units and links is available at all units at all times. From a security viewpoint, however, this may be considered a drawback.

Model-Building algorithms tend to require more storage than other approaches, because a representation of the entire network must be

stored. Model-Building algorithms also tend to require more computation than other methods, because each unit must do its own analysis of the whole network, rather than taking advantage of its neighbor's analyses (as in Delay Table Passing) or getting route information directly from messages (as in Flooded Path Searching).

D. Selecting the Best Route

In any routing system, routing decisions are made based on certain routing criteria. Some of the types of routes that can be used are briefly reviewed below.

Fewest Hops Routing — Messages are sent on the path which traverses the fewest intermediate units. This method assumes that all links are of equal delay and thus may not give good results if some links are much slower than others. There is no adaptation to traffic conditions with this method.

Shortest Path Routing — For this type of routing, each link in the system is considered to have a certain link "weight", which is generally some measure of the link delay. Messages are sent on the path that has the smallest total link weight. Some subset of the following factors are generally used to compute link weights:

a. link transmission rate

b. delay or queue length at the link sender

c. delay or queue length at the link receiver

d. buffer space utilization at the link reciever

An attractive feature of using time-varying link weights (such as b, c, or d above) is that they enable a network to adapt to changing

traffic conditions. Ideally, such weighting schemes should cause messages to detour around heavily loaded portions of the network. Caution must be exercised, however, in implementing such "traffic-adaptive" routing schemes, for several reasons:

- too much adaptation to temporary traffic conditions can lead to network instability in the form of wildly fluctuating queue lengths or looping messages [7, 8].

- adapting to traffic conditions allows messages to take longer paths. Thus when network utilization is high all over the network, adaptive routing tends to decrease throughput [8, 10].

- even if some gains are obtained by adapting to traffic conditions, the gains may be small. (Gains have been shown to be small for most networks with uniform loading [8], but these results may not apply to a TACS network).

- having each unit compute its own optimal routing based on path weights does not, in general, lead to optimal routing for the network as a whole [6].

- adapting to traffic conditions increases the chances that message packets will arrive out of order. Thus more computation may be required to reassemble messages at the destination.

Split-Path Routing — The above methods may be termed "single-path" routing methods, since only one route is calculated to each destination. It has been shown that higher network throughput can be achieved by allowing message to travel along a variety of paths, with larger portions of the message traffic traveling along the faster paths [6]. This type

of routing may be termed "split-path", "traffic splitting", or "bifurcating." Messages may be assigned to paths by stochastic [6] or deterministic [9] rules. Traffic-adaptive features may or may not be incorporated.

PAST WORK

In this section, several algorithms for routing in systems with one-way links are briefly described. These are simplified summaries, intended to give only a qualitative understanding of each algorithm. For fully detailed descriptions of the Warner-Spragins and Brayer methods, see reference [4] and [5]. The Wech algorithm has not been published.

A. Warner-Spragins Methods

Warner and Spragins described several routing algorithms, all of which are extensions of the Flooded-Path-Search approach.

They began with Simple Backwards Learning, a method for networks with two-way paths only. In this method each unit x periodically floods to all units a message containing a "delay" field. The delay field holds the number of hops the message has traveled, or some other estimate of the time expired since the message was originally sent. Each receiver of x's flooded message uses the message delay x as a measure of the delay back to x. This estimate is used to update one entry in a table of estimated delays to each unit via each port. As this table entry is updated, other table entries representing delays to unit x over other ports are increased. In this way, old values in the table are "forgotten" over time. Changes in the table values are always made gradually so as

not to overreact to a temporary situation.

In order to apply this "Backwards Learning" method to networks containing one-way links, an extra bit, called the Non Reciprocal Path Bit (NRPB), is added to each flooded message. The NRPB is initially zero, but is set to one if the message traverses any one-way link. (It is assumed that units can easily establish whether an inward link is bad and if so can set the NRPB's of the messages sent out of the corresponding outward link.) By performing Backwards Learning using only those messages that arrive over two-way paths, units are able to determine the best two-way path to every destination (if a two-way path exists).

Two extensions are proposed for handling one-way links. In the "Rapid Reciprocal path Search" method, if a directed message must be sent to some destination to which there is no two-way path, then the message is flooded to all neighbors. The flood spreads only over those units that have no two-way path to the destination. With this method, a directed message will always arrive at its destination if any path exists, but if one-way paths are involved then multiple copies of the message may arrive.

An additional feature is added to speed the networks adaptations when a one-way link becomes a two-way link. Every time a flood passes through a unit, the delay and entry port of the first flood message to arrive, whether one-way or two-way, is always recorded in a special table. If a shortest path that was one-way becomes two-way, the new shortest two-way delay is available for immediate use, rather than

having to be gradually "learned" over time.

A second method for handling one-way links is the "Reciprocal Path Search Algorithm With Delay Vectors." Again units use Backwards Learning with the Non Reciprocal Path Bit to construct a table of delays via two-way paths. The following Delay-Table-Passing mechanism is added to find paths over one-way links: The unit at the receiving end of each one-way link periodically floods out a special message containing:

- the location of the one-way link and

- a table of its own delays to all other units, each augmented by the one-way link delay.

This special message is called a "delay vector." A unit receiving a delay vector can determine its delays to every destination through the one-way link by combining its own delay to the sending end of the one-way link with each delay vector value. Units keep track of not only the best two-way paths but also the best path through one-way links. The shortest path is used for routing messages. Entries in the "one-way path" table expire if they are not renewed within a certain time limit.

Warner and Spragins also proposed a third method combining ideas from the two just described. Their test results, however, indicated poorer performance for this third method.

B. Wech Method

O. Wech has proposed another algorithm of the Flooded-Path-Search type. The algorithm will be presented first as two distinct steps (although we will see that the two steps are really performed by

iterations of a single procedure).

First, each unit floods out an exploratory message to the network. The flooding process used differs from normal flooding in the following important detail: As the flood origin issues a message, it labels each copy of the message with the port on which it is being sent. Only the flood origin does this.

When the first copy of such a flooded message is received by any unit, the receiver records both the originating unit of the flood and the port used by that unit. After one round of floods, each unit y has a record of what port every other unit should use to send to y. This data is called the "experience vector" of unit y. The experience vector is of no value to y, but would be useful to any other unit that wishes to send a message to y.

In the second step, each unit floods its experience vector back to all the other units. When a unit x receives the experience vector of unit y, it finds its own entry, which indicates the port x should use to send to y. Unit x enters this data in its routing table. By the end of the second round of floods, all routing tables are complete.

Though the above explanation treats the initial "exploratory" flood and the second "experience vector" flood as different steps, a single combination procedure actually performs both steps. Every set of flooded messages contains both the first step data (sender and port number) and the second step data (sender's experience vector). Thus each flood from x both explores paths away from x, and at the same time distributes the current experience vector of x. At system

start-up all tables are empty, but after one round of floods, the experience tables are complete, and after two rounds of floods, the routing tables are complete.

Each unit continues periodic flooding so that any changes in the network will be sensed. To promote faster adaptation to network changes an immediate flood is sent whenever a unit detects a change in its experience vector. Routing table entries expire if they are not renewed within a certain time limit.

C. Brayer Method

The Brayer method is a Model-Building type method. Each unit's information about the network links is contained in its n x n "connectivity matrix." An entry of 1 in position c_{ij} of the matrix indicates that unit i has a link to unit j. An entry of 0 indicates no such link is known.

The connectivity matrix is constructed by means of path records appended to all messages. The path record is a list of all the units traversed by the message so far. Every time a unit handles a message, it examines the message's path record and adds any new links to its connectivity matrix. Then it adds itself to the message path record before sending the message onward.

Each message is routed along the "fewest-hops" path calculated from the connectivity table. If the connectivity table indicates no path, the message is randomly sent to any other unit which it has not already visited. Such "wandering" messages help explore new paths.

(Note that a message may traverse units it has visited before on the way to the next unit, but it will not loop indefinitely.) If the message finally arrives at its destination, an acknowledge message is returned to the sender. This may again require some "wandering." If a message acknowledgement is not returned to the sender within a preset timeout, the sender will try again. After a certain number of failed attempts to send a message, the sender will conclude that the destination is unreachable.

So that failed links will be detected, each link is kept active by its sending end. If a unit stops receiving traffic on a link, that unit issues a "bad link" message. The "bad link" message is randomly routed to all other units, informing them that they should remove a certain link from their connectivity tables. There is also provision for removing entries from the connectivity table if after repeated tries to send over a certain link no acknoweldges are received.

Computing the routing table from the connectivity matrix is a two-step process. First, the distance matrix is computed from the connectivity matrix. Each entry d_{ij} in the distance matrix indicates how many hops it takes to get from unit i to unit j. The algorithm used is of complexity O (D·e·n), where D is the network diameter (i.e., the maximal number of units in the fewest hops path between two units). A second algorithm, of complexity O(n), computes the routing table from the distance matrix.

VI. PROPOSED ROUTING METHOD

In this section the proposed routing algorithm is informally presented along with some of the reasoning which led to this algorithm. A more

detailed and concise description of the algorithm is given in Appendix I.

The method proposed here is a Model-Building type algorithm, based on the idea that the unit which first notices the failure or establishment of a link is best suited to inform the rest of the network of the change. In the proposed scheme, each link is kept active by its sending end. If there are no messages to send, the link is kept active with dummy traffic. Each unit constantly monitors the traffic on its input ports, and if a change is noted, floods an "update" message to the rest of the system. Update messages contain the identity of the source unit and information about the set of links on which that unit is receiving messages. Link information includes the identity of the link sender and the port number used by that sender.

From the udpates received, each unit is able to construct a model of the part of the communication network accessible to that unit. This model is stored in graph form (unit records and link records, attached by pointers) so that the memory requirements are $O(n+e) = O(n)$. An algorithm of complexity $o(n+e) = O(n)$ is used to generate the routing table from the graph model. (See Section VI.D.)

The proposed routing algorithm is quite similar to the one currently used in the ARPANET [7]. Both are Model-Building algorithms in which each unit distributes information about its own links. The proposed algorithm, however, is designed to make use of one-way links, while the ARPANET algorithm is not. Some major differences between the algorithms are:

- In the ARPANET, each unit reports the status of its outward links, whereas in the proposed method, units can only report the status of

inward links; outward link status may not be known.

- In the proposed algorithm, special methods are provided for adding and deleting nodes from the system model in the presence of one-way links.

- In the proposed algorithm, special measures are taken to avoid dependence on periodic updates from all units.

- The proposed algorithm for generating the routing table from the system model is different from the one used in ARPANET.

A. Addition or Deletion of Links

Any unit "x" which begins to receive a new inward link or fails to receive data on a formerly working inward link, floods an udpate to all other units. All other units then revise their system models to show the new set of inward links of unit x.

If a change in the network causes some units to become inaccessible to others or causes formerly inaccessible units to become accessible, then special action is taken (see following sections) on addition and deletion of units.

B. Addition of Units

A problem arises when a new network (or a new single unit) is added to an operating network. The new connections will trigger udpates informing the entire network about the new links along the border of the two networks. However, units on one side of the border will not find out about units behind the border on the other side unless those units experience some change that causes them to produce updates. This difficulty can be handled through use of the following four techniques:

a. A strightforward approach used by ARPANET is to require that all units issue periodic updates even if they experience no changes in their links. Such reports could be at relatively long intervals (ARPANET uses one minute) so that these extra flooded messages do not impose a large load on the network. Any new units would be able to construct a complete network model within one time interval of connection. Such periodic updates are a prudent safety feature to correct false routing information in the system. It is desirable, however, to minimize the reliance on periodic updates and instead disseminate new routing information as soon as it becomes available. Hence the following methods are proposed.

b. In the case of a single unit making its first connection to the network, the new unit could request a copy of its new neighbor's entire system model. Once the model is obtained, the new unit could keep up with future system changes like any other unit. Copying the model from a neighbor is an attractive method since it eliminates the need to generate floods from every unit.

c. In the case of two subnetworks joining, some method is needed to cause the units behind the border to produce udpates so that the units on the other side of the border can construct complete models. One way to do this is by the following rule: When a unit "x" receives an update indicating the presence of a new unit "y", x issues an update. The presence of new units thus stimulates the rest of the system to report back to those new units.

If two subnetworks join, each unit would receive a rapid sequence of updates indicating new units. Rather than responding each time with unnecessary copies of the same update, the unit should only respond to the first, then postpone further stimulated updates for a certain timeout period.

d. The methods given above will produce the desired updates in most situations. These methods, however, are not sufficient in the case when two subnetworks join by first establishing a link in one direction and later establishing a link in the other.

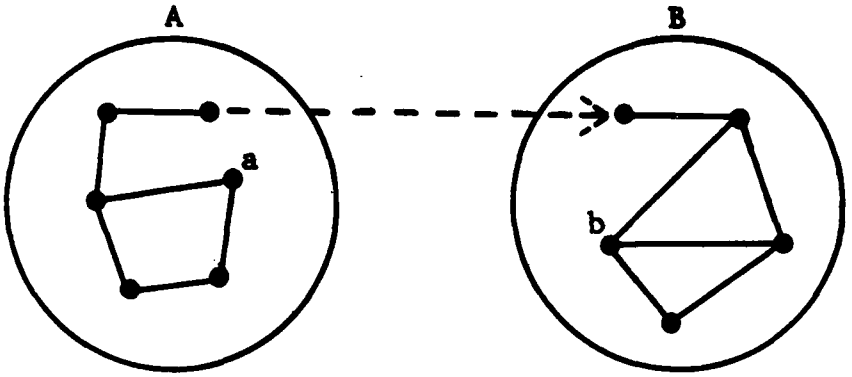
Suppose that a link exists from subnetwork A to subnetwork B (Figure 1a). Suppose further that due to recent changes in subnetwork A, all units in A have recently produced updates. These udpates are, of course, received by the units of both A and B. Thus units in B have complete models of the sytem, while units in A only have models of A. Now assume that a link is established from unit "b" in B to unit "a" in A (Figure 1b). Unit "a", sensing a change in inward links, will produce an update reporting the link from b. All other units in A, upon receiving news of the previously unknown unit "b", will also send updates (by rule 3). No updates, however, will be stimulated from units in B, since these units were already aware of all units in A. Thus units in A remain ignorant of B indefinitely.

To resolve this difficulty we add one more rule for "stimulated" updates: Unit x sends an update whenever it calculates that it has a route to some unit y to which x previously had not route. In the case of Figure 1, when the udpate from a reaches the units of B, B's units

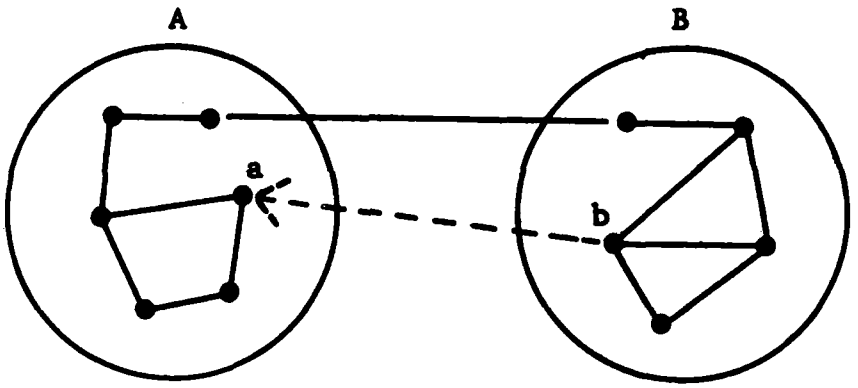
all calculate that they now have routes to the units of A. Hence, all units in B produce the needed updates, completing the models of units in A.

Rules 3 and 4 solve the problems of insufficient udpates, but unfortunately can also lead to some unnecessary updates. For example, in the case of Figure 1, when the units of A finally receive updates from B's units, A's units will respond with a second set of udpates. These updates are generated because the units of A have discovered the units of B (rule 3) and because the units of A now have routes to the units of B (rule 4). Since all units already have complete models, this second set of updates is unnecessary. Investigation shows that in most situations where units are added to the system, one such extra set of updates is produced. The excess updates are to a certain extent unavoidable since units have no knowledge of which parts of the network were or were not reached by previous updates. The additional traffic is not expected to pose an unacceptable load on the network.

Postponing updates for a short time after one update is issued (as mentioned earlier) helps to reduce the number of excess updates. The length of the timeout is not critical—if too long, routing changes at some units may be delayed, but the changes will occur eventually; if too short, more unnecessary updates will be issued.



(a) Link is established from subnetwork A to subnetwork B. Units in A know only about A. Units in B obtain complete models through miscellaneous updates from A.



(b) Link is established from unit b to unit a. Unit a sends update (rule 2). All units in A respond with stimulated updates (rule 3). Units in B do not respond. Units in A still do not know about units in B.

Figure 1. Illustration of case in which rules 2 and 3 do not produce sufficient udpates.

The four methods just described can be combined as follows: When a single node enters a network it would use method 2 (i.e., inform the network of its presence and get a network model from a neighbor). The first update from the single new unit would have to include a special "no stimulate bit" so that all the other units would not respond with stimulated updates. When two networks, each containing more than one unit combine, methods 3 and 4 would cause all units on both sides to produce sufficient udpates. Finally, a unit which senses no network changes would, after a relatively long delay, generate an update on its own (method 1). This is to assure that any routing information which is somehow lost or garbled will eventually be corrected.

C. Deleting Units from the Network

A second problem arises from the fact that only inward links are reported. Suppose some unit y becomes disconnected from the network. The neighbors of units y will flood out updates indicating that y is no longer sending. Also unit y will flood an update indicating that its inward links are not working, but this update cannot reach the rest of the network. Hence the main network will not find out that unit y can no longer receive; it will only find out that unit y can no longer send. If no special action is taken, units could continue to try to send messages to unit y through broken links. The same reasoning holds if an entire subnetwork becomes disconnected.

One way to avoid this situation would be to timestamp all updates and have all model data expire if it is not renewed within a given timeout period. After one timeout interval, a disconnected network would

decompose into two separate networks, each aware only of its own units. Units attempting to send messages to the other network would find that the destination does not exist and take alternative action. A serious disadvantage of this solution is that units must send out periodic updates even if no network changes are observed. If the network is to react quickly to failures, the model timeout must be short, and the interval between updates must be even shorter. The vast majority of these updates, of course, will carry no new information.

Fortunately, a better method, requiring no extra updates, is available. When some unit y becomes disconnected from a network, all units will be informed through normal udpates that y can no longer send. If y can no longer send, there is no way for the network to establish whether y exists. Thus y can be deleted from each unit's model on the basis that there are no paths out of y.

The method for handling deletion of units, then, is as follows. Whenever unit x receives an udpate which causes it to delete some arc from its model, the node "y" representing the arc sender is checked to see if it still has a path of x. If not, node y is removed from x's model, along with any arcs attached to node y. Removal of these arcs may, in turn, result in the removal of still more nodes. The entire process is accomplished by an algorithm of complexity $O(n+e) = O(n)$.

D. Algorithm for Computing Routing from a System Model

A simple algorithm is used to compute fewest hops routes. This algorithm mimics the action of a flooded path search. The unit in which this algorithm takes place is referred to as u. Let M be the set

of all nodes in u's current system model. For each node x in M, the set of nodes that x has arcs to is referred to as S(x) (successors of x).

Unit u contains a route table ROUTE which has an entry for each unit in M. A route table entry ROUTE(x) specifies the port that unit u should use to send to unit x. The purpose of the REROUTE algorithm is to fill in as many entries of the ROUTE table as possible. Before the REROUTE algorithm can operate, however, the ROUTE entries for u's immediate neighbors must be filled in. This is done by the following rule. Whenever u receives an update listing a link from u to some node x, ROUTE(x) is filled in with the port which u used to send to x.

The algorithm is given in a psuedo-code form. In the algorithm, Q is a queue (first-in-first-out list) of modes. (A slightly more detailed version is in the appendix.)

```
Procedure REROUTE
  for each node x in M - S(u) do ROUTE(x) = null;
  add S(u) to Q;
  while Q is not empty do begin
    remove a node x from Q;
    for each node y in S(x) do begin
      if ROUTE(y) = null;
      then begin
        ROUTE(y) = ROUTE(x);
        add y to Q;
      end then
    end for
  end while
end procedure
```

VII. COMPARISON OF ROUTING METHODS

In this section some observations are made about the expected overhead and performance of the various routing methods. These comments are intended to assist in future research by pointing out possible strengths and weaknesses of the methods. Before any conclusions should be drawn about the "best" method for a particular application, empirical data must be obtained comparing the performance of the various algorithms under realistic conditions. Without such data it is difficult to assess the relative significance or insignificance of particular factors.

In the following, it should be kept in mind that for the graphs considered in this report, $O(e) = O(n)$ and $O(d) = O(1)$. (See section IIIB.)

A. Communications Requirements

The Warner-Spragins methods and the Wech method require each unit to flood messages to all other units periodically. In both of the Warner-Spragins methods these flooded messages are of length $O(1)$, resulting in a total communication load of (n units sending) · (e edges traversed by each message) = $O(n \cdot e) = O(n^2)$. In the Rapid Reciprocal Path Search Algorithm, two-way paths are used whenever possible, which may cause more congestion than necessary if alternative one-way paths are available. If no two-way paths are available to a destination, the network is subject to partial flooding of directed messages. This is the only method which requires flooding of normal directed messages. Such flooding could result in severe congestion problems. (See Figure 2.) In the other Warner-Spragins method, additional "delay vector" messages of length $O(n)$ are flooded out by every one-way link receiver, increasing the order of the comm load to some value between n^2 and n^3.

In the Wech method, the periodic messages are of length $O(n)$, resulting in a communications load of $O(n^3)$. Note, however, that the n elements of a message are port numbers, which can be very short, since units are not expected to have a large number of ports.

In the Brayer method, the communication load is mainly in the form of path records which are attached to all messages. Whether a given path record represents a large or small overhead in a given system depends on the relative sizes of the path record vs. the rest of the message. The average size of path data is proportional to the average path length of a message, which grows very slowly as the number of units increases. (For a network with uniform connectivity and uniform traffic, the average path length increases roughly as log n [7].) When changes occur in the network, loading becomes larger due to the longer path lengths of messages randomly searching out new paths or taking nonoptimal paths. Messages looking for non-existant destinations, and link-out messages (which visit every unit) can accumulate extremely long paths. These effects, however, are difficult to quantify.

In the proposed routing method, initial model building requires flooding an "update" message of d items from every unit. This is a total communication load of (n units sending) (e links traversed per message) (d items per message) = $O(n \cdot e \cdot d) = O(n^2)$. After this initialization, however, repeated frequent flooding by all units is <u>not</u> necessary. When links or units fail, only the direct neighbors of the failure need to generate updates. Flooded updates from all units are needed only during initial system start-up, or when two networks are joined. Periodic

updates may be used as an extra reliability measure, but intervals between these updates can be extremely long.

B. Storage Requirements

The Warner-Spragins and Wech methods require tables in each unit of size $O(n \cdot d) = O(n)$. The proposed method requires more space (since a system model must be stored) but the size order is still $O(n+e) = O(n)$. The matricies used in the Brayer method are of size $O(n^2)$.

C. Computation Requirements

The algorithms used in the Warner-Spragins methods are of complexity $O(d) = O(1)$, except for the algorithm for accepting delay vectors, which is of complexity $O(n)$. The complexity of the algorithms for the Wech method is $O(1)$, for the Brayer method approximately $O(n^2 \log n)$, and for the proposed method $O(n)$.

D. Performance in an Unchanging Network

Warner-Spragins Rapid Reciprocal Path Search method always uses the shortest two-way path if one exists. If no two-way path exists, the message is flooded out until it reaches units which do have a two-way path (if any). These policies can cause unnecessary congestion.

Warner-Spragins' "Delay Vector" method, the Wech method, and the proposed method, all use the shortest available path, be it one-way or two-way.

The Brayer method has not been tested in systems with one-way links, and there is reason to suspect that the performance of the algorithm will be significantly degraded in such networks. Since each unit gathers network information from paths of arriving messages, each unit mainly

finds out about the paths leading towards itself. If these cannot be assumed to be two-way paths, the information is of little value. The only way a unit x can find out about paths leading away from itself is through randomly routed messages which happen to traverse a path away from x, and then wander back through x. It may take considerable time for randomly routed messages to loop back to x, and it seems unlikely that such messages would ever give x a complete map of the system.

One modification to the Brayer method that might give some performance improvement would be to include in each acknowledgement message the entire forward path. Such acknowledgements would provide another way for units to learn about their outward paths.

E. Response to Network Changes

In both the Warner-Spragins and Wech methods, the time required to adapt to network changes is governed by the rate of the periodic updates. In the Wech method, changing unit x's route to unit y entails the following steps:

(1) Unit x floods an update (average delay = 1/2 update period).

(2) The update travels from x to y (delay = shortest path transit time).

(3) Unit y floods an update (immediatley upon observing a change in the experience vector).

(4) The update travels from y to x (delay = shortest path transit time).

Thus the total average delay is 1/2 update period plus the round trip shortest path delay from x to y and back to x. Note that corrections in

x's routes to units far from x take a relatively long time even if a network change occurs quite close to x.

In the Warner-Spragins methods, correction of a two-way route from x to y requires at least the following steps:

(1) Unit y floods an update (average delay = 1/2 update period).

(2) The update travels from y to x (delay = shortest path transit time).

The total is 1/2 update period plus the one-way path delay from y to x. However, due to the "gradual learning" method employed, for table modifications, the first update received at x may not cause a change in routing. One or two additional update cycles may be required before the new information actually takes effect.

With the "delay vector" algorithm, routes through a single one-way link are constructed quickly--as soon as the flooded delay vector arrives from the one-way link. It may take multiple flood interations, however, to find paths through multiple one-way links.

The ability of the Brayer method to adapt to changes is difficult to predict. Large delays are possible if it becomes necessary to send out a randomly routed message looking for a path. It is also possible that, due to incomplete random exploration, units will not always take advantage of new paths when they appear.

The proposed method is expected to adapt to network changes very quickly. The units first detecting network changes flood the new information to all other units. Thus the time before unit x adapts to a network change is limited only by the message transmission time from the damaged

unit or link to unit x. This is a fundamental limit regardless of routing method.

The proposed method is expected to produce minimal overhead traffic when network failures occur. After a failure, units never send out flooded or wandering messages to search out new paths, since each unit already has all the current information about alternative paths in its model. Upon being informed of a failure, a unit will either derive a new path from its existing system model, or conclude that no path exists. The system-wide information gathered by each unit before failures effectively reduced the unit's need for new information immediately after a failure. This may be an important advantage in TACS applications.

Another advantage of the proposed method is its minimal dependence upon time-outs. The methods of purging old information and adding new information to the system model do not depend upon messages arriving within critical time-out intervals. There is no need for readjustment of routing parameters according to the size of the network. The only time-outs used are fine tuning measures to help avoid unnecessary updates:

(1) Generation of stimulated updates is postponed for a short time after one update has been sent.

(2) New nodes may remain in unit u's model for a short "expire-time" even if the nodes have no route to u. This delay is to compensate for the fact that updates may arrive out of order, and it should be a little longer than delay (1). The lengths of these time-outs are not critical to proper routing operation.

VIII. RECOMMENDATIONS

A number of aspects of routing need further work. Most of the routing algorithms studied here include, or can be extended to include, methods for adapting routing to local traffic loads. The effectiveness of these adaptive schemes needs further study. Policies for flow control and task scheduling should be studied along with routing, since these mechanisms interact [8]. For routing in very large networks, some sort of "regional" routing scheme might be considered, wherein messages are deliverd first to the proper region, then to the proper unit within the region.

As previously mentioned, empirical data is needed in order to make meaningful comparisons between routing algorithms. Such data could be obtained through careful computer simulations. It is important in such simulations that representative size, topology, data rate, traffic patterns, and damage patterns be used. A simulation program partially constructed for this project could serve as a starting point.

Many larger issues involved in the design of a distributed TACS network also need to be addressed. The following are only a few:

- Routing methods deal mainly with the task of delivering a message to a specified nodal unit. The problems of determining the unit in which a particular destination process resides also needs attention. (Brayer has considered this in his routing scheme.)

- The quantities, types, sources, and destinations of information to be sent are of critical importance in designing a communication system. Better characterization of future TACS communication loads are urgently needed.

- Possible network control methods (store and forward, packet switching, broadcast, etc.) for the TACS network need further study. The different types of information involved in TACS command and control impose conflicting demands on the network, making these design decisions very complex. Hence, appropriately focussed research in all of these areas seems indicated.

REFERENCES

1. M. Schwartz and T. E. Stern, "Routing Techniques Used in Computer Communication Networks," IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980, pp. 539-552.

2. P. M. Merlin and A. Segall, "A Failsafe Distributed Routing Protocol," IEEE Transactions on Communications, Vol. COM-27, No. 9, September 1979, pp. 1280-1287.

3. J. M. Jaffe and F. H. Moss, "A Responsive Distributed Routing Algorithm for Computer Networks," IEEE Transactions on Communications, Vol. COM-30, No. 7, July 1982, pp. 1758-1762.

4. C. J. Warner and J. D. Spragins, "Simulation of Communications Protocols for a Tactical Radar Network," Final Report for AFOSR Grant 78-36/9, Directorate of Mathematical and Information Sciences, Bolling AFB, August 1979.

5. K. Brayer, "Implementation and Performance of Survivable Computer Communication with Autonomous Decentralized Control," IEEE Communications Magazine, Vol. 21, No. 4, July 1983, pp. 34-41.

6. L. Fratta, M. Gerla and L. Kleinrock, "The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design," Networks, Vol. 3, No. 2, 1973, pp 97-133.

7. J. M. McQuillan, I. Richer, and E. C. Rosen, "The New Routing Algorithm for the ARPANET," IEEE Transactions on Communications, Vol. COM-28, No. 5, May 1980, pp. 711-719.

PREVIOUS PAGE IS BLANK

8. H. Rudin and H. Mueller, "Dynamic Routing and Flow Control," IEEE Transactions on Communications, Vol. COM-28, No. 7, July 1980, pp. 1030-1039.

9. T. P. Yum, "The Design and Analysis of a Semidynamioc Deterministic Routing Rule," IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 498-504.

10. W. Chou, A. W. Bragg, and A. A. Nilsson, "The Need for Adaptive Routing in the Chaotic and Unbalanced Traffic Environment," IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 481-490.

APPENDIX — COMPUTER SIMULATION

A. Introduction

An interactive computer simulation program has been written to demonstrate that the proposed routing algorithm does work and to facilitate study of network behavior with various topologies, damage, patterns, and traffic loads. An attempt has been made to write the program in such a fashion that it would be easy to substitute other routing algorithms instead of the one proposed here. Eventually it is hoped that the simulation might be used for direct comparisons between routing algorithms.

The program enables the user to specify system units and the links among these units interactively. The user can then run as many simulation steps as desired and see the resulting route tables and message queues. Links and units can be added or destroyed at any time and the network can be monitored as it adapts to the changes.

Currently the simulation generates only routing messages, but the program contains provisions for addition of other types of message traffic.

The following sections describe the routing simulation in more detail. In Section B, the data structures used by the simulation are reviewed. Section C describes the types of messages that can be sent in the simualted system, and Section D contains psuedocode versions of the procedures used for routing.

B. Data Structures

A system is modeled as a set of "units" and "links." Associated with each unit are a table of up to "MaxPorts" outward links and another table

of up to "MaxPorts" inward links. Associated with each link are a sender unit, a receiver unit, and two queues of messages. The "SendQ" contains messages which have been assigned to the link by its sender and which are waiting to be sent. The "RecvQ" represents messages which are in transit (i.e., sent but not yet received). The send rate and the transit delay of links can be individually specified.

Each timestep is modeled by the following steps:

(1) Send: Each outward link of each unit transfers as many messages as its send rate allows from its SendQ to its RecvQ.

(2) Receive: Each unit accepts messages from the RecvQs of its inward links. All messages scheduled to arrive in the current timestep are accepted. The action taken upon receipt of a message depends upon the message destination, type, and content.

Each unit "u" has a "model" of the network in graph form, indicating all the units and links of the system which, according to u's most recent information, are operational. In the model, representations of units are referred to as "nodes" and representations of links are referred to as "arcs." Using this graph model, u can quickly determine, for any node y, the set of units to which y can directly send (referred to as S(y)—successors of y), and the set of nodes that can send directly to y (referred to as P(y)—predecessors of y). Each node y in u's model also includes a field to hold the port that unit u has calculated to be its best outward port for sending messages to y. This field is referred to as "Route (y)" and is filled in by the "Reroute" procedure. Each node also has a field which can be "marked" or "unmarked" (initially unmarked).

UNIT STRUCTURE

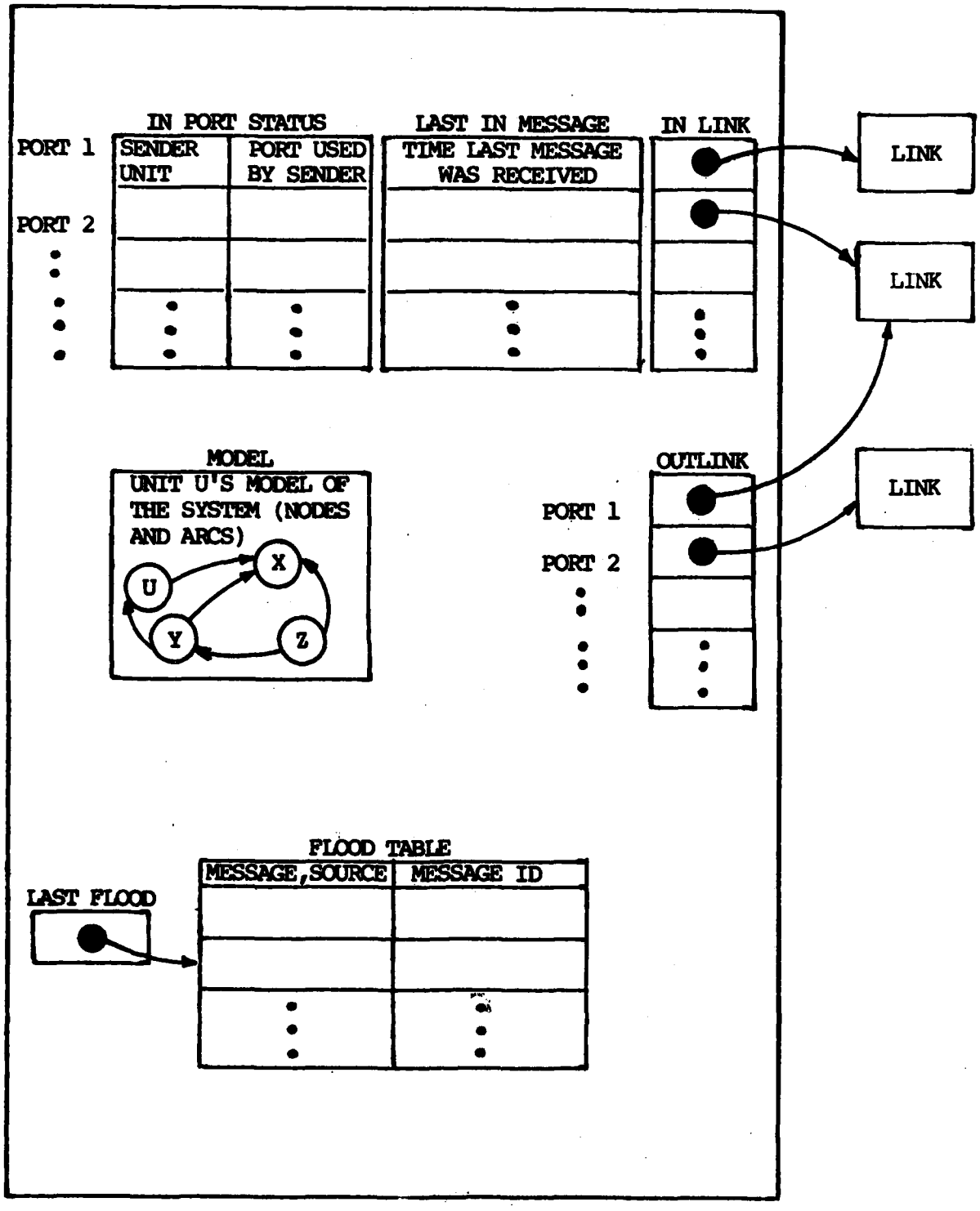


FIGURE A-1

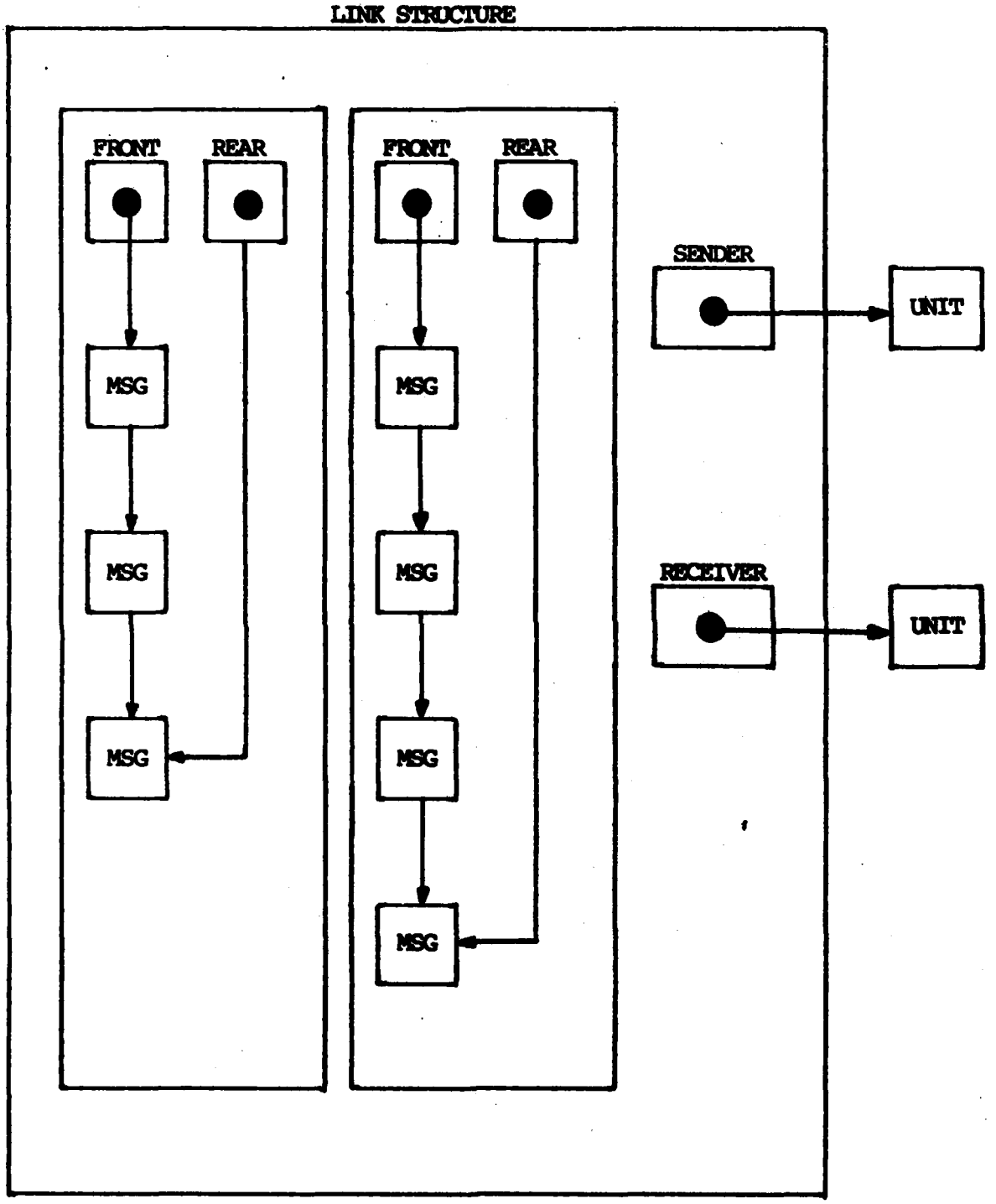

LINK STRUCTURE
FRONT
REAR
FRONT
REAR
SENDER
UNIT
MSG
MSG
MSG
MSG
MSG
MSG
RECEIVER
UNIT

FIGURE A-2

When unit u is first created, its system model contains only the node representing u itself. Unit u builds its model through use of update messages sent from other units.

u has an "InPortStatus" table indicating, for each incoming port "p", the unit currently sending to port p (referred to as Sender(p)), and the port used by Sender (p) to send to u (referred to as SenderPort (p)). The InPortStatus table is initially empty. u keeps track of when the last message was received on each inward port in the LastInMesg table. u also maintains a FloodTable indicating the source and message ID's of recently received flooded messages.

C. Messages

Three major types of message addressing are provided for by the simulation program: Directed messages, Flooded messages, and Hello messages.

A Directed message includes a "Destination" field identifying the unit to which the message is to be delivered. When a Directed message arrives at u and its destination is $y \neq u$, u send the message out over the port Route(y). It is expected that much of the network traffic would consist of Directed messages. The routing algorithm itself, however, uses no Directed messages.

Flooded messages are addressed to "All." When a flooded message arrives at u, u first searches its flood table to see if another copy of the message has already arrived. If not, then u sends the Flooded message out of all its output ports (except the port back to the unit which just sent the message to u, if any). u also processes the message itself.

Each unit informs the rest of the system of changes in its inward links through flooded "update" messages. An update from some unit x contains a copy of x's InPortStatus table, that is, the message tells which units are currently sending to x and the ports used by those units.

Hello messages are addressed to "Any" unit, and when received are not passed on. They are only used to provide test messages on links which would otherwise be idle.

All messages contain the following fields:

LastSender -- the unit which most recently sent the message.

LastOutPort -- the output port used by LastSender to send the message. These fields, of course, are changed each time the message is forwarded over another link.

Messages also include a priority rating, so that high priority messages will traverse the network faster.

D. Routing Procedures

In this section, the procedures used for routing in the simulated network are described in informal psuedocode. Only routing procedures are given, not other procedures involved in simulating and modifying the network. For more detail, see the program listing.

All procedures are expressed from the point of view of a particular unit "u." The procedures are the same for every other unit.

Routing procedures are called in two ways. Some routines are called as part of the process of handling any incoming message. Other procedures are called at every timestep, regardless of traffic. Procedures for handling an incoming message are described first.

Whenever a message "Mesg" arrives on port "Port" of unit u,

RecvMesg is called:

```
procedure RecvMesg(Msg,Port);

begin

    record the current time in LastInMesg(Port) to show when port was
    most recently used;

    if Sender and Port fields of Mesg do not match Sender and
        SenderPort fields of InPortStatus table

    then begin
        place new values in InPortStatus(Port);

        UpdateToBeSent = true;  (*This will cause an update to be flooded
                                  to all other units.  See Receive
                                  procedure.*)
        update u's own model to show new port data;
end;

duplicate = false;

if Mesg destination is "All"

then if the message is not in the flood table

    then begin

        replace the oldest entry in the flood table with the
        message Sender and ID;

        send copies of Mesg to all adjacent units except the
        one from which the message came;

    end

    else duplicate = true;

end;

if duplicate

then dispose of Mesg

else
```

```
    if Mesg destination is "u", "Any", or All"

    then case Mesg type of

        Hello: dispose of Mesg;

        Update: call RecvUpdate(Mesg);

            .
            .
            .

           (*Similarly, other message types would result in
           calls to appropriate handling routines.*)

    end

    else begin

        let y = destination of Mesg;

        send Mesg along port specified in Route(y);

    end;

end; (* end of RecvMesg*)

procedure RecvUpdate(Mesg);

begin

    let the source of Mesg be called y;

    find the node of u's model that represents y;

    if no such node exists in current model

    then begin

        add a new mode y to the model;

        set the expiration time of y to current time + ExpireTime;
            (so that y will not be removed immediately as an
            isolated node)

end;

compare the inward arcs of node y to the set of links reported in
Mesg;
```

```
if any new links are reported in Mesg
then for each new link do begin
    let x be the sender of the new link;
    if x is not in u's model
    then add a new node x to the model;
    add an arc from x to y;
    if this is the first inward arc for y  (*Rule 3 for producing updates*)
    then UpdateToBeSent = true;
end;
if any arcs in the model are not in Mesg
then for each such arc do begin
    let x be the sender of the old arc;
    delete the arc from x to y;
    PruningNeeded = true; (*This will cause a search for isolated
                           nodes which should be removed.  See
                           Receive procedure*)
end;
if any changes were made in the model
then RerouteNeeded = true;
if any link listed in the update is from unit u
then copy the corresponding SenderPort listed in the update
    into Route(y);
end; (*end of RecvUpdate*)
```

The following "Receive" procedure is executed at every simulation step. the procedures that follow are all called by this procedure.

```
procedure Receive

begin for each of u's input ports begin

    if the InPortStatus table indicates that the port is active
    and if LastInMesg(port) indicates that there has been no
    traffic for a period of "HellotimeLim"

    then begin

        change the InPortStatus table entry to null;

        UpdateToBeSent = true;

        SelfUpdateNeeded = true;

    end;

end;

if SelfUpdateNeeded

then begin

    construct an update message "Mesg" containing the
    InPortStatus table;

    call RecvUpdate(Mesg);

    SelfUpdateNeeded = false;

end;

if PruningNeeded

then call RemoveIsolates;

if RerouteNeeded

then call Reroute;

if (an update has not been sent for MaxUpdateInterval)
or (UpdateToBeSent = true and MinUpdateInterval has elapsed
since the last update was sent)
```

```
then begin

    construct an update message containing the InPortStatus table;
    flood this message to all units;

end;

    for each outward link from u

        if there are no messages queued on this link

        then send a "Hello" message;

end; (* end of Receive *)

procedure RemoveIsolates

begin

    (* This procedure uses a list of nodes called the JustReached list.

     Recall that P(x) is the set of nodes that have arcs to x and S(x)

     is the set of nodes that have arcs from x.  Also, all nodes are

     initially "unmarked." *)

    initialize the JustReached list to contain only the node
    representing u;

    mark node u;

    while the JustReached list is not empty do begin

        remove a node x from the JustReached list;

        for each node y in P(x) do

            if y is not yet marked

            then begin

                mark y;

                add y to the JustReached list;

            end;

end;        (* All nodes that can reach u (according to u's model) are
            now marked. *)
```

```
for each unmarked node z

    if z has passed its expire time

    then begin

        delete all arcs to and from z from the model
        delete z from the model;
        RerouteNeeded = true;

    else set Route(z) to be "Suspended";

    remove marks from all nodes;

    PruningNeeded = false;

end; (* end of RemoveIsolates *)

procedure Reroute;

begin
    (* This procedure uses a queue of nodes called Q.  Note that
    all nodes are initially unmarked.*)

    mark node u;

    mark the nodes in S(u);

    initialize Q to be empty

    for each node x in S(u) do

        if Route(x) is not "Suspended"

        then add x to Q;

    while Q is not empty do begin

        remove a node x from Q;

        for each node y in S(x) do

            if y is not marked

            then if Route(y) is "Suspended"

                then PruningNeeded = true

                else begin
```

```
                    if Route(y) = null (* meaning there was previously
                                          no route to y*)

                    then UpdateToBeSent = true; (*Rule 4 for producing
                                                  updates*)

                    Route(y) = Route(x);
                    mark node y;
                    add y to Q;
                end
            end
        end
    end;

    for each node x in the model

        if x is marked

        then unmark x

        else Route(x) = null;

        end;

        RerouteNeeded = false;

    end; (* end of Reroute *)
```

DISTRIBUTION LIST

HQ USAF/XOORC
Washington, D.C. 20330 (1)

Air Force Systems Command
Andrews AFB, MD 20332
AFSC/XR (1)
AFSC/XRK (2)

Electronic Systems Division
Hanscom AFB, MA 01731
AFGL/SULR (3)
AFGL/SULL (1)
ESD/CC (1)
ESD/DC (1)
ESD/IN (1)
ESD/OC (1)
ESD/TC (2)
ESD/TO (1)
ESD/YW (1)
ESD/XR (2)
ESD/XRC (2)
ESD/XRX (2)
ESD/XRT (10)
ESD/XRW (1)

ESD DET 9
APO NY 09021 (2)

Rome Air Development Center
Griffiss AFB, NY 13441
RADC/CC (1)
RADC/CA (2)
RADC/DC (2)
RADC/CO (2)
RADC/OC (2)
RADC/XP (2)

HQ ESC/XOX
Kelly AFB
San Antonio, TX 78243 (2)

Tactical Air Command
Langely AFB, VA 23665
TAFIG/II (2)
TAC/DRC (2)

Tactical Air Command Systems Office
Hanscom AFB, MA 01731
TACSO-E (2)

The MITRE Corporation
Bedford Operations
P.O. Box 208
Bedford, MA 01730
ATIN: Mr Norm Briggs (2)

DARPA/ITPO
1400 Wilson Blvd
Arlington, VA 22209 (1)

USA/CECOM
Ft. Monmouth, NJ 07703 (1)

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314 (12)

AU Library
Maxwell AFB, AL 36112 (1)

PREVIOUS PAGE IS BLANK