NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A DISTRIBUTED ROUTING PROTOCOL
FOR A PACKET RADIO NETWORK

by

Robert Heritsch

March 1982

Thesis Advisor: J. Wozencraft

Approved for public release, distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. A118271 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Distributed Routing Protocol for a Packet Radio Network	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis March 1982
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert R. Heritsch	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 39490	12. REPORT DATE March 1982
	13. NUMBER OF PAGES 167
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report)
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Packet Radio, Routing Protocol, Distributed Routing Algorithm

Digital Communications, Network Management

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

Packet Radio is a digital communications concept which offers the user the capability to pass voice and other data in a radio network which may link high power computers with small mobile radios containing microprocessors. The technique of routing digital traffic from source to destination depends on the operational requirements of the network. Most routing concepts today centralize network control (in varying degrees) for normal

operations. This thesis describes a concept for completely decentralized control of a packet radio network. The basic protocol is relatively simple and robust, but suffers from the usual build-up of overhead traffic with network size. Another related routing protocol is proposed which, under certain operational situations, reduces routing traffic and memory requirements compared to the basic algorithm. A concept for use of alternate links in the event of a broken link is also suggested.

Approved for public release; distribution unlimited

A Distributed Routing Protocol for a Packet Radio Network

by

Robert Heritsch
Major, United States Army
B.S., University of Wisconsin-Milwaukee, 1969

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1982

Author Robert Heritsch

Approved by: John M. Wozencraft
Thesis Advisor

Kenneth J. Moore
Second Reader

Robert D. Strum
Chairman, Department of Electrical Engineering

William M. Tolles
Dean of Science and Engineering

ABSTRACT

Packet Radio is a digital communications concept which offers the user the capability to pass voice and other data traffic in a radio network which may link high power computers with small mobile radios containing microprocessors. The technique of routing digital traffic from source to destination depends on the operational requirements of the network. Most routing concepts today centralize network control (in varying degrees) for normal operations. This thesis describes a concept for completely decentralized control of a packet radio network. The basic protocol is relatively simple and robust, but suffers from the usual build-up of overhead traffic with network size. Another related routing protocol is proposed which, under certain operational situations, reduces routing traffic and memory requirements compared to the basic algorithm. A concept for use of alternate links in the event of a broken link is also suggested.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

The opportunity to work closely with Prof. Wozencraft, who patiently placed many of these concepts in my mind, was truly the high point of my studies at NPS. The constant support and understanding of my wife Sandy, who placed most of these concepts in print, has helped make this entire period of study richly rewarding and enjoyable.

I. INTRODUCTION

A. THE PACKET RADIO CONCEPT

Packet Radio technology extends the application of packet switching into the mobile radio environment. It offers a convenient and efficient way to communicate among a large number of mobile users. This is particularly important in a tactical environment where rapid deployment and mobility are required.

Users in a packet radio network essentially share common radio channels. Use of these channels is (to varying degrees) controlled by microprocessors in the user's radio in a manner which is transparent to the user. Packet Radio is a digital communications concept which in principle can accommodate voice as well as digital data traffic provided that adequate traffic capacity is available. The use in packet radio of spread spectrum communications is particularly attractive to military applications because of potential capabilities for a low probability of intercept (LPI) and excellent antijamming (AJ) characteristics.

Although the military is pressing development in packet radio technology, it is also, in a sense, part of the natural evolution of the computer age. Almost all computer

networks are bound to the cables which connect the computers. Yet as computers get smaller and potentially more mobile, the need for wireless links become more important.

Two packet radio network testbeds are currently in operation. The Bay Area PRNET (packet radio network) in San Francisco has been operational since 1976, and is the primary site for development and evaluation of network protocols and application concepts. The Army Data Distribution System (ADDS) testbed PRNET at Ft. Bragg, North Carolina, became operational in 1979 with the objectives of providing potential users of packet radio technology with exposure to the technology early in its development, giving timely feedback to developors and offering the users an opportunity to experimentally determine the impact on tactical doctrine of mobile access to computer-based command and control.

B. ROUTING

The goal of a properly operating packet radio network is to route packets (groups of bits) thru a series of radios from the sender to the receiver in an efficient manner. Enroute, the packet is automatically processed and passed on

by a series of radios in a manner transparent to those users. Through multiplexing or other concepts, a radio may provide input/output service to its user and also forward other traffic simutaneously. Because of the limited power of mobile radios, a transmitter may often not have a direct link with the ultimate receiver. In a military application, low power transmissions may also enhance survivability. Therefore the routing from every potential message source to every potential destination requires the application of a network-wide intelligence to determine the most efficient links along which to forward the message. There are two main, different approaches to routing algorithms for solving this problem.

1. Centralized And Backbone Systems

Both the Bay Area PRNET and the Ft. Bragg PRNET use network components called stations to manage the routing in different portions of the network. In the Ft. Bragg network, each station is a (DEC) PDP 11/40. The purpose of the station is to monitor the relative activity level in each radio under its jurisdiction, and to aid in the routing of traffic that passes thru, originates or terminates in its portion of the network.

There may be many stations in a network, each controlling a certain number of user radios. Together they provide the network-wide intelligence which maintains and implements the dynamic routing scenerio. Network control is centralized in the stations. This scheme is both practical and efficient. However in a military sense, stations are a vulnerability since only a few of them control the operation of all the radios in the network.

Use of a backbone network offers similar advantages and shortcomings. A backbone is a network superimposed over a common user network which improves the efficiency of the network by providing high volume, high speed and/or long distance trunks. Traffic from the common user is placed on and taken off of the backbone in accordance with a routing process such as the station concept mentioned above. Once again, the vulnerability of the network is directly related to the vulnerability of the backbone.

2. Completely Decentralized Routing

A completely decentralized network does not have stations or a backbone. Conceivably, every user has a packet radio containing a microprocessor which is no different than any other communications/processing component

(other packet radios) in the network. Depending on the topographical situation, there may also be unattended packet radios in the network. These radios do not have users which use the radio as a terminal into and out of the network. They are usually placed in positions in the network to provide additional communication paths or links increasing the number of routing alternatives. However these unattended radios function essentially the same as a terminal user's radio insofar as message processing is concerned. In other words, in a decentralized network, every radio is the same and there is no centralized or semi-centralized component controlling how the network operates. It is the collection of packet radios themselves which must combine their processing capabilities to create the network-wide intelligence needed to build, maintain and implement an efficient routing scheme for user traffic. The advantage is a reduction in the vulnerabilities inherent in any system which tends to centralize its control capabilities. The disadvantages are increased complexity, increased overhead traffic (which represents competition with user traffic for a finite channel capacity), and possibly a reduction in speed.

The objective of this study is to present and investigate the performance of a simple algorithm which could be programmed into each packet radio in a completely decentralized network. Assuming that each radio in the network has a very limited range compared to the diameter of the network, the algorithm allows each radio to relay information about other radios (called nodes from now on) throughout the network. The algorithm uses this information, as it works its way through the network, to create relatively efficient communication paths (links) between every pair of radios (nodes) in the network. The end result automatically gives users throughout the network the appearance of direct access to every other node in the network, albeit with some delay. The dynamic routing of traffic as it is created and enters the network enables many channels of communications to exist simutaneously across the network.

II. NETWORK MODELING

Although this study is based on what is considered a practical concept for a military radio network, the theory can be considered very general in nature. Therefore, the network is modeled as a combination of nodes and links between nodes. Furthermore, the nodes and links are affected dynamically by events such as routine traffic, the gain or loss of a link, and network maintenance traffic. This chapter defines the modeling components and functions, relates them to physical components or requirements, and makes some assumptions.

A. NODES

In the model, nodes represent receiver-transmitters. Nodes also contain processors. It is convenient to picture many functions in each node performed by parallel processors so that all unrelated processing can be performed simutaneously. Conversely, only those operations which must be performed in a sequence with a significant execution time are subject to conflicts and queuing delays.

All nodes in the network have exactly the same capabilities. However, depending on its processing

instructions, each node may process a given message differently. For example, one node may be a terminal for a specific user. Therefore, this node may accept routine traffic for a certain list of addresses, determine which traffic is addressed to its assigned user, deliver that traffic, and retransmit the remainder to the appropriate addresses. Another node may be solely a transmitter which only relays routine traffic and does not serve as a terminal for a user.

When one node can pass traffic directly to another node, the other node is considered a neighbor to the first node. These nodes are connected by a link. Although every node in our network can contact every other node, each node has only a limited list of neighbors which may vary with time.

B. LINKS

A link exists whenever two nodes are in direct contact with each other. A link is considered broken when one or both nodes lose the capability to transmit to, or receive from, the other node. Therefore, a link implies two-way communications between specific node pairs. Of course the actual method of communications in a radio network is through antenna transmissions. These may be either

directional or omni-directional antennas. And of course, these transmissions could potentially be received by many nodes other than a particular partner in a node pair. Conceptually, this can be accommodated by assuming that all traffic/packets contain the address of the intended receiving node for a given link. Then any node which receives traffic not addressed to it simply ignores the message.

Another more sophisticated concept has a link representing a unique center frequency which one node uses to transmit to another. In creating the link, the two nodes determine which frequency bands are mutually available, and then each selects an available transmission frequency to communicate with the other node. Now, when either node wishes to transmit to the other, it uses its selected frequency band. Conceptually, only one node within range of a given transmitter will accept traffic in a particular frequency band. In this manner more than one link to a single node may be operating simutaneously. Other more familiar techniques such as Code Division Multiplexing could also be used to establish a link.

C. CHANNEL VALUE

Assuming that a network consisting of many links has been established, one needs an efficient way to use this network. Clearly, an unacceptable technique would be to retransmit every message on every link to ensure that the addressee receives the message. Although it may ensure that a single message gets to its destination, it represents work for every node in the network. Assuming that different messages could be initiated by many nodes in the network, and that much of this traffic could be present in the network at the same time. The inefficiencies of broadcasting quickly lead to saturating nodes or links in the network, since nodes indiscriminantly relay everything they hear. Smart nodes should be able to do much better.

What is needed is a way of selecting one link over another link. Once that decision is made, traffic for a given destination uses only the best path, or optimum series of links, from the source of a message to its destination. One way to quantify the connection between two nodes is to assign a weight or cost value to each link or channel in the network. Then, summing the costs for a given path between two nodes, one can assign a value to every

possible path, and thereby (theoretically) pick the lowest cost path between a source and destination.

There may be many ways to assign channel values. One practical technique would be to count the backlog of traffic (or packets) waiting to use a particular link. This queue or delay represents a portion of the total time it takes for a message to reach its destination. Normally it is desired that traffic move through the network as quickly as possible. This is particularly important if the network is to accommodate real-time speech. Therefore, a channel value which reflects net transmission time is useful. This is the technique used in this study.

D. NETWORK DYNAMICS

Nodes and links represent the static network structure. But a practical network must accommodate changes which may be represented as the creation or destruction of nodes or links. Furthermore, there must be a concept for passing network maintenance information and, most importantly, user traffic.

1. Routine or User Traffic

A network exists to pass routine traffic. Traffic could be either inter-active voice (characterized by

real-time conversations), or data (characterized by one-way transmissions assembled or stored at the receiving end for later review).

In a digital network, both voice and data traffic are transmitted in the form of digital packets. For voice traffic, the most important thing is that packets arrive at a relatively uniform rate. Voice packets are created by sampling the voice signal. The number of voice bits required per unit time is a function of the encoding technique and the desired quality of the received signal. Any additional bits are unnecessary and therefore waste channel capacity. Fewer bits, in the form of delayed or lost voice packets, may degrade the reception. Note that once a voice packet is delayed one inter-packet period,it is no longer useful.

For data traffic, it is not necessary to have a smooth flow of traffic. Bursty traffic is quite acceptable. The important thing is that after the message is divided into packets for transmission from the source node, all these packets are recovered and reassembled properly at the destination node to recreate the original message.

The ability for data packets to move satisfactorily in a bursty manner allows them to complement the rigid schedule of voice packets. A concept for the integration of voice and data traffic is discussed in more detail in Chapter III.

2. Broken Links

As defined earlier, a link implies the capability for two-way communications between two nodes. A broken link is recognized in a node when it is discovered that this two-way capability no longer exists. Depending on the situation, as explained in Chapter III, the two nodes on each side of a link may realize a link is broken at different times.

In modeling a network, a broken link may be used to represent various events. If a node is lost, it could be reflected as a broken link between the lost node and each of its neighbors. If the transmission path between neighbors is interrupted, this can be represented as a loss of a single link between the two nodes. If links are broken in a particular pattern, it may indicate that a particular node is moving away from its neighbors.

3. New Links

As opposed to a broken link, a new link is created when the nodes on each end establish communication with each other. This will typically require some interaction between the two nodes.

New links would be created when an inactive node becomes active, when a moving node moves into range of other nodes, or when other conditions change to enable two-way communications between two nodes where conditions previously prevented this link.

It is apparent that a network can be dynamically modeled by allowing links to be broken or created to represent physical activities such as changing signal paths, nodes entering and leaving the network (being turned on or off), node movements and other situations.

4. Network Maintenance Traffic

If the nodes in a network are to be as organized and resourceful as described above, then they must be programmed to communicate with each other, passing information related to their activity and capabilities. In a network with fully distributed control, the objective is to achieve efficient network-wide communication under the constraint that each

node can only transmit and receive directly with a limited number of neighbors. There is no central control facility to route and monitor traffic between non-adjacent nodes.

Every user in the network must be able to reach every other user in the network in a manner which is transparent to all users, even in a dynamic environment where links are created or broken randomly. Therefore, over and above user traffic, nodes must pass network maintenance traffic. This traffic should be transparent to the user. This means that the nodes measure or sense their operational status and are programmed to automatically report information to their neighbors. Neighbors process the information and may then automatically relay the processed information to selected neighbors until every node requiring the information eventually receives it. A program or algorithm that generates and processes network maintenance traffic is commonly called a protocol. At a minimum, to model a practical network, network maintenance traffic must accommodate new links, broken links, and changes in channel values which may represent more efficient ways of routing routine traffic through the network. The concept of protocols for distributed networks is discussed in much greater detail in Chapter III.

III. LEVELS OF DISTRIBUTED PROTOCOL

A distributed protocol in a packet radio network is defined as algorithms which are executed independently in each node to process both network maintenance and routine traffic. The effect should be the overall efficient use of network resources, approaching the efficiency of a centrally controlled network.

A particular protocol may be based on many design considerations. Of course the designer must consider the capabilities of available or proposed equipment and the characteristics of the operating medium. But within these constraints, the designer may be free to trade off such things as simplicity and robustness for speed and sophistication. And of course these qualities are not mutually exclusive. Therefore, the examples of distributed protocols in the literature vary from rather limited, simple ones such as Yen's algorithm [Ref. 1], to more sophisticated and complicated algorithms such as Segall's [Ref. 2].

It may be helpful to break down network operations performed by each node into three groups or levels of protocol. In this way activities can be isolated,

controlled and analyzed in a modular fashion while assuming the remainder of the node's functions are unaffected and operating as expected. This study assumes three levels of protocol. The first is node to node protocol, the second is network management protocol and the third is user service protocol. Concepts and examples of node to node protocol and user service protocol are discussed in some detail in this chapter. Network management protocol is mentioned only briefly in this chapter. However, a detailed concept and example is developed and analyzed in the remainder of this study.

A. NODE TO NODE PROTOCOL

There are several activities required in an active network which basically involve only two nodes. Perhaps the most fundamental interaction is recognizing each other. This mutual recognition is considered a link. Links exist to pass traffic, which leads to another important inter-nodal function, that of the receiving node informing the sending node that it has received its message.

1. Establishing and Monitoring a Link

A node to node protocol should provide for establishing communications between two nodes. This could

be accomplished by each node asynchronously transmitting a beacon message on a designated frequency. The beacon message would contain the identity of its originator. Any other node receiving the beacon message with an adequate S/N ratio checks its list of neighbors. If the node addressed in the beacon message is not found on the receiving node's neighbor list, the receiving node would initiate an acknowledgement message addressed to the node which sent the beacon message. If the original node now receives the acknowledgement, it adds the node which sent the acknowledgement message to its neighbor list and sends that node a notice message that two way communications exist. Finally, the node which initially responded to the beacon message adds the originator of the beacon messageto its neighbor list and a new link is born.

As mentioned in Chapter II, the implementation of a link may vary by design. If, for example, the two way link actually consists of two frequency bands which enable simultaneous transmission between two nodes on a single link, then the interchange of information in establishing the link would include the determination of mutually available frequency bands. If, on the other hand, the

network used Carrier Sense Multiple Access (CSMA), which was the technique actually used in the DARPA packet radio testbed which operated in the San Francisco Bay area [Ref. 3], then different information must be passed to establish a link.

Once established, the status of a link must be monitored. The beacon message could also be used for this function. A node should expect to receive beacon messages from every node on its neighbor list. Therefore, when it receives the beacon message there is no need to reply. However, it may note the time it received the last beacon message from each of its neighbors. Failing to receive a beacon message from a neighbor over an established period of time would prompt a node to conclude that it had lost two way communications. This may have an impact on many other nodes in the network and would therefore initiate a reaction by the Network management protocol as discussed in paragraph 2 below. When a node discovers it has lost a link, the corresponding node on the other end of the link must be removed from its list of neighbors.

There is another more immediate way for a node to discover that it has lost a link. This would occur when a

node attempted to send a packet to a neighboring node but does not receive an appropriate acknowledgement for a successful transmission. If this is the case, the sending node could try to retransmit at least one more time, but eventually it may conclude that the link has been broken. Once again this may initiate activity by a higher level protocol. This also demonstrates how one node may discover that a link has been broken before it is discovered by its corresponding neighbor. In any packet radio concept, the establishment and monitoring of links is a fundamental activity that can be delegated to a low level protocol.

2. Packet Acknowledgement

Another node to node function is the acknowledgement by the receiving node to the sending node that a packet has been successfully transmitted across a link. Under any practical operating concept for a packet radio network, there are significant opportunities for a node to improperly receive a packet. A few of these situations include multipath interference, intentional or unintentional jamming, fading or improper synchronization. A conservative design consideration would preclude a transmitting node from purging a transmitted packet from its memory until it has received acknowledgement that the packet has been received.

In the ALOHA net, operated by the University of Hawaii, this acknowledgement is accomplished as the sending node monitors the retransmission of the receiving node. If the retransmission matches what was sent, the sending node eliminates the packet from its memory. If it did not, the packet is resent. This is an adequate technique for an ALOHA-type network. But if a node uses different frequencies for each link, this technique may be impractical.

Another concept is to terminate each packet with check bits. The number of check bits per packet would be a function of the expected probability of error per bit for an average link. If all check bits are properly received, the receiving node reports its successful reception to the sending node in a brief message. Lack of such a report after an established period of time may prompt a node to retransmit a packet. Receipt of an acknowledgement would cause a node to eliminate the packet from its memory, considering it successfully transmitted. There are check bit schemes for fail-safe communications which are not only more efficient than a bit-for-bit check, but are also more reliable [Ref. 4].

Other verification concepts may offer still other advantages. However, because of the inherent potential and significant effects if bit error in radio communications, it is very likely that some technique of ensuring accurate packet transmission will be required in any packet radio network.

B. NETWORK MANAGEMENT PROTOCOL

The primary objective of a communications network is to move user traffic from source to destination. A network management protocol is intended to organize the network so that traffic moves efficiently under all conditions.

If one assumes that node to node activities are appropriately handled by a lower level protocol as described above, then he can treat the loss or addition of another link as a routine event, and process the information as it would affect the entire network. Therefore, network management protocol is not concerned with how or under what conditions a link is established. It only acts on the information that a link does or does not exist.

1. Update

The fundamental network-wide management operation is the update. In an operational network, traffic on each link

is constantly changing. To efficiently use the network to pass traffic between two given nodes, it is desirable to find the "Best Path" between the two nodes. Exactly what is measured may be a subjective decision. But once made, this quantity can be used to compare various alternatives and select a best path. Yet the best path can be expected to vary with time, for loading on each link of a network may be constantly changing. Therefore best paths must be updated periodically to accommodate network dynamics.

In a distributed control network, each node could initiate its own update. The form of this update message and exactly how it is processed in the network depends on the selected protocol. There is always a design trade-off involving the frequency of updates with the corresponding generation of update messages (management traffic) versus the effects of old or outdated best paths. This tradeoff should not be a casual decision. In a network of n nodes, there are at least n(n-1) best paths. With some of the most efficient algorithms, it may take at least (n**3) node to node messages to complete one network-wide update under the worst conditions (see Appendix C). Therfore it is desirable to find an effective update frequency which provides for realistic and efficient network traffic flow.

In addition to updating existing paths, the updating process can serve to introduce new links into the network. In some protocols such as Segall's (Ref 2) the arrival of a new link has an immediate impact on the network update process. As in the case of broken links discussed next, Segall immediately initiates a new update message whenever a node experiences a change in its link status. This creates a situation where update messages initiated by the same node may be negotiating the network at the same time. Therefore there must be provisions to prioritize these messages so that the most recent message takes precedence over the outdated messages. This is normally accomplished by introducing cycle numbers as part of each update message and many other network management messages. The problem with cycle numbers is that they can potentially grow larger than the allotted buffer space. Segall places a bound on his cycle numbers by using a procedure devised by Finn [Ref. 5].

One of the objectives of this study is to investigate a network management protocol that does not require cycle numbers. In this concept new links are introduced to the network only during routine updates. It is assumed that the delay involved may be traded for increased simplicity.

2. Link Breakdown

Broken links may have various impacts on a network. If a link is under heavy use, a break may have a serious effect on net traffic flow. Heavy use may also indicate that many other nodes rely on this particular link in their best paths to other distant nodes. On the other hand, some links may serve very few nodes, and in fact be inactive at the time a break is discovered.

The objective of any reaction to a broken link is to minimize its impact on the flow of traffic and other network activity. Ideally, traffic should immediately and automatically be switched to the next best path. One way to incorporate alternate links under certain circumstances is described in Appendix A and is considered along with the proposed network management protocol in Chapter IV.

When it cannot always immediately reroute traffic, the network management protocol must take action to stop or reduce traffic intended for a broken link, cause the network to find new best paths for traffic affected by the break, or a combination of both. Finding new best paths is typically done during an update operation. It is a function of a protocol to indicate how an update may be initiated.

In some protocols discovery of a break may initiate an update. For example, the discovering node may broadcast a special update request message addressed to each destination for which the discovering node had considered the broken link as part of a best path. Other nodes echo the request and eventually the destination nodes receive their requests and initiate an update. In this concept, some type of cycle number would be required to mediate conflicts between new and outdated updates from a single destination node which could exist in the network at the same time.

Alternatively, a protocol can be designed to routinely issue updates from each node at a rate that ensures that any previously issued update message from a particular node had already passed out of the network, yet often enough to tolerate freezing traffic blocked by a broken link until the next routine update provides a new best path. This is the basis of the network management protocol proposed in Chapter IV.

C. USER SERVICE PROTOCOL

Once a network is constructed and operational, the last question is how routine user traffic will be packaged and

processed by each node in the network. This could be considered the User Service Protocol level. In this case the designer may assume that lower level protocols will work independently to do things such as update best paths, react to new or broken links, and acknowledge transmissions across a link. The User Service Protocol uses selected information from lower-level protocols to efficiently accomplish its primary function of passing user traffic.

The basic characteristic of a user service protocol in a packet radio network is that, like other lower level protocols, it should be transparent to the user. Decisions such as packet size, content, and processing depend on the capabilities of the selected equipment and the priorities of the network designer. In this section a particular algorithm is discussed as an example of a typical user service protocol. It is presented to illustrate one possible technique for managing routine traffic within the framework of a network operating with other lower level protocols.

1. Voice Traffic

Several assumptions must be made in order to gain physical appreciation of the requirements of a conceivable

packet radio network. Some of the parameters selected both here and in the remainder of this study are based on a theoretical packet radio concept propcsed for a Marine Amphibious Brigade by Bond [Ref. 6], and Lucke [Ref. 7].

It is assumed that the network will move both voice and data traffic. In this section we discuss the characteristics and requirements of each type of traffic. As mentioned in Chapter II, voice must flow at a consistent, periodic rate. Data, on the other hand, can move in bursts as channel capacity becomes available.

In a digital network, voice must be converted to a digital signal (vocoding). This is done by sampling the analog voice signal and converting each sample to a digital value. This produces a voice packet. In a real-time conversation, any delay of more than approximately 0.1 sec between speakers becomes noticeable. Therefore a voice packet should take no more than 0.1 sec to move from the source node to the destination node. Assuming that packets will be relayed by a maximum of 10 nodes in our theoretical network, and further assuming that processing time in each node is far more significant than the propagation time between nodes, then the maximum processing delay per node is

$$\text{Delay} = \frac{.1 \text{ sec}}{10 \text{ nodes}} = .01 \text{ sec/node/packet.}$$

Because of the periodicity requirements of voice, there are certain advantages to establishing a virtual link between the traffic source and destination. In a packet radio network, a virtual link may consist of reserving a time slot on each link along the best path from the source to destination at the time the virtual link is established. Once a virtual link is established, it is used until the source node has finished the voice conversation (unless a link is broken), regardless of whether or not subsequent update operations have found other best paths during the course of the conversation. This ensures periodicity in the voice traffic, for each voice packet passes thru the same number of nodes, with the same net processing time for a given source-destination pair.

In a practical network, a link would probably be required to accommodate traffic for more than one node at a time. As implied in the previous paragraph, this may be accomplished by transmitting traffic for a specific destination node in an assigned time slot on each link. This is also called time division multiplexing. The

particular slot on each link enroute to a destination is determined as the call is being initiated and the virtual link is being built. Once established, this slot will only carry voice packets for its assigned destination until the virtual link is broken or dismantled.

It is convenient to define the series of time slots which can each carry a separate virtual link as a Frame. Then in each frame, one slot represents one virtual link to a destination. During normal link operation, each frame is followed by another frame carrying the next voice packet in the assigned slot for each virtual link (see Fig. 3.1).

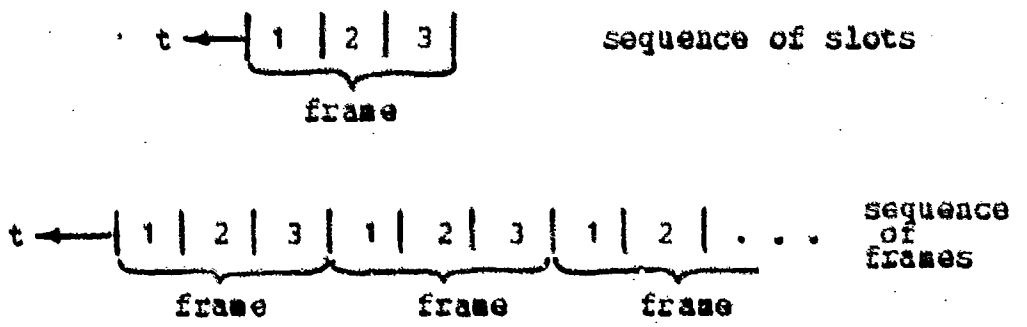


Fig. 3.1. Slot/Frame Concept

It was estimated above that if a maximum of 10 nodes were used in a virtual link, each node can take up to .01 sec to retransmit one voice packet. If it is further

assumed that each frame must handle up to 10 virtual links (slots), then each slot (which carries one voice packet) can be no more than 1 msec because each frame can last no more than .01 sec.

$$\text{Slot Duration} = \frac{.01 \text{ sec/frame}}{10 \text{ slots/frame}} = 1 \text{ msec/slot.}$$

It is estimated that good quality digital adaptive Delta-mod voice requires a bit rate of 16 x (10**3) bits/sec. In the multiplexing system mentioned above, each voice channel has only a 1/10th duty cycle. Therefore when active, a virtual link must pass traffic at a rate of 160 x (10**3) bits/sec.

For this example, if the radios in this network operate with a bandwidth of approximately 100MHz (spread spectrum), a significant post detection processing gain could be obtained

$$G = \frac{\text{Transmission Bandwidth}}{\text{Bandwidth of Message}} = \frac{10**8}{160 \text{ x } (10**3)}$$

$$= 625 = 28 \text{ dB.}$$

2. Data Traffic

Data traffic would not normally have the stringent timing requirements that voice traffic may require. On the other hand, within reason, voice traffic could afford to randomly lose packets while experiencing a graceful degradation in the actual flow of information, whereas any lost data packets represent an absolute loss of information. Therefore the network may pass data traffic more slowly, but must do so more accurately.

Because of the periodicity requirement of voice traffic, voice packets need to have priority over data packets. Under this network concept, data traffic would be integrated as a filler in available slots during pauses in voice traffic. The result is bursts of data traffic, which does not lend itself to the virtual link concept described for voice traffic. In fact it may be simpler to picture each data packet as an individual message containing the address of the destination, and being released by the source node to find its way to the destination node. One advantage of this concept is that if the network updates its best paths while a source node is releasing data packets for a particular destination, later packets have the advantage of

using the updated best paths to their destination. By contrast, in the virtual link concept considered here, once a virtual link is established, traffic is confined to it even though better paths may become available.

If data traffic is to be moved on the network developed earlier, it must be able to work within the frame/slot concept devised for voice traffic. Assuming a slot has a duration of 1 msec with a data rate of 160 x (10**3) bits/sec, then each slot contains approximately 160 bits of information. In the virtual link concept this may be perfectly acceptable because once the virtual link is established, nearly all bits passed on the virtual link are user traffic. However, if each data packet is to move independently from the source node to the destination node, each packet must contain certain overhead information which is commonly lumped together at the beginning of the packet in a preamble.

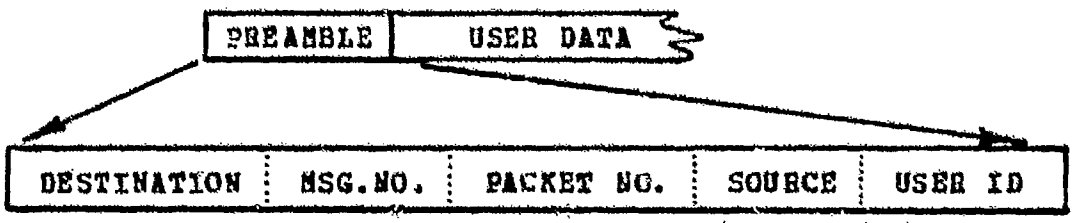


Fig. 3.2. Preamble

The preamble in Fig. 3.2 illustrates some of the information that might be required in the heading of a data packet. If this information were to require a portion of the available bits in every slot, it would seriously degrade the rate at which user data could be passed. An alternative is to use much larger data packets.

Data packets are typically created as the source node divides up a stream of data from a buffer which is being fed by a console, facsimile device, etc. The size of the packets is dictated by the user service protocol. Therefore the number of data packets needed to carry the users entire message is obviously a function of the messages size and the size of a data packet. On the receiving end, not only must all data packets be received (correctly), but it may be required to sort the packets to place them in the proper order, meaning each packet must be serial numbered. Information such as this does not contribute to the net flow of user information. Therefore to pass the largest possible ratio of user information to preamble information with the slot technique, a data packet including preamble should be some larger multiple of a voice packet.

When data packets are transmitted across a link, the sending node reads the preamble of the data packet and assigns a slot number on the best path link toward the destination node. The sending node then divides the data packet into sub-packets which are the size of a slot. Depending on the standard size of a data packet, the sending node sends the remainder of the data packet in the appropriate slot in consecutive frames. The receiving node is also programmed to accept a standard number of sub-packets once it has agreed to accept a data packet in a particular slot. In this way only one preamble is sent per data packet and the effective ratio of user information actually passed could be significantly increased.

This procedure is essentially another version of the virtual link. Depending on the number of sub-packets and system priorities for handling sub-packets, a virtual link for a data packet may vary in size. For example, if nodes are programmed to relay sub-packets as soon as they are successfully received, several nodes on the best path may be relaying portions of a single data packet at the same time. In fact, the destination node may be receiving the first subpackets before the last subpackets are transmitted. The

difference is that these virtual links have a fixed finite lifespan. They are limited by the amount of time the designer wants to make a slot unavailable to voice traffic. An extension of the same idea has two or more slots in the same frame being used to pass sub-packets of the same data packet. This provides a more efficient use of a link which may have little voice traffic and is consistent with the bursty nature of traffic.

3. Integrated Management Traffic

With the exception of the preamble, there has been no mention of management traffic which is required by node to node and network management protocols. Typically this traffic consists of relatively short messages. It is conceivable that these messages could be tagged on the end of user packets placed in each slot. In this situation it would appear to the network that 100 percent of channel capacity was available to user traffic. If this is not practical, then slots could be used on an as-needed basis to pass groups of Management messages.

There is another aspect of traffic management that may be considered. Once voice traffic is interrupted, it is important that the speaker be notified. This could be a

programmed response to the network's reaction to a broken link. The result would be for the speaker to quit talking.

Similarly, for data traffic it is practical for the source node to release only a limited number of data packets into the network and wait for a receipt acknowledgement from the destination node as data packets arrive. This is called "flow control". This prevents a source node from loading interim nodes with excessive traffic which the network may not be able to process because of a lost link to the destination. It also allows the source node to selectively retransmit packets that were not successfully received and erase those that were. Finally, it provides assurance that the data traffic was received.

IV. A DISTRIBUTED NETWORK MANAGEMENT PROTOCOL CONCEPT

This chapter describes a particular concept for a distributed control network management protocol. This protocol is limited by design to fit into the larger concept of independent levels of protocol which handle different classes of messages, processed as described in Chapter III. Analysis of the protocol developed here by a computer simulation is discussed in Chapter V.

A. SETTING THE FRAMEWORK

The following network management protocol is based on the assumption that an adequate node to node protocol is performing necessary functions such as periodically testing links, discovering new as well as broken links, and confirming when a packet has been successfully transmitted across a link.

It is further assumed that the result of this protocol, which is intended to be a flexible network which can react to link changes and find new best paths based on the latest channel values, will be used by a higher level user service protocol. This higher protocol could resemble that described in Chapter III. But it is not necessary to define

a particular user service protocol in order to investigate a lower level network management protocol. Therefore the remainder of this study will minimize any assumptions about the form of higher level protocols which may use the results of this network management protocol.

B. DEFINITIONS

All the most common components of our network, such as nodes and links, have already been mentioned. However it is necessary here to further describe certain previously defined components, and to present additional components or concepts needed to explain the protocol.

1. The Basic Group

The Basic Group is what has been defined as the network up to this point. A basic group is a collection of nodes, each having a unique identification, each being connected to at least one other node in the basic group, and each node being considered an equal member of the group (See Fig. 4.1). By using only links belonging to the basic group, it is possible to send a message from any node in the basic group (called the Source) to any other node (called the Destination).

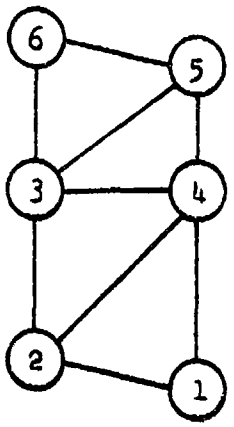


Fig. 4.1. Example of a Basic Group

Our network management protocol will initially be developed with nothing more than a basic group. Later, a version of the protocol involving "Related Groups" and "Families of Groups" will be introduced. However this will have little impact on the basic concept.

In order to move user traffic efficiently, the protocol must be able to calculate the best route from a source to destination node. To do this, each link is assigned a channel value, and these values are summed and compared to determine the best path from the source to destination node. It is not essential to specify in advance the exact physical nature of these channel values, or

distances as they are sometimes called. But whatever channel value physically amounts to, it should reflect the relative "cost" of sending traffic over a link at the time it is measured.

A best path implies that, based on existing links and current channel values at the time it was measured, there is at least one combination of links whose net channel value represents the most efficient path from the source to destination. This is frequently considered the minimum delay route. Best paths can become outdated for two reasons: either one of its links is broken making movement impossible, or another combination of links develops a lower net channel value.

It should be noted that each link is a two way communications channel, and usually the current channel value in one direction has no relationship to the channel value in the other direction. In Fig. 4.2 below, the channel value from nodes A to B is 1. However the channel value from nodes B to A is 5. This means that for any two nodes in a basic group, the best path from the first node to the second is not necessarily the best path from the second node to the first. Thus in any basic group of N nodes, there are N(N-1) or approximately N**2 possible best paths.

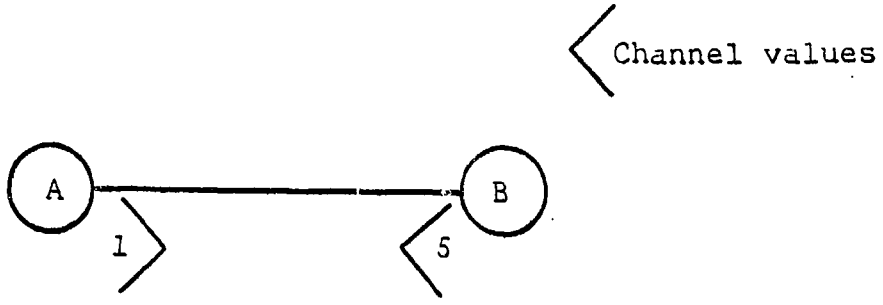


Fig. 4.2. Channel Values on a Two-way Link

2. Activities

In the course of maintaining the network, the protocol will cause each node to initiate and participate in several management activities. Most have already been mentioned and will only be discussed briefly here.

The best path update is the fundamental operation of this level of protocol. As channel values change and best paths become outdated, steps must be taken to find the new best path. This process is automatically and asynchronously initiated by each node, and when it is completed (which may require the origination of several update cycles as discussed below), every other node in the basic group knows the latest best path to the initiating node. This operation is periodically required of every node in the network. The

reason why complete updating of the best paths may require more than one initiation or an update cycle can be seen in a simple example. In the network in Fig. 4.3, node A sends out an update message to nodes B and C. Node C updates its channel value to A from 5 to 4 but still retains its old best path thru node B believing it has a total channel value of 3. Finally after node B relays A's update message to C, node C learns that the actual channel value thru node B to A is now 7. When A initiates its next update, node C will change its best path to node A to be the direct A-C link.

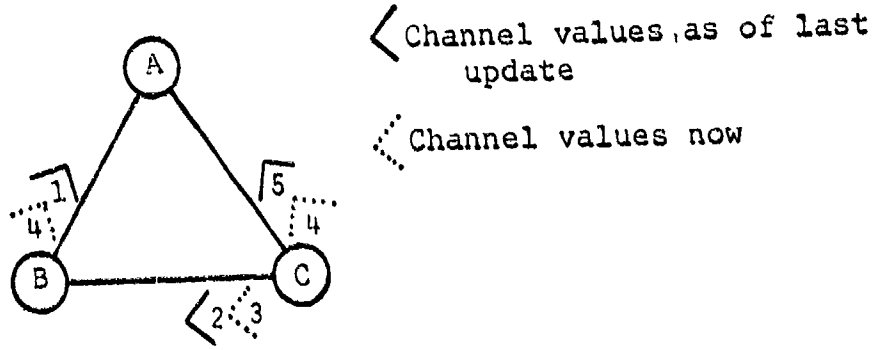


Fig. 4.3. Update Iterations

A broken link can be a traumatic event in the network. Therefore the protocol will react to broken links in an attempt to minimize the effect on user traffic flow.

First it will attempt to switch all traffic hampered by the broken link to an alternate link. An alternate link is defined only in respect to individual nodes, and if one is available, it may be used by a node if the node is faced with an inability to move traffic over a previous best path which now contains a broken link. An alternate link can only be considered as a temporary fix. Its only guarantee is that if used, it will not create a loop situation. A loop is defined as a closed path consisting of a series of links. Therefore traffic leaving a loop node will eventually return to that node. In Fig. 4.4, node 2 cannot consider the link to node 4 as an alternate link if node 4 routes traffic destined for node 1 through node 3. This creates a loop. However if node 2 can be assured that node 4 will not route traffic destined for node 1 on any path which eventually moves through node 2, then node 2 can switch traffic to node 4 after a break with confidence that it retains a loop-free network.

Although switching traffic of a best path implies a decrease in efficiency, the alternate may be to stop all traffic routed over a broken link. Of course this may be even less efficient. But a node faced with the decision may

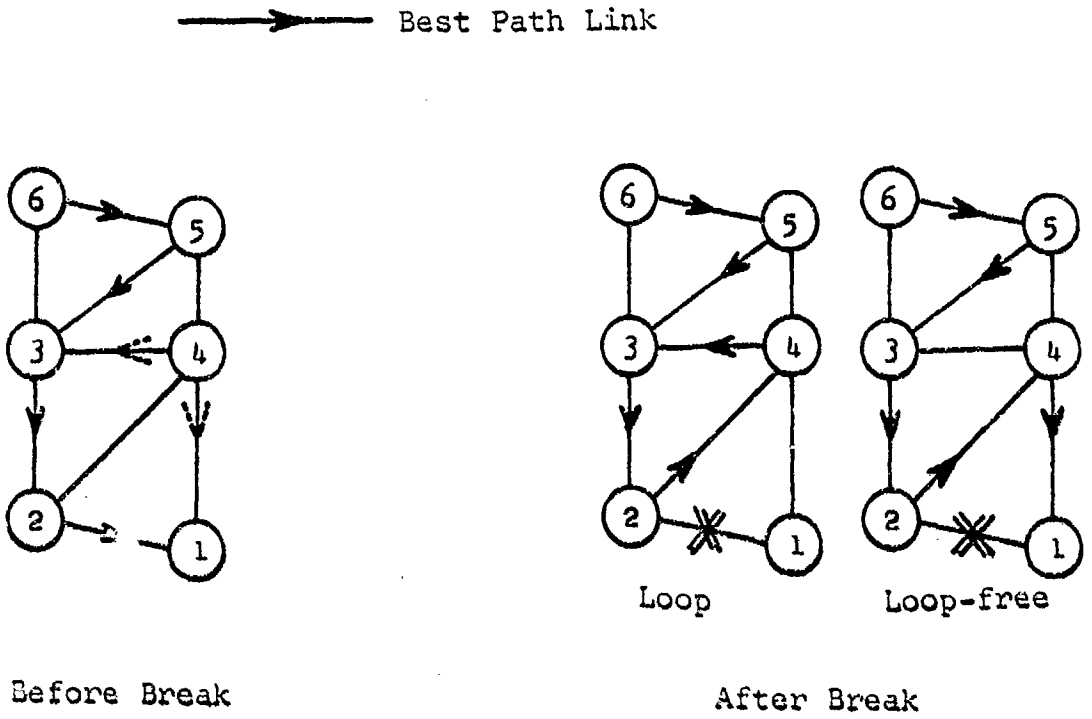


Fig. 4.4. Loop

not always have the option of an alternate link. See paragraph C3 below and Appendix A for a discussion and proof of an alternate link concept which is compatible with the network management protocol described in this chapter.

Clearly, it is required that a node be able to cope with a situation where it may lose all access to one or more nodes. Recovery is defined as eventually establishing another path to the disconnected nodes. The efficiency of

recovery is defined as the speed at which a path is reestablished over the new best path.

To accomplish the above activities, each node in the network will create, process and relay messages from other nodes. The processing will frequently change components of a message that a node receives from a previous node, adding information to the message before relaying it. Nodes are also selective as to which other nodes it will send or relay a message. The net result is to improve network-wide operation and efficiency.

3. Messages

The network management protocol is required to send two types of overhead messages related to the maintenance activities mentioned in the previous paragraph. Each message will have several elements which will be abbreviated and represented in a message argument.

a. Update Message

The symbol for an update message and its components are shown below. The letter l identifies the last node to relay the update message (or U-msg). The letter d identifies the originator of the U-msg. Note that when the originator first sends the U-msg, l=d. D(l) is the cummulative channel value on the best path from l to d.

Update Message ==> U(l,d,D(l))

b. Broken Path Message

The symbol for a broken path message and its components are shown below. The argument d represents the destination node for which the broken link is blocking traffic, and corresponds to the d in the U-msg. The d in the U-msg is the identity of the initiating node, and represents the destination to which the best paths created by this U-msg will point. The d in the X-msg indicates that the best path to d is broken.

Broken Path Message ==> X(d)

C. THE CONCEPT

The objective of this network management protocol is to provide a single algorithm that can operate autonomously in each node of a network to provide completely decentralized network control, yet provide for efficient traffic routing. This algorithm was also chosen for its relative simplicity and potential robustness. Its primary departure from most other algorithms of this nature is that it attempts to

accommodate new and broken link events without requiring cycle numbers. The algorithm is given in Appendix B.

1. Normal Operations without New or Broken Links

It is most convenient initially to study the update process while freezing the status of nodes and links. We will also initially assume each node has a best path to every other node. As mentioned earlier, the basic group or network consist of N nodes. The number of links between these nodes will normally exceed the number of nodes. Normally, if they are evenly distributed, the more links into an average node, the more robust is the network.

To efficiently use a network, traffic should take the best path from the source to destination node. To identify and use this path, each node along the way must know the destination of the traffic, and what neighboring node is downstream on the best path to each destination. Downstream will imply movement toward the destination, that is, relaying the traffic to another node with a smaller cummulative channel value to the destination. Upstream implies movement away from the destination, normally backwards along the best path. The update message allows each node to determine which neighbor is on its best path to

every other node in the network. Each node periodically initiates a U-msg to all its neighbors. In Fig. 4.5 node 1

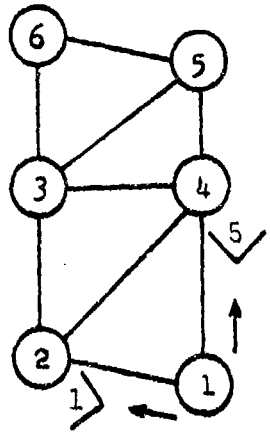


Fig. 4.5. Initiating an Update

initiates an update by sending U(1,1,0) to nodes 2 and 4. When a node receives a U-msg ititiated by d, it computes the cummulative channel value to d thru 1 and compares it to the last cummulative channel value along the node's current best path to d. For example, in Fig. 4.5, suppose node 4 had previously selected the direct link, with channel value = 5, as its best path to node 1. Meanwhile node 2 has also received a U-msg from node 1, has determined that this is its best path to node 1 because no other path offers a

cummulative channel value of 1, and has relayed node 1's U-msg. Node 2 sends a modified U-msg to all of its neighbors except the neighbor from which it received the U-msg. Now the U-msg is updated with the cummulative distance from node 2 to node 1. Let d(i,1) be the channel value on the link from a node i to any neighbor 1. Then the cummulative channel value from node 2 to node 1 is

$$d(2,1) + D(1) = 1 + 0 = 1.$$

D (1) is taken from the U-msg received by node 2 from node 1. d(1,2) is calculated at some earlier designated time when all nodes in the network simutaneously calculate and fix channel values to each of their neighbors (this procedure is discussed in greater detail in Chapter V). Therefore the U-msg relayed to node 2's neighbors is U(2,1,1) which states that node 2 is relaying a U-msg from node 1 and the cummulative channel value through node 2 to node 1 along its best path is 1.

When node 4 receives the U-msg from node 2, it once again processes the message in a standard fashion. As shown in Fig. 4.6, the channel value from node 4 to node 2 is 3 (d(4,3)=3). Now upon receiving the U-msg from node 2, node 4 calculates the cummulative channel value through node 2 to

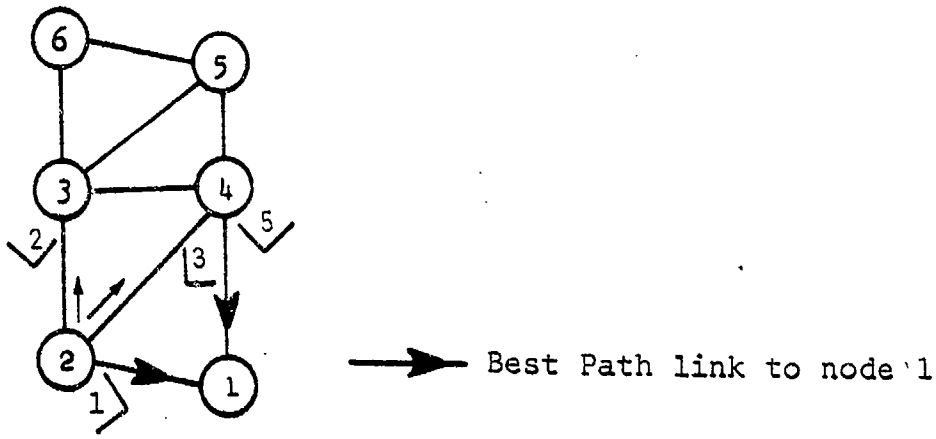


Fig. 4.6. Node 2 Relays Node 1's U-msg

the initiator of the U-msg (d=node1). For node 4 in this example

$$d(4,2) + D(2) + 3 + 1 = 4.$$

When node 4 compares this value with the latest cummulative channel value for its best path to node 1 (which node i will define as the symbol B(d)) it will find that it is more efficient to go thru node 2 to get to node 1, or B(1)=2. Note that in future discussions the term "Best Path" will imply the optimum series of links, whereas B(d) will indicate a specific neighboring node which a transmitting node considers as the next downstream node on the best path to destination d.

In this example, node 4 receives a U-msg which enables it to improve its best path to d. Any node which changes its best path or the cummulative channel value for its current best path must, in turn, relay this information to all of its neighbors (except B(d). This is necessary because, given this new information, an upstream neighbor may have an opportunity to update its B(d). On the other hand, if a node receives a U-msg which does not change the node's B(d) or cummulative channel vlaue to d, it will not relay the U-msg. This is acceptable because the upstream nodes already have access to the current route which is considered more efficient than a route through the node which relayed the last U-msg.

Deletion of update messages is an important function. If the network were not allowed to eliminate useless messages, it could impose a significant unnecessary burden on the management traffic load. In a network of N nodes, there are approximately N**2 best paths. If, when each node initiated an update operation, every other node indiscriminately relayed the update message, there would be a minimum of approximately Nx(number of links) update messages generated when each node originates an update in a

network of N nodes. Therefore to control this growth, the first node to receive a useless U-msg eliminates it.

Fig. 4.7 shows the complete network with channel values and best paths from all nodes, to node 1, before and after update. The order in which U-msgs arrive at a node can significantly affect the number of U-msgs generated in reaching the optimum solution. But (assuming static values for the channel values) the end result will always be optimum, even though it may require several update initiation cycles to stabilize. The following is a sequence of events that could have occurred to update the network in Fig. 4.7.

Node 1 generates U(1,1,0) and sends it to Nodes 2 and 4.

Node 4 receives Node 1's U-msg. Since this is already its B(1), it updates its net channel value, generates U(4,1,5) and sends it to nodes 2,3, and 5.

Meanwhile Node 2 receives Node 1's U-msg upstream along its B(1), updates its net channel value, generates U(2,1,1) and sends it to Nodes 3 and 4.

Node 2 receives Node 4's U(4,1,5), compares it to B(1), and discards it.

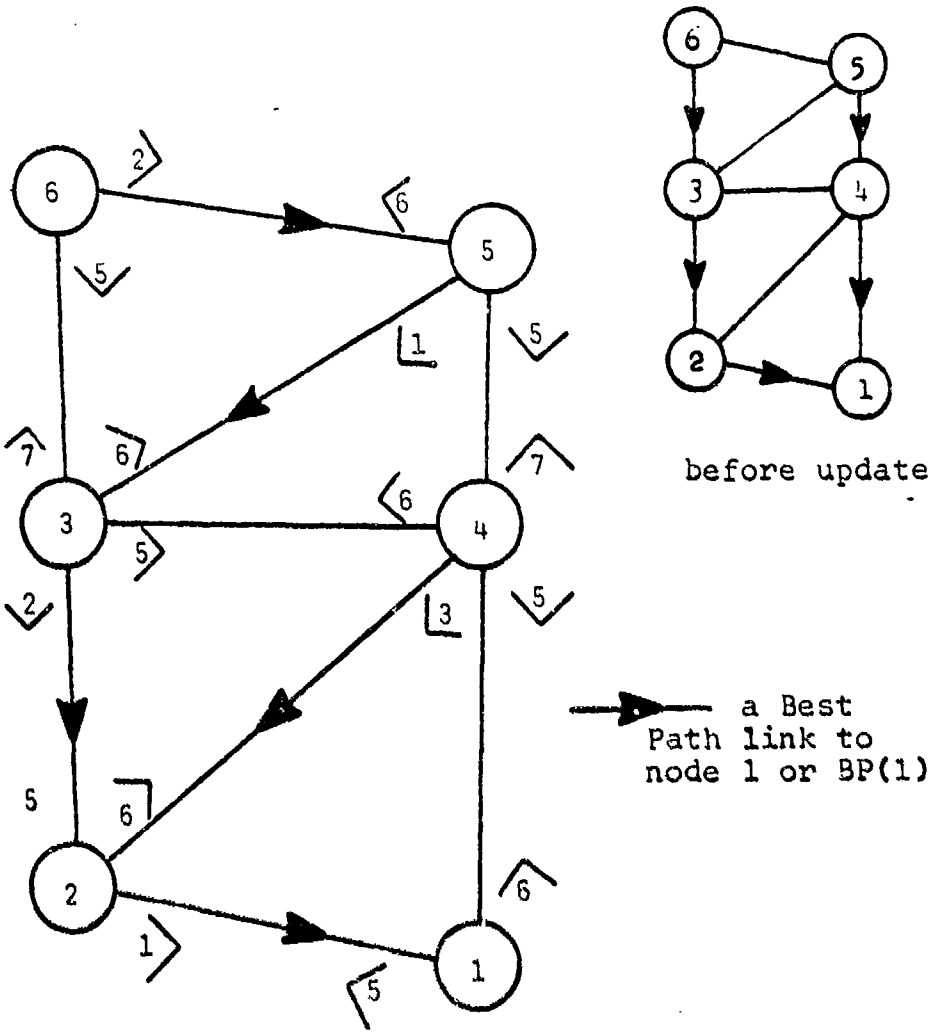


Fig. 4.7. Complete network with Channel Values and BP(1)'s

Node 3 receives Node 4's U(4,1,5), compares it to its latest B(1) and discards it.

Node 5 receives U(4,1,5) upstream from its B(1), generates U(5,1,10) and sends it to Nodes 3 and 6.

Now node 4 receives Node 2's U(2,1,1), compares it to its last B(1) and selects a new B(1)=2. Since it changed B(d), Node 4 issues U(4,1,4) to Nodes 3 and 5 which will eventually be discarded by both nodes.

Meanwhile Node 3 receives Node 2's U(2,1,1), updates its B(1) and generates U(3,1,3) for Nodes 4,5 and 6.

Node 5 receives Node 3's U(3,1,3), finds this better than its previous B(1) and sets B(1)=3. Now Node 5 must also issue U(5,1,4) to Node 4 and 6.

Node 3 receives Node 5's previous U(5,1,10) and discards it. Node 4 receives Node 5's later U(5,1,4) and also discards it.

Node 6 initially received Node 5's U(5,1,10) but retained its old B(1)=3. Later Node 6 received U(5,1,4). This time it finds this path much better and sets B(1)=5. It also issues U(6,1,6) to node 3. Eventually Node 6 receives U(3,1,3) but discards it. Finally, Node 3 receives U(6,1,6) and discards it.

In the above example, the senerio would have been slightly changed had messages arrived in a different order, but the ultimate best path results would be the same.

2. Introducing New and Broken Links

Realistically a network must integrate new links and recover from broken links. Later the "Alternate Link", as an interim fix, will be discussed. But initially we shall assume that there are no known routes remaining from the node which detects the broken link, to some destination. Assume also that traffic for this destination may already be stored in the detecting node, or enroute to it under the assumption that the broken link is still intact. The network management protocol must provide for a graceful recovery.

In order to eliminate the added complexity of cycle numbers, the protocol is restricted to initiating one U-msg from any one node in the network at one time. This means that there must be enough time for an update cycle or session initiated by a node to propagate thru the entire network, updating all best paths as it goes. When a break cuts off access to a node, it is important that a new update from that node (or nodes) be initiated and propagated thru

the network as soon as possible in order to identify new best paths so that stalled traffic can continue to their destinations. In order to address this problem, the protocol assigns the highest processing priority to U-msgs. This is intended to allow U-msgs to perform the update (and therefore eliminate themselves from the net) as soon as possible. At the same time, the protocol sets the frequency at which each node periodically initiates a new U-msg. The idea is to establish a practical U-msg initiation frequency so that the event of a broken link does not require a request for initiation of a special update message, and yet does not leave user traffic stranded for a long time.

It might be helpful to consider an example of this in terms of the user service protocol example in Chapter III. If the average distance between nodes is approximately 3 km (based on the Marine Amphibious Brigade model) then assuming speed of light propagation the signal travel time between nodes is

$$\text{travel time} = \frac{3 \text{ km}}{3 \times 10^{5} \text{ km/sec}} = 10^{-5} \text{ sec} = 10 \text{ usec.}$$

Furthermore assume a network or basic group of 50 nodes, and assume a longest best path of 30 nodes. Then the maximum total travel time is

30 links x 10 usec/link = 300 usec.

Assume an additional 20 usec processing time in each node (when protocol messages are given top priority). Then the total time for a U-msg to process thru the entire net is approximately 1 msec. Therefore if the protocol required each node to initiate a U-msg every .1sec (or once every 10 frames), approximately .1sec + 1msec is the longest any traffic should be stranded due to a broken link. This should not significantly affect data traffic which is bursty in nature anyhow. Although detectable in voice traffic, it would not be serious unless failures occurred repeatedly. This situation could be improved by increasing the frequency of the update at the cost of more network management traffic.

This protocol requires that traffic which is stranded due to a broken link wait to be rescued by a routine U-msg from the destination node to which the traffic is addressed. Yet there are still actions which can be taken to make good use of the broken link information and minimize the trauma of recovery. Since there is no assurance that any of the nodes close to the break will be on the new best path, it is probably helpful to freeze data

traffic enroute to a broken link. This is one of the functions of a broken path message (X-msg).

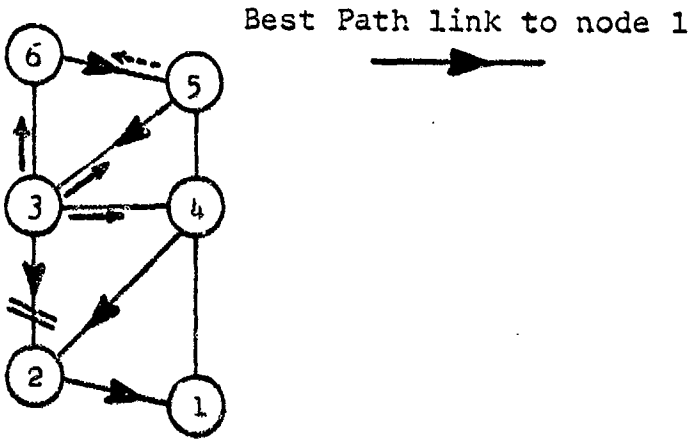


Fig. 4.8. Sending the X-msg Upstream

When a node discovers a broken link on one of its best paths, it initiates a X-msg for every destination node for which the discovering nodes considered the broken link as part of the best path. These nodes are easy to identify because this is the same information used for normal routing operations. The initiating node sends the X-msgs to all of its neighbors. For example, as shown in Fig. 4.8, when Node 3 discovers that the link to Node 2 (and B(1)) is broken, it will initiate an X-msg which is X(1). Note in this example

that the broken link could also be Node 3's B(2), also requiring a X(2) message. But for simplicity it is assumed that Node 1 is the only destination in this network. The X(1) is sent to all of Node 3's neighbors. Node 4 and 6 do not consider Node 3 to be on their best path to Node 1. Note that for Node 4, this is a correct assumption. But for Node 6 this assumption is not correct. In any event, if a node receives a X-msg from a non-best path neighbor, it discards the X-msg and takes no other action. This is how useless X-msgs are eliminated.

Node 5, on the other hand, receives X(1) from its B(1). This indicates that it has lost its best path to Node 1. As with Node 3, when a node discovers that its best path to d is broken, it freezes any data traffic in its buffer for d, and issues an X-msg. Therefore Node 5 now issues X(1) to Nodes 4 and 6. Once again Node 4 ignores the X-msg. However this time Node 6 has received the X-msg from its B(1). This would cause Node 6 to stop sending data traffic until a new best path is found.

At the User Protocol level, which may employ virtual links as described in Chapter III, the X-msg may not be enough to stop all traffic routed over a given link when a

break is discovered. A virtual link fixes a route at the time it is created, for the duration of the traffic session. In the meantime, subsequent update cycles may have caused nodes along an established virtual link to select other nodes as B(d) while maintaining its virtual link thru the node which was the B(d) at the time the virtual link was created. Therefore, in addition to sending X-msgs to all neighbors for all destinations for which a broken link was considered a best path, it may also be necessary to define a Virtual Disconnect message which would be relayed upstream to break down virtual links. In fact something like this would probably be required in any network using virtual links to break down the virtual links when users have completed a routine traffic session.

Because of the frequency of the U-msg, it may not be necessary for a X-msg to work its way all the way upstream to the most remote node on the best path. In Fig. 4.9 Node 2 could have issued its X(1) after Node 1 issued U(1,1,0) to Node 4. In this case, if Node 4 had not yet processed U(1,1,0) when it received the X(1) from its B(1), Node 4 would immediately adopt the direct link as its B(1) and issue U(4,1,5) to Nodes 2, 3, and 5. Since Node 2 already

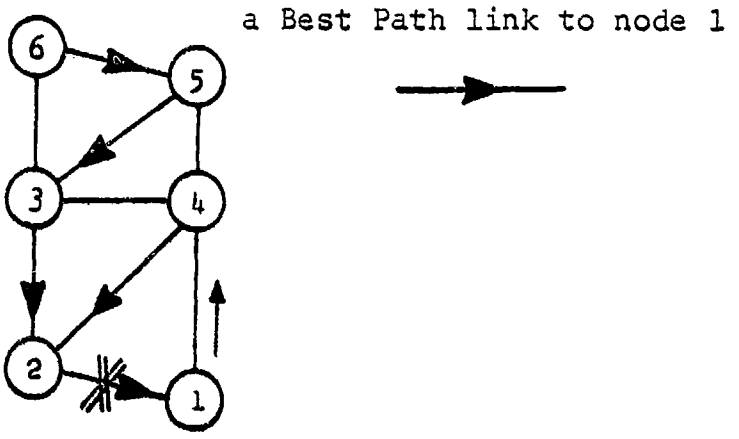


Fig. 4.9. X-msg Meets U-msg

froze traffic for Node 1, it would immediately release the traffic to Node 4 considering Node 4 as its B(1). If Node 3 had already frozen traffic for Node 1, the same would apply. But in this situation it is linkely that a new best path would be established before traffic in Nodes 5 and 6 were even affected by the broken link.

New links do not have the traumatic impact of broken links. A new link represents the addition of a new neighbor for the two nodes on each side of the link. Since the link is initially unloaded, it is likely to become a prime candidate for a link in several best paths because of its low channel value. It is necessary to guard against

oscillations here which can be done by assigning arbitrary initial average channel value to the new link which would enable the link to be gracefully integrated into the network. In time, as the link becomes used, the effect of this arbitrary assignment will disappear. Once again, because of the frequency of initiating U-msgs, there is no need to request special updates upon the discovery of a new link. It will be integrated into the network quickly enough just by normal updates.

3. Alternate Link - an Interim Fix

It is not always necessary to stop traffic in the face of a broken link. Ideally every best path would have a backup path so that when a broken link is discovered, traffic is immediately switched to the backup path with minimum ripple in network traffic flow. But this may not be possible, and the additional complexity in the protocol as well as the increase in the volume and content of network management messages appears significant.

However, as explained in Appendix A, the protocol as described in this chapter provides sufficient information to manage the basic update and broken link functions, and with a slight increase in processing at each node, the same

network management messages can occasionally provide a real-time routing alternative to a broken link. This is called an alternate link.

The alternate link is identified during the update operation. For example, during a routine Update for Node 1 of the network in Fig. 4.10, Node 2 will send U(2,1,2) to Nodes 3 and 4. Node 4 will not select Node 2 as its B(1), and would normally discard the U-msg. However if Node 4 made one additional comparison, it may still find the Node 2 route to Node 1 useful.

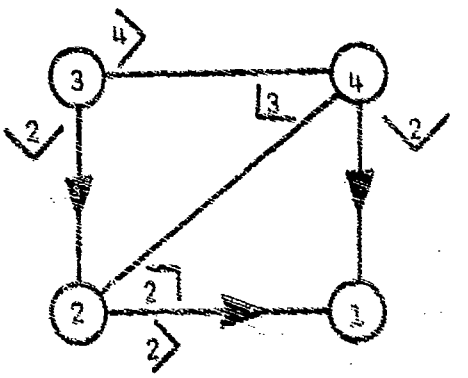


Fig. 4.10. The Alternate Link

For any node j, by comparing the cumulative channel value (D(l)) of the last relaying node (l) with node j's current cumulative channel value along its best path to d,

node j can determine if traffic passing thru node 1 can also eventually pass back thru node j. Assuming all links have a minimum channel value >0, if node j's cummulative best path channel value >= D(1), then node j is assured that node 1 does not pass traffic thru node j in order to get to d. This minor conclusion provides node j with a loop-free alternative path to d if it should discover a break in its best path. This alternative says nothing about multi-destination or implied loops. It only offers a temporary fix for a node which has experienced a broken link.

Going back to the example in Fig. 4.10, Node 4 notes in U(2,1,2) that D(2) = 2 which also equals the cummulative channel value for Node 4's B(1). This causes Node 4 to list Node 2 as an alternate link to Node 1. In larger networks, one node can certainly have alternate links for several d's as well as several alternate links for a single d. Node 3 in the example sets B(1) = Node 2. However when it receives U(4,1,2), it will list Node 4 as its alternate link to Node 1. Likewise Node 2 will pick Node 4 as its alternate link to Node 1. But note that Node 4 can not rely on Node 3 as an alternate link. When Node 4 received U(3,1,4) from Node

3, it has no way of insuring that the route is not as shown in Fig. 4.11. Therefore in the absence of any further overhead traffic, Node 4 discards this information as unreliable.

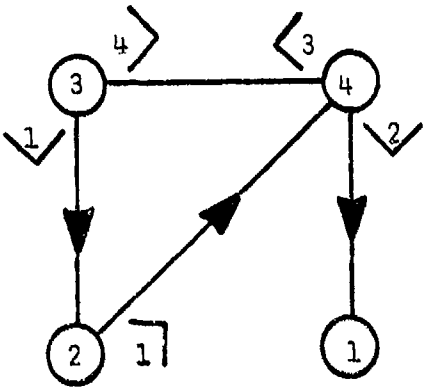


Fig. 4.11. Potential Loop Situation

The impact of alternate links is not clear. If a network is very evenly weighted and richly connected, each node could have one or more alternate links to most of the other nodes in the network. This implies that broken links may only require a shift in traffic. A less evenly distributed network would have some nodes with alternate links and others without. In this case some X-msgs might be avoided, others curtailed, and yet others unaffected. At a minimum, the alternate link concept appears to add

additional robustness to the network. At best it may allow the Update frequency to be decreased, cutting down the rate of management traffic.

D. EXPANDING TO RELATED GROUPS AND FAMILIES

Although there is no theoretical limit on the number of nodes in a basic group, there may be practical considerations which make it attractive to limit this number. For example, when all nodes are considered part of a single basic group, then every node in the basic group (which includes the entire network) must record a B(d) for every other node in the network. This further implies that updates for every individual node can potentially span the entire network. As the number (N) of nodes in a richly connected network grows, the number of U-msgs generated (to complete an update for one source) under worst-case conditions approaches 3(N**2). (See Appendix C). Therefore it may be convenient to partition the network along operational or geographical boundaries. To investigate this, several additional definitions must be introduced.

1. More Definitions

In the top half of Fig. 4.12, all the nodes in a given network fall into one of six groups numbered 100 thru

600. Nodes within a particular group consider that group its basic group. Within a basic group each node has a unique node identity. For example, in Group 400, there is only one Node 2. Outside of a node's basic group are other groups. For each group in the top of Fig. 4.12, there are five other groups called Related Groups. Note that related does not imply that two groups have a common border. For example, Group 400 does not border Group 200. By combining the group and node identity, each node can, once again, have a unique identity in the network. For example, Node 2 in group 400 can uniquely be called Node 402.

Furthermore, the six groups in the top of Fig. 4.12 can be grouped together and called a Family. This family could also have a unique identity, such as 3000 in Fig. 4.12, and be one of a number of families which combine to form a large network of nodes. In the bottom of Fig. 4.12, there are four families numbered 1000 through 4000. Once again, every group, in every family in the bottom of Fig. 4.12 could have a Node 2. But when group and family identities are added to the node identity, each node retains a unique identity. To fully identify Node 2 mentioned earlier, it can now be called Node 3402. By using this

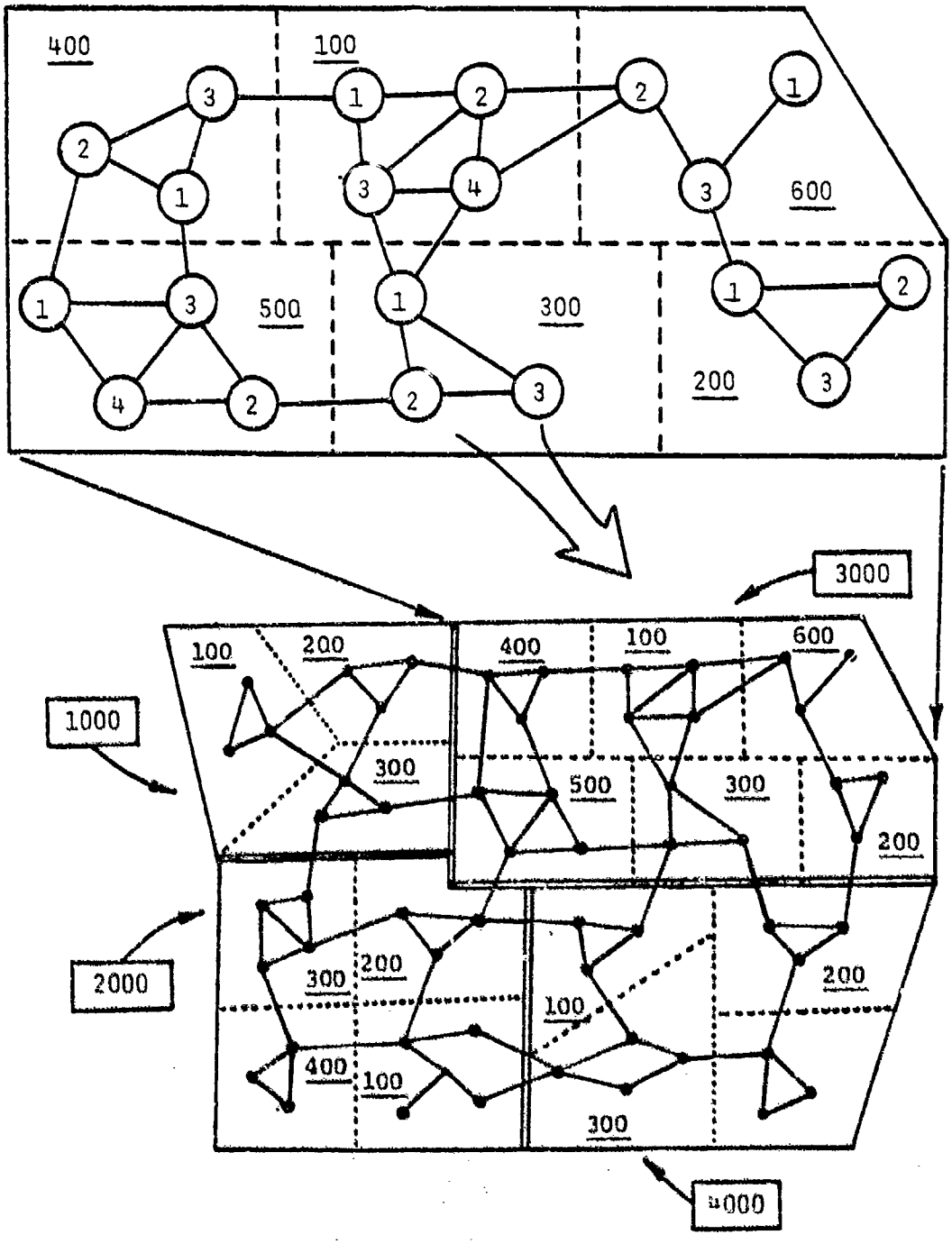


Fig. 4.12. Related Groups and Families

three tier identity concept, there is a potential to reduce network management traffic. There is no requirement to stop at three tiers; however three tiers suffice to demonstrate the principles.

One requirement of this structuring principle is that groups and families must retain some continuity. That does not mean that groups and families cannot move in relation to each other. It simply means that entire groups and families can not distribute all their nodes randomly around the network. If the network must tolerate complete random node movement, then the single basic group concept seems best suited to control the network. However there may be several situations wherein clusters of nodes are likely to remain geographically and operationally close while being fluid in a larger network of nodes. Military organizations are a good example of this structuring.

2. Efficiencies of Grouping

In the remainder of this study, any reference to node identity will imply the full identity including the node's basic group and family, if applicable. All message formats and contents are also the same. It will be assumed that all nodes know their assigned group and family. In a

military network for example, the group could represent the battalion, and the family could represent the Brigade.

As shown in Fig. 4.12, individual nodes continue to establish links with neighboring nodes regardless of arbitrary boundaries. Efficiency is available by changing the processing of messages that cross these boundaries. The goal is to reduce the number of network management messages that travel to remote nodes in the network when there is small likelihood that the best paths being updated by these messages will ever be used.

Basic groups are organized to contain a group of nodes which communicate frequently with each other. Every node in the basic group has a best path to every other node in the basic group. For these nodes, which constitute a mini-network, the basic network management protocol described in Section C above applies directly. However, the node to node protocol will establish a link with any node it can contact. Therefore a node may find that it has a link with another node outside of its basic group. This is where group/family processing begins.

Fundamentally, grouping causes nodes to treat related groups and related families as single nodes, while

still maintaining the capability to contact each node in the network. Therefore for our example in Fig. 4.12, Node 3402 has two other nodes in its basic group, five related groups, and three related families. Thus Node 3402 maintains a best path to a total of 10 network elements. By contrast, without groups Node 3402 would be required to maintain best paths to 57 individual nodes in order to contact every node in the network. It should be noted that basic groups of only three nodes is probably unrealistic. Basic groups of 10 to 25 nodes, families of 3 to 5 groups and networks of 3 to 5 families would fit typical military organizations. Although there is probably an optimum combination for a given traffic profile, there are no rigid requirements on grouping sizes.

During the course of a normal Update, an initiating node, or a relaying node will send a U-msg to all of its neighbors. The receiving node (j) checks the identity (l) of the node which last relayed the U(l,d,D(l)) message. If l is not in Node j's basic group, and if l does not equal d (indicating the last relaying node did not initiate the message), Node j discards the U-msg. If l=d, this indicates to Node j that a neighbor outside Node j's basic group has

initiated an Update. If the neighbor is from a related group (same family), Node j compares its cummulative best path channel value for that related group to the net channel value through l. If this is an improvement, Node j alters the U-msg with its d(i,l), and relays the U-msg to all neighbors. This procedure continues until this U-msg can not offer an improved best path to any other node or until it reaches the family boundary. If Node j's previous best path was superior to the new possibility, Node j would discard the U-msg. If the neighboring node (l) which initiated the message was from another family, node j would check its current cummulative best path channel value to l's family, and compare it to the net channel value through l. As in the case of a related group, if there is an improvement, the U-msg is relayed, otherwise it is discarded.

Fig. 4.13 serves to illustrate Updates across boundaries. The process may become clearer by tracking a possible sequence of events during a routine update operation. In Fig. 4.13, the dotted triangles pointing away from a node represent that node's (pre-update) best path to a related family (if the U-msg creating the best path had

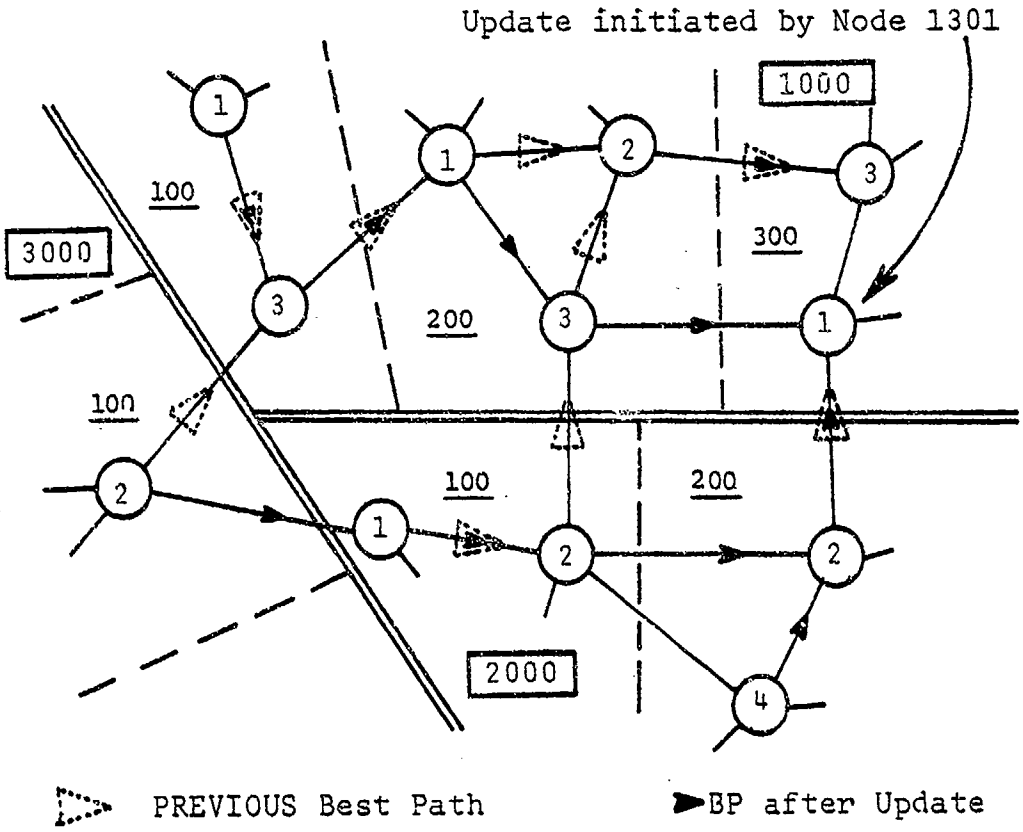


Fig. 4.13. Update Across Boundaries (partial network)

crossed a family boundary after being initiated) or best path to a related group (if the U-msg creating the best path had crossed a related group boundary after being initiated). The dark triangle represents the updated best paths after Node 1301 issues an update.

When Node 1301 issues a U-msg to all of its neighbors, the nodes in basic group 1300 will update like any basic group. Nodes 1203 and 2202 will also receive

U(1301,1301,0). Node 1203 sees that this U-msg was initiated by one of its related group neighbors, and after comparing channel values with its old best path through Node 1202, selects Node 1301 as its new best path to Group 1300 (or B(1300)=Node 1301). This also requires Node 1203 to adjust and relay the U-msg to all of its neighbors. Node 2102 receives Node 1203's U-msg across a family boundary, notes that it was relayed but not initiated by Node 1203, and discards it. It is discarded because this particular version of Node 1301's four original update messages (one for each link) initially crossed a Group boundary. Therefore all subsequent versions of this U-msg serve to update best paths to Group 1300 within its family. Therefore when Node 1203 relayed an offspring of the "group" version outside its family, Node 2102 discarded it. Node 1202 keeps its best path to Group 1300 through Node 1303. Node 1201 changes its best path through Node 1203 and relays the U-msg to its neighbors.

Now Node 1103 notes that a Group 1200 node has relayed an U-msg initiated by a third related group in Node 1103's family; therefore it will evaluate this message in an effort to improve its path to Group 1300. Note that once a

U-msg successfully crosses a group boundary leaving its basic group, it continues to cross other group boundaries until it no longer offers a shorter best path, and is discarded.

The same considerations apply when initially crossing a family boundary, as will be seen below. Node 1103 updates its best path to Group 1300 and relays the U-msg to Nodes 1101 and 3102. Node 1101 retains its path to Group 1300 through Node 1103. It so happens that Node 3102 currently has Node 1103 as its best path to the 1000 Family. However when it receives the U-msg relayed by Node 1103, it notes that it was not initiated by Node 1103 and discards it since Node 3102 is not interested in establishing a path to Group 1300, or any other individual group in the 1000 Family.

Meanwhile Node 2202 has also received the U-msg initiated by Node 1301. Node 2202 updates its net channel value retaining this best path, and relays U(2202,1301,D(2202)) to all of its neighbors. Node 2102 accepts this new route as its best path to the 1000 Family in preference to its less efficient link through Node 1203. It then relays the U-msg to its neighbors. Node 2101 updates and relays again.

Now Node 3102 has again received a version of the U-msg initiated by Node 1301. But this time it was passed by a node which was not in the 1000 Family, indicating that the cummulative channel value in the U-msg up to this point represents the distance along this proposed path to the edge of the 1000 Family. Node 3102 compares this to its current best path channel value to the 1000 Family (which is direct to Node 1103), and picks the best path. In this example, Node 3102 found that it was more efficient to travel to the 1000 Family through the 2000 Family, than to cross the direct link to Node 1103 (rather unusual).

To use this routing information, the Source node addresses traffic to the destination node and sends the traffic on its way. If the destination is in the same basic group as the source, the source has a best path direct to the destination node. If the destination is in a related group (same Family), the source node sends the traffic on the best path to the destination node's basic group. As soon as it crosses the basic group boundary, the traffic will reach a node which now has a best path to the specific destination node. Similarly for inter-family traffic: it is routed on the source node's best path to the family of the

destination. When it crosses the family boundary, it is routed by the destination's group and finally when it crosses the basic group boundary, it is routed to the specific node.

The cost in routing inefficiency entailed by the tremendous reduction in overhead traffic offered by this scheme is obvious from the example in Fig. 4.13. If Node 3102 wanted to communicate with Node 1101 after the Node 1301 update (dark triangles), the traffic would ultimately travel through nearly every node in the figure, when if fact Node 1101 is only two links away from Node 3102. Although it has been established that it is shorter to go from Node 3102 to Node 1301 than to Node 1103 in this case, the full trip would normally be shorter by the more direct route.

Besides the reduction in overhead traffic, it should also be noted that group and family boundaries would normally be selected on operational boundaries, so that a relatively small amount of traffic would be expected to suffer from this self-inflicted inefficiency.

There are some minor exceptions to the above rules which would be helpful if integrated into this scheme. For example, any node on a boundary with a non-best path direct

link to a node in another group or family, need not reject or purposely break this link simply because it is programmed only to use that node's group or family address. The minor additional overhead of retaining direct links to all neighboring nodes can be useful in recovering from broken links. Both the broken path and alternate link concepts described for the basic group can be applied, virtually unchanged, to the group/family processing concept.

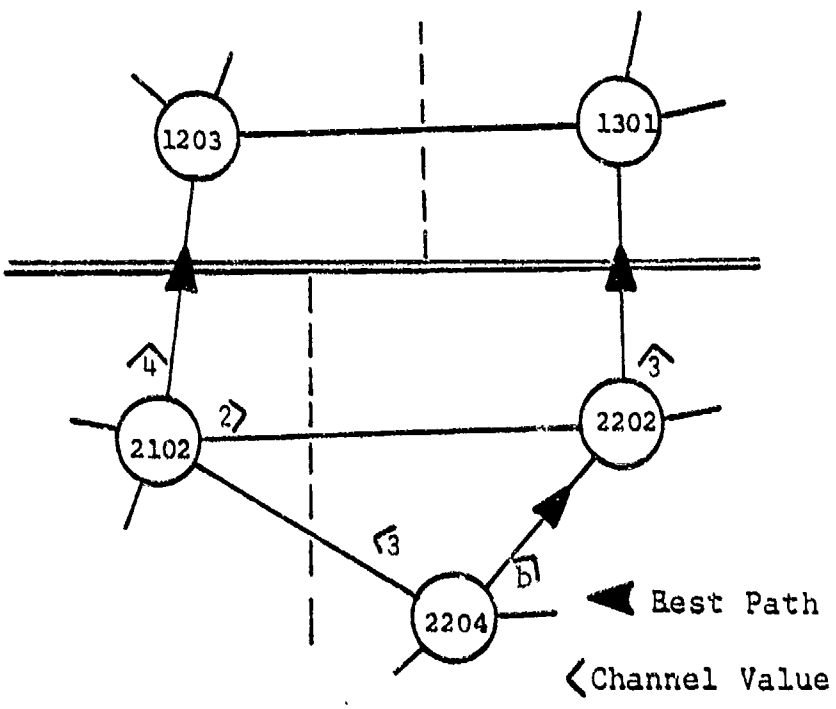


Figure 4.14. Broken Paths and Alternate Links Across Boundaries

A broken link on a best path to a related group or family causes the discovering node to first look for an

alternate link. In Fig. 4.14, a portion of a network including boundary crossings and link channel values is shown. If these channel values had existed when Node 1203 last updated, Node 2102 would have retained its B(1000)=Node 1203. However when Node 2202 relayed U(2202,1301,3) to Node 2102, Node 2102 would see that D(2202)=3 to the 1000 Family, which is less than its cummulative best path channel value to the 1000 Family. Therefore Node 2102 would keep Node 2202 as an alternate link to the 1000 Family. Then if Node 2102 lost its direct link to the 1000 Family, it could immediately switch traffic to Node 2202 with the assurance that traffic would not enter a loop. Conversely, Node 2202 could not pick up Node 2102 as its alternate link to the 1000 Family because Node 2102's cummulative channel value (4) is greater than Node 2202's cummulative channel value (3).

If Node 2202 experienced a broken link to the 1000 Family in Fig. 4.14, it would be required to initiate a broken path message to all of its neighbors. When Node 2102 received X(1000) from Node 2202, it would disregard the X-msg since its best path is not affected. When Node 2204 received X(1000), it would find that its B(1000)= Node 2202

indicating it had lost its best path to the 1000 Family, and look for an alternate link. If during the last update by Node 1203 the channel value b in Fig. 4.14 were such that

b + Node 2202's D(1000) >= Node 2102's D(1000)

or

b + 3 >= 4,

then Node 2204 would switch traffic for the 1000 Family thru Node 2102. If

b + 3 < 4,

then Node 2204 would relay the X-msg or X(1000) to all of its neighbors. And the process would continue outward from the broken link just as within a basic group.

In this discussion, it should be noted that the basic group concept of network management protocol can be applied directly to the group/family organization of the network, with some requirements on the structure of the group and families. The idea of detached nodes in the group/family concept is a special case which will be discussed in Chapter VI. Whether or not the group/family concept should be imposed on the network is a function of

network size, the user traffic profile and efficiency tradeoffs.

V. SIMULATION

A primary objective of the simulation was to test the basic algorithm by selecting and fixing some network parameters, and then making multiple runs in which the remaining parameters were varied. Though limited in scope, the simulations validated some of the mechanics of the algorithm. This included originating and relaying update messages, which further resulted in selecting and updating best paths based on calculated channel values. Both the basic group and family/group concepts were tested. Two methods of calculating channel values, both using a variable time duration called a window, were also investigated. The test network is shown in Fig. 5.1.

Simulation results were initially compared to results obtained using static routing via fewest number of hops over the same network. Later, selected parameters were varied to observe the stability and robustness of the network control. As a result, several basic observations were made about the attributes, efficiencies and limitations of this management protocol concept.

The broken link and alternate link concepts were not part of this initial simulation. If the basic link

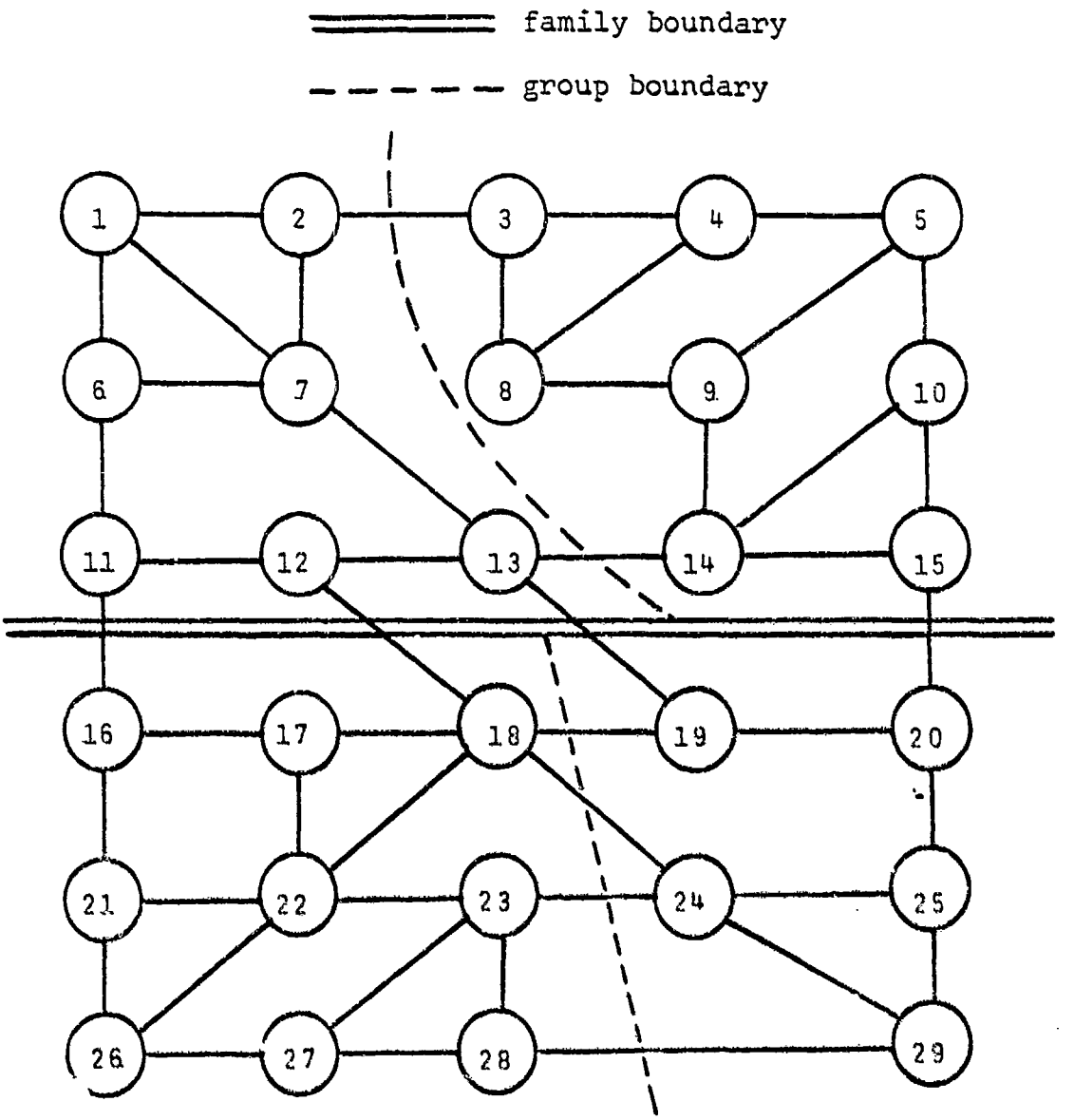


Fig. 5.1. Test Network

management concept ultimately proves to be worthwhile, as these initial tests suggest, then the next logical step would be to test the algorithm under the added strain of gaining and losing links.

The simulation was conducted using the SIMSCRIPT II.5 simulation language. The encoded algorithm is listed in Appendix E. Several SIMSCRIPT encoding decisions are discussed later in this chapter.

A. SIMULATING A USER AND MEASURING EFFECTIVENESS

In order to observe and measure the relative effectiveness of the algorithm, a simple user service protocol involving only data packets was integrated into the system. User traffic sessions were generated with an exponentially random inter-arrival rate and with a uniformly random number of packets. Both the rate and number of packets were controlled by input variables. A packet either moved thru the network, or waited in a queue if the required link was busy, until it arrived at its destination where it was discarded after performance data was collected. All traffic sessions (and therefore all packets) had a source node determined by a uniform random function based on a transmit factor assigned to each node. Each packet also had a destination assigned by a similar process. Packets created in a single traffic session all had the same source and destination.

One measure of relative efficiency was the average time (per total nodes hopped) it took packets to reach their destination. Other, perhaps more significant, measures of effectiveness involve the amount of queuing delay or queue sizes that occurred during the test. This was observed in several ways.

The maximum queue size per simulation was recorded for every link and listed after every run. This information varied significantly and appeared to be influenced by the large influx of packets during the initiation of traffic sessions.

Half way thru the simulation, a group of links having the longest queues during the first half of the test were selected to be sampled during the second half of the test. The number of links selected was an input variable. The number of samples per links was also an input variable, but was normally set at 1000. The resulting distribution of sample sizes for the busiest links in the network, appeared to offer a stable, more representative measure of the algorithm's ability to process packets. The average sample queue size and its standard deviation was also calculated.

Other checks and measures included the average number of nodes hopped per packet, the average number of links used for a node's update cycle, and the longest best path established anytime during the test. Finally there are several checks to report if packets were excessively delayed, particularly due to dynamical changes in best paths during message transmission, resulting in an abnormal number of hops to the destination.

B. PROGRAMMING SCHEME

The simulation program was organized as a set of subroutines controlled by a simulation clock (Fig. 5.2) which is an inherent feature of SIMSCRIPT. Before the simulation begins, the routine is initialized by the main program which includes reading input variables, dimensioning arrays and printing out various input parameters. The main program also schedules the events on the simulation clock which starts several activity chains resulting in the generation of user traffic, the periodic update of the network and the collection of performance data.

The Main program schedules the first update originated by each node in the network. This is begun at a random time thru an event routine called NEW.UPDATE.MESSAGE. For the

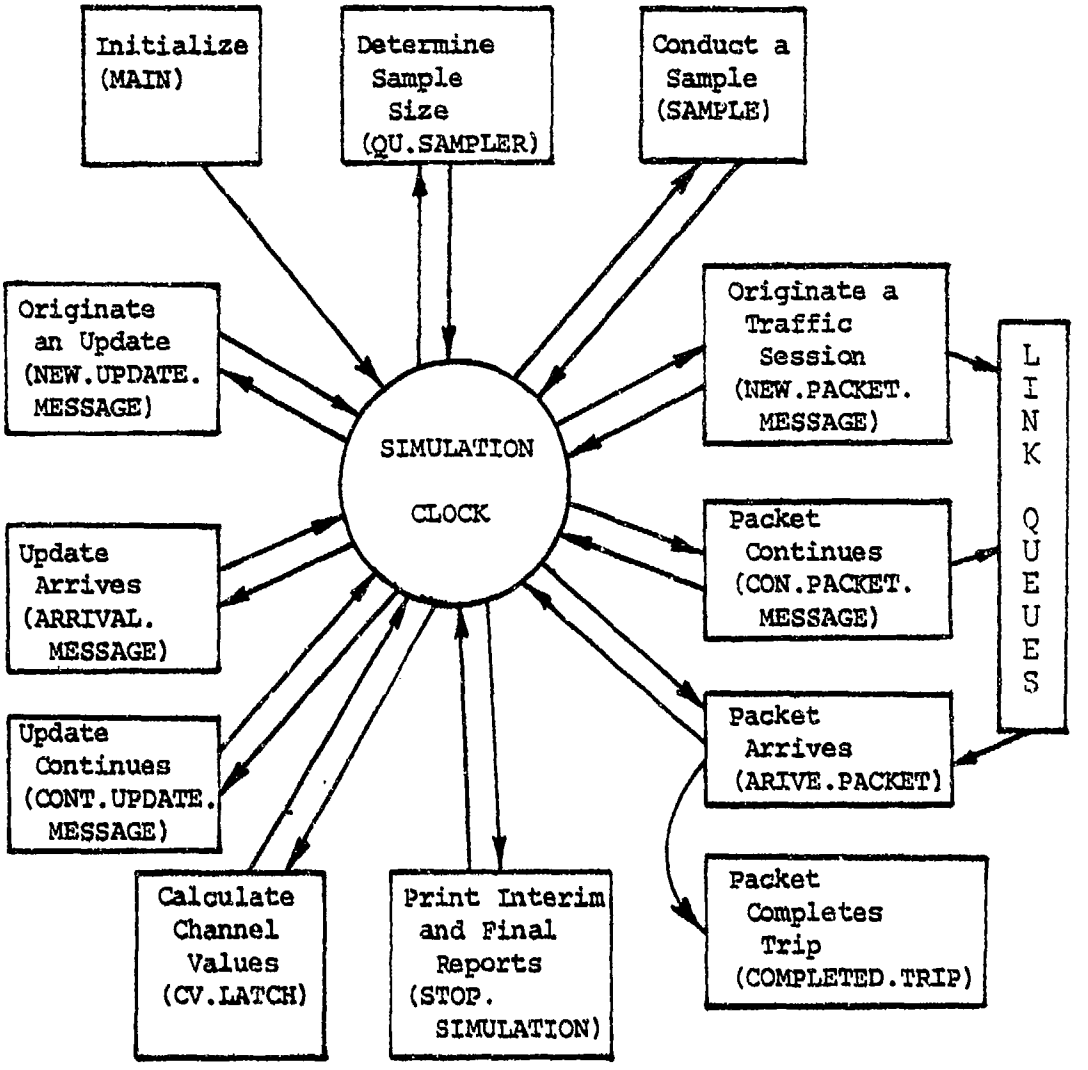


Fig. 5.2. Program Organization

designated node, this routine generates a U-msg for each of its neighbors and places the messages on the link to each neighbor. The routine schedules the arrival of each U-msg

after a pause to account for propagation time plus message duration. A time of 2ms was selected for the simulation. Finally this routine reschedules the designated node for its next update origination in an interval which was based on an input variable explained below.

Once a U-msg was originated, it was scheduled to arrive at neighboring nodes. The arrival of a U-msg was handled by a routine called ARRIVAL.MESSAGE. This routine is the heart of the update operation and implements the update portion of the algorithm in Chapter IV. The ARRIVAL.MESSAGE routine determines whether or not this U-msg should be relayed to the neighbors of the receiving node, which neighbors to relay it to, and what the contents of the relayed message will be. It simulates the processing time by scheduling retransmitted U-msgs to continue after a brief processing time. U-msg processing time was set at 0.1 x the packet processing time (.0001 sec) to reflect the priority of U-msgs and their small relative size.

After the processing time, the U-msg is placed on the next link by the CONT.UPDATE.MESSAGE routine and the U-msg is again scheduled to arrive at the next node in the selected transmission time of 2ms. This process continues

until the U-msg arrives at a node, is processed by the ARRIVAL.MESSAGE routine and considered no longer suitable for retransmission (due to an excessive net channel value). The result is the creation of a best path to the node (group or family) which originated the U-msg from every other node thru which the message successfully passed prior to discard.

During the course of the simulation, as link queues vary in size, channel values change. One of the most significant observations affecting the fundamental algorithm made during the simulations, concerns the timing of when channel values may be calculated. It was initially conceived that during an update cycle, a node could calculate its channel value to a neighbor whenever that node received a U-msg from that neighbor. However it was found that under a relatively high traffic rate, some node (i) might relay U-msgs having selected a best path node (j), but by the time node i received relayed versions of its own U-msg its channel value to node j might have changed dramatically, resulting in a loop (See Fig. 5.3). To remedy this problem, updates were constrained to start anytime during a (relatively large) time interval. This interval was followed by another equal size interval during which no updates could be started, but

existing updates could be processed. The minimum size of these intervals was large enough to insure that any existing update cycles would work their way out of the network before the next series of updates was allowed to begin. The calculation of channel values was synchronized in each node to take place once (and only once) near the beginning of the first (origination) interval. The operational feasibility of this synchronization requirement is not unreasonable, for very good network synchronization will be a likely requirement in order to take advantage of the benefits of spread spectrum modulation techniques, position location or other attractive capabilities of digital communication.

In the simulation, the Main program schedules the first channel value calculation with the routine CV.LATCH. Since this takes place at time zero of the simulation and no traffic has started, all links are initialized to the basic channel value of 1. CV.LATCH also calculates the update origination interval, based on the specified update interval which is an input variable (UP.DATE.PERIOD), and reschedules itself for every node in the network.

After the first update, the CV.LATCH routine uses historical queue information for each link to calculate a

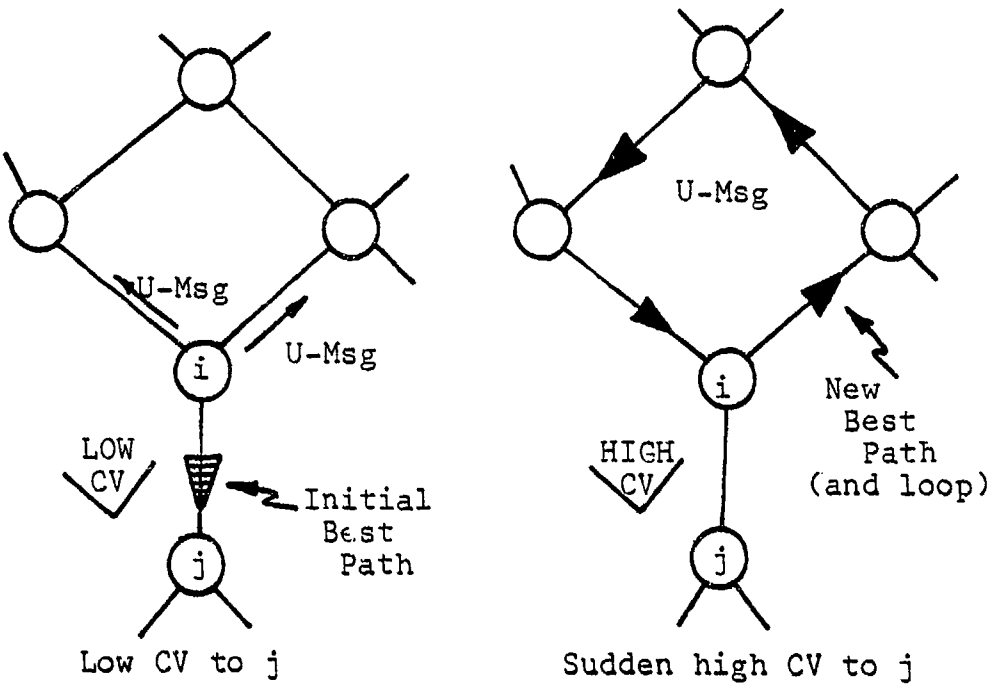


Fig. 5.3. Possible Result of Frequent CV Changes

new channel value for that link. This value is based on a time average of past queue sizes existing at that node over a time period called the WINDOW. The channel value is the integer part of

$$\frac{\tau_i Q_i}{\text{WINDOW}} + 1$$

where

Q_i = queue size of i th queue
τ_i = time interval over which queue = Q_i

summed for all queue measurements not older than the WINDOW for a given link.

The Main program begins to schedule traffic with an exponential arrival rate based on an input variable (AVE.NEW.TRAFFIC.INTERVAL). Traffic is started by calling a routine called NEW.PACKET.MESSAGE. The first traffic is generated after the network is allowed to complete one update cycle, thus insuring that all nodes have a best path to the other nodes, groups or families as appropriate. A traffic message (referred to as "session" in Appendix E) involves randomly selecting a source node, destination node and the number of packets in the message. The selection of the nodes is a function of input variables assigned to each node which dictate the relative frequency with which nodes will transmit and receive. The routine can also restrict destination nodes to be in the same group or family as the source node for a given percentage of the traffic messages (sessions) based on additional input data. The routine will send the first packet on the best path to its destination if the link is idle. If not it will place the packet in a link queue. In either case, all other packets in the message are placed in the queue.

When a packet leaves its source node, it is assigned the current simulation time which is checked again upon arrival at the packet's destination. This information is used to compute the average and peak times for packets to hop N nodes. Each packet counts the hops or nodes it passes thru enroute to its destination as explained later in this section.

When a packet leaves for its first best path neighbor, it is scheduled to arrive after an interval representing the packet transmission time, which is an input variable called PKT.XMN.TIME. For the simulation this value was fixed at 50ms, based on performance factors mentioned in Chapter III.

Finally NEW.PACKET.MESSAGE reschedules itself for the next traffic session which will have the same exponential inter-arrival rate mentioned above, but will result in the random selection of a new source, destination and message size (number of packets).

Enroute to their destination, packets arrive at neighboring nodes which is simulated in the ARIVE.PACKET routine. In this routine a packet is checked to see if it has reached its destination. If so it is processed in a routine called COMPLETED.TRIP discussed below. If not the

packet is processed; routed to the next best path neighbor based on the ID of the family, group or node of the destination node; and then either forwarded if the link is idle, or placed in the link's queue. ARIVE.PACKET schedules each arriving packet thru the CON.PACKET routine after a processing time delay which was a test parameter fixed at 0.1msec per packet. Finally ARIVE.PACKET goes back to the queue of the node which sent the last packet. If another packet is in the queue, it is placed on the link (by scheduling an ARIVE.PACKET for that packet) to the node which just received the last packet. If the queue was empty, it is designated as idle.

Meanwhile, when the packet scheduled for the CON.PACKET routine arrives, if the link to its next node is idle, it is placed on the link and scheduled to arrive (ARIVE.PACKET) at the next node in the packet transmission time (PKT.XMN.TIME) mentioned above. If not, it is placed in the queue for that link. In order to minimize large-scale loading shifts from one link to another, the algorithm does not change the routing of a packet coming out of a queue to be transmitted if, during the time the packet was waiting in the queue, the best path to its destination has changed. A packet keeps

its original routing unless the link has been broken (not covered in these simulations) and newly arriving packets are routed thru the best path node.

Eventually the packet reaches its destination. Here it is processed by the COMPLETED.TRIP routine. This routine collects and computes performance data including the number of nodes hopped by the packet, and trip time. It increments a counter which sums the number of packets hopping N nodes and records the highest trip time for N nodes. It keeps track of the number of packets arriving for each session and sums all the trip times for N nodes so it can later be divided by the total number of nodes making N hops to calculate the average trip time for N hops.

After four equal intervals (quarters), the simulation is stopped with the STOP.SIMULATION routine. This routine reprints selected input data. It also calculates and/or prints performance data for the simulation up to that point. Appendix E contains an example of the full printout which includes the average and peak packet transit time for N hops, and the maximum queue for every link. It also presents results of a statistical sampling of the links with the largest maximum queues during the first half of the simulation.

The Main program schedules the QU.SAMPLER routine at the mid-point of the simulation. This routine will identify the M links with the highest queues in the first half of the simulation. M is an input variable (SMP.LINKS). QU.SAMPLER then schedules a routine called SAMPLE which samples these M links in the second half of the simulation with an exponentially distributed time between samples with mean 1/S, where S is another input variable (NO.OF.SAMPLES). The queue sizes found during these samples increment a queue size counting array called QU.DISTR. STOP.SIMULATION prints the results of this queue sample (QU.DISTR) as well as calculates the average queue size and its standard deviation. After four reports STOP.SIMULATION halts the test.

C. ARRAYS AND TEMPORARY ENTITIES

SIMSCRIPT is an excellent programming language, particularly for its readability and simulation oriented functions. The encoded algorithm and related routines in Appendix E are written in SIMSCRIPT and also have additional documentation. However the organization of the arrays and attributes of the "message" and "pack" temporary entities contain several subjective encoding decisions.

Understanding these organizational decisions will help when reading Appendix E.

The arrays used in the program are listed in Fig. 5.4 There are one, two and three dimensional arrays (1-D,2-D,3-D). The array name is followed by its dimension(s), which may either be variable or constant, and by the different meanings for the subscripted variable (e.g. node ID). Below each array name is the meaning of the first (1-D), second (2-D) or third (3-D) subscript. Together the subscripts identify a variable location which may be used during the simulation.

For example, the first array (FAM.OF.GRP) is 1-dimensional. Its size is the sum of the number of nodes, plus the number of groups plus 25. Arguments of this array will be the program numbers for groups (program numbers are explained in Appendix B). The content of this subscripted variable is the family of the group in the argument.

The 2 and 3 dimensional arrays are read similarly. For example SMP.SET is a 2-dimensional array. The first argument is the count number of the link to be sampled which is determined by the QU.SAMPLER routine. The second argument identifies whether the variable is the "to" or

```
1-D

  FAM.OF.GRP (nc. of nodes+ grps+ 25) --> family ID
    1st-D (program ID for group i)

  QU.DISTR (250) --> sample count
    1st-D (queue size)

2-D

  LINK.ABLE (no. of links in network, 2) --> node ID
    1st-D (link number)
    2nd-D ( 1= 1st node | 2= 2nd node)

  TRACER (2 x test duration/ave. session interval, 2)
         --> no. of pkts
    1st-D (session number)
    2nd-D ( 1= original pkt count for this session
          | 2= pkts which reached destination)

  CLOCK.DATA (4 x no. of nodes, 2) --> time
    1st-D (no. of hops = N)
    2nd-D ( 1= net time for all pkts hopping N nodes
          | 2= highest individual trip time for
               a N-hop pkt)

  HOP.COUNT (4x no. of nodes, 2) --> no. of pkts
    1st-D (number of hops = N)
    2nd-D ( 1 = Not Used
          | 2= no. of pkts hopping N hops)

  SMP.SET (a selected no of links, 2) --> node ID
    1st-D (sample link ID number)
    2nd-D ( 1= "from" node ID | 2= "to" node ID)
```

Fig. 5.4. SIMSCRIPT Arrays (1 and 2 Dimensional)

"from" node for that link. The subscripted variable is the actual identity of the node.

Finally for the NEIGHBOR.LIST array, the first argument is a simple counting integer corresponding to one of the above node's neighbors (a node may have up to 6 neighbors). The third argument describes whether the subscripted variable will contain the ID of the neighbor node (1), an integer (1 or 0) indicating whether or not the neighbor is active (2), or the channel value to this neighbor (3).

```
3-D

  NEIGHBOR.LIST (no.of nodes, 6, 3)
                --> node ID or UP/DOWN or CV
    1st-D (node n's ID)
    2nd-D (a number listing from 1 to 6 of
           of node n's neighbors)
    3rd-D ( 1= neighbor ID | 2= link status
                | 3= CV from node n to neighbor)

  BEST.PATH (no. of nodes, no. of nodes+ groups
             families, 2) --> node ID or CV
    1st-D ("from" node ID)
    2nd-D ("to" ID of either node, group or family)
    3rd-D ( 1= best path neighbor
                | 2= CV thru best path neighbor)

  LINK.MONITOR (no. of nodes, no. of nodes, 3)
                --> busy signal (1) or Q size
    1st-D ("from" node ID)
    2nd-D ("to" node ID)
    3rd-D ( 1= idle/busy status
                | 2= current queue size
                     | 3= max queue size thus far)
```

Fig. 5.5. SIMSCRIPT Arrays (3 Dimensional)

Another advantage of SIMSCRIPT is the ability to create and destroy multivalued variables called temporary entities. By using these entities, groups of data can be shuffled and processed thru queues relatively easily. Another advantage is an efficient utilization of memory space because entities which are no longer needed can be destroyed and the memory freed for reuse.

The algorithm in Appendix E used two temporary entities extensively. The first is the MESSAGE entity. As shown in Fig. 5.6 the MESSAGE entity is used for both U-msgs and packets. There is more information in the SIMSCRIPT U-msg

MESSAGE

	UPDATE	PACKET
Type	1	2
Relayer	last relaying node	
Next.Stop	next receiving node	
Destination	originating node	packet's destination
Info 1	channel value	session number
Info 2	family of originator	packet serial number
Info 3	N/A	sum of nodes hopped
Info 4	N/A	time released from source
Info 5	group of originator	family or group of non-basic group node

PACK

Number	queue size due to last change
Entry.Time	time queue changed to above size
Pac.Neighbor	neighboring node to which the queue has been changed

Fig. 5.6. SIMSCRIPT Temporary Entities

than in the theoretical U-msg of Chapter IV simply for efficiency of programming and data collection. Note that several attributes of the Update MESSAGE and Packet MESSAGE have the same meaning and others do not.

The PACK entity (Fig. 5.6) is used in the CV.LATCH routine to calculate all link channel values. A pack is

created every time a queue size increases or decreases. PACKS are kept in a queue (called TIME.QUEUE) assigned to each node. They are kept until the PACK's ENTRY.TIME (the time it entered the queue) is older than the window. Each PACK has a NUMBER which is the new queue size which caused the PACK to be created. Each node may have several links, but all PACK are kept in the same queue. Therefore PAC.NEIGHBOR identifies which PACKs belong to each link for a given node.

D. SELECTION OF TEST PARAMETERS

For the purpose of the simulation, certain test parameters were selected to be fixed and others varied. For the fixed parameters, approximations were made based on the estimated performance characteristics of a typical system as described in Chapter III. Fig. 5.7 lists the major system and simulation parameters. There is no explicit distinction between fixed and varying parameters. However the estimate of 16,000 bps bit rate in Chapter III helped settle on a set of processing and transmission times for messages estimated to range from less than 100 bits (U-msg) to approximately 1000 bits for packets. The ranges of the varying parameter were also affected by the 16 Kbps bit rate estimate on one

end, and by a performance threshold on the other. These results are discussed in greater detail in Chapter VI.

FIXED PARAMETERS	
Pkt Processing Time in a Node	.0001 sec
U-msg Processing Time in a Node	.00001 sec
Pkt Transmission Time per Link	.05 sec
U-msg Transmission Time per Link	.002 sec
Number of pkts per session (uniformly distributed)	1 - 20
Links to be sampled	10
No. of Samples per Link	1000
Receiving/Transmitting factors (for each node)	1

VARYING PARAMETERS	
Period Between Updates	.01 - 1 sec
Simulation Time Limit	<= 2000 sec
Period Between New Traffic Sessions	.05 - .1 sec
Window Size	1 - 20 times update period
% Inner-Group/Family	0 - 75 %

Fig. 5.7. System Parameters

Other input data consisted of a description of the test network (Fig. 5.1) including the identification of nodes, groups, families and links. The network topology was fixed for all simulation runs.

VI. RESULTS, CONCLUSIONS AND RECOMMENDATIONS

The results and conclusions discussed in this chapter primarily involve the simulation of the update portion of the protocol in Chapter IV. Furthermore, in view of the wide variety of parameters that could have been varied in these simulations, many were fixed at what was considered to be reasonable approximations based on performance figures used to describe the theoretical system in Chapter III. The analysis centered around several parameters which were considered potentially to have the broadest affect on the system response, including the interval between update messages (or the rate at which updates were originated), the average arrival rate of new traffic sessions (or at the rate at which packets were created), and window size.

A. RESULTS AND OBSERVATIONS

One of the first simulations involved assigning a fixed channel value of 1 to all links in the test network (Fig. 5.1), defining the best path between any two nodes as the first path found with the lowest net channel value (also called the shortest path), and then fixing all best paths for the entire simulation. This is a static routing scheme,

using shortest direct paths in the sense of minimum number of hops. This fundamental network scheme was used to establish a minimum performance level and was used to measure the effectiveness of the update program in Appendix E. For the network in Fig. 5.1, all best paths were calculated and frozen at the beginning of the simulation. Then traffic sessions were originated at intervals of .05 sec and .08 sec. The network quickly congested. At an interval of 0.1 sec the network settled down with average sampled queue lengths from 2.2 to 3.1 for a 2000 sec simulation.

The remainder of the simulations involved the update algorithm applied to the same network (Fig. 5.1). The tests were divided into two groups. The first and largest group of simulations considered the whole network to be one basic group (all groups and family ID's were the same). This is essentially unfreezing the static network by applying the update algorithm. The second group of simulations involved the partitioning of nodes into groups and families which could then be compared to the basic group simulations.

1. Basic Group Tests

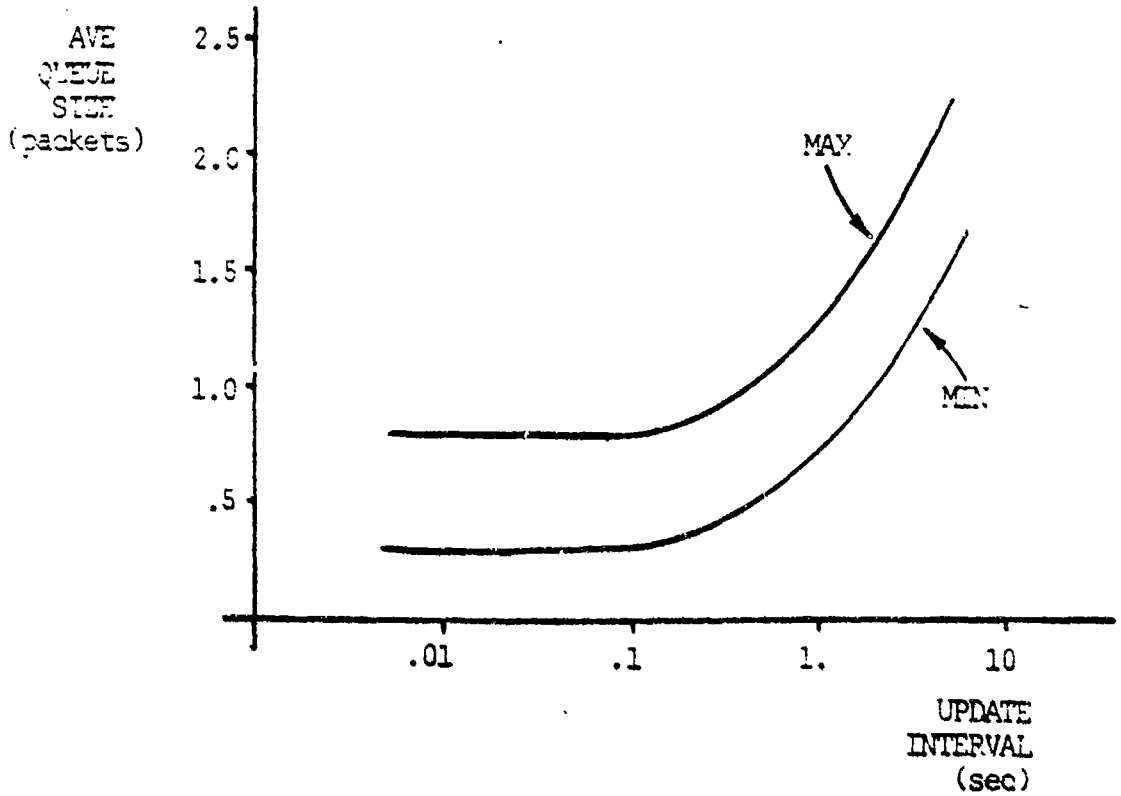
For the test network (Fig. 5.1), the frozen path baseline scheme could not function at a new traffic session period of less than 0.1 sec. The basic group tests results (Appendix D) suggested that 0.1 sec was also a good limit for the Update algorithm. However runs at new traffic session periods of .08 sec and .05 sec indicated a gradual loss of efficiency indicated by excessive queue lengths. The static network, on the other hand, had demonstrated catastrophic failure at these intervals. Runs at intervals averaging greater than 0.1 sec caused very little strain on the update algorithm, therefore the traffic inter-arrival interval of 0.1 sec average was selected for the large majority of the basic group (and group/family) tests.

At an average traffic inter-arrival interval of 0.1 sec, a message (session) averaging 10 packets was added randomly to the network at a rate of 10 messages (sessions) per second. Simulation results indicated that after 100 seconds of simulation clock time, any residual effects of starting the simulation were undetectable. Therefore test results were taken for simulation runs varying from 100 to 1000 sec. Runs greater than 1000 sec gave no indication of

new information about the length of queues or the rate at which packets could be processed. Therefore using an average message (session) arrival time of 0.1 sec and a test duration from 100 to 1000 sec, the primary focus of simulations were on the update period (or rate at which new update cycles were started) and window size.

The update period directly reflects the amount of overhead required by the network. In the static network, the overhead requirements are minimal, amounting to that required to initially establish the best path network. Therefore it is reasonable to expect better performance for an increase in overhead traffic. The basic group test indicated this improvement.

The average queue size was the primary measure of performance. The figures derived are conservative calculations since the sampling in the second half of the test was made of the busiest links found in the first half of the test. The static network's average queue length for a traffic inter-arrival interval of 0.1 sec was 2+ packets. The basic group algorithm approaches the static network as the Update period approached infinity. Fig. 6.1 shows the decrease in average queue size as the traffic session



The curve labelled "MAX" is the plot of the largest average queue sizes over the set of experiments.

The curve labelled "MIN" is the plot of the smallest average queue sizes over the set of experiments.

Fig. 6.1. Queue Size vs Session Interval (Basic Group)

interval size decreases. As expected, for relatively long update intervals (>1 sec), average queue sizes ranged between 1 and 3 packets. From there, as the update interval decreased, average queue sizes dropped quickly. Around 0.1 sec, the average queue size settled into a range of values between approximately .25 and .75 packets. Many runs were made in this range and there was no tendancy for results to

prefer any particular part of this range as the test duration was varied. Results were very stable (see Appendix D).

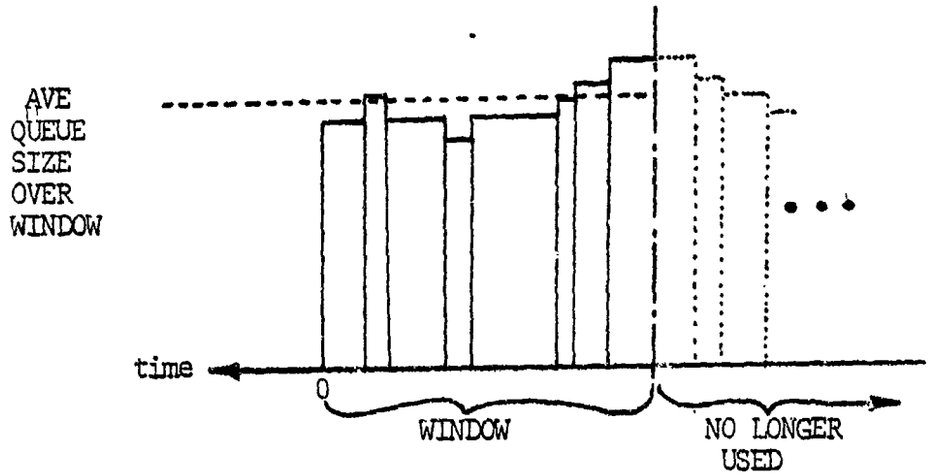


Fig. 6.2. Window Calculations

Window size was also varied to determine its impact on average queue size. At first the channel values were calculated over various window sizes based on a linear weighted time average. The weighting scheme gave the highest weights to the most recent queue sizes. However simulation results showed that this scheme resulted in larger average queue sizes than a straight unweighted time average as shown in Fig. 6.2 The height of the blocks

represent the size of the queue. Their width represents the length of time that the queue did not change in size. The window indicated how far back on the time line (in Fig. 6.2) the program would go to calculate the average queue, and therefore the channel value for a particular link.

Window size also proved to be a very stable parameter. Window sizes between 1 and 10 times the update period gave no indication of influencing the curves shown in Fig. 6.1 As the window size increased to over 20 times the update period, there were slight increases in average queue size.

The standard deviation of the average queue size for basic group tests varied slightly from 1.2 to 2.3 packets, with the smallest standard deviation for update periods of 0.1 sec.

2. Family/Group Tests

The basic group test results were compared to a fundamental static network routing scheme. The family/group tests results (Appendix D) were primarily compared to the basic group results. Although the proposed advantages of the family/group aspect of the update algorithm presented in this paper are based on the assumption that the majority of

the traffic is confined to inner group or inner family transactions, most of the family/group tests placed no restrictions on which node sent or received traffic. Under these conditions, the same set of messages with the same sources and destinations used in the basic group tests were used in the family/group tests. Runs involving restricted traffic is briefly mentioned at the end of this section.

As expected, the average queue size increased in the family/group tests. For update periods ranging from .05 to 0.5 sec, average queue sizes varied from 0.5 to 1.5 packets. Additionally, the standard deviation increased to approximately 3.7 to 4.2 packets. The benefits due to this drop in performance was the decrease in overhead traffic. Typically, the average number of links used by U-msgs as the result of a single node originating one update cycle during the basic group test ranged from 75 to 90 links. This is because every node had to have a best path for every other node. For a typical family/group test, the average number of links used dropped to around 23 to 27 links. Therefore for a (roughly) 50 percent improvement (decrease) in maximum average queue size, a 200 percent increase in overhead traffic was required.

Another effect of the family/group algorithm is an increase in the average number of nodes hopped by all packets. Because of the addressing given to a packet starting out for a node in a different family (it starts out with a family best path neighbor), more average links are normally required in family/group tests. An interesting observation is that occasionally a few (most often 1 or 2 in more than 20,000) packets in a family/group test would make an unusual number of hops, clearly indicating that it is looping due to changes in its best paths. However, invariably the total time required for this packet to finally reach its destination was well within the average times required by other packets which used far fewer hops. The cause of these (relatively rare) oscillations is not obvious. It is probably due to the large number of links which cross a group or family boundary. During an update each link represents an entry port to that particular related group or family and each acts as an originator of a U-msg for that larger entity (or super-node). Therefore from update to update, the best path port to that super-node could change by a large physical distance. However it is not clear if this observation indicates a serious problem,

since the packets still invariably arrive in a timely fashion. As a safeguard, an additional routine was added to the simulation program to check the hop count on all outstanding packets when the test ended. At no time was there an indication that an undelivered packet was looping or making excessive hops.

Compared to the basic group simulations, relatively fewer runs were made for the family/group tests. However these results suggested that as the ratio of window size to session interval increased over the range from 1 to 10, average queue size also increased slightly. Additional testing may indicate that a relatively flat performance band similar to Fig. 6.1 also exists in this case.

Finally several runs were made with the same test network with the additional restriction that 50 percent of all traffic is inner group and 50 percent of the remaining traffic is inner family. There were too few runs to establish a trend. However the results suggested a decrease in average queue size and standard deviation.

B. CONCLUSIONS

This was very much a preliminary investigation of this network management protocol. It would be improper to

identify much more than broad performance characteristics or trends.

The update algorithm clearly functions properly. Both the basic group concept and the family/group concept responds to changing channel values and provides routes that can be used under a reasonable traffic load. The algorithm is also very stable and robust. Fig. 6.1 indicates that for a traffic session interval of 0.1 sec and longer, the algorithm has a good and very flat performance curve for an update period of approximately 0.1 sec and less.

Investigating the update interval should continue to be a focal point in future analysis. For a given network performance level it will always be important to minimize the update interval, since it reflects the overhead traffic that the network must process.

On the other hand, as new and broken links are integrated into the simulation, they will add counter-arguments to the continued increase in the update periods. As links are broken, if alternate links are not available, nodes rely on U-msgs to unfreeze traffic and provide new best paths. Therefore future analysis must find a balance that will optimize this situation.

The family/group concept provides very good economy (and robustness, as compared to gateway nodes) with a modest decrease in performance. For networks which operate extensively within smaller sub-network boundaries (such as a typical military network), the family/group concept appears to offer a significant savings in overhead traffic compared to the same network operating as one basic group.

C. RECOMMENDATIONS FOR FURTHER STUDY

This preliminary study indicates that the update portion of the decentralized routing protocol described in Chapter IV accomplishes the fundamental requirement of routing user traffic. Follow-on investigations are needed to integrate the new and broken link concepts described in Chapter IV. Wherever possible, the program in Appendix E was written to facilitate this next step in the investigation. It can similate the loss, gain, and the planned movement of nodes by scheduling the failure and awakening of links on the simulation clock. Both single node and group movements could be simulated.

This protocol was conceived to be simple and practical. A degree of simplicity has been retained. However to become a practical protocol, there are several topics which need to be considered, that are not addressed in this study.

The primary problem is how to cope with splintering. Splintering may be a single node which crosses a group or family boundary. If this problem is applied to a single basic group network, it then reduces to the problem of a node leaving the network or a new node entering the network.

Splintering may also be defined as groups of nodes being completely cut off from the other nodes in its basic group. This sub-group may be left to operate autonomously or find itself in the middle of another basic group. A practical example based on the military organizations mentioned earlier would be the movement of a company, which is part of a battalion's basic group net, thru another battalion's sector. It is conceivable that the company may lose all direct links with its basic group during the movement. A practical protocol must also accommodate this splintering to the point where a basic group is divided into two or more equal parts, leading to the question of determining which group is the splinter and which is the remainder of the original basic group.

Finally, assuming a practical protocol can be fully developed, there is the much more difficult problem of measuring its efficiency both in an absolute sense, and in relation to other existing decentralized algorithms.

APPENDIX A

ALTERNATE LINK THEORY

CLAIM: During a shortest path update process, it may be possible to identify an alternate link which may be used to maintain traffic flow (although at some degraded level) in the event that a node's IMMEDIATE DOWNSTREAM link in a particular shortest path is broken. The switch will result in a non-optimum but LOOPFREE network. LOOPFREE in this discussion implies loopfree in the narrow sense. That is, traffic leaving a node for a given destination is assured that it will not loop back into the sending node.

DISCUSSION: Simply stated, the concept is that some optimum path routing algorithms may acquire information that is normally discarded, but may be used at individual node level to switch traffic to an alternate link if a break is found in that node's immediate downstream link along the optimum path. This switch does not necessarily leave the rest of the network optimally routed, but it is LOOPFREE. It may be useful as a temporary fix until the next update process is received.

PROOF: Any set of shortest path routes in a network can be expressed as a spanning tree, which is always loopfree.

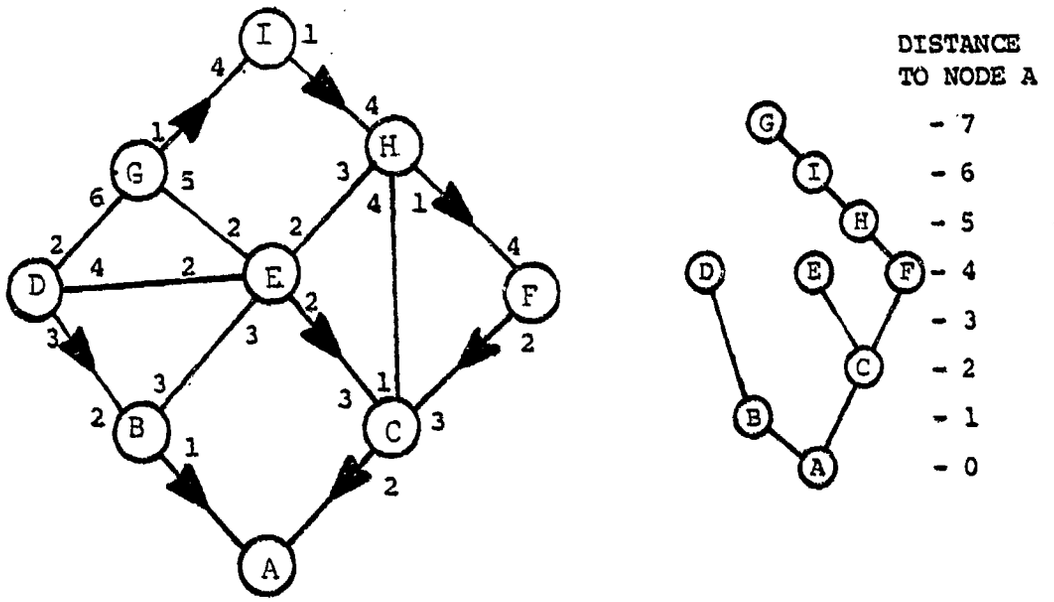
In the following examples, the spanning tree is vertically scaled to represent the channel value/distance to the tree root. Every link is assumed to have a minimum value of 1, however this is not critical in the proof.

A loop implies that a route passes through the same node more than once.

If every link has a minimum value of 1, then the total distance for each node in an optimum spanning tree to the root must be at least one larger than the next node downstream.

Therefore a loop cannot exist in a spanning tree because once traffic leaves a node in a spanning tree, it will never arrive at a node of equal distance to the root. Under normal operations, if more than one node has the distance to the root of d, then a particular message for that root will only pass through (at most) one of these nodes of distance d. Furthermore, once traffic reaches a node of distance less than d, it will never pass thru any node of distance d under normal traffic flow conditions.

Consider the following network and spanning tree with A as the root.



Applying the concept that a message will only pass thru one node of distance d to the above spanning tree, we see that if a message is 4 units away from node A, it is EITHER at node D, E or F. The message will be at one of these three and will never pass thru the other two.

Similarly, if a message at node G was transplanted in node D or a message in node H was transplanted in node C (where C and D both have distances less than H and G) the message could continue to the root without ever passing back thru the initial node (H or G). Note that this is not the case if a message in node C was transplanted in node H. Note also that node H's distance is greater than node C's

distance to the root. This is an indicator that a loop condition is possible.

Reviewing the above network diagram we see that there are many unused links if traffic to A is restricted to the Best Paths (⟶). However if an established best path is broken, these links provide a potential bridge which may transplant traffic from the node which discovers a broken link, to an adequate adjacent node which circumvents the broken link. The necessary and sufficient factor to determine whether an adjacent node is adequate is its distance to the root A. As long as the adjacent node's distance to A is less than or equal to the distance via the broken Best Path link, traffic transferred to it will never loop back to the transferring node (and presumably will proceed to node A).

APPLICATION: Consider a simple algorithm where cummulative distance to the root/sink was used to determine the Best Path. At each node, one best path would ultimately be selected over others. In making this selection each node learns the best path distance to the sink thru its adjacent neighbors, adds its distance to each adjacent neighbor and picks the Best Path. However the best path distance from

the non-selected adjacent neighbors may still be useful. If this distance is <= the node's best path distance to the sink, and if the node detects a break in its best path link, it may transfer traffic around the broken link by using one of the adjacent nodes, with the assurance that the new path is loop free. It is perfectly conceivable that more than one alternate link may be available to a given node at one time. Using the network and the tree shown above as an example, we have:

NODE/ (distance to A)	POSSIBLE ALTERNATE LINK(S)/ (distances)
B/(1)	None
C/(2)	None
D/(4)	E/(4)
E/(4)	B/(1), D/(4)
F/(4)	None
G/(7)	D/(4), E/(4)
H/(5)	C/(3), E/(4)
I/(6)	None

Note that the transferring node must still add the link distances from it to the selected adjacent node before picking the shortest alternate path.

APPENDIX B

DISTRIBUTED PROTOCOL ALGORITHM

```
**********************************************************************
SYMBOLS AND DEFINITIONS

     A(d) ==> Set of alternate neighbor nodes for destination d
     L(d) ==> Distance from B(d) to d (G/d or F/d)* on best path
   AL(d,l) ==> Distance from A(d) = l to d (G/d or F/d)* on l's best path
    AX(d) ==> Message to neighbor indicating it has been selected B(d)
     B(d) ==> Neighboring node on best path to d (G/d or F/d)*

  * (G/d or F/d) indicates that group or family identities can be
  use in the argument of these symbols in place of d to define
  intra-group and intra-family operations.

      G/n ==> Group identity of node n
      F/n ==> Family identity of node n
    d(i,n) ==> Channel value (distance) of link from node i to node n
     D(n) ==> Distance from node n to node d on its best path
        d ==> d, G/d or F/d for inner group, intra-group or
              intra-family operations as required
    TRANS ==> Originate or relay a message to appropriate neighbor(s)
     EXIT ==> No further action required
**********************************************************************

A.  UPDATE INITIATION

  I. Xmt U(i,d,D(i)) to all neighbors where
       i:= i
       d:= i
       D(i) = 0

 II. Schedule next Update origination in designated Update period.

B.  RECEIVE / PROCESS UPDATE

  I. Rcv U(i,d,D(i)) at node i, & G/i = G/i

     1) G/i ne (not equal) G/i
        EXIT

     2) If B(d) = i
        L(d):= D(i); D(i):= D(i)+ d(i,i)
        TRANS U(i,d,D(i))
        EXIT

     3) If B(d) ne i & D(i)+ d(i,i) <= D(i)
        If L(d) < D(i)+ d(i,i)
           A(d):= A(d) U B(d)
           AL(d,B(d)):= L(d)
        If any i∈ A(d)
           A(d):= A(d)-i
        B(d):= i
        L(d):= D(i); D(i):= D(i)+ d(i,i)
        TRANS U(i,d,D(i))
        EXIT

     4) If B(d) ne i & D(i) <= D(i)
        A(d):= A(d) U i
        AL(d,i):= D(i)
        EXIT

     5) If D(i) > D(i) & i∈ A(d)
        A(d):= A(d)-i
        EXIT

 II. RCV U(i,d,D(i)) AT NODE I & G/I NE G/D & F/I = F/D
```

```
1) If F/i ne F/1
     EXIT

2) If G/1 = G/d & 1 ne d
     EXIT

3) If B(G/d) = 1
     L(G/D) := D(L) ; D(G/D) := D(L)+ D(I,L)
     TRANS U(i,G/d,D(G/d))
     EXIT

4) If D(1)+ d(i,1) <= D(G/d) & B(G/d) ne 1
     If I(G/d) < D(1)+ d(i,1)
        A(G/d) := A(G/d) U B(G/d)
        AL(G/d,B(G/d)) := L(G/d)
     If 1 e A(G/d)
        A(G/d) := A(G/d) - 1
     B(G/d) := 1
     L(G/d) := 1; D(G/d) := D(1)+ d(i,1)
     TRANS U(i,G/d,D(G/d))
     EXIT

5) If B(G/d) ne 1 & D(1) <= D(G/d)
     A(G/d) := A(G/d) U 1
     AL(G/d,1) := D(1)
     EXIT

6) If D(1) > D(G/d) & any A(G/d) = 1
     A(G/d) := A(G/d) - 1
     EXIT
```

III. RCV U(L,D,D(L)) AT NODE I & F/I NE F/D

```
1) If F/1 = F/d & 1 ne d
     EXIT

2) If B(F/d) = 1
     L(F/D) := D(L) ; D(F/D) := D(L)+ D(I,L)
     TRANS U(i,F/d,D(F/d))
     EXIT

3) If D(1)+ d(i,1) <= D(F/d) & B(F/d) ne 1
     If L(F/d) < D(1)+ d(i,1)
        A(F/d) := A(F/d) U B(F/d)
        AL(F/d,B(F/d)) := L(F/d)
     If 1 e A(F/d)
        A(F/d) := A(F/d) - 1
     B(F/d) := 1
     L(F/d) := D(1) ; D(F/d) := D(1)+ d(i,1)
     TRANS U(i,F/d,D(F/d))
     EXIT

4) If B(F/d) ne 1 & D(1) <= D(F/d)
     A(F/d) := A(F/d) U 1
     AL(F/d,1) := D(1)
     EXIT

5) If D(1) > D(F/d) & any A(F/d) = 1
     A(F/d) := A(F/d) - 1
     EXIT
```

IV. Any traffic frozen due to X(d') is released as soon as a new B(d') is selected.

C. BROKEN LINK PROCESS

I. Node i discovers link with node j is broken.

For each d' s.t. B(d') = j :

```
1) If A(d') ne 0
     n:= A(d') for which AL(A(d'),d')+ d(i,A(d'))
          is min over A(d')
     s:= AL(n,d')
     B(d') := n
     L(d') := s
     TRANS AX(d') to n
     EXIT

2) If A(d') = 0
     TRANS X(d') to all neighbors
     Freeze traffic destined for d'
     B(d') = 0
     L(d') = ∞
     EXIT
```

II. Rcv X(d') at node i from node l

```
1) If B(d') = l
     PARA. I.1) & 2) ABOVE

2) If B(d') ne l
     EXIT
```

III. Rcv AX(d') at node i from node l

```
If any A(d') = l
A(d') := A(d') - l
EXIT
```

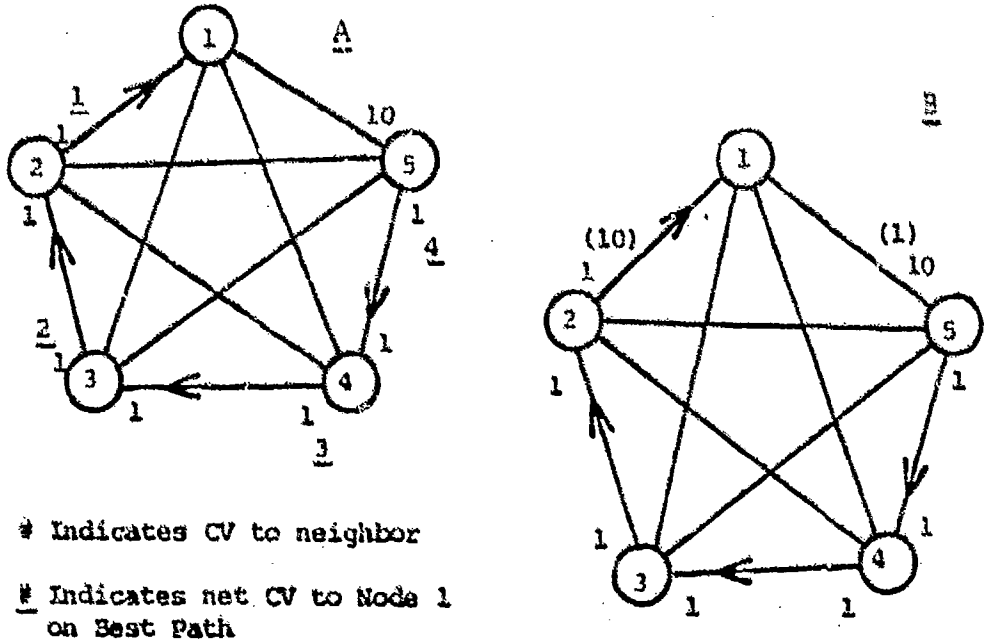
APPENDIX C

WORST CASE GROWTH OF UPDATE MESSAGES

It is unlikely that any distributed routing algorithm, which attempts to balance traffic on all links in a busy network and takes a finite time to update a network, will ever produce a network-wide routing scheme which is truly optimized. For example, delays due to propagation time and processing time in each node will cause a time difference between the node which originated the update and the last node which was affected by that origination. During this period, particularly under heavy traffic loads, conditions which existed in and around the originating node at the time the update was started may be very different from those existing by the time the most distant node is updated. Generally, the longer it takes for an update cycle to propagate throughout the network, and the longer the time between update cycles, the more conditions could change, thereby degrading an ideal routing scheme. Therefore it is germane to consider how long it would take for an algorithm to optimize a network if a sudden change in link status caused a worst case situation.

It is important to understand that the following analysis assumes that after the status (e.g. loading) of the links have changed, they are theoretically frozen until the network achieves re-optimization. Without this assumption, as stated above, it may be impossible to arrive at a fully optimized routing scheme in a changing network at any point in time.

For the algorithm presented in this paper, under worst case conditions, it could take up to (approximately) 3x(n**3) update messages to optimize a network of n nodes. The following figure shows a richly connected network.



* Indicates CV to neighbor

_ Indicates net CV to Node 1 on Best Path

(#) Indicates new CV at next update

Network A shows the best paths to Node 1 as of the last update. To maintain the worst case conditions, we will assume that the channel values of the links inside the network (the star) are always so large that they will never be selected as a best path link to Node 1. However each of these links will require that another U-msg be sent each time a node at either end updates its current best path or adopts a new one.

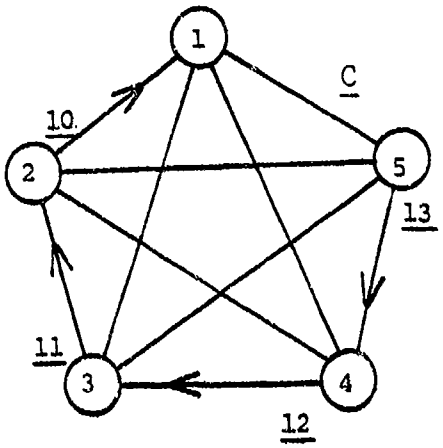
The channel values in parenthesis in network B represent changes in the channel values since the last update and will be used in the next update cycle originated by node 1. As the first update cycle begins, all nodes receive a U-msg from node 1 (for n=5 nodes, n-1 U-msgs are initially sent out at the beginning of a cycle). It is assumed that (under worst case conditions) the U-msg is relayed counter-clockwise (CCW) thru node 2 upstream along the best path arriving at node 5 sometime after nodes 3,4 and 5 have rejected the direct U-msg from node 1 (because they compared their outdated net best path channel value to the current channel values in the U-msgs.).

As the U-msg thru node 2 works itself upstream along the best path, it updates each node's net best path channel

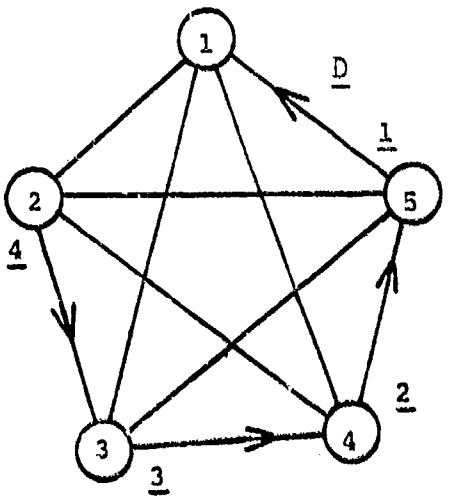
value and causes each node to relay the update to all neighbors except the sending node (n-2 relays for n-1 nodes) causing a relay of (n-1)(n-2) U-msgs. Adding the original n-1 U-msgs from node 1;

$$(n-1)(n-2) + (n-1) = (n-1)(n-2+1) = (n-1)**2.$$

In this first update cycle, (n-1)**2 U-msgs have gone out and not a single node has changed its best path neighbor to node 1. But this was still a significant step because each node now knows the true distance along its best path to node 1 as shown in Network C.



# Indicates net CV to Node 1 on Best Path



Some time later node 1 originates its next U-msg cycle. This time (worst case) it is assumed that the U-msg thru

node 2 works its way CCW to node 5 before node 5 receives its U-msg direct from node 1. This is nearly a repeat of the previous cycle causing another (n-1)**2 U-msg to be initiated. However when the U-msg direct from node 1 finally arrives at node 5, node 5 picks a new best path (direct to node 1) and relays the U-msg to all neighbors (except the sending node). When node 4 receives node 5's U-msg, it selects this new best path, informs all neighbors and the process continues until the network has now selected the optimum routing scheme as shown in Network D.

This final series of U-msgs involving (n-1)(n-2) transmissions brings the total U-msgs generated over the two update cycles (originated by node 1) to

$$2(n-1)^{**}2 + (n-1)(n-2)$$

which is approximately

$$3((n-1)^{**}2) \longrightarrow 3(n^{**}2).$$

Since there are n nodes in the network, each initiating its own update during a single network update cycle, the total (worst case) number of U-msgs possible is 3(n**3) (over two network-wide update cycles in this example).

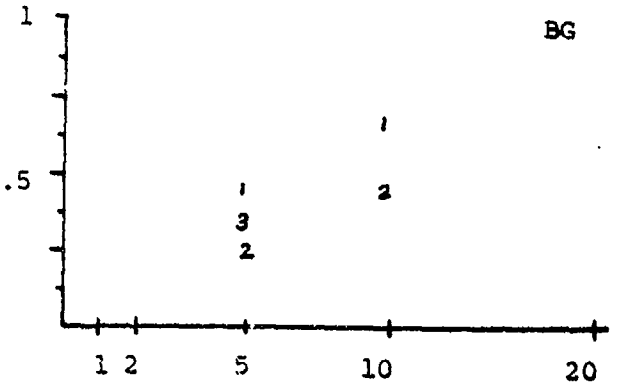
APPENDIX D

SIMULATION RESULTS

This appendix illustrates many of the results of the simulation runs for the program in Appendix E. For all results in this annex, the only parameters varied were the Update period, window size and time limit/duration of the simulation run. Data is divided into two major areas; basic group test results (BG) and family/group test results (F/G). Each plot corresponds to the Update period size indicated to its left. The plotted data is the average queue size for a run derived from sampling 10 links (those having the highest average queues over the first half of the simulation run) approximately 1000 times each during the second half of the simulation run. Results for a given test duration are represented by a number corresponding to the size listed adjacent to the plot. The vertical axis is average queue size. The horizontal axis is the Window/Update Period ratio.

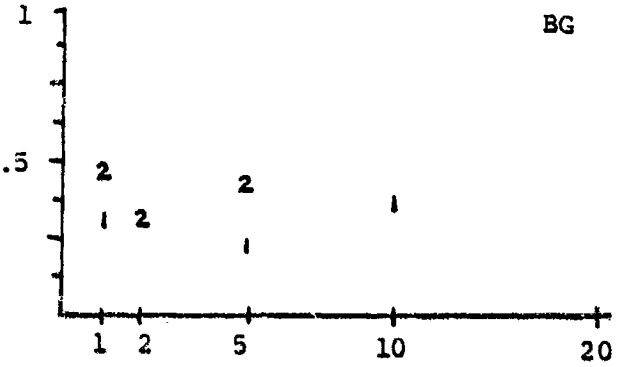
UPDATE PERIOD = .01 sec

Test Duration
1 - 100
2 - 150
3 - 200



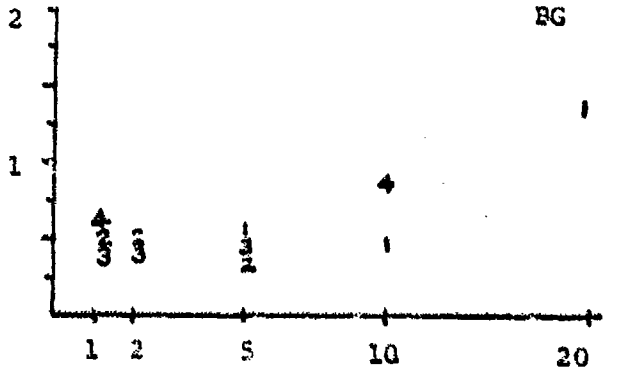
UPDATE PERIOD = .025 sec

Test Duration
1 - 100
2 - 250



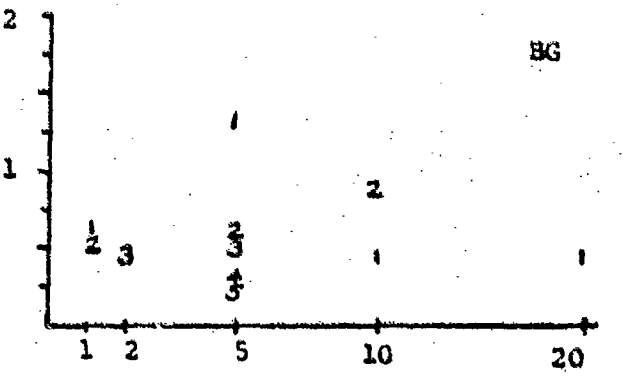
UPDATE PERIOD = .05 sec

Test Duration
1 - 100
2 - 150
3 - 200
4 - 250



UPDATE PERIOD = .1 sec

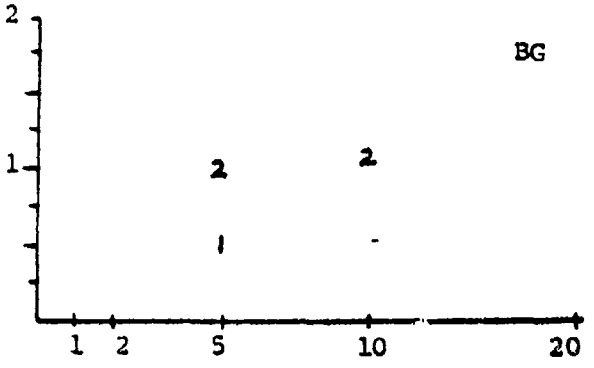
Test Duration
1 - 100
2 - 200
3 - 500
4 - 600
5 - 800



ALL PLOTS ARE AVE QUEUE SIZE VS WINDOW/UPDATE PERIOD

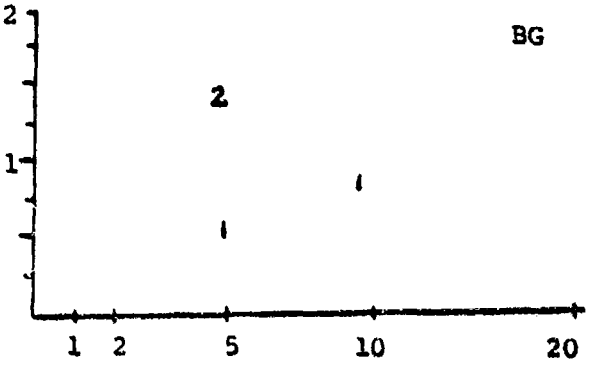
UPDATE PERIOD = .25 sec

Test Duration
1- 100
2- 500



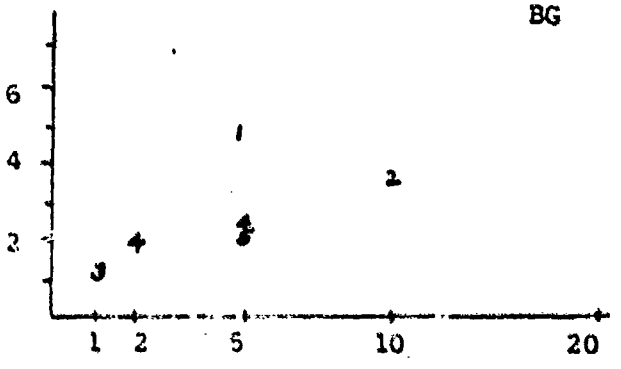
UPDATE PERIOD = .5 sec

Test Duration
1- 60
2- 100



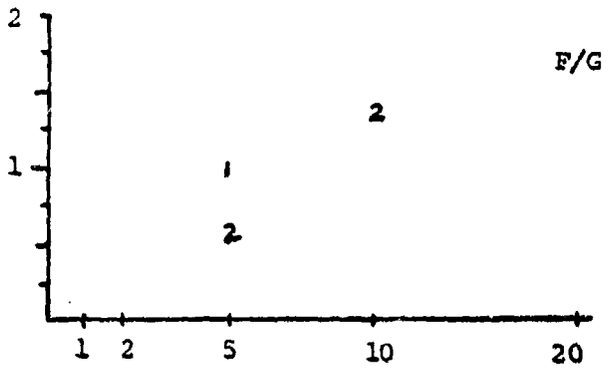
UPDATE PERIOD 1 sec

Test Duration
1- 40
2- 50
3- 60
4- 100
5- 200



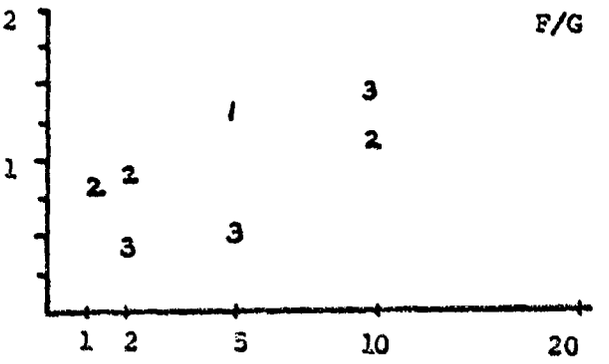
UPDATE PERIOD = .05 sec

Test Duration
1 - 200
2 - 500



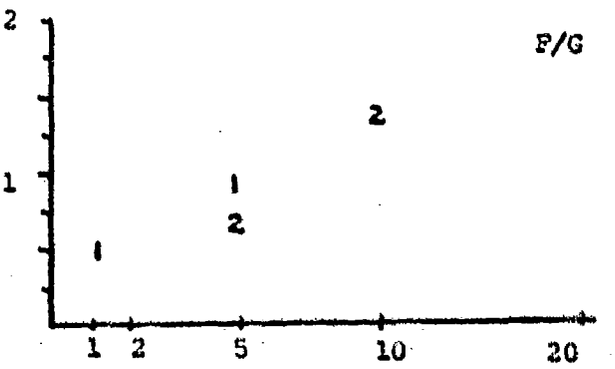
UPDATE PERIOD = .1 sec

Test Duration
1 - 100
2 - 200
3 - 1000



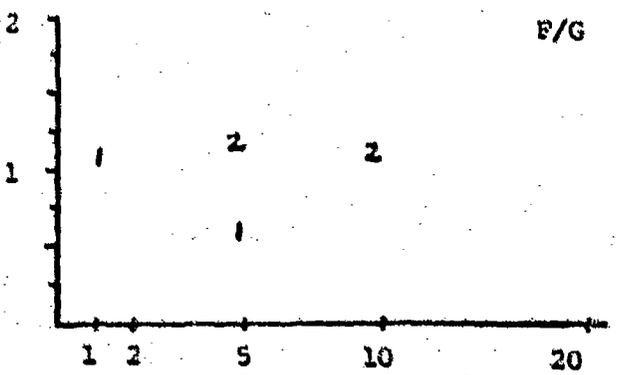
UPDATE PERIOD = .25 sec

Test Duration
1 - 100
2 - 500



UPDATE PERIOD = .5 sec

Test Duration
1 - 200
2 - 500



APPENDIX E

SIMULATION PROGRAM

```
FILE: DOC      S        A   NAVAL POSTGRADUATE SCHOOL

//HOBNAME1 JOB (2034,0058),'HERITSCH 1052',CLASS=A
//*MAIN LINES=(15)
//  EXEC SIM25CLG,REGION.GO=1024K,PARM.GO='MAP,SIZE=760K'
//SYSPRINT DD SYSOUT=A
//*YSPRINT DD DISP=SHR,UNIT=3350,VOL=SER=MVS004,DSN=S2034.COMPILE,
//*DCB=(RECFM=FB,LRECL=133,BLKSIZE=4123)
//SIM.SYSIN DD *
PREAMBLE
''
NORMALLY MODE IS INTEGER
GENERATE LIST ROUTINES
''
PERMANENT ENTITIES
   EVERY NODE HAS A TRANSMIT.PERCENT, A RECEIVE.PERCENT, A GROUP,
    A FAMILY, OWNS A QUEUE AND A TIME.QUEUE
     DEFINE TRANSMIT.PERCENT AND RECEIVE.PERCENT AS REAL VARIABLES
''
TEMPORARY ENTITIES
   EVERY MESSAGE HAS A TYPE, A RELAYER, A NEXT.STOP, A DESTINATION,
    AN INFO1, AN INFO2, AN INFO3, AN INFO4, AN INFO5 AND MAY BELONG TO
     A QUEUE
     DEFINE INFO4 AS A REAL VARIABLE
   EVERY PACK HAS A NUMBER, AN ENTRY.TIME, A PAC.NEIGHBOR AND MAY
    BELONG TO A TIME.QUEUE
     DEFINE TIME.QUEUE AS A LIFO SET
     DEFINE ENTRY.TIME AS A REAL VARIABLE
''
EVENT NOTICES INCLUDE STOP.SIMULATION, QU.SAMPLER, SAMPLE, CV.LATCH
   EVERY NEW.UPDATE.MESSAGE HAS A SENDING.NODE, AND A TYPE.MESSAGE
   EVERY ARRIVAL.MESSAGE HAS AN ID.MESSAGE.NUMBER
   EVERY CONT.UPDATE.MESSAGE HAS A LAST.NODE, A NEXT.NODE, A NET.CV,
    A SOURCE, A FAMILY, A HOP.CNT AND A GR.OUP
   EVERY ARIVE.PACKET HAS AN ID.NUMBER
   EVERY CON.PACKET.MESSAGE HAS AN IDENT.MESSAGE.NUMBER
   EVERY NEW.PACKET.MESSAGE HAS A T.MESSAGE
   EVERY COMPLETED.TRIP HAS A MES.NUM
''
DEFINE UP.DATE.PERIOD, PROCESSING.TIME, PKT.XMN.TIME AND TIME.LIMIT
 AS REAL VARIABLES
DEFINE TRNS.PCNT AND RCV.PCNT AS REAL VARIABLES
DEFINE LINKS AS A VARIABLE
DEFINE MSG.HLT AS A VARIABLE
DEFINE NEIGHBOR.LIST AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE BEST.PATH AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE CHANNEL.VALUE TO MEAN INFO1
DEFINE FAM.LY TO MEAN INFO2
DEFINE GRP TO MEAN INFO5
DEFINE GRPS, FMLYS AND NGPS AS VARIABLES
DEFINE LINK.ABLE AS A 2-DIMENSIONAL ARRAY
DEFINE LNK.MONITOR AS A 3-DIMENSIONAL ARRAY
DEFINE TRACER AS A 2-DIMENSIONAL ARRAY
DEFINE CLOCK.DATA AS A 2-DIMENSIONAL REAL ARRAY
DEFINE HOP.COUNT AS A 2-DIMENSIONAL ARRAY
DEFINE SMP.SET AS A 2-DIMENSIONAL ARRAY
DEFINE QU.DISTR AS A 1-DIMENSIONAL ARRAY
DEFINE FAM.OF.GRP AS A 1-DIMENSIONAL ARRAY
DEFINE IDLE TO MEAN 0
DEFINE BUSY TO MEAN 1
DEFINE PACKET TO MEAN 2
DEFINE AVE.NEW.TRAFFIC.INTERVAL, PKT.MIN AND PKT.MAX AS REAL VARIABLES
DEFINE EST.MAX.OP.TRAFFIC AND TOT.NEW.TRAFFIC AS VARIABLES
DEFINE TRANS.NUMBER TO MEAN INFO1
DEFINE PACK.NUMBER TO MEAN INFO2
DEFINE NODES.HOPPED TO MEAN INFO3
DEFINE RELEASE.TIME TO MEAN INFO4
DEFINE FM.GP TO MEAN INFO5
DEFINE YES TO MEAN 1
DEFINE NO TO MEAN 0
DEFINE UPDATE TO MEAN 1
DEFINE TRAF.LIMIT AS A VARIABLE
DEFINE WINDOW AS A REAL VARIABLE
DEFINE UP.PAC.RATIO AS A REAL VARIABLE
```

```
FILE: DOC      5        A   NAVAL POSTGRADUATE SCHOOL

DEFINE PRNT AS A VARIABLE
DEFINE IN.GROUP, IN.FAMILY, SELECTOR AS REAL VARIABLES
DEFINE NET.U.LINKS, UP.STARTS, MAX.U.HOPS AS VARIABLES
DEFINE TIMER AS A REAL VARIABLE
DEFINE SMP.LINKS, NO.OF.SAMPLES, SMP.CNTR AS VARIABLES
DEFINE EARLIEST.UPDATE AND LATEST.UPDATE AS REAL VARIABLES
DEFINE U.XMN.TIME AS A REAL VARIABLE
DEFINE PEEK AS A VARIABLE
END ''OF PREAMBLE
''
''
MAIN
''
''¢¢¢¢ SET SOME VARIABLES
''¢¢
''¢¢ SET THE RANGE OF PACKETS PER SESSION (PKT.MIN / MAX)
''¢¢ SET A LIMIT ON THE NUMBER OF SESSIONS (TRAF.LIMIT)
''¢¢ SET AN UPDATE MSG TO PACKET PROCESSING RATIO (U.PAC.RATIO)
''¢¢ SET AN UPDATE MSG TERMINATION TIME (U.XMN.TIME)
''¢¢
LET PKT.MIN = 1.
LET PKT.MAX = 21.
LET TRAF.LIMIT =95000
LET UP.PAC.RATIO = .1
LET U.XMN.TIME = 0.002
''
''¢¢¢¢ READ AND PRINT INPUT DATA
''
READ N.NODE
PRINT 2 LINES AS FOLLOWS
NODE      TRANSMIT   RECEIVE   GROUP      FAMILY
 NO.       FACTOR     FACTOR   (PGM #)    (PGM #)
CREATE EVERY NODE
FOR EVERY NODE
  READ  TRANSMIT.PERCENT(NODE), RECEIVE.PERCENT(NODE), GROUP(NODE),
   FAMILY(NODE)
''
''¢¢ TRNS.PCNT AND RCV.PCNT ARE THE SUM OF TRANSMIT AND RECEIVE FACTORS.
''¢¢ INPUT GROUP NUMBERS ARE ADDED TO N.NODE TO GET PROGRAM GROUP NUMBERS.
''¢¢ INPUT FAMILY NUMBERS ARE ADDED TO N.NODE * THE HIGHEST GROUP NUMBER
''¢¢  TO GET THE PROGRAM FAMILY NUMBER.
''
FOR I = 1 TO N.NODE, DO
  LET TRNS.PCNT = TRNS.PCNT + TRANSMIT.PERCENT(I)
  LET RCV.PCNT = RCV.PCNT + RECEIVE.PERCENT(I)
  IF GRPS < GROUP(I)
    LET GRPS = GROUP(I)
  REGARDLESS
''
''¢¢ SET PROGRAM GRP NUM
''
  LET GROUP(I) = GROUP(I) + N.NODE
LOOP
RESERVE FAM.OF.GRP (*) AS (GRPS + N.NODE + 25)
FOR I = 1 TO N.NODE, DO
  IF FMLYS < FAMILY(I)
    LET FMLYS = FAMILY(I)
  REGARDLESS
''
''¢¢ SET PROGRAM FAM NUM
''
  LET FAMILY(I) = N.NODE + GRPS + FAMILY(I)
  LET FAM.OF.GRP (GROUP(I)) = FAMILY(I)
LOOP
LET NGPS = N.NODE + GRPS + FMLYS
FOR I = 1 TO N.NODE, DO
  PRINT 1 LINE WITH I, TRANSMIT.PERCENT(I), RECEIVE.PERCENT(I),
   (GROUP(I) - N.NODE), GROUP(I), (FAMILY(I) - N.NODE - GRPS), FAMILY(I)
   AS FOLLOWS
 **        **.***    **.***    **(**)    **(**)
LOOP
SKIP 1 OUTPUT LINE
```

```
FILE: DOC      S        A   NAVAL POSTGRADUATE SCHOOL


READ UP.DATE.PERIOD
READ PROCESSING.TIME
READ PKT.XMN.TIME
READ TIME.LIMIT
READ AVE.NEW.TRAFFIC.INTERVAL
READ WINDOW
''
''¢¢ IN.GROUP MEANS THE PERCENTAGE OF GENERATED TRAFFIC THAT WILL NOT
''¢¢  LEAVE ITS BASIC GROUP.  SIMILARLY FOR IN.FAMILY.
''¢¢
''¢¢ PRNT IS AN INTEGER WHICH CONTROLS THE LEVEL OF DIAGNOSTIC PRINTING.
''¢¢   0 -> INPUT DATA + RESULTS
''¢¢   1 -> 0 + TRACES ALL PACKETS + LISTS INITIAL NEIGHBORS + LIST
''¢¢        BEST PATHS W/ CV AT END OF RUN
''¢¢   2 -> 1 + ANNOUNCES CHANGES IN BEST PATHS
''¢¢   3 -> 2 + TRACES ALL UPDATES + ANNOUNCES ALL NEW BEST PATHS
''¢¢
''¢¢ SMP.LINKS IS THE NUMBER OF LINKS TO BE SAMPLED
''¢¢ NO.OF.SAMPLES IS THE EXPONENTIAL MEAN NUMBER OF SAMPLES TO BE TAKEN
''¢¢  AT EACH OF SMP.LINKS LINKS IN THE LAST HALF OF TEST.
''¢¢ LINKS IS THE TOTAL NUMBER OF LINKS IN THE NETWORK.
''
READ IN.GROUP, IN.FAMILY
READ PRNT
READ SMP.LINKS, NO.OF.SAMPLES
READ LINKS
''
PRINT 8 LINES WITH UP.DATE.PERIOD, PROCESSING.TIME, PKT.XMN.TIME,
 TIME.LIMIT, TRAF.LIMIT, AVE.NEW.TRAFFIC.INTERVAL, WINDOW,
 PKT.MIN, PKT.MAX, IN.GROUP, IN.FAMILY AS FOLLOWS
UPDATE PERIOD IS **.****** SEC
PROCESSING TIME IN EACH NODE FOR ANY PACKET IS .****** SEC
PACKET TRANSIT TIME BETWEEN ANY TWO NODES IS .****** SEC
TEST DURATION IS ***.****** SEC. TEST LIMITED TO ***** TRAFFIC SESSIONS.
NEW TRAFFIC SESSIONS ARE STARTED AT AN AVERAGE INTERVAL OF **.****** SEC
CHANNEL VALUE CALCULATION WINDOW IS ***.****** SEC
EACH TRAFFIC SESSION VARIES FROM ** TO ** PACKETS
AT LEAST **. % OF TRAFFIC IS INNER GROUP, ANOTHER **. % IS INNER FAMILY.
SKIP 1 OUTPUT LINE
''
''¢¢ SEE CHAPTER 6 FOR DESCRIPTION OF ARRAYS
''
RESERVE LINK.ABLE(*,*) AS LINKS BY 6
RESERVE LNK.MONITOR(*,*,*) AS N.NODE BY N.NODE BY 3
RESERVE HOP.COUNT(*,*) AS (4*N.NODE) BY 2
RESERVE CLOCK.DATA(*,2) AS (4*N.NODE) BY 2
LET EST.MAX.OF.TRAFFIC = 2 * INT.F(TIME.LIMIT / AVE.NEW.TRAFFIC.INTERVAL)
RESERVE TRACER(*,*) AS EST.MAX.OF.TRAFFIC BY 2
RESERVE SMP.SET(*,*) AS SMP.LINKS BY 2
RESERVE QU.DIST(*) AS 250
''
PRINT 1 LINE AS FOLLOWS
   LINK
''
FOR I = 1 TO LINKS, DO
  FOR J = 1 TO 2, READ LINK.ABLE(I,J)
''
PRINT 1 LINE WITH LINK.ABLE(I,1), LINK.ABLE(I,2) AS FOLLOWS
 **  **
''
LOOP
''
''¢¢¢¢ SCHEDULING INITIAL EVENTS
''¢¢
''¢¢ SCHEDULE THE FIRST UPDATE FOR EACH NODE (NEW.UPDATE.MESSAGE)
''¢¢ FREEZE ALL CV'S FOR THIS UPDATE (CV.LATCH)
''¢¢ SCHEDULE FIRST PACKET SESSION
''¢¢ SCHEDULE TEST FOR MAX QUEUES HALF WAY THRU TEST
''¢¢
FOR EACH NODE
  SCHEDULE A NEW.UPDATE.MESSAGE GIVEN NODE, UPDATE IN (UNIFORM.F
   (0.0,.1,1)) UNITS
```

```
FILE: DOC      S        A   NAVAL POSTGRADUATE SCHOOL

  SCHEDULE A CV.LATCH AT 0.00
  SCHEDULE A STOP.SIMULATION IN TIME.LIMIT/4. UNITS
  SCHEDULE A NEW.PACKET.MESSAGE GIVEN PACKET IN (2 * .1
   + EXPONENTIAL.F(AVE.NEW.TRAFFIC.INTERVAL, 5)) UNITS
  SCHEDULE A QU.SAMPLER IN (TIME.LIMIT/2.) UNITS
RESERVE BEST.PATH(*,*,*) AS N.NODE BY NGPS BY 2
**
**<<<< IDENTIFY NEIGHBORS, SET INITIAL CV'S AND PRINT
**
FOR I = 1 TO N.NODE, DO
  LET BEST.PATH (I,I,1) = I
LOOP
RESERVE NEIGHBOR.LIST(*,*,*) AS N.NODE BY 6 BY 3
  FOR I = 1 TO LINKS, DO
    FOR J = 1 TO 6, DO
      IF NEIGHBOR.LIST(LINK.ABLE(I,1),J,1) = 0
        LET NEIGHBOR.LIST(LINK.ABLE(I,1),J,1) = LINK.ABLE(I,2)
        LET NEIGHBOR.LIST(LINK.ABLE(I,1),J,3) = 1
        LET M = J
        GO NEXT.STEP
      ELSE
    LOOP
'NEXT.STEP'
    FOR J = 1 TO 6, DO
      IF NEIGHBOR.LIST(LINK.ABLE(I,2),J,1) = 0
        LET NEIGHBOR.LIST(LINK.ABLE(I,2),J,1) = LINK.ABLE(I,1)
        LET NEIGHBOR.LIST(LINK.ABLE(I,2),J,3) = 1
        GO LAST.STEP
      ELSE
    LOOP
'LAST.STEP'
    LET NEIGHBOR.LIST(LINK.ABLE(I,1),M,2) = YES
    LET NEIGHBOR.LIST(LINK.ABLE(I,2),J,2) = YES
  LOOP
**
IF PRNT > 0
SKIP 3 OUTPUT LINES
PRINT 1 LINE AS FOLLOWS
NEIGHBORS AND CHANNEL VALUES
FOR I = 1 TO N.NODE, DO
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE WITH I AS FOLLOWS
  NODE ** NEIGHBORS AND CV
  FOR J = 1 TO 6, DO
    PRINT 1 LINE WITH NEIGHBOR.LIST(I,J,1) AND NEIGHBOR.LIST(I,J,3)
     AS FOLLOWS
            **          **
  LOOP
LOOP
REGARDLESS
**
**<<<<
START SIMULATION
**<<<<
**
END **OF MAIN
**
**
**<< THIS ROUTINE HALTS THE PROGRAM AND GIVES SEVERAL STATISTICAL
**<<  REPORTS ON THE STATUS OF THE SIMULATION.  AFTER FOUR REPORTS
**<<  THE ROUTINE STOPS THE SIMULATION.
**
EVENT STOP.SIMULATION
DEFINE TOT.HOPS, TOT.PACKETS AND DELIVERED AS VARIABLES
DEFINE AVE.TIME AND AVE.NODES.HOPPED AS REAL VARIABLES
DEFINE RATIO AND IDEAL.TIME AS REAL VARIABLES
DEFINE Y AS A REAL VARIABLE
DEFINE SD,I3,Z2,Z2SUM AS REAL VARIABLES
**
**<< PEEK COUNT THE FOUR REPORTS
**<< TOT.PACKETS SUMS THE TOTAL PKTS GENERATED UP TO THIS POINT
**<< DELIVERED SUMS THE TOTAL PKTS REACHING THEIR DESTINATION
```

```
FILE: DOC       $         A    NAVAL POSTGRADUATE SCHOOL

!!<< TOT.HOPS SUMS ALL HOPS MADE BY ALL PKTS
!!<< SUM SUMS SAMPLED Q SIZES
!!<< X IS THE TOTAL NUMBER OF SAMPLES
!!<< Z2SUM IS THE SUM OF SAMPLE SIZES SQUARED
!!
LET PEER=PEER+1
LET TOT.PACKETS = 0
LET DELIVERED = 0
LET TOT.HOPS = 0
LET SUM = 0
LET X = 0
LET Z2SUM = 0.0
!!
!!<< PRINT BP NEIGHBORS AND CV
!!
IF PRNT > 0
SKIP 3 OUTPUT LINES
FOR I = 1 TO N.NODE, DO
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE WITH I AS FOLLOWS
BEST PATHS  FROM NODE **  TO -- DESTINATION - BP.NEIGHBOR - CV FRM BP.NODE
  FOR J = 1 TO N.NODE, DO
    IF (GROUP(I) = GROUP(J) AND I NE J)
    PRINT 1 LINE WITH J,BEST.PATH(I,J,1), BEST.PATH(I,J,2) AS FOLLOWS
                                  **                **              ***
    REGARDLESS
  LOOP
  FOR J = (N.NODE+1) TO NGPS, DO
    IF (GROUP(I) NE J AND FAMILY(I) NE J AND J > N.NODE)
     IF BEST.PATH (I,J,1) NE 0
     IF (FAM.OF.GRP (J) NE FAMILY(I) AND J <= (N.NODE + GRPS))
      GO OMIT.PRINT
     ELSE
      PRINT 1 LINE WITH J, BEST.PATH(I,J,1), BEST.PATH(I,J,2) AS FOLLOWS
                                  **                **              ***
'OMIT.PRINT'
    REGARDLESS
    REGARDLESS
  LOOP
LOOP
REGARDLESS
!!
!!<< COUNT PKTS CREATED AND DELIVERED
!!
FOR I = 1 TO TOT.NEW.TRAFFIC, DO
  LET TOT.PACKETS = TOT.PACKETS + TRACER (I,1)
  LET DELIVERED = DELIVERED + TRACER (I,2)
LOOP
!!
!!<< PRINT SELECTED INPUT DATA
!!
BEGIN REPORT ON A NEW PAGE
PRINT 8 LINES WITH UP.DATE.PERIOD, PROCESSING.TIME, PKT.XMT.TIME,
 TIME.LIMIT, TRAF.LIMIT, AVE.NEW.TRAFFIC.INTERVAL, WINDOW,
 PKT.MIN, PKT.MAX, IN.GROUP, IN.FAMILY AS FOLLOWS
UPDATE PERIOD IS **.******* SEC
PROCESSING TIME IN EACH NODE FOR ANY PACKET IS .****** SEC
PACKET TRANSIT TIME BETWEEN ANY TWO NODES IS .****** SEC
TEST DURATION IS ***.******* SEC. TEST LIMITED TO ***** TRAFFIC SESSIONS.
NEW TRAFFIC SESSIONS ARE STARTED AT AN AVERAGE INTERVAL OF **.****** SEC
CHANGE VALUE CALCULATION WINDOW IS ***.****** SEC
EACH TRAFFIC SESSION VARIES FROM ** TO ** PACKETS
AT LEAST **. % OF TRAFFIC IS INNER GROUP, ANOTHER **. % IS INNER FAMILY.
!!
!!<< PRINT PKT STATISTICS
!!
SKIP 1 OUTPUT LINE
PRINT 2 LINES AS FOLLOWS
NODES        NO.              MEAN TIME          PEAK TIME         IDEAL
HOPPED      PKTS              PER PKT              TIME             TIME
FOR I = 1 TO (2**N.NODE), DO
  IF HOP.COUNT (I,PACKET) NE 0
```

```
FILE: DOC      S        A   NAVAL POSTGRADUATE SCHOOL

    LET TOT.HOPS = TOT.HOPS + (I * HOP.COUNT(I,PACKET))
    LET AVE.TIME = CLOCK.DATA(I,1) / REAL.F(HOP.COUNT(I,PACKET))
    LET IDEAL.TIME = I*PKT.XMN.TIME + (I-1)*PROCESSING.TIME
    PRINT 1 LINE WITH I, HOP.COUNT(I,PACKET), AVE.TIME, CLOCK.DATA(I,2),
     IDEAL.TIME AS FOLLOWS
 **        ***          *.******          *.******          *.******
  REGARDLESS
LOOP
''
''** PRINT ALERT MSG IF A PKT HOPPED MORE THAN TOTAL NUMBER OF NODES
''
IF MSG.HLT NE 0
  SKIP 2 OUTPUT LINES
  PRINT 1 LINE AS FOLLOWS
***** NOTE AT LEAST 1 PACKET HOPPED MORE THAN THE TOTAL NUMBER OF NODES *
  SKIP 2 OUTPUT LINES
REGARDLESS
''
LET AVE.NODES.HOPPED = REAL.F(TOT.HOPS) / REAL.F(DELIVERED)
''
''** PRINT SELECTED STATISTICAL DATA
''
SKIP 3 OUTPUT LINES
PRINT 1 LINE WITH AVE.NODES.HOPPED AS FOLLOWS
MEAN NUMBER OF NODES HOPPED PER PACKET IS  **.*
SKIP 1 OUTPUT LINE
PRINT 1 LINE WITH TOT.NEW.TRAFFIC AND TOT.PACKETS AS FOLLOWS
A TOTAL OF  *** NEW XMNS WERE STARTED (TOTALING  ***** PACKETS ).
PRINT 1 LINE WITH (TOT.PACKETS - DELIVERED) AS FOLLOWS
  OF THESE,  *** PACKETS WERE UNDELIVERED WHEN THE TEST WAS ENDED.
SKIP 1 OUTPUT LINE
LET RATIO = REAL.F(NET.U.LINKS)/REAL.F(UP.STARTS)
PRINT 1 LINE WITH RATIO AS FOLLOWS
FOR EACH NEW UPDATE, AN AVERAGE OF ***.* LINKS WERE USED.
SKIP 1 OUTPUT LINE
PRINT 1 LINE WITH MAX.U.HOPS AS FOLLOWS
LONGEST BEST PATH AT ANY TIME WAS *** LINKS.
''
''** SKIP TO END OF ROUTINE IF STILL IN FIRST HALF OF TEST
''
IF PEEK < 3
  GO COM.TIDUE
ELSE
''
''** PRINT THE NUMBER OF LINKS SAMPLED AND TOTAL SAMPLES PER LINK
''
SKIP 1 OUTPUT LINE
PRINT 3 LINES WITH SMP.LINKS, SMP.CNTR AS FOLLOWS
QUEUE LENGTH DISTRIBUTION
  ** LINKS WERE SAMPLED
  *** SAMPLES / LINK WERE TAKEN
END **OF BACKGROUND DATA
''
''** PRINT MAX Q LENGTH
''
IF PRNT >= 0
  SKIP 2 OUTPUT LINES
  PRINT 2 LINES AS FOLLOWS
MAXIMUM QUEUE LENGTH:
    FROM   TO   MAX
  FOR I = 1 TO LINKS, DO
    LET A = LINK.ADDR(I,1)
    LET B = LINK.ADDR(I,2)
    PRINT 1 LINE WITH A,B,LNK.MONITOR(A,B,3) AS FOLLOWS
     **   **   ***
    PRINT 1 LINE WITH B,A,LNK.MONITOR(B,A,3) AS FOLLOWS
     **   **   ***
  LOOP
REGARDLESS
''
''** PRINT THE SAMPLING COUNT OF Q SIZES FROM 0 THRU 250, AND COMPUTE
''**  SUMS TO CALCULATE AVERAGE AND STANDARD DEVIATION
```

```
FILE: DOC      S       A   NAVAL POSTGRADUATE SCHOOL

''
BEGIN REPORT ON A NEW PAGE
PRINT 1 LINE AS FOLLOWS
  Q-SIZE  - SAMPLE DENSITY
FOR I = 1 TO 250, DO
  LET SUM = (I-1)*QU.DISTR(I) + SUM
  LET X=X+QU.DISTR(I)
  LET Z2= QU.DISTR(I)*((I-1)**2)
  LET Z2SUM= Z2SUM+ Z2
  IF QU.DISTR(I) NE 0
    PRINT 1 LINE WITH (I-1) AND QU.DISTR(I) AS FOLLOWS
    ***       ***
  REGARDLESS
LOOP
LET Y = SUM/X
PRINT 1 LINE WITH Y AS FOLLOWS
AVERAGE Q LENGTH = ***.***
LET XR = X
LET SD= SQRT.F((Z2SUM- XR*(Y**2))/(XR-1.))
SKIP 1 OUTPUT LINE
PRINT 1 LINE WITH SD AS FOLLOWS
STANDARD DEVIATION = **.****
''
''** CHECK ALL UNDELIVERED PKTS.  REPORT ANY WITH A HOP COUNT > N.NODE
''
PRINT 1 LINE AS FOLLOWS
UNUSUAL DELAYS FOR PACKETS NOT DELIVERED DESCRIBED BELOW
FOR NO.DE = 1 TO N.NODS, DO
FOR EACH MESSAGE IN QUEUE(NO.DE) WITH TYPE(MESSAGE)= PACKET, DO
  IF NODES.HOPPED(MESSAGE) >= N.NODE
    PRINT 1 LINE WITH RELEASE.TIME(MESSAGE), NODES.HOPPED(MESSAGE)
      AS FOLLOWS
PACKET RELEASED AT ***.****** AND HAS *** HOPS
  REGARDLESS
LOOP
LOOP
''
IF PEEK NE 0
  GO CON.TINUE
ELSE
  STOP
'CON.TINUE'
''
''** RESCHEDULE THE NEXT STOP.SIMULATION
''
SCHEDULE A STOP.SIMULATION IN TIME.LIMIT/4. UNITS
END ''OF STOP.SIMULATION
RETURN
END
''
''
''** THIS ROUTINE IS CALLED WHEN A NODE ORIGINATES AN UPDATE MESSAGE.
''**  THE INITIAL U-MSG IS SENT TO ALL OF THE INITIATING NODE'S NEIGHBORS
''
EVENT NEW.UPDATE.MESSAGE GIVEN SENDING.NODE AND TYPE.MESSAGE
LET UP.STARTS = UP.STARTS + 1
  FOR I = 1 TO LINES, DO
''
''** CHECK EACH LINK TO SEE IF SENDING.NODE IS ON ONE END
''
    IF (LINK.ABLE(I,1) = SENDING.NODE OR LINK.ABLE (I,2) = SENDING.NODE)
      CREATE A MESSAGE
      LET TYPE(MESSAGE) = UPDATE
      LET RELAYER(MESSAGE) = SENDING.NODE
        IF LINK.ABLE(I,1) = SENDING.NODE
''
''** IF OPPOSITE NODE IS IN ANOTHER FAMILY:
''
          IF FAMILY (LINK.ABLE(I,2)) NE FAMILY (SENDING.NODE)
            LET FAM.LY (MESSAGE) = FAMILY (SENDING.NODE)
            LET DESTINATION(MESSAGE) = FAMILY (SENDING.NODE)
            GO LIST.NEXT.STOP
```

```
          ELSE
;;
;;** IF OPPOSITE NODE IS IN ANOTHER GROUP (SAME FAMILY):
;;
          IF GROUP (LINK.ABLE(I,2)) NE GROUP (SENDING.NODE)
            LET GRP (MESSAGE) = GROUP (SENDING.NODE)
            LET DESTINATION (MESSAGE) = GROUP (SENDING.NODE)
            GO LIST.NEXT.STOP
          ELSE
;;
;;** IF OPPOSITE NODE IS IN SAME BASIC GROUP
;;
          LET DESTINATION(MESSAGE) = LINK.ABLE(I,1)
'LIST.NEXT.STOP'
          LET NEXT.STOP(MESSAGE) = LINK.ABLE(I,2)
        ELSE
        IF FAMILY (LINK.ABLE(I,1)) NE FAMILY (SENDING.NODE)
          LET FAMILY (MESSAGE) = FAMILY (SENDING.NODE)
          LET DESTINATION (MESSAGE) = FAMILY (SENDING.NODE)
          GO INCL.NEXT.STOP
        ELSE
        IF GROUP (LINK.ABLE(I,1)) NE GROUP (SENDING.NODE)
          LET GRP (MESSAGE) = GROUP (SENDING.NODE)
          LET DESTINATION (MESSAGE) = GROUP (SENDING.NODE)
          GO INCL.NEXT.STOP
        ELSE
          LET DESTINATION(MESSAGE) = LINK.ABLE(I,2)
'INCL.NEXT.STOP'
          LET NEXT.STOP (MESSAGE) = LINK.ABLE(I,1)
        REGARDLESS
;;
;;** SCHEDULE ARRIVAL OF U-MSG AT OPPOSITE NODE
;;
          SCHEDULE AN ARRIVAL.MESSAGE GIVEN MESSAGE IN
            U.XMN.TIME UNITS
    REGARDLESS
  LOOP
;;
;;** SCHEDULE THE NEXT ORIGINATION OF A U-MSG FOR THIS NODE
;;
  SCHEDULE A NEW.UPDATE.MESSAGE GIVEN SENDING.NODE AND UPDATE AT
    (TIME.FROM.NOW.P(EARLIEST.UPDATE, LATEST.UPDATE, 3))
RETURN
END **OF NEW.UPDATE
;;
;;
;;** THIS ROUTINE CREATES CONTINUED U-MSGS REPRESENTING THE RELAYING
;;**   OF A U-MSG FROM A NODE AFTER AN UPDATE
;;
EVENT CONT.UPDATE.MESSAGE GIVEN LAST.NODE, NEXT.NODE, NET.CV,
  SOURCE, FAMILY, HOP.CNT AND GR.OUP
  CREATE A MESSAGE
    TYPE(MESSAGE) = UPDATE
    RELAYER(MESSAGE) = LAST.NODE
    NEXT.STOP(MESSAGE) = NEXT.NODE
    DESTINATION(MESSAGE) = SOURCE
    CHANNEL.VALUE(MESSAGE) = NET.CV
    GRP(MESSAGE) = GR.OUP
    FAMILY(MESSAGE) = FAMILY
  LET NODES.HOPPED(MESSAGE) = HOP.CNT
;;
;;** THIS RELAYED U-MSG IS SCHEDULED TO ARRIVE AT THE NEXT DESTINATION
;;**   AFTER A SELECTED TRANSMISSION TIME
;;
  SCHEDULE AN ARRIVAL.MESSAGE GIVEN MESSAGE IN
    U.XMN.TIME UNITS
RETURN
END **OF CONT.UPDATE
;;
;;
;;** THIS ROUTINE PROCESSES AN UPDATE MESSAGE AS IT ARRIVES IN A NODE,
;;**   RELAYING IT TO NEIGHBORS IF APPROPRIATE
```

```
FILE: DOC      S        A   NAVAL POSTGRADUATE SCHOOL

''
EVENT ARRIVAL.MESSAGE GIVEN ID.MESSAGE.NUMBER
LET MESSAGE = ID.MESSAGE.NUMBER
LET THIS.NODE = NEXT.STOP(MESSAGE)
LET NET.U.LINKS = NET.U.LINKS + 1
LET NODES.HOPPED(MESSAGE) = NODES.HOPPED(MESSAGE) + 1
''
''** ID CV OF LAST RELAYING NODE
''
FOR I = 1 TO 6, DO
   IF NEIGHBOR.LIST(THIS.NODE, I, 1) = RELAYER(MESSAGE)
      LET CV.OF.LINK = NEIGHBOR.LIST (THIS.NODE, I, 3)
      GO SELECT.BEST.PATH
   ELSE
LOOP
'SELECT.BEST.PATH'
LET TOTAL.CV.OF.PATH = CV.OF.LINK + CHANNEL.VALUE(MESSAGE)
''
''** ID PREVIOUSLY SELECTED BP NEIGHBOR
''
LET BP.NEIGHBOR = BEST.PATH(THIS.NODE,DESTINATION(MESSAGE),1)
''
''** IF RELAYER = CURRENT BP NEIGHBOR, UPDATE ITS CV TO THE DESTINATION
''**  AND RELAY UPDATE
''
IF RELAYER(MESSAGE) = BP.NEIGHBOR
   LET BEST.PATH(THIS.NODE,DESTINATION(MESSAGE),2) = CHANNEL.VALUE(MESSAGE)
''
   IF PRNT >= 3
   SKIP 1 OUTPUT LINE
   PRINT 1 LINE WITH THIS.NODE, DESTINATION(MESSAGE), BP.NEIGHBOR,
    CHANNEL.VALUE(MESSAGE), C.V.OF.LINK, TOTAL.CV.OF.PATH, TIME.V
    AS FOLLOWS
NODE ** UPDATES CV THRU SAME BP TO ** (THRU **) AS ***.**** *** AT **.****** SEC
   SKIP 1 OUTPUT LINE
   REGARDLESS
''
   GO RELAY.UPDATE.TO.NEIGHBORS
ELSE
''
''** IF THERE WAS NO BP NEIGHBOR, ADOPT RELAYER AS BP NEIGHBOR AND
''**  RELAY UPDATE
''
IF BP.NEIGHBOR = NONE
   LET BEST.PATH(THIS.NODE,DESTINATION(MESSAGE),1) = RELAYER(MESSAGE)
   LET BEST.PATH(THIS.NODE,DESTINATION(MESSAGE),2) = CHANNEL.VALUE(MESSAGE)
   LET BP.NEIGHBOR = RELAYER(MESSAGE)
''
IF PRNT >= 3
   SKIP 1 OUTPUT LINE
   PRINT 1 LINE WITH THIS.NODE, DESTINATION(MESSAGE), BP.NEIGHBOR, TIME.V,
    TOTAL.CV.OF.PATH AS FOLLOWS
    NEW BEST PATH FROM ** TO ** NOW THRU ** AT **.****** SEC. BEST NET CV= ***
   SKIP 1 OUTPUT LINE
REGARDLESS
''
   GO RELAY.UPDATE.TO.NEIGHBORS
ELSE
''
''** IF THE RELAYER IS NOT THE BP NEIGHBOR, AND IF THE NEW PATH IS
''**  SHORTER THAN THE OLD BEST PATH, MAKE RELAYER THE NEW BP NEIGHBOR
''**  AND RELAY THE UPDATE
''
FOR I = 1 TO 6, DO
   IF NEIGHBOR.LIST(THIS.NODE,I,1) = BP.NEIGHBOR
      LET CV.TO.BP.NEIGHBOR = NEIGHBOR.LIST(THIS.NODE,I,3)
      GO COMPARE.CVS
   ELSE
LOOP
'COMPARE.CVS'
IF (BEST.PATH(THIS.NODE,DESTINATION(MESSAGE),2) + CV.TO.BP.NEIGHBOR) >
 TOTAL.CV.OF.PATH
```

```
FILE: DOC      S        A   NAVAL POSTGRADUATE SCHOOL

  LET OLD.BP = BP.NEIGHBOR
  LET OLD.CV = BEST.PATH (THIS.NODE, DESTINATION(MESSAGE), 2)
  LET LNK.CV = CV.TO.BP.NEIGHBOR
  LET BEST.PATH(THIS.NODE, DESTINATION(MESSAGE), 1) = RELAYER(MESSAGE)
  LET BEST.PATH(THIS.NODE, DESTINATION(MESSAGE), 2) = CHANNEL.VALUE
   (MESSAGE)
  LET BP.NEIGHBOR = RELAYER(MESSAGE)
''
IF PRNT >= 2
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE WITH THIS.NODE, DESTINATION(MESSAGE), BP.NEIGHBOR, TIME.V,
   CHANNEL.VALUE(MESSAGE), CV.OF.LINK, TOTAL.CV.OF.PATH AS FOLLOWS
   NEW BEST PATH FROM ** TO ** NOW THRU ** AT **.****** SEC. CV= ***.***= ***
  PRINT 1 LINE WITH OLD.BP, OLD.CV, LNK.CV, (OLD.CV + LNK.CV) AS FOLLOWS
OLD BP THRU ** HAD CV OF *** + *** = ***
  SKIP 1 OUTPUT LINE
REGARDLESS
''
  GO RELAY.UPDATE.TO.NEIGHBORS
ELSE
'' IF NEW PATH IS NOT BETTER THAN THE OLD BEST PATH, DISCONTINUE U-MSG
''
  GO DISCONTINUE.ORIGINAL.MESSAGE
''
''** IF A NEW BP IS SELECTED OR AN OLD BP IS UPDATED, PREPARE INFORMATION
''**  FOR THE NEXT U-MSG TO ALL NEIGHBORS
''
'RELAY.UPDATE.TO.NEIGHBORS'
FOR I = 1 TO 6, DO
  IF NEIGHBOR.LIST (THIS.NODE, I, 1) = BP.NEIGHBOR
    LET CV.TO.BP.NEIGHBOR = NEIGHBOR.LIST (THIS.NODE, I, 3)
    GO COMPUTE.NET.CV
  ELSE
LOOP
'COMPUTE.NET.CV'
LET NET.CV.FROM.THIS.NODE = BEST.PATH (THIS.NODE, DESTINATION(MESSAGE),2)
 + CV.TO.BP.NEIGHBOR
FOR I = 1 TO 6, DO
  IF NEIGHBOR.LIST (THIS.NODE, I, 2) = YES
   IF NEIGHBOR.LIST (THIS.NODE, I, 1) NE RELAYER (MESSAGE)
    LET UPSTREAM.NODE = NEIGHBOR.LIST (THIS.NODE, I, 1)
''
''** IF UPSTREAM NODE IS IN ANOTHER FAMILY AND THIS IS A INTRA-FAMILY
''**  U-MSG, RELAY IT
''
    IF (FAMILY(MESSAGE) NE 0 AND FAMILY(UPSTREAM.NODE) NE FAMILY(MESSAGE))
      GO RETRANS.UPSTREAM
    ELSE
''
''** IF UPSTREAM NODE IS IN ANOTHER GROUP AND THIS IS A INTRA-GROUP (SAME
''**  FAMILY) U-MSG, RELAY IT
''
    IF (GRP(MESSAGE) NE 0 AND GROUP(UPSTREAM.NODE) NE GRP(MESSAGE))
      GO RETRANS.UPSTREAM
    ELSE
''
''** IF UPSTREAM NODE IS IN THE SAME BASIC GROUP AND THIS IS A BASIC
''**  GROUP ORIGINATED U-MSG, RELAY IT
''
    IF (GROUP(UPSTREAM.NODE) = GROUP(THIS.NODE)
     AND FAMILY(MESSAGE) = GRP(MESSAGE))
      GO RETRANS.UPSTREAM
    ELSE
''
''** IF NONE OF THE ABOVE, UPSTREAM NODE DOES NOT GET A U-MSG
''
      GO SKIP.PRINT
'RETRANS.UPSTREAM'
    LET HOPS = NODES.HOPPED(MESSAGE)
    SCHEDULE A CONT.UPDATE.MESSAGE GIVEN THIS.NODE, UPSTREAM.NODE,
     NET.CV.FROM.THIS.NODE, DESTINATION(MESSAGE), FAMILY(MESSAGE), HOPS,
     AND GRP(MESSAGE) IN (PROCESSING.TIME * UP.FAC.RATIO) UNITS
```

```
'SKIP.PRINT'
  REGARDLESS
  REGARDLESS
LOOP
''
''** IN THE SIMULATION, ALL U-MSGS ARE DESTROYED AFTER TRAVELLING ONE
''**  LINK.  HOWEVER THE UPDATE CYCLE PROCEEDS ACCORDING TO THE BASIC
''**  CONCEPT BECAUSE THE ARRIVING U-MSG CAUSES NEW U-MSGS TO BE
''**  INITIATED IF A NEW BP WAS SELECTED OR AN OLD BP WAS UPDATED.
''**  IF A NODE COULD NOT USE AN INCOMING U-MSG, IT IS DESTROYED
''**  WITHOUT GENERATING ANY NEW U-MSGS.
''
'DISCONTINUE.ORIGINAL.MESSAGE'
IF NODES.HOPPED(MESSAGE) > MAX.U.HOPS
  LET MAX.U.HOPS = NODES.HOPPED(MESSAGE)
REGARDLESS
DESTROY MESSAGE CALLED ID.MESSAGE.NUMBER
RETURN
END ''OF ARRIVAL.UPDATE
''
''
''** THIS ROUTINE CALCULATES CHANNEL VALUES BASED ON A TIME-WEIGHTED
''**  AVERAGE OF QUEUE SIZES OVER A SPECIFIED TIME CALLED THE WINDOW.
''**  QUEUE SIZE INFORMATION OLDER THAN THE WINDOW TIME IS DISCARDED.
''
EVENT CV.LATCH
DEFINE EDGE, LAST, SUM, LAS.QU, SPAN, MID, AREA, WEIGHT, BLOCK
  AND REMAINDER AS REAL VARIABLES
DEFINE NONE TO MEAN 0
FOR THIS.NODE = 1 TO N.NODE, DO
''
''** DESTROY QUEUE INFORMATION BEYOND WINDOW SIZE.  "PACK" IS A PACKAGE
''**  OF INFORMATION DESCRIBED LATER.
''
FOR EACH PACK IN TIME.QUEUE (THIS.NODE) WITH ENTRY.TIME(PACK) <
  (TIME.V - WINDOW), DO
  REMOVE PACK FROM TIME.QUEUE (THIS.NODE)
  DESTROY PACK
LOOP
''
''** CALCULATE THE CV TO EACH NEIGHBOR
''
  FOR J = 1 TO 6, DO
    IF NEIGHBOR.LIST (THIS.NODE, J, 2) = YES
      LET NEIB = NEIGHBOR.LIST (THIS.NODE, J, 1)
''
LET EDGE = 0.0
LET LAST = TIME.V
LET SUM = 0.0
LET LAS.QU = 0.0
LET ANY.PACKS = NONE
FOR EACH PACK IN TIME.QUEUE (THIS.NODE) WITH PAC.NEIGHBOR (PACK) =
  NEIB, DO
  LET ANY.PACKS = YES
  LET SPAN = LAST - ENTRY.TIME (PACK)
  LET MID = SPAN/2. + EDGE
  LET AREA = REAL.F(NUMBER(PACK)) * SPAN
  LET BLOCK = AREA
  LET SUM = SUM + BLOCK
  LET EDGE = EDGE + SPAN
  LET LAST = ENTRY.TIME(PACK)
  LET LAS.QU = REAL.F(NUMBER(PACK))
LOOP
IF ANY.PACKS = NONE
  LET LAS.QU = LNK.MONITOR (THIS.NODE, NEIB, 2)
REGARDLESS
LET REMAINDER = WINDOW - EDGE
LET MID = (REMAINDER/2.) + EDGE
LET AREA = LAS.QU * REMAINDER
LET BLOCK = AREA
LET SUM = SUM + BLOCK
LET CV.OF.LINK = INT.F(SUM/WINDOW + 1.)
```

```
FILE: DOC      S        A   NAVAL POSTGRADUATE SCHOOL

LET NEIGHBOR.LIST (THIS.NODE, J, 3) = CV.CE.LINK
''
   REGARDLESS
  LOOP
LOOP
''
''ee SCHEDULE THE NEXT CV CALCULATION FOR ALL NEIGHBORS.  IN THE
''ee  SIMULATION, THIS PROCESS IS SYNCHRONIZED FOR EVERY NODE IN
''ee  THE NETWORK (SEE CHAP VI).
''
SCHEDULE A CV.LATCH IN (2*UP.DATE.PERIOD) UNITS
''
''ee EARLIEST.UPDATE AND LATEST.UPDATE SAT THE NEXT INTERVAL DURING
''ee  WHICH ALL NODES WILL RANDOMLY INITIATE AN UPDATE CYCLE.  THE
''ee  NEXT CV.LATCH FOR ALL NODES IN THE NETWORK OCCURS AT THE VERY
''ee  BEGINNING OF THIS INTERVAL.  AFTER THIS PERIOD, THERE IS ANOTHER
''ee  EQUAL SIZED PERIOD DURING WHICH NO UPDATE CYCLES ARE INITIATED.
''ee  BUT THIS PERIOD INSURES THAT ALL CYCLES STARTED DURING THE
''ee  EARLIEST.UPDATE TO LATEST.UPDATE PERIOD WILL BE COMPLETED.
''ee  DURING THESE TWO PERIODS, THE CV FOR ALL LINKS ARE FROZEN.
''
LET EARLIEST.UPDATE = TIME.V + (2* UP.DATE.PERIOD)
LET LATEST.UPDATE = TIME.V + (3* UP.DATE.PERIOD)
RETURN
END ''OF CV.LATCH
''
''
''ee THIS ROUTINE GENERATES A TRAFFIC SESSION MADE UP OF A RANDOM
''ee  NUMBER OF PKTS (BETWEEN PRESCRIBED LIMITS).  PKTS ARE SENT OUT
''ee  ON IDLE LINKS IF AVAILABLE, OR STORED IN QUEUES IF LINKS ARE BUSY.
''
EVENT NEW.PACKET.MESSAGE GIVEN T.MESSAGE
DEFINE CK.XMTR, CK.RCVR, X.TOT.PERCENT AND R.TOT.PERCENT AS REAL
  VARIABLES
LET X.TOT.PERCENT = 0
LET R.TOT.PERCENT = 0
LET CK.XMTR = UNIFORM.F(0.0, TRNS.PCNT, 2)
''
''ee SELECTOR IS USED IF A PERCENTAGE OF THE TRAFFIC IS REQUIRED
''ee  TO BE INNER-GROUP/FAMILY
''
LET SELECTOR = UNIFORM.F(0.0, 100., 9)
''
''ee SELECT THE TRANSMITTING NODE
''
FOR I = 1 TO N.NODE, DO
  LET X.TOT.PERCENT = X.TOT.PERCENT + TRANSMIT.PERCENT(I)
  IF CK.XMTR <= X.TOT.PERCENT
    LET XMTR = I
    GO FIND.RECEIVER
  ELSE
LOOP
''
''ee SELECT THE RECEIVER
''
'FIND.RECEIVER'
LET CK.RCVR = UNIFORM.F(0.0, RCV.PCNT, 3)
FOR J = 1 TO N.NODE, DO
  LET R.TOT.PERCENT = R.TOT.PERCENT + RECEIVE.PERCENT(J)
  IF CK.RCVR <= R.TOT.PERCENT
    LET RCVR = J
    GO CK.GROUPS.AND.FAMILIES
  ELSE
LOOP
''
''ee IF THE RECEIVER MUST BE INNER-GROUP OR FAMILY, KEEP LOOKING UNTIL
''ee  AN ADEQUATE RECEIVER IS FOUND
''
'CK.GROUPS.AND.FAMILIES'
IF SELECTOR < IP.GROUP
  IF GROUP(XMTR) = GROUP(RCVR)
    GO SEE.IF.XMTR.EQ.RCVR
```

```
   ELSE
     LET R.TOT.PERCENT = 0.0
     GO FIND.RECEIVER
ELSE
IF SELECTOR < (IN.GROUP + IN.FAMILY)
   IF FAMILY(XMTR) = FAMILY(RCVR)
     GO SEE.IF.XMTR.EQ.RCVR
   ELSE
     LET R.TOT.PERCENT = 0.0
     GO FIND.RECEIVER
ELSE
'SEE.IF.XMTR.EQ.RCVR'
IF RCVR = XMTR
   LET R.TOT.PERCENT = 0.0
   GO FIND.RECEIVER
ELSE
'DERIVE.NUMBER.OF.PACKETS'
LET PKT.COUNT = INT.F(UNIFORM.F(PKT.MIN, PKT.MAX, 4))
*'
*'** COUNT TOTAL TRAFFIC SESSIONS.  THE TRACER ARRAY KEEPS TRACK OF
*'**  SESSION INFORMATION
*'
LET TOT.NEW.TRAFFIC = TOT.NEW.TRAFFIC + 1
LET TRACER (TOT.NEW.TRAFFIC, 1) = PKT.COUNT
*'
*'** CREATE A MESSAGE FOR EACH PACKET
*'
FOR I = 1 TO PKT.COUNT, DO
   CREATE A MESSAGE
     LET TYPE(MESSAGE) = PACKET
     LET RELAYER(MESSAGE) = XMTR
     LET FG = 0
*'
*'** ADDRESS PKT TO NODE, GROUP OR FAMILY OF DESTINATION AS APPROPRIATE
*'
     IF FAMILY(XMTR) NE FAMILY(RCVR)
       LET BP.NODE = BEST.PATH (XMTR, FAMILY(RCVR), 1)
       LET FM.GP(MESSAGE) = FAMILY(RCVR)
       LET FG = FAMILY(RCVR)
       GO ADD.DESTINATION
     ELSE
     IF GROUP(XMTR) NE GROUP(RCVR)
       LET BP.NODE = BEST.PATH (XMTR, GROUP(RCVR), 1)
       LET FM.GP(MESSAGE) = GROUP (RCVR)
       LET FG = GROUP (RCVR)
       GO ADD.DESTINATION
     ELSE
       LET BP.NODE = BEST.PATH (XMTR, RCVR, 1)
'ADD.DESTINATION'
     LET DESTINATION(MESSAGE) = RCVR
     LET TRANS.NUMBER(MESSAGE) = TOT.NEW.TRAFFIC
     LET PACK.NUMBER(MESSAGE) = I
     LET NEXT.STOP(MESSAGE) = BP.NODE
*'
*'** IF LINK TO BP NEIGHBOR IS IDLE, SEND OUT PACKET
*'
   IF LNK.MONITOR (XMTR, BP.NODE, 1) = IDLE
     SCHEDULE AN ARIVE.PACKET GIVEN MESSAGE IN PKT.XMN.TIME UNITS
     LET LNK.MONITOR(XMTR, BP.NODE, 1) = BUSY
     LET RELEASE.TIME (MESSAGE) = TIME.V
   ELSE
*'
*'** IF LINK IS BUSY, STORE PKT AT END OF QUEUE FOR THAT LINK
*'
     FILE MESSAGE IN QUEUE (XMTR)
     LET LNK.MONITOR(XMTR, BP.NODE, 2) = LNK.MONITOR(XMTR, BP.NODE, 2) + 1
     IF LNK.MONITOR(XMTR,BP.NODE,2)>LNK.MONITOR(XMTR,BP.NODE,3)
       LET LNK.MONITOR(XMTR,BP.NODE,3) = LNK.MONITOR(XMTR,BP.NODE,2)
     REGARDLESS
*'
*'** IF LINK QUEUE CHANGES, CREATE A "PACK" (A PACKAGE OF INFORMATION)
*'**  WHICH CONTAINS THE NEW QUEUE SIZE AND THE TIME IT WAS CHANGED
```

```
FILE: DOC      S       A   NAVAL POSTGRADUATE SCHOOL

''<< FOR THE LATER CALCULATION OF CE
''
    CREATE A PACK
      LET NUMBER(PACK) = LNK.MONITOR (XMTR,BP.NODE,2)
      LET ENTRY.TIME(PACK) = TIME.V
      LET PAC.NEIGHBOR(PACK) = BP.NODE
    FILE PACK IN TIME.QUEUE(XMTR)
  REGARDLESS
''
IF PRNT >= 1
  PRINT 1 LINE WITH TOT.NEW.TRAFFIC, I, XMTR, BP.NODE,FG, RCVR, TIME.V,
    LNK.MONITOR(XMTR, BP.NODE, 2) AS FOLLOWS
*****/** INITIATED FROM ** THRU **(**) TO ** AT **.****** SEC. QU= ***
SKIP 1 OUTPUT LINE
REGARDLESS
''
LOOP
''
''<< RESCHEDULE NEXT TRAFFIC SESSION UP TO MAX SET BY TRAFFIC.LIMIT
''
IF TOT.NEW.TRAFFIC < TRAF.LIMIT
  SCHEDULE A NEW.PACKET.MESSAGE GIVEN PACKET IN
    EXPONENTIAL.F(AVE.NEW.TRAFFIC.INTERVAL, 1) UNITS
REGARDLESS
RETURN
END ''OF NEW.PACKET
''
''
''<< THIS ROUTINE PROCESSES PKTS AS THEY ARRIVE INA NODE
''
EVENT ARRIVE.PACKET GIVEN ID.NUMBER
LET MESSAGE = ID.NUMBER
LET THIS.NODE = NEXT.STOP(MESSAGE)
LET PAST.NODE = RELAYER(MESSAGE)
''
IF PRNT >= 1
PRINT 1 LINE WITH TRANS.NUMBER(MESSAGE), PACK.NUMBER(MESSAGE),
  RELAYER(MESSAGE), NEXT.STOP(MESSAGE), TIME.V AS FOLLOWS
*****/** ARRIVES FROM ** INTO ** AT **.****** SEC
REGARDLESS
''
''<< IF THE PKT HAS REACHED ITS DESTINATION, GO TO A PROCESSING ROUTINE
''
IF NEXT.STOP(MESSAGE) = DESTINATION(MESSAGE)
  SCHEDULE A COMPLETED.TRIP GIVEN MESSAGE NEXT
''
IF PRNT >= 1
  PRINT 1 LINE AS FOLLOWS
    AND STOPS
REGARDLESS
''
''<< IF PKT IS TO CONTINUE, ADDRESS IT TO THE NEXT BP NEIGHBOR BASED
''<<  ON THE NODE, GROUP OR FAMILY ID OF THE DESTINATION AS APPROPRIATE
''
ELSE
  LET RELAYER(MESSAGE) = THIS.NODE
  LET FM.GP(MESSAGE) = 0
  LET FG = 0
  IF FAMILY (THIS.NODE) NE FAMILY (DESTINATION (MESSAGE))
    LET FG = FAMILY (DESTINATION (MESSAGE))
    LET BP.OBJ = FG
    LET FM.GP (MESSAGE) = FG
    GO ASGN.NEXT.STOP
  ELSE
  IF GROUP (THIS.NODE) NE GROUP (DESTINATION (MESSAGE))
    LET FG = GROUP (DESTINATION (MESSAGE))
    LET BP.OBJ = FG
    LET FM.GP (MESSAGE) = FG
    GO ASGN.NEXT.STOP
  ELSE
    LET BP.OBJ = DESTINATION (MESSAGE)
'ASGN.NEXT.STOP'
```

```
FILE: DOC      S       A   NAVAL POSTGRADUATE SCHOOL

  LET NEXT.STOP(MESSAGE) = BEST.PATH (THIS.NODE, BP.OBJ,1)
  LET NODES.HOPPED(MESSAGE) = NODES.HOPPED(MESSAGE) + 1
''
''** SCHEDULE A PROCESSING COMPLETION TIME WHEN THE PKT WILL BE READY
''**  FOR RETRANSMISSION.
''
  SCHEDULE A CON.PACKET.MESSAGE GIVEN MESSAGE IN PROCESSING.TIME UNITS
REGARDLESS
''
''** GO TO THE QUEUE OF THE NODE WHICH RELAYED THE ABOVE PKT.  IF EMPTY
''**  DEFINE THE LINK AS IDLE; IF NOT, PLACE THE NEXT PKT ON THE LINK
''**  AND ADJUST THE QUEUE INFORMATION BY CREATING A NEW PACK.
''
  FOR EACH MESSAGE IN QUEUE(PAST.NODE) WITH NEXT.STOP(MESSAGE)=THIS.NODE,
    FIND THE FIRST CASE
    IF NONE
      LET LNK.MONITOR( PAST.NODE, THIS.NODE, 1) = IDLE
    ELSE
      REMOVE MESSAGE FROM QUEUE(PAST.NODE)
      LET LNK.MONITOR(PAST.NODE, THIS.NODE, 2) =
       LNK.MONITOR(PAST.NODE, THIS.NODE, 2) - 1
      CREATE A PACK
        LET NUMBER(PACK) = LNK.MONITOR (PAST.NODE, THIS.NODE, 2)
        LET ENTRY.TIME(PACK) = TIME.V
        LET PAC.NEIGHBOR(PACK) = THIS.NODE
      FILE PACK IN TIME.QUEUE(PAST.NODE)
      IF NODES.HOPPED(MESSAGE) = 0
        LET RELEASE.TIME (MESSAGE) = TIME.V
      REGARDLESS
''
''** SCHEDULE THE ARRIVAL OF THE PKT JUST RELEASED FROM THE QUEUE.
''
      SCHEDULE AN ARIVE.PACKET GIVEN MESSAGE IN PKT.XMN.TIME UNITS
''
IF PRNT >= 1
      PRINT 1 LINE WITH TRANS.NUMBER(MESSAGE), PACK.NUMBER(MESSAGE),
       RELAYER(MESSAGE), NEXT.STOP(MESSAGE), PM.GP(MESSAGE), DESTINATION
       (MESSAGE), TIME.V, LNK.MONITOR(PAST.NODE, THIS.NODE, 2) AS FOLLOWS
*****/** LEAVES ** THRU ** FOR **(**) AT **.****** SEC (FROM Q) QU=  ***
REGARDLESS
''
    REGARDLESS
RETURN
END ''OF ARIVE.PACKET
''
''
''** THIS ROUTINE CONTINUES THE PACKET ON IT BEST PATH AFTER PROCESSING
''**  IF THE LINK IS IDLE, OR PLACES IT IN A QUEUE.
''
EVENT CON.PACKET.MESSAGE GIVEN IDENT.MESSAGE.NUMBER
LET MESSAGE = IDENT.MESSAGE.NUMBER
LET THIS.NODE = RELAYER (MESSAGE)
''
''** IF LINK IS AVAILABLE, SEND OUT PKT.  LIST LINK AS BUSY.
''
IF LNK.MONITOR ( THIS.NODE, NEXT.STOP(MESSAGE), 1) = IDLE
  SCHEDULE AN ARIVE.PACKET GIVEN MESSAGE IN PKT.XMN.TIME UNITS
''
IF PRNT >= 1
  PRINT 1 LINE WITH TRANS.NUMBER(MESSAGE), PACK.NUMBER(MESSAGE),
   RELAYER(MESSAGE), NEXT.STOP(MESSAGE), PM.GP(MESSAGE),
   DESTINATION(MESSAGE), TIME.V AS FOLLOWS
*****/** LEAVES ** THRU ** FOR **(**) AT **.****** SEC. NO WAIT.
REGARDLESS
''
  LET LNK.MONITOR(THIS.NODE, NEXT.STOP(MESSAGE), 1) = BUSY
ELSE
''
''** IF LINK WAS BUSY, PLACE PKT IN QUEUE AND CREATE A PACK WITH NEW
''**  QUEUE INFORMATION.
''
  FILE MESSAGE IN QUEUE(THIS.NODE)
```

```
FILE: DOC      S       A   NAVAL POSTGRADUATE SCHOOL

  LET LNK.MONITOR (THIS.NODE, NEXT.STOP(MESSAGE),2) =
   LNK.MONITOR (THIS.NODE, NEXT.STOP(MESSAGE), 2) + 1
  IF LNK.MONITOR (THIS.NODE, NEXT.STOP(MESSAGE), 2) >
   LNK.MONITOR (THIS.NODE, NEXT.STOP(MESSAGE), 3)
    LET LNK.MONITOR (THIS.NODE, NEXT.STOP(MESSAGE), 3) =
     LNK.MONITOR (THIS.NODE, NEXT.STOP(MESSAGE), 2)
  REGARDLESS
  CREATE A PACK
    LET NUMBER(PACK) = LNK.MONITOR (THIS.NODE, NEXT.STOP(MESSAGE), 2)
    LET ENTRY.TIME(PACK) = TIME.V
    LET PAC.NEIGHBOR(PACK) = NEXT.STOP(MESSAGE)
  FILE PACK IN TIME.QUEUE (THIS.NODE)
''
IF PRNT >= 1
  PRINT 1 LINE WITH TRANS.NUMBER(MESSAGE), PACK.NUMBER(MESSAGE),
   THIS.NODE, NEXT.STOP(MESSAGE), TIME.V, LNK.MONITOR(THIS.NODE,
   NEXT.STOP(MESSAGE), 2) AS FOLLOWS
*****/** ENTERS QU IN ** FOR ** AT **.****** SEC. QU =  ***
REGARDLESS
''
REGARDLESS
RETURN
END ''OF CON.PACKET
''
''
''** THIS ROUTINE COLLECTS STATISTICAL DATA WHEN A PKT REACHES ITS
''**  DESTINATION.
''
EVENT COMPLETED.TRIP GIVEN MES.NUM
DEFINE DEL.TIME AS A REAL VARIABLE
LET MESSAGE = MES.NUM
LET CNTR = NODES.HOPPED(MESSAGE) + 1
''
''** PRINT ALERT IF NODES HOPPED >= TOTAL NODES IN NETWORK.
''
IF (CNTR >= N.NODE AND MSG.HLT = 0)
  PRINT 1 LINE AS FOLLOWS
  PROBLEM -- MORE HOPS THAN NODES
  LET MSG.HLT = 1
REGARDLESS
''
''** INCREMENT COUNTER FOR TOTAL NODES HOPPED FOR THIS PKT AND SUM NET
''**  TIME FOR GIVEN NUMBER OF HOPS.
''
LET HOP.COUNT (CNTR, PACKET) = HOP.COUNT (CNTR, PACKET) + 1
LET DEL.TIME = TIME.V - RELEASE.TIME (MESSAGE)
LET CLOCK.DATA (CNTR, 1) = CLOCK.DATA (CNTR, 1 )+ DEL.TIME
''
''** NOTE IF THIS TRANSIT TIME IS A NEW MAX FOR THIS NUMBER OF HOPS.
''
IF DEL.TIME > CLOCK.DATA (CNTR, 2)
  LET CLOCK.DATA (CNTR, 2) = DEL.TIME
REGARDLESS
''
''** INCREMENT TRACER ARRAY WHICH KEEPS THE NUMBER OF PKTS GENERATED
''**  AND THE NUMBER REACHING THEIR DESTINATION.
''
LET TRACER (TRANS.NUMBER(MESSAGE), 2)= TRACER(TRANS.NUMBER(MESSAGE),2)+1
DESTROY MESSAGE CALLED MES.NUM
RETURN
END ''OF COMPLETED.TRIP
''
''
''** THIS ROUTINE IDENTIFIES A SET OF THE BUSIEST LINKS OVER THE FIRST
''**  HALF OF THE SIMULATION SO THE LINKS CAN BE SAMPLED DURING THE
''**  SECOND HALF OF THE TEST.
''
EVENT QU.SAMPLER
''
PRINT 1 LINE AS FOLLOWS
-------------------------------------MID POINT
''
```

```
FILE: DOC      S       A   NAVAL POSTGRADUATE SCHOOL

''&& ID THE LARGEST QUEUE SIZE IN THE FIRST HALF OF THE SIMULATION.
''
LET I = 1
FOR F = 1 TO N.NODE, DO
  FOR T = 1 TO N.NODE, DO
    IF MAX < LNK.MONITOR (F,T,3)
      LET MAX = LNK.MONITOR (F,T,3)
    REGARDLESS
  LOOP
LOOP
''
''&& SMP.LINKS IS AN INPUT VARIABLE LISTING THE NUMBER OF LINKS TO BE
''&&  SAMPLED FOR QUEUE SIZE.
''&& SMP.SET IS AN ARRAY OF NODE PAIRS FOR THE "SMP.LINKS" BUSIEST
''&&  LINKS OVER THE FIRST HALF OF THE SIMULATION.
''
'FILL.SMP.SET'
FOR F = 1 TO N.NODE, DO
  FOR T = 1 TO N.NODE, DO
    IF LNK.MONITOR (F,T,3) = MAX
      LET SMP.SET(I,1) = F
      LET SMP.SET(I,2) = T
      IF I = SMP.LINKS
        GO BEGIN.SAMPLING
      ELSE
        LET I = I+1
        GO NOP.NEW.MAX
    ELSE
    IF NEW.MAX < LNK.MONITOR(F,T,3) AND LNK.MONITOR(F,T,3) < MAX
      LET NEW.MAX = LNK.MONITOR(F,T,3)
    REGARDLESS
'NOP.NEW.MAX'
  LOOP
LOOP
LET MAX = NEW.MAX
GO FILL.SMP.SET
''
''&& SCHEDULE FIRST SAMPLE OF ALL LINKS IN SMP.SET
''
'BEGIN.SAMPLING'
LET TI.MEA = TIME.LIMIT/(2.* REAL.F(NO.OF.SAMPLES))
SCHEDULE A SAMPLE IN (EXPONENTIAL.F(TI.MEA, 8)) UNITS
RETURN
END '' OF QU.SAMPLER
''
''
''&& THIS ROUTINE SAMPLES THE LINKS IDENTIFIED IN QU.SAMPLER WITH AN
''&&  EXPONENTIAL SAMPLING RATE.
''
EVENT SAMPLE
''
''&& COUNT ACTUAL SAMPLES TAKEN.
''
LET SMP.CNTR = SMP.CNTR + 1
''
''&& INCREMENT COUNTER BASED ON QUEUE SIZE.
''
FOR I = 1 TO SMP.LINKS, DO
  LET QU.DIST(LNK.MONITOR(SMP.SET(I,1),SMP.SET(I,2), 2)+1) =
    QU.DIST(LNK.MONITOR(SMP.SET(I,1),SMP.SET(I,2), 2)+1) + 1
LOOP
''
''&& SCHEDULE THE NEXT SAMPLE.
''&& TI.MEA IS DEFINED IN QU.SAMPLER
''
SCHEDULE A SAMPLE IN (EXPONENTIAL.F(TI.MEA,8)) UNITS
RETURN
END ''OF SAMPLE
/*
//SIM006  DD SYSOUT=A
//*G.SIM006  DD  VOL=SER=MVS004,UNIT=3350,DISP=SHR,DSN=S2036.RUN,
//*  DCB=(RECFM=FB,LRECL=133,BLKSIZE=4123)
```

```
FILE: D29W      D         A    NAVAL POSTGRADUATE SCHOOL


  --> TYPICAL INPUT FILE <--

29                          <- NUMBER OF LINKS
1.  1.   1   1              <- TRANSMIT FACTOR:
1.  1.   1   1                 RECEIVE FACTOR:
1.  1.   2   1                   GROUP NUMBER:
1.  1.   2   1                     FAMILY NUMBER
1.  1.   1   1                       "
1.  1.   1   1
1.  1.   2   1
1.  1.   2   1
1.  1.   1   1
1.  1.   1   1
1.  1.   2   1
1.  1.   2   1
1.  1.   1   2
1.  1.   1   2
1.  1.   2   2
1.  1.   2   2
1.  1.   1   3
1.  1.   1   3
1.  1.   2   3
1.  1.   2   3
1.  1.   1   4
1.  1.   1   4
1.  1.   2   4
1.  1.   2   4
.1                          <- UPDATE PERIOD
.0001                       <- PKT PROCESSING TIME
.05                         <- PKT TRANSMISSION TIME
500.                        <- TEST DURATION
.1                          <- SESSION INTERVAL
5                           <- WINDOW
00.  00.                    <- % INNER-GROUP; % INNER FAMILY
0                           <- DIAGNOSTIC PRINT LEVEL
10 1000                     <- NUMBER OF LINKS TO SAMPLE; NUMBER OF SAMPLES EACH
48                          <- NUMBER OF LINKS
1   2                       <- NODE PAIRS FOR EACH LINK
                               "
1   3
2   3
2   4
3   4
3   5
4   5
4   6
5   6
5   10
6   7
6   11
7   8
7   9
8   9
8   16
10  15
10  11
11  12
11  16
12  13
12  16
13  14
13  15
14  15
15  20
16  17
16  21
17  22
```

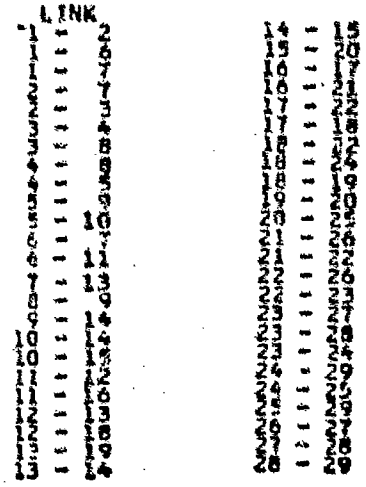
```
FILE: D290      D        A   NAVAL POSTGRADUATE SCHOOL

17   18
18   22
18   24
18   19
19   20
20   25
21   26
21   30
22   26
22   23
23   27
23   28
23   24
24   29
24   25
25   30
25   27
26   27
27   28
28   29
/*
//
```

OUTPUT EXAMPLE

NODE NO.	TRANSMIT FACTOR	RECEIVE FACTOR	GROUP (PGM #)	FAMILY (PGM #)
1	1.000	1.000	1(30)	1(34)
2	1.000	1.000	1(30)	1(34)
3	1.000	1.000	2(31)	1(34)
4	1.000	1.000	2(31)	1(34)
5	1.000	1.000	2(31)	1(34)
6	1.000	1.000	1(30)	1(34)
7	1.000	1.000	1(30)	1(34)
8	1.000	1.000	2(31)	1(34)
9	1.000	1.000	2(31)	1(34)
10	1.000	1.000	2(31)	1(34)
11	1.000	1.000	1(30)	1(34)
12	1.000	1.000	1(30)	1(34)
13	1.000	1.000	1(30)	1(34)
14	1.000	1.000	2(31)	1(34)
15	1.000	1.000	2(31)	1(34)
16	1.000	1.000	3(32)	2(35)
17	1.000	1.000	3(32)	2(35)
18	1.000	1.000	4(33)	2(35)
19	1.000	1.000	4(33)	2(35)
20	1.000	1.000	4(33)	2(35)
21	1.000	1.000	3(32)	2(35)
22	1.000	1.000	3(32)	2(35)
23	1.000	1.000	4(33)	2(35)
24	1.000	1.000	4(33)	2(35)
25	1.000	1.000	4(33)	2(35)
26	1.000	1.000	3(32)	2(35)
27	1.000	1.000	3(32)	2(35)
28	1.000	1.000	4(33)	2(35)
29	1.000	1.000	4(33)	2(35)

UPDATE PERIOD IS .100000 SEC
PROCESSING TIME IN EACH NODE FOR ANY PACKET IS .000100 SEC
PACKET TRANSIT TIME BETWEEN ANY TWO NODES IS .050000 SEC
TEST DURATION IS 500.000000 SEC. TEST LIMITED TO 55000 TRAFFIC SESSIONS.
NEW TRAFFIC SESSIONS ARE STARTED AT AN AVERAGE INTERVAL OF .100000 SEC
CHANNEL VALUE CALCULATION WINDOW IS .500000 SEC
EACH TRAFFIC SESSION VARIES FROM 1 TO 21 PACKETS
AT LEAST 0. % OF TRAFFIC IS INNER GROUP, ANOTHER 0. % IS INNER FAMILY.



OUTPUT EXAMPLE

NODE NO.	TRANSMIT FACTOR	RECEIVE FACTOR	GROUP (PGM #)	FAMILY (PGM #)
1	1.000	1.000	1(30)	1(34)
2	1.000	1.000	1(30)	1(34)
3	1.000	1.000	2(31)	1(34)
4	1.000	1.000	2(31)	1(34)
5	1.000	1.000	2(31)	1(34)
6	1.000	1.000	1(30)	1(34)
7	1.000	1.000	1(30)	1(34)
8	1.000	1.000	2(31)	1(34)
9	1.000	1.000	2(31)	1(34)
10	1.000	1.000	2(31)	1(34)
11	1.000	1.000	1(30)	1(34)
12	1.000	1.000	1(30)	1(34)
13	1.000	1.000	1(30)	1(34)
14	1.000	1.000	2(31)	1(34)
15	1.000	1.000	2(31)	1(34)
16	1.000	1.000	3(32)	2(35)
17	1.000	1.000	3(32)	2(35)
18	1.000	1.000	4(33)	2(35)
19	1.000	1.000	4(33)	2(35)
20	1.000	1.000	4(33)	2(35)
21	1.000	1.000	3(32)	2(35)
22	1.000	1.000	3(32)	2(35)
23	1.000	1.000	4(33)	2(35)
24	1.000	1.000	4(33)	2(35)
25	1.000	1.000	4(33)	2(35)
26	1.000	1.000	3(32)	2(35)
27	1.000	1.000	3(32)	2(35)
28	1.000	1.000	4(33)	2(35)
29	1.000	1.000	4(33)	2(35)

UPDATE PERIOD IS .100000 SEC
PROCESSING TIME IN EACH NODE FOR ANY PACKET IS .000100 SEC
PACKET TRANSIT TIME BETWEEN ANY TWO NODES IS .050000 SEC
TEST DURATION IS 500.000000 SEC. TEST LIMITED TO 55000 TRAFFIC SESSIONS.
NEW TRAFFIC SESSIONS ARE STARTED AT AN AVERAGE INTERVAL OF .100000 SEC
CHANNEL VALUE CALCULATION WINDOW IS .500000 SEC
EACH TRAFFIC SESSION VARIES FROM 1 TO 21 PACKETS
AT LEAST 0. % OF TRAFFIC IS INNER GROUP. ANOTHER 0. % IS INNER FAMILY.

LINK		
1	-	2
1	-	7
1	-	8
1	-	10
1	-	11
1	-	13
1	-	14
1	-	15
1	-	20
2	-	7
2	-	8
2	-	10
2	-	11
2	-	13
2	-	14
2	-	15
2	-	20
3	-	8
3	-	10
3	-	11
3	-	13
3	-	14
3	-	15
3	-	20
4	-	8
4	-	10
4	-	11
4	-	13
4	-	14
4	-	15
4	-	20
5	-	8
5	-	10
5	-	11
5	-	13
5	-	14
5	-	15
5	-	20
6	-	7
6	-	8
6	-	10
6	-	11
6	-	13
6	-	14
6	-	15
6	-	20
7	-	8
7	-	10
7	-	11
7	-	13
7	-	14
7	-	15
7	-	20
8	-	10
8	-	11
8	-	13
8	-	14
8	-	15
8	-	20
9	-	10
9	-	11
9	-	13
9	-	14
9	-	15
9	-	20
10	-	11
10	-	13
10	-	14
10	-	15
10	-	20
11	-	13
11	-	14
11	-	15
11	-	20
12	-	13
12	-	14
12	-	15
12	-	20
13	-	14
13	-	15
13	-	20
14	-	15
14	-	20
15	-	20
16	-	17
16	-	22
17	-	22
18	-	22
18	-	24
19	-	22
20	-	22
21	-	22
22	-	23
22	-	26
23	-	26
24	-	26
25	-	26
26	-	27
26	-	28
27	-	28
28	-	29

```
UPDATE PERIOD IS    .100000 SEC
PROCESSING TIME IN EACH NODE FOR ANY PACKET IS .000100 SEC
PACKET TRANSIT TIME BETWEEN ANY TWO NODES IS .050000 SEC
TEST DURATION IS 500.000000 SEC. TEST LIMITED TO 95000 TRAFFIC SESSIONS.
NEW TRAFFIC SESSIONS ARE STARTED AT AN AVERAGE INTERVAL OF    .100000 SEC
CHANNEL VALUE CALCULATION WINDOW IS    .500000 SEC
EACH TRAFFIC SESSION VARIES FROM  1 TO 21 PACKETS
AT LEAST  0. % OF TRAFFIC IS INNER GROUP, ANOTHER  0. % IS INNER FAMILY.
```

NODES HOPPED	NO. PKTS	MEAN TIME PER PKT	PEAK TIME TIME	IDEAL TIME
1	5769	.050045	.050244	.050000
2	7475	.148045	1.400245	.100100
3	8365	.267807	2.068256	.150200
4	7718	.371600	2.439480	.200300
5	6757	.473230	2.486388	.250400
6	5470	.600169	2.949171	.300500
7	4253	.716920	4.290086	.350600
8	3234	.839226	4.190083	.400700
9	1974	.994289	4.094386	.450800
10	1145	1.146388	4.239901	.500900
11	704	1.376437	3.966151	.551000
12	372	1.514461	4.153350	.601100
13	191	1.590849	5.253336	.651200
14	79	1.771252	3.439857	.701300
15	24	2.387163	6.051393	.751400
16	17	2.272257	3.954242	.801500
17	8	1.901922	3.539663	.851600
18	6	2.303152	3.253066	.901700
19	6	2.162179	4.206535	.951800
20	2	1.829691	2.367598	1.001900
25	2	1.994116	2.502305	1.252399

```
MEAN NUMBER OF NODES HOPPED PER PACKET IS   4.6

A TOTAL OF  4956 NEW XMNS WERE STARTED (TOTALING  93664 PACKETS ).
  OF THESE,    73 PACKETS WERE UNDELIVERED WHEN THE TEST WAS ENDED.

FOR EACH NEW UPDATE, AN AVERAGE OF  23.1 LINKS WERE USED.

LONGEST BEST PATH AT ANY TIME WAS  16 LINKS.

QUEUE LENGTH DISTRIBUTION
  10 LINKS WERE SAMPLED
 1030 SAMPLES / LINK WERE TAKEN
```

MAXIMUM QUEUE LENGTHS

FROM	TO	MAX
1	2	21
2	1	20
1	6	24
6	1	20
1	7	35
7	1	35
2	4	8
4	2	8
2	3	33
3	2	33
3	4	48
4	3	39
4	3	36
3	8	20
8	3	20
4	8	32
8	4	28
4	5	26
5	4	26
5	9	22
9	5	22
5	10	34
10	5	34
6	7	16
7	6	16
6	11	28
11	6	28
7	12	33
12	7	33
8	9	24
9	8	24
9	14	46
14	9	46
10	14	35
14	10	35
10	15	25
15	10	25
11	12	24
12	11	24
11	16	28
16	11	28
12	13	33
13	12	33
12	18	26
18	12	27
13	19	29
19	13	29
13	14	34
14	13	35
14	15	24
15	14	24
15	20	32
20	15	32
16	17	34
17	16	28
16	21	28
21	16	22
17	22	21
22	17	20
17	18	36

18	17	20
18	22	19
22	18	33
18	24	21
24	18	26
18	19	27
19	18	28
19	20	22
20	19	20
20	25	27
25	20	31
21	26	20
26	21	19
21	22	33
22	21	23
22	26	16
26	22	23
22	23	20
23	22	20
23	27	20
27	22	18
23	28	20
28	23	29
23	24	22
24	23	20
24	29	18
29	24	35
24	25	24
25	24	21
25	29	20
29	25	31
26	27	19
27	26	37
27	28	20
28	27	19
28	29	34
29	28	20

- Q-SIZE -	SAMPLE DENSITY
0	5613
1	115
2	73
3	58
4	34
5	60
6	45
7	49
8	22
9	26
10	25
11	15
12	61
13	11
14	13
15	17
16	13
18	8
19	8
20	3

AVERAGE Q LENGTH = .435

STANDARD DEVIATION = 2.0461
UNUSUAL DELAYS FOR PACKETS NOT DELIVERED DESCRIBED BELOW

LIST OF REFERENCES

1. Yen, J. Y., A Decentralized Algorithm for Finding the Shortest Path in Defense Communications Networks, Naval Postgraduate School research report (NPS55-79-015), July 1979.

2. Segall, Adrian, "Advances in Verifiable Fail-Safe Routing Procedures", IEEE Transactions on Communications, Vol. Comm-29, No.4, April 1981.

3. Kahn, Robert, et.al., "Advances in Packet Radio Technology", Proceedings of the IEEE, Vol. 66, No.11, Nov. 1978.

4. Golestaani, Seyyed, Design of a Retransmission Strategy for Error Control in Data Communications Networks, Massachusetts Institute of Technology, Report ESL-R-674, July 1976.

5. Finn, Steven, "Resynch Procedures and a Fail-Safe Network Protocol", IEEE Transactions on Communications, Vol. Com-27, No.6, June 1979.

6. Bond, James, Self-interference in a Packet Radio Network, Masters Thesis, Naval Postgraduate School, Monterey, California, Dec. 1980.

7. Lucke, Edmund, An Investigation of Distributed Communications Systems and their Potential Applications to Command and Control Structures in the Marine Corps, Masters Thesis, Naval Postgraduate School, Monterey, California, Dec. 1979.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93940	2
2.	Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, CA 93940	1
3.	Prof. J. M. Wozencraft, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, CA 93940	3
4.	PM, Test Measurement and Diagnostic Systems ATTN: DRCPM-TMDS (MAJ Robert Heritsch) Ft. Monmouth, NJ 07703	4
5.	C3 Division (Code D102) Development Center Marine Corps Development Education Command Quantico, VA 22134	2
6.	Dr. Michael Athans Laboratory for Information and Decision Systems Massachusetts Institute of Technology Boston, MA 01239	1
7.	Prof. Robert G. Gallager Laboratory for Information and Decision Systems Massachusetts Institute of Technology Boston, MA 01239	1
8.	Dr. Vint Cerf Information Processing Techniques Office Defense Advanced Research Projects Agency Washington, D.C. 20390	1
9.	Dr. Barry M. Leiner Information Processing Techniques Office Defense Advanced Research Projects Agency Washington, D.C. 20390	1
10.	Naval Electronics Systems Command Code 03 Washington, D.C. 20390	1
11.	Naval Electronics Systems Command Marine Corps Representative, Code PME-154 Washington, D.C. 20390	1

12. Office of Naval Research
Marine Corps Representative, Code 100M
Washington, D.C. 20390 — 1

13. Dr. Adam Feit
14 Apt. 7 Kazan St.
Ranana, 43000 Israel — 1

14. Commander, Naval Ocean Systems Center
Code 447 (Library)
San Diego, CA 92152 — 2

15. Capt. Kenneth L. Moore, Code 62 MZ
Department of Electrical Engineering
Naval Postgraduate School
Monterey, CA 93940 — 1

16. HQ CECOM
ATTN: DRSCL-POD-P (Mr. Resnic)
Ft. Monmouth NJ 07703 — 2

17. Defense Technical Information Center
Cameron Station
Alexandria, VA 22314 — 2

18. Deputy Under Secretary of the Army
for Operations Research
Room 2E261, Pentagon
Washington, D.C. 20310 — 1

19. LCDR Anthony W. Lengerich, USN
SMC Box 1101
Naval Postgraduate School
Monterey, CA 93940 — 1

20. LT George Wasenius, USN
SMC Box 1162
Naval Postgraduate School
Monterey, CA 93940 — 1