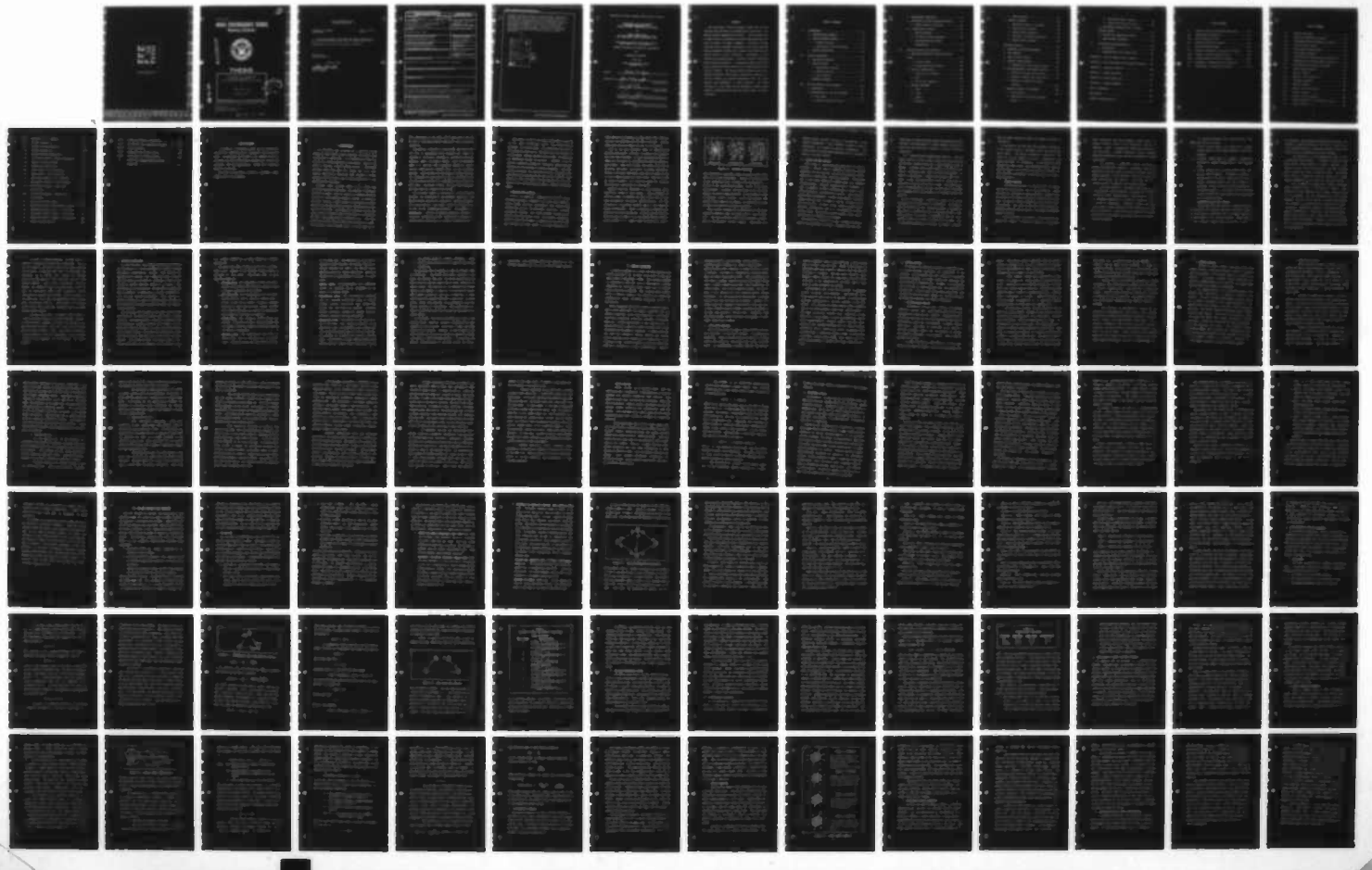
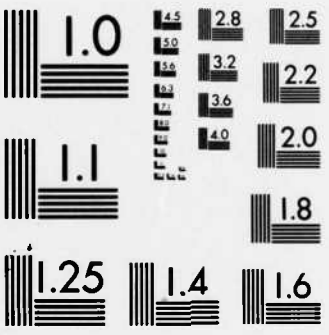
AD-A139 147 APPLICATION OF A DISTRIBUTED ROUTING ALGORITHM TO A PACKET-SWITCHED COMMUNICATIONS NETWORK(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA R R LOGAN DEC 83 1/4

UNCLASSIFIED NPS62-83-060 F/G 17/2 NL





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

(2)

NPS62-83-060

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD A139147



THESIS

APPLICATION OF A DISTRIBUTED
ROUTING ALGORITHM TO A
PACKET-SWITCHED COMMUNICATIONS NETWORK

by

Robert Reade Logan

December 1983

Thesis Advisor: J. M. Wozencraft

DTIC FILE COPY

DTIC ELECTE MAR 2 0 1984 S E D

Approved for public release; distribution unlimited

Prepared for:
Naval Ocean Systems Center
San Diego, California

84 03 19 084

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Commodore R. H. Shumaker
Superintendent

David A. Schrady
Provost

This thesis prepared in conjunction with research supported in part by Naval Ocean Systems Center under work request N66001 83WR00093.

Reproduction of all or part of this report is authorized.

Released as a
Technical Report by:

William M. Tolles

WILLIAM M. TOLLES
Dean of Research

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS62-83-060	2. GOVT ACCESSION NO. AD-A139 147	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Application of a Distributed Routing Algorithm to a Packet-Switched Communications Network		5. TYPE OF REPORT & PERIOD COVERED Engineer's Thesis; December 1983 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert Reade Logan		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62721N: CF21245X4S N66001 83WR00093
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Ocean Systems Center 271 Catalina Boulevard San Diego, California 92152		12. REPORT DATE December 1983 13. NUMBER OF PAGES 335
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 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 Switching; Communications Networks; Routing Protocols; Distributed Routing Protocols; Dynamic Routing Protocols; Computer Simulation; Network Analysis; Yen Algorithm

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

Two distributed routing procedures based upon the Yen shortest path algorithm are developed for application in a packet-switched communications network. The algorithm uses a unique method in calculating shortest paths based upon the time of arrival of update messages. The first routing procedure uses a proposed "combination" link weight function having parameters involving both current link queue size and recent history of link utilization. Performance of this procedure is analyzed under a

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

variety of network conditions using computer simulation. A comparison study is done with both a least hop routing protocol and a multiple path static routing protocol. The second routing procedure has a hierarchical structure which offers substantial reductions in routing traffic and memory requirements over the first version when implemented in large networks. The major conclusion is that these routing procedures exhibit robust operating characteristics which are almost optimal in simple situations.

Accession For
NTIS GRA&I [X]
DTIC TAB [ ]
Unannounced [ ]
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

DTIC COPY INSPECTED

Approved for public release; distribution unlimited.

Application of a Distributed Routing Algorithm to a Packet-Switched Communications Network

by

Robert Reade Logan
Captain, United States Marine Corps
B.S., Washington State University, 1976

Submitted in partial fulfillment of the requirements for the degrees of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

and

ELECTRICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
December 1983

Author: Robert R. Logan

Approved by: John M. Wozencraft
Thesis Advisor

Carl R. Jones
Second Reader

Chairman, Department of Electrical Engineering

J. M. Dyer
Dean of Science and Engineering

ABSTRACT

Two distributed routing procedures based upon the Yen shortest path algorithm are developed for application in a packet-switched communications network. The algorithm uses a unique method in calculating shortest paths based upon the time of arrival of update messages. The first routing procedure uses a proposed "combination" link weight function having parameters involving both current link queue size and recent history of link utilization. Performance of this procedure is analyzed under a variety of network conditions using computer simulation. A comparison study is done with both a least hop routing protocol and a multiple path static routing protocol. The second routing procedure has a hierarchical structure which offers substantial reductions in routing traffic and memory requirements over the first version when implemented in large networks. The major conclusion is that these routing procedures exhibit robust operating characteristics which are almost optimal in simple situations.

TABLE OF CONTENTS

I. INTRODUCTION 14

A. COMMUNICATIONS NETWORKS 16

B. PACKET-SWITCHED NETWORKS 19

1. Message Services 21

2. Application with Voice Traffic 23

C. NETWORK ARCHITECTURE 26

II. ROUTING PROCEDURES 31

A. LINK WEIGHT FUNCTION 32

B. ROUTING CLASSES 34

1. Deterministic Routing 34

2. Adaptive Routing 37

3. Hybrid Routing 45

C. CONGESTION CONTROL 47

III. THE YEN SHORTEST PATH ALGORITHM 54

A. DESCRIPTION 55

B. MODIFICATIONS TO THE YEN ALGORITHM 66

1. Time Delay 66

2. Multiple Destination Nodes 73

C. COMPUTATIONAL EFFICIENCY 74

D. APPLICATION TO PACKET SWITCHED NETWORKS . . . 78

1. Update Insertion Technique 80

E. A HIERARCHICAL VERSION 86

1. Warning Messages 88

2. Originating Warning Messages 90

3. Synchronizing Unit Update Transmissions 92

IV. NETWORK MODELING AND SIMULATION 98

A. ANALYTICAL TECHNIQUES INVOLVING QUEUING THEORY 99

1. Little's Result 100

2. Pollaczek-Khinchin Formula 102

B. NOTES ON SIMSCRIPT II.5 107

1. Entities and Attributes 107

2. Sets 109

3. Statistical Analysis 111

C. NETWORK COMPONENTS 112

1. Nodes 112

2. Links 114

3. Queues 115

4. Messages 118

5. Update Packets 120

D. NETWORK OPERATION 123

1. Preparation Prior to Simulation 125

2. Update Routing Protocol 128

3. Message Packet Transport 134

4. Evaluation of Network Performance . . . 135

5. Example Update Sequence 138

V. SIMULATION RESULTS 140

A. MEASURING ALGORITHM PERFORMANCE 141

B. TEST NETWORK 143

C. STATIC ROUTING RESULTS 145

D. DYNAMIC ROUTING RESULTS 149

1. Determining the Link Weight Function . . 150

E. THE "COMBINATION" LINK WEIGHT FUNCTION . . . 166

1. Results of Varying Window Period 174

2. Results of Varying the Queue Factor . . 177

3. Results of Varying Node Update Interval 182

F. TESTING THE NODAL YEN ALGORITHM 189

1. Timing Errors 190

2. Geometrically Distributed Message Lengths 197

3. Unbalanced Traffic Patterns 201

4. Datagram and Virtual Circuits 207

G. PERFORMANCE COMPARISON WITH A ROUTING FRACTION SCHEME 210

H. TESTING THE HIERARCHICAL VERSION 212

I. LARGE NETWORK SIMULATION 219

VI. CONCLUSIONS AND RECOMMENDATIONS 225

A. CONCLUSIONS 225

B. RECOMMENDATIONS FOR FURTHER STUDY 229

APPENDIX A: THE YEN SHORTEST PATH ALGORITHM 233

APPENDIX B: THE "SUCCESSOR" NODE SELECTION ALGORITHM . 236

APPENDIX C: SIMULATION PROGRAM 245

APPENDIX D: SAMPLE INPUT DATA 312

APPENDIX E: SAMPLE OUTPUT DATA 313

APPENDIX F: GRAPHICS PROGRAMS 321

LIST OF REFERENCES 330

BIBLIOGRAPHY 333

INITIAL DISTRIBUTION LIST 334

LIST OF TABLES

I. Actions of Basic and Modified Algorithms 72

II. Computational Efficiency 77

III. Output Sequence of FSR 96

IV. Simulation Best Path Results 139

V. Least Hop Routing Table 146

VI. Simulation Validation using Little's Result . . 149

VII. Effects of Update Queueing Delays 196

VIII. Average Number of Packets in Network 200

IX. Performance Comparison of the Algorithms 212

X. Performance of Hierarchical Version 216

LIST OF FIGURES

1.1	Network Topclogies	18
3.1	Algorithm Demonstration Network	59
3.2	Transmission Delay Demonstration Network	69
3.3	Time Delay Test Network	71
3.4	Update Packet Configuration	82
3.5	Group Update Sequence	89
3.6	Feedback Shift Register: DSSS Configuration . .	95
3.7	Feedback Shift Register: FHSS Configuration . .	96
5.1	Test Network for Nodal Yen Algorithm	144
5.2	Static Routing Results	148
5.3	Method 1 Results	153
5.4	Method 1 Link Utilization	154
5.5	Windowing Technique	156
5.6	Method 2 Results	157
5.7	Method 2 Link Utilization	158
5.8	Method 3 Results	159
5.9	Method 3 Link Utilization	160
5.10	Average Queue Size vs. Utilization Factor . .	162
5.11	Method 4 Results	164

5.12 Method 4 Link Utilization 165

5.13 Method 5 Results I 167

5.14 Method 5 Link Utilization I 168

5.15 Method 5 Results II 169

5.16 Method 5 Link Utilization II 170

5.17 Link Weight Methods : Stable Performance . . . 171

5.18 Varying the Window Period 176

5.19 Varying the Queue Factor 178

5.20 Queue Factor : 0.50 Utilization 180

5.21 Queue Factor : 0.80 Utilization 181

5.22 Node Update Interval : Low Overhead 184

5.23 Node Update Interval : High Overhead 186

5.24 Node Update Interval : Total Link Utilization 187

5.25 Node Update Interval : Update Link Utilization 188

5.26 Varying the Clock Interval 194

5.27 Geometrically Distributed Message Lengths . . 198

5.28 Unbalanced Traffic: Static - 30 pkts/sec . . . 204

5.29 Unbalanced Traffic: Dynamic - 50 pkts/sec . . 205

5.30 Unbalanced Traffic: Dynamic - 100 pkts/sec . . 206

5.31 Virtual Circuit Results 208

5.32 Virtual Circuits vs. Datagrams 209

5.33 Test Network for Hierarchical Version 213

5.34 96 Node Network 221

5.35 Large Network: Static Routing 222

5.36 Large Network: Dynamic Routing 224

B.1 Network for Demonstrating Successor Algorithm 242

ACKNOWLEDGMENT

I consider it a privilege to have had this opportunity to work with Professor John M. Wozencraft during the past year. He epitomized my concept of the academic leader by providing not only knowledgeable guidance and stimulating insights, but also encouraging, rather than stifling, personal initiative.

This work is dedicated to my wife, Elizabeth, whose support made it all possible.

I. INTRODUCTION

In describing the activities of a military recruit during the first phase of basic training, the phrase "hurry up and wait" is often used. An illustration of this practice is the double-time march to the barber, only to wait hours in line for a thirty second haircut. The recruit probably feels there must be a better way to do this. Similar feelings may also occur when stopped in freeway traffic or while trying to get an open telephone line to the grandparents on Christmas.

Although these problems appear to be taken from distinctly different situations, they possess a fundamental property in common; they all relate to processes involving the flow of some commodity through a channel or network of channels. In fact, a wide variety of large physical systems exhibit all the characteristics which mathematicians associate with networks. The freeway and telephone systems may have nothing in common by way of installations and equipment yet both have similar network representations. Both have junctions or nodes which are connected by channels or links. In the links of these networks there is a flow (cars in one

and conversations in the other) and in both cases links can be used to form paths between nodes which are not directly connected.

The need for solutions to the problems associated with the flow of commodities within networks has caused an increase of activity in branches of mathematics involving queueing, graph and network theory. Ford and Fulkerson [Ref. 1] in the late 1950's developed methods to solve a railway transportation problem known as the Hitchcock problem. Their solution was the design of network flows so as to minimize the costs related to use of the network while still satisfying certain flow requirements which existed at some nodes. The result of their work was a more efficient and cost effective transportation system.

In the events leading to the solution of the Hitchcock problem, the importance of the commodity carried by the network helped provide the impetus for rigorous research. In a similar fashion, the current focus of attention is on an item of ever increasing importance: the commodity of <u>information</u>. Finding increasingly efficient methods to "transport" information through communications networks has indeed become a major goal to a variety of research efforts.

This thesis is part of an ongoing effort being conducted at the Naval Postgraduate School with communications networks, and in particular, the packet-switched network. The purpcse of this wcrk is the application and analysis of a distributed routing procedure which guides the ccmmodity cf information through the network. Chapters I and II provide background irforma tion concerning packet-switched networks and the function of routing procedures. The third chapter introduces the distributed routing algorithm which was mcdified and adapted for use within such a network. The remainder of the work deals with the thorough test and evaluatien of the routing protocol using a computer simulation model.

A. COMMUNICATIONS NETWORKS

As with all networks, communications networks can be represented as being composed of the three basic elements; nodes, links, and flow commodities. The commodity has already been seen tc be information which must be passed through the network. The nodes represent communicatens centers which can receive, store, and transmit information. Nodes are connected by means of communication channels which are the links with other nodes. A detailed descriptien of

these elements is purposely not given since their configuration may vary depending upon the type of network. However, their basic functions remain the same in all networks.

Due to the many different types of communications networks which exist, a variety of methods for categorization have been developed. A basic classification method is based upon the number of communications channels present in the network. When only a single link exists which must be shared by all nodes, the network is termed "broadcast". However if multiple links exist, such that all nodes are not in direct communications with each other, then the network is "point-to-point". In this case nodes not directly connected must communicate via other nodes.

Networks are also described by their topology using terminology common to graph theory. The arrangement of nodes and links, for example, may form a star, tree or loop. Each configuration has certain characteristics, with the selection of a particular topology being a very important design consideration. Often in reference to communications networks, the words centralized, decentralized and distributed will also be used to give the general network configuration (Figure 1.1). These three configurations tend to

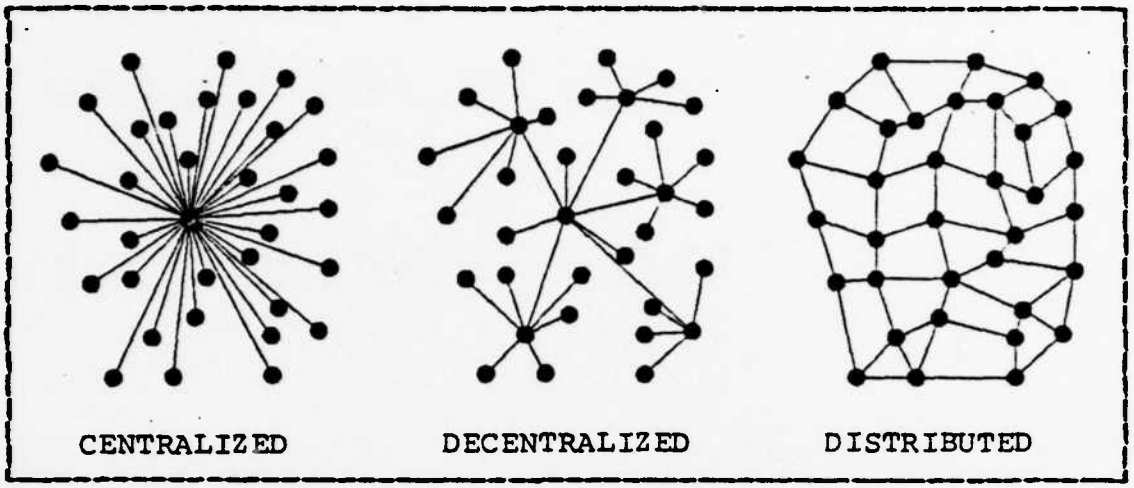


Figure 1.1 Network Topologies

lend themselves to particular types of network control which often use the same terminology. Control of the network may be the responsibility of a single node (centralized), a portion of the nodes (decentralized), or all of the nodes (distributed). Confusion can occur when it is not clear whether the network being described is distributed in terms of topology, control or both. In the next chapter, the network control function of routing is categorized in terms of where the algorithm is actually performed.

Point-to-point networks can be subdivided into two basic categories depending upon the method used to pass information. The first technique called line-switching is analogous to the telephone (voice) network. Here calls and

message routing are set up prior to the beginning of the message transmission. Once a complete circuit or route is established, the message is ready for transmission. The second method is called packet-switching and is discussed in detail in the next section.

B. PACKET-SWITCHED NETWORKS

The concept of packet-switching arose from the need to provide a "better way" to exchange information between computers than was possible with the line-switching techniques used in conventional telephone networks. The problem lay in the basic difference between human and computer "conversations". Computer communication is "bursty", in that it consists of short transmissions of data at high rates followed by long periods of silence. This is quite unlike human conversation in which calls usually last much longer than the time it takes to set up the call in the switching center. In other words, human-to-human traffic requires long-duration use of a low-bandwidth channel, whereas (some) computer-to-computer traffic needs short-duration use of a high-bandwidth channel.

The alternative switching strategy of packet-switching removes the requirement of having a dedicated physical path

established in advance between source and destination nodes. Therefore the inefficiency of having computer conversations which take less time than is required to set up the call is eliminated.

In a packet-switched communications network, when the source node has a block of data to send, that block is first divided into subunits which are a fixed number of bits in length. Attached then to each subunit is addressing and control information such that the resulting "packet" can reach its destination. Packet size (usually 1000 bits or less) is kept relatively small so that no single user can monopolize a link for too long at a time. The basic operation of packet-switching can best be described by following the movements of the first packet as it proceeds through the network.

At the source node, a decision is made as to which outgoing link the packet will be transmitted. The decision rule is referred to as a routing procedure or algorithm. The status of the selected link is then checked and if that link is idle, the packet is transmitted over the link. However if the link is busy, the packet joins a queue; this is accomplished physically by storing the packet in memory.

When the packet reaches the head of the queue, the packet is transmitted.

In arriving at the next node, the addressing information of the packet is used to determine if the destination has been reached. The routing procedure is repeated if the packet has not yet completed its trip. Eventually, the packet will reach its destination. It is clear why these networks are often called store-and-forward nets, since in passing through a node packets are stored, if necessary, and then forwarded (transmitted) to the next node on the way to their destination.

1. Message Services

The remaining packets of the data block can travel through the network using one of two techniques. These techniques or services are usually selected depending upon the nature of the traffic, which may be computer file transfers, interactive terminal sessions or even digitized voice conversations. The first is the pure packet-switching or datagram technique, and the second is the packetized virtual circuit technique.

Using datagrams, each packet is routed independently through the network. The packets making up a particular

data block could each take different routes through the network and thus arrive at the destination node out of sequence. Hence, a resequencing mechanism is required at the destination. Packets necessarily contain complete addressing information since they move independently through the network.

In contrast, a packetized virtual circuit is a circuit-switching-like approach. Information flow is in packetized form and virtual circuits carry the traffic. Packets belonging to the same data block will travel the same route through the network. This route (i.e the virtual circuit) is created in the following manner.

The first packet of a particular data block acts as a "trailblazer" which establishes the route as it passes through each node. Nodes maintain a "circuit table" so that all subsequent packets will be routed in the same way. Once a circuit is established, subsequent packets need only contain simple header information which identifies them with the initial packet. Connections can be created with virtual circuits which will remain until broken down at the termination of the session.

In comparing datagrams and virtual circuits some summarizing points are given:

1. Virtual circuit packets are shorter than datagram packets because of the reduced addressing information required.
2. Datagrams require nodes to possess a resequencing protocol for packets arriving out of their original order.
3. Virtual circuits tend to route packets less efficiently than datagrams. Subsequent packets in a virtual circuit may not be taking the best path if network conditions change while the circuit is in operation. On the other hand, datagram packets would be routed according to the current routing procedure at each node and thus their paths are not constrained by an outdated decision.

2. Application with Voice Traffic

Packet-switching was originally proposed because it was recognized that computers were very inefficient users of the line-switching techniques which had worked quite well for human speech in the telephone system. Rather than try to make a computer "speak" like a man so that it could use

the telephone, a new switching technique was developed. The success of packet-switching and its obvious benefits have led to research efforts in developing methods by which voice traffic could be transmitted using this technique.

The difficulty in this implementation does not lie in making the voice packets. A number of methods are used which convert analog speech to a digital waveform. Data rates as low as 2.4 to 4.8 kbits/sec have been achieved using vocoders with linear predictive coding. The sampled voice data is divided into discrete packets with packet length corresponding to an interval of the analog signal.

The fundamental problem which plagues packetized voice networks relates to the nature of speech, which requires both high throughput and low delay, a feature which is not consistant with the capabilities of most packet networks. In order that the quality of speech be preserved, voice traffic in a packet-switched network must maintain continuity. Voice gaps caused by end-to-end delays are generally undesireble. Traditional line-switched networks exhibit fixed delays to voice traffic and thus the relative speech timing was preserved. However, the variable delays due to the possible queueing of voice packets can cause loss in intelligibility.

In order to achieve regularity in voice traffic a variety of schemes have already been investigated. The use of both datagrams and virtual circuits have been suggested and the merits of each have been cited by Gruber [Ref. 2]. Datagrams, it is pointed out, provide the shortest end-to-end delay and thus voice continuity would be more readily achieved. Also, due to the redundant nature of speech, the effects of lost or damaged packets would not have the catastrophic effect which accompanies similar occurrences in computer data networks. Virtual circuits, on the other hand, do not have the resequencing delays associated with datagram service. Additionally, the shorter packet length would cause less congestion than the larger datagram packet caused.

The issue of how best to implement voice traffic in a packet-switched network is by no means resolved. The goal of reducing delay, however, continues to be germane to all packet-switched networks, and in the case of voice traffic may be the most important issue. Work aimed at developing routing techniques which are applicable to these networks continues to be relevant, if not critical, to future success.

C. NETWORK ARCHITECTURE

As stated earlier, this thesis involves the development and application of a routing procedure for a packet-switched distributed communications network. The function of selecting which outgoing link a packet will take is one of the many operations that must be performed in order for the network to provide effective service to its users. The complexity of the relationships between these functions can make it difficult to understand how routing procedures fit into the total network architecture. This task can be greatly simplified by using a conceptual model proposed by the International Standards Organization (ISO) for communications networks.

The ISO model was developed as the first step in standardization of the various protocols which exist in these networks. The model provides a basis for categorizing the many complicated network operations into "layers" so that rules and procedures belonging to a particular layer perform a well defined service. The layers are organized in a hierarchical structure where each one is built upon its predecessor. The impact of using this layer model is that higher layers need not be concerned with the details of how

the services offered by the lower layers are actually implemented.

The seven layer model arrived at by the ISO is presented below with a summary of the distinct function which is performed at each layer. Terminology used is from Tannenbaum [Ref. 3].

1. Physical layer - The transmission of data bits over a channel which involves the mechanical, electrical and procedural interfacing of the network.
2. Data Link layer - The transformation of a raw transmission facility into a line that appears free of errors to the Network layer. Protocols employed break up the input data into data frames, transmit the frames sequentially, and process the acknowledgement frames sent back from the receiver.
3. Network layer - The routing of packets within the network. Additional protocols divide messages from the Transport layer into packets and provide measures for control of congestion.
4. Transport layer - The acceptance of data from the Session layer which is then passed to the Network layer. Protocols ensure that data arrives correctly

at the other end. The complexity of this layer is therefore highly dependent upon whether datagram or virtual circuits are used in the Network layer. Additional protocols handle hierarchical addressing schemes and the multiplexing of a number of connections.

5. Session layer - The establishment of a connection between two processes and the management of the dialog in an orderly manner.

6. Presentation layer - The performance of functions that are requested sufficiently often to warrant finding a general solution for them, rather than letting each user solve the problem. Protocols include encryption routines for message security, character conversion and text compression.

7. Application layer - The content of this layer is up to the individual user. Since the contents of the application layer are developed by the network user vice the network designer (as with the lower layers), few national or international standard protocols exist. Nevertheless, current research dealing with issues which are common to many applications may be

the future source of accepted protocols. These topics include distributed data bases and distributed computation.

As the layer level increases, so does its relative level of abstraction. The lowest three layers (Physical, Data Link, and Network) deal with the actual transportation of data within the network itself and are often refered to as subnet layers. The functions of the remaining four higher layers can be compared to those of a virtual machine system. These layers provide the user with a connection to their destination which appears dedicated to them alone. In other words, though many users may be simultaneously using the shared resources of the network, their connections will remain isolated from each other through the functions of the subnet layers.

In this model, the routing procedures are a part of the Network layer. Numerous different algorithms have been proposed in the literature which claim to find the "best paths" within a network. However, as will be shown in the next chapter, not all are well suited for application within a distributed packet-switched network. In providing the general characteristics of the wide variety of routing

algorithms, the rationals behind the selection of the specific procedure used in this work should become apparent.

II. ROUTING PROCEDURES

A routing procedure for a packet-switched communications network prcvides a set of rules which guide packets from their scurce to destination nodes. This routing procedure cr algorithm is then that part of the computer software responsible for deciding on which output line an inccming packet should be trarsmitted. The resulting path a packet takes is defined simply as the collection of sequential communicatien links ultimately connecting source to destinatien.

Befcre discussing the varicus routing procedures used to select these paths there are certain properties that are generally desirable in all such algorithms: correctness, simplicity, robustness, stability, fairness and optimality [Ref. 3]. Correct performance of the algorithm must be established prior to use and a simple structure aids in both implementation and speed of execution. A robust algorithm can cope with changes in network topology and traffic withcut requiring majcr modifications to take place in its structure or use. Stability refers to a routing algorithm's ability tc converge to a state of equilibrium. When an

unchanging network has staticnary traffic inputs then the algorithm should converge to a stable traffic pattern. An algorithm which never converges, no matter how long it runs, is unstable. Fairness and optimality are certainly desirable, yet often contradictory goals. This contradiction is best seen by example. Suppose two high volume traffic users share a single communications link while transmitting to another party. Due to their combined traffic, the link is used to full capacity and thus exhibits maximum traffic throughput. If the criterion for optimization is maximum traffic flow, then the situation is optimal. However, if a lower volume user requests to send traffic over that link, it will be denied since the traffic flow would be reduced if the user were allowed to use the link. Evidently, some compromise between global efficiency and fairness to individual connections is needed.

A. LINK WEIGHT FUNCTION

Most routing algorithms used in packet networks turn out to be variants, in one form or another, of shortest path algorithms. These procedures achieve optimality by routing packets from source to destination over a path of "least cost". In establishing a criterion function the designer

assigns a positive cost to each link which a packet incurs when it travels over that link. These "link weights" are evaluated based upon a specific commodity which the designer is seeking to minimize. The link weight function, then, assigns a numeric value to all links which the routing algorithm uses in its computation of the least cost or best paths of travel throughout the network.

The parameters upon which the link weight is dependent are set by choice of the designer. Some networks use a fixed weight for each link in the network, the weight being inversely proportional to the link capacity in bits per second. For a network with equal capacity links, minimization of the cost produces a least "hop" routing scheme, a hop being a trip over a single link. Links with a high error rate or poor signal to noise characteristics may be given higher weights, steering traffic away from them. Weights may be assigned depending upon the type of traffic transmitted with a higher weight given to computer file transfers than to interactive terminal usage. In attempting to minimize average packet delay, link weights could be based upon estimates of average packet delay times over that link.

B. ROUTING CLASSES

In choosing the link weight function, the designer makes a crucial decision concerning the performance of the network. If link weights are not a function of parameters involving estimates of current traffic and topology, the algorithm will be deterministic. However, if link weights are a function of these parameters then the algorithm will be adaptive. A third class, called hybrid algorithms, use link weight functions which behave both deterministically and adaptively depending upon network conditions.

1. Deterministic Routing

Deterministic routing (i.e. static, non-adaptive, directory routing) uses fixed routing tables which were computed with unchanging link weights. With deterministic strategies, the path for any source-destination node pair is determined a priori and is independent of normal traffic variations. This class of algorithms do not adapt to changes in network traffic but are designed to provide satisfactory performance, on the average, over a range of traffic intensities.

A simple example of a deterministic routing algorithm is the least hop scheme in which all link weights are

equal. This method could be selected when the traffic pattern is balanced, i.e. the link utilization of different links is approximately the same. Under such conditions, the average delay of a packet will be minimized.

A more complex, though still deterministic algorithm, is to expand the routing table to include alternate paths. Therefore, for each possible destination, the node would have a first, second, third, etc. outgoing link whose selection would be based on a predetermined plan. A better basis of assignment is to have a specified fraction of entering packets corresponding to a particular source-destination pair be routed over each outgoing link.

The selection of these "routing fractions" can be made in an ad hoc fashion. However, if the incoming traffic statistics for the network are known then better, although more complicated, iterative approaches can be used. The determination of the routing policy can be based upon minimizing the average source-to-destination packet delay. This type of minimization problem is a convex multicommodity problem on a convex constraint set. The important feature of this type of minimization problem is that the solution is a unique local minimum, which is also the global minimum.

This means that the solution will be the optimal routing solution for the given static situation. A number of optimal techniques for the sclution of multicommodity flow problems are found in the literature which have a wide range cf ccmputational efficiencies [Ref. 4].

Ancther type cf deterministic approach is called "flooding". Each node receiving a packet simple retransmits it over all outgoing links. This technique is simple and robust. Flooding always chocses the shortest path, because it chocses every path in parallel. The obvious drawback is the network being cverwhelmed with multiple copies cf a message.

The advantage of deterministic routing is its simplicity of implementation, since little, if any, dynamic calculatien or signaling information is required. Important too is the fact that these methods provide good, if not cptimal, performance under conditions for which they were designed. The main drawback cf non-adaptive schemes is just that, they do not adapt. Static routing is often designed based upon mean traffic patterns, but for chaotic and unbalanced traffic environments, they are not adequate [Ref. 5].

2. Adaptive Routing

Adaptive or dynamic routing algorithms utilize link weight functions containing parameters which are measurements or estimates of the current traffic or topology of a network. In general, adaptive routing procedures can usually be differentiated by two criteria; how dynamic they are and where the algorithm is implemented.

The dynamic feature of an algorithm concerns how rapidly and in what manner it adapts to changes in the network. The best, although nonrealizable, algorithm would be one where instantaneous and global knowledge of the network status was available upon which routing decisions were made. This "ideal observer" would always route traffic in the optimal fashion. Since this is not possible, adaptive algorithms depend upon making estimates of traffic conditions, reflecting these estimates using link weights, and then computing new routes. While adaptive routing strategies appear more attractive than deterministic ones, the overhead traffic caused by routing updates and the practical difficulty involved in accurately reflecting the current traffic conditions can make the choice unclear.

a. Centralized Routing

Under centralized control, a central node in the network receives status information from all other nodes, processes this information, and then computes the new best path routing table based upon this global knowledge. The attractivenes of this scheme lies in the ability of the central node to make routing decisions based upon global knowledge and the alleviation of routing computation by other nodes.

However the drawbacks are serious, if not fatal. Putting "all its eggs in one basket", this system is hardly robust, with failure of the central node being catastrophic for the network. There are also problems related to a heavy concentration of routing traffic near the central node. Finally there exists the more basic "chicken and egg" predicament where one needs routes to transmit status information required to establish routes.

Numerous centralized algorithms can be found in the literature. An algorithm proposed by Dijkstra [Ref. 6] deserves mention due to its popularity. Because of its computational efficiency, it is used as a basis of comparison in the next chapter. The Dijkstra algorithm uses a

"tree growing" technique in which the source node is the base of the tree. Through an iterative process, nodes which constitute the shortest path from the source to the destination receive a "permanent" label. This label contains the identity of the node which is the next hop to the source node. The algorithm, in its basic form, terminates when the destination node is permanently labeled. If it is desired to find the shortest paths from the source to all nodes within the network, then the algorithm is simply repeated until all nodes are permanently labeled. To complete a centralized routing table for the entire network, a more computationally efficient matrix implementation of the algorithm is available.

b. Isolated Routing

On the opposite end of the spectrum from centralized algorithms are procedures in which nodes make routing decisions based only on information they themselves have gleaned. With isolated routing, nodes exchange no routing information with other nodes. These methods still, however, try to adapt to changes in topology and traffic.

A simple example of an isolated scheme is the "hot potato" algorithm. When a packet arrives, the node

tries to get rid of it as fast as possible by putting it on the queue of the output link which is shortest.

The advantage of these algorithms is the absence of update traffic while the disadvantages are related to much less than optimal path selection due to limited knowledge. This brings out the ongoing battle between the benefits of up-to-date status information for decision making and the burden this places on the network in the form of high overhead traffic.

c. Distributed Routing

With distributed algorithms, each node constructs its own routing table using periodic updating information from neighboring nodes. The algorithms are distributed in the sense also that no central routing tables are required and there is no global knowledge of the topology of the routes. Each node knows only its adjacent nodes and from amongst them, chooses a "preferred" neighbor for each destination node.

Distributed algorithms have many features which make them attractive for packet-switched communications networks. They are robust due to their lack of dependence upon a central node for routing information. They also

possess the ability to react quickly to local disturbances at the point of disturbance with slower "fine tuning" in the rest of the network.

The routing algorithm used in the ARPANET since May 1979 employs a modified version of the Dijkstra algorithm in a distributed manner [Ref. 7]. Each node independently computes its own routing table using this Dijkstra modification which is called the "shortest-path-first" algorithm. The modifications allow the algorithm to handle various possible changes in link wieghts without having to recalculate the whole spanning tree.

Link weights are evaluated in terms of time delays on the link. Each node calculates an estimate of the delay on each of its outbound links by averaging the total packet delay over a ten second interval. Since all nodes must be informed of link weight changes, a flooding technique is used to forward the information throughout the network. To reduce overhead, a link weight update is transmitted only when the change in weight since the last transmission has exceeded a certain threshold. Upon receipt of a new link weight update, the shortest-path-first algorithm restructures its shortest path tree.

Gallager [Ref. 8] proposed a distributed routing algorithm which makes use of the routing fraction concept discussed earlier in the deterministic routing section of this chapter. Unlike the static scheme, these routing fractions are functions of information communicated between adjacent nodes. The information in the update message is related to the marginal delay to each destination. The algorithm sends packets over routes which minimize the overall delay of all messages, vice attempting to send packets over a route that minimizes their own delay with no regard to that of other packets. The basic notion which the algorithm exploits is that a change in link flow causes a marginal change in network delay.

Gallager demonstrates that the algorithm converges to the minimum average delay for a network with staticnary inputs and links. The question as to whether the algorithm could adjust fast enough to keep up with changing statistics is still the subject of study. Faster network changes would necessitate more frequent updating of the algorithm. Besides the usual problems of increased overhead, frequent updates would also cause noisier measurements of marginal link delays and node flows.

Another distributed algorithm is presented by Segall and Merlin [Ref. 9] which uses the spanning tree concept of graph theory. A spanning tree is a connected partial graph (all nodes and a subset of links) containing no loops. A normal update cycle would begin from the destination node at the base of the tree. The cycle would propagate up-tree (on the spanning tree), modifying the weight estimates from each node to the destination node and then propagate down-tree while updating the best path neighbors for each node. Each cycle tends to find the best path from each node to the destination node. As with Gallager's algorithm, the paths are also loop free at every iteration.

In comparing the last two distributed algorithms, there is a fundamental difference in how they seek to minimize their cost functions. Gallagher's algorithm produces a set of routing fractions which tell the node what percentage of traffic for a particular destination is to be placed on each of its outgoing links. To dramatize the point assume 100 packets destined for node J arrive at node I. The current routing fractions state that 50% of all traffic for node J is to travel on link (I,A), 30% on link (I,B) and 20% on link (I,C). Node I would then place 50

packets in link queue (I,A), 30 packets in link queue (I,B) and 20 packets in link queue (I,C).

Segall's algorithm, on the other hand, produces a routing table which lists a "preferred" neighbor for each destination. If the same situation had occurred using this algorithm then all 100 packets would have been placed into one link queue. It would appear in the case of algorithms providing a single path routing table, that this does not mean that at any given instant all traffic arriving at a node destined for another should be sent via the single path. The policy of only incrementally increasing the fraction of traffic routed to the preferred neighbor, while (incrementally) easing the load to other routes seems more suitable. In this way, the best estimated route becomes somewhat more loaded, whereas the others become less loaded, thereby improving network performance.

Another distributed routing algorithm which operates on a substantially different principle was written by Yen [Ref. 10]. That principle, and the other characteristics of the Yen algorithm are investigated in the following chapters.

3. Hybrid Routing

Hybrid routing is a strategy based upon the hypothesis that a combination of deterministic and adaptive schemes will probably achieve the best results. Chow [Ref. 5] states that for the routing procedures which have been implemented, adaptive routing procedures are good for certain traffic conditions and deterministic ones for others. Chow concludes that a hybrid algorithm must behave deterministically when traffic so warrants and adaptively otherwise.

Hybrid algorithms must utilize a link weight function which is able to produce nonvarying weights when deterministic network conditions prevail and highly variable weights when adaptive network conditions exist. This link weight function, then, must not be sensitive to small variations in the traffic. That is, if there are only incremental variations and thus small fluctuations in path weights of possible routes, then the current route does not change. This prevents detrimental oscillations or "churning" between closely weighted paths which can result in unnecessary packet delays.

The ARPANET is an operational packet-switched network which implements a hybrid scheme. Since 1979 this network has used a link weight function [Ref. 11] which can be generalized into

$$W(I,J) = b + cQ(I,J)$$

where $Q(I,J)$ is the link queue size at the time of routing update and $b > c$. In this scheme when traffic is reasonably balanced, the constant b determines a fixed path for each source-destination pair and the routing strategy behaves like a deterministic one. Otherwise, the difference in queue buildup of alternate paths becomes substantial, the impact of the $cQ(I,J)$ term increases, and the routing becomes adaptive. Chow considers a third term, $dQ^2(I,J)$, such that the link weight function is

$$W(I,J) = b + cQ(I,J) + dQ^2(I,J)$$

and $c > d$. His simulation verified that this link weight function provides a good compromise. The actual values of b, c and d and the frequency of updates depended upon the network topology and traffic condtions.

The development of a hybrid routing system based upon the Yen distributed routing algorithm and a newly

devised link weight function is presented in the following chapters.

C. CONGESTION CONTROL

In presenting the functions of the Network layer, an additional protocol besides routing was mentioned, namely congestion control. Congestion control was introduced as a separate area, but as one can surmise from the material in this chapter, both these topics are intimately related. To see their relationship, the characteristics of congestion must first be understood.

Congestion can be described as the state of a network where more offered traffic results in less carried traffic. Congestion is not the graceful approach to the network's maximum throughput but rather the deterioration away from this value as traffic intensity increases beyond some point. This point can occur when network resources are no longer able to accomodate the traffic. Whether congestion begins simultaneously throughout the network or in a particular region, if left unchecked it tends to feed upon itself and become worse. This can lead to such a degraded condition that no transmission can take place at all. When deadlock or lockup happens, the network ceases to function and will stay that way unless special procedures are invoked.

The relationship between the congestion process and the available network resources will be illustrated beginning with the use of an "ideal" packet-switched network. Ideal means that each node has infinite buffer capacity, each microprocessor has a perfect routing algorithm, and each link has infinite capacity. Within this network, since packets travel instantaneously from node to node over the best path, congestion cannot occur. Offered traffic can never exceed the limits of a network with unlimited capacity.

Suppose now that the links are made more realistic and given finite capacity with attendent propagation delays. Nothing else in the network is changed. In this case, since the routing algorithm is perfect, incoming traffic will be distributed within the network such that the limiting resource (the link capacity) is efficiently used. However, as the intensity increases, the finite link capacity will eventually reach saturation resulting in queue build up and packet delays. With a perfect routing algorithm, when the network limit is reached, several nodes or links will become blocked at the same time. Had the routing algorithm been less than perfect, the congestion would have started at a

smaller traffic intensity since the network resources would be less skillfully allocated.

The final touch of realism for the network is the use of finite capacity queues. Under this limitation, all packets arriving at a node for which there is no queue space are simply discarded. Congestion would not occur in this network, but the process of packet discarding (although used in some networks) seems a drastic solution to the problem.

From this scenario, the role of routing is seen as congestion "avoidance" vice congestion "elimination". Even perfect routing can not stop congestion when the network traffic limit is exceeded. Although good routing protocols will increase the load that a network can take, it follows that the elimination of congestion must start prior to packets entering the network. Once a packet has been accepted into the network, the best that can be done is to provide routes so as to avoid congestion. But if too many packets are allowed into the network, congestion is inevitable.

The methods of congestion control presented in the following paragraphs are those which act prior to a packet entering the network. The "isarithmic" method by Davies

[Ref. 12] is a technique which places a limit on the total number of packets in the network. The network contains permits which circulate about. A packet requires a permit to travel in the network and upon reaching the destination, the permit is recirculated. This simple rule ensures that the number of packets in the network will not exceed the number of permits.

If virtual circuits are used in a packet-switched network, a simple congestion control scheme is to allow each virtual circuit to reserve buffer space at each node along its path. If buffer space at any node is not available, the caller gets a "busy" signal and the circuit is not completed. This preallocation of buffers, however, results in some inefficiency since resources assigned but not actually being used in connections are not available to anyone else.

Another method proposed by Majithia [Ref. 13] uses a mechanism which is triggered only when the system is congested. The "warning" of impending congestion is transmitted throughout the network by means of choke packets. The choke packets act as a negative feedback source which is then used by nodes to throttle incoming traffic.

The mechanism known as flow control is often included in the set of solutions for the problem of congestion control in packet-switched networks. The term is used throughout the literature and not without some confusion. In understanding the possible applications of flow control, it is first necessary to clarify what flow control is and how it differs fundamentally from congestion control.

Most authors use flow control to mean the mechanism by which a receiver throttles a sender to prevent data from arriving at a rate faster than the receiver can handle it. Flow control seeks to allocate network resources for user-receiver pairs as long as they are available. Usually the implementation of flow control is at the Transport layer between source and destination nodes (end-to-end flow control) but sometimes the term is applied to protocols between neighboring nodes.

Numerous flow control schemes are presented in the literature although usually they involve the implementation of one of only a limited number of throttling tools. Pouzin [Ref. 14] summarizes them as being in the following categories:

1. Stop and Go - The source either can send traffic without limit, or it is barred from transmitting.
2. Credit - The sender cannot transmit unless it has received from the receiver an indication about the amount of traffic that can be accepted.
3. Rate - The sender cannot transmit traffic above a predetermined rate.
4. Delay - As delay increases between source and destination, the sender is throttled.
5. Class - Traffic is offered with a class indicator. When enough resources are available, all offered traffic is accepted. Otherwise, some classes are restricted depending upon criteria which appears to strike an acceptable balance between conflicting user demands.

The flow control schemes using Credit and Delay appear particularly suited for controlling congestion. Situations can occur where many nodes within a network are transmitting to the same destination node. Even if their individual transmission rates are low, the destination node may not be able to handle the combined traffic load. Both the credit and delay schemes would be able to sense the beginnings of

congestion through the termination of credits from the receiver or the increase in packet delay. Traffic to the destination node could then be throtteled to avoid congestion.

In contrast, the Stop and Go and Rate schemes may not be as effective in controlling congestion. These flow control schemes require the sender to stop sending at some point and wait for an explicit go-ahead message, or permit the receiver to simply discard packets at will with impunity. The ability to select the cut-off "point" for the protocol which satisfies a total network requirement is the real problem. If the point is determined based upon average network operating levels then these particular flow control schemes will provide poor service for bursty traffic. On the other hand, if peak levels are used then they end up providing no control at all.

III. THE YEN SHORTEST PATH ALGORITHM

With the abundance of shortest path algorithms in the literature [Ref. 15], there was a need to develop criterion for selection of one method over another. Immediately, all centralized shortest path algorithms were eliminated due to the goal of creating a truly decentralized communications system. This reduced the number of choices by a substantial amount. The following characteristics were then sought after in the remaining decentralized shortest path algorithms.

1. Algorithms should not require knowledge of the complete network topology.
2. Algorithms should be efficient (i.e. require as few as possible computational steps for the node's microprocessor).
3. Information exchanged between network nodes required for algorithm to function should be a minimum.

Based upon these criterion, the Yen shortest path algorithm was chosen [Ref. 10]. Both Gallagher's and Segall's algorithms satisfied condition (1), however Gallagher's was not as computationally efficient and Segall's required more

exchange cf information (the dual nature of its spanning tree update cycle) between nodes than the algorithm proposed by Yen. Not only does the Yen algorithm satisfy the desired requirements, but it also exhibits unique features which were cf particular interest for this research. In addition, and perhaps foremost, was the fact that this algorthim has not been thoroughly studied in actual network simulations.

A. DESCRIPTION

The key word in understanding and analyzing the Yen shortest path algorithm is "time". The manner in which the algorithm makes use cf time in arriving at the shortest path between nodes of a network is quite unique. This was the cnly algorithm reviewed by this author that even approached the problem in this fashion. The necessary assumptions for applying the algcrithm are as follows:

1. Each node in the network is equipped with transmission and computation facilities and a timing device called a clock.
2. Each node J knows a set of nodes, called Neighbor(in) ncdes, each cf which is connected to a node J by a directed link from the Neighbor(in) node to ncde J.

3. Each node J knows a set of nodes, called Neighbor(out) nodes, each of which is connected to node J by a directed link leading from node J to the Neighbor(out) node.

4. Each node J knows the weight of each of the links connecting node J to the Neighbor(out) nodes. Weight, again refers to the assigned cost of using that link.

5. Each node J maintains a list consisting of certain Neighbor(out) nodes, called an Update Transmission List, which are to receive the transmission of update message "J".

For most data communication networks, there will be a full duplex link between node pairs so that the set of Neighbor(in) and Neighbor(out) nodes would be identical for a given node. To simplify the discription to follow, Neighbor will refer to a terminating node of a full duplex link. The clock is the only item which is peculiar to the application of this algorithm. However, since each microprocessor has a clock associated with it, no additional hardware is required.

The algorithm for finding the shortest paths from all nodes to a destination node K is contained in Appendix A. The activity of the algorithm begins with node K transmitting the update message "K" to all of its Neighbors at the same time. The actions which a node takes upon reception of an update are broken down into five parts within Step 2 of the algorithm. These actions are summarized as follows:

1. Revise the Update Transmission List (Step 2.A). This list contains those Neighbors which are to receive the update sent from this node. Each time a node receives an update from a Neighbor, that Neighbor is removed from the Update Transmission List. This process prevents update message "looping".
2. Calculate the Update Reception Weight (Step 2.B). The basic operation of the algorithm is the mapping of the time of reception of an update message into an Update Reception Weight. The relationship between the time of reception and the Update Reception Weight is a direct proportion (using the constant C) such that updates arriving at later times will result in larger Reception Weights.

3. Determine the Tentative Shortest Path Distance (Step 2.C). The Update Reception Weight is added to the weight of the link to the node from which the update message had been sent. If node 2 received an update from node 4 this "reverse link weight" would be W(2,4). The total of the two weights is the shortest path distance to the destination node via the reverse link. To provide a degree of differentiation, the term "distance" will be used in reference to the total weight of a path or route consisting of one or more links. This path distance is then compared to the previously computed shortest path distance with the smaller of the two becoming the new tentative shortest path.

4. Schedule the Tentative Time of Update Transmission (Step 2.D). The tentative shortest path distance is mapped into a corresponding time value. When this time occurs, the update message will be sent to all Neighbors in the Update Transmission List.

5. Transmit the Update (Step 2.E). At the moment the time of update transmission occurs, the message is sent to all nodes in the Update Transmission List.

Tc aid in understanding the operation of the algorithm a simple example is given. Figure 3.1 presents a fcur ncde network which is used to demonstrate the algorithm in its tasic form. The numbers near the beginning of each link are

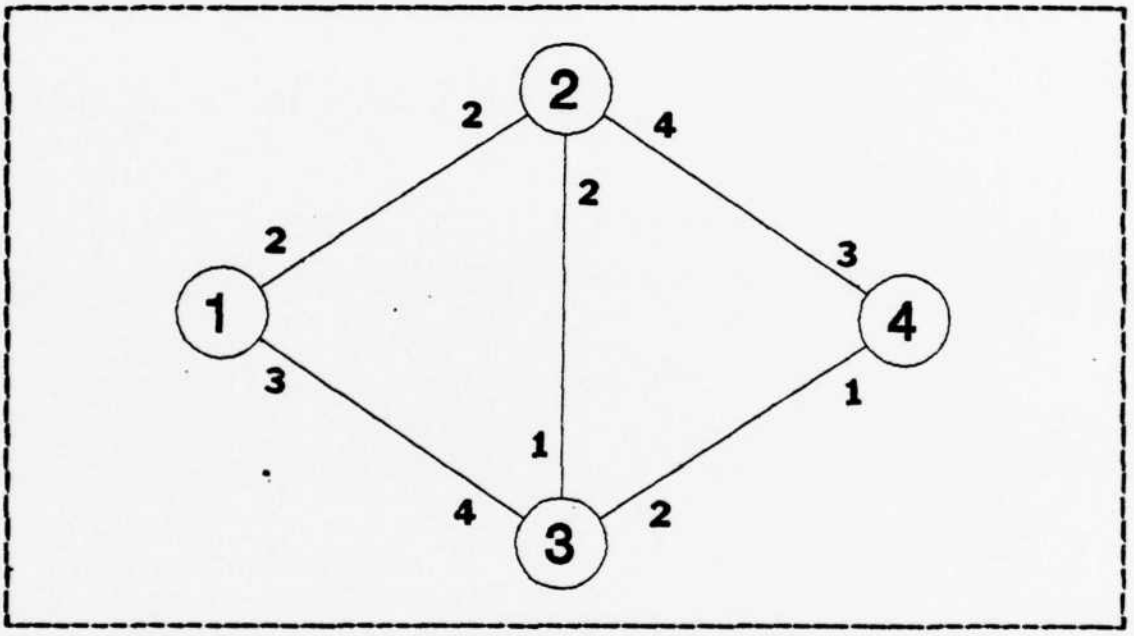


Figure 3.1 Algorithm Demonstration Network

their weights, W(I,J). Before proceeding with this example it must be pointed out that the Yen algorithm in its tasic form cannot be applied to an actual network. The propagation and processing delays present in real systems must be taken into account and thus some modifications have to be made. Also when the algcrithm is applied to find

simultaneously the shortest paths to many destination nodes, the simple update message presented in the algorithm must be changed to include additional identifying information. Both of these modifications were performed prior to implementation into the simulation but are not essential in understanding the fundamentals of the algorithm.

In this example, node 1 will be the destination node. At the conclusion of the algorithm all nodes in the network will know the next node along the best path to take to node 1. The algorithm begins by initializing all tentative shortest path distances and their corresponding time value to infinity. The Update Transmission List of each node initially contains all the node's Neighbors. The constant C which relates path distance to time is set to unity for ease in understanding the operation of the algorithm.

At time 0, node 1 sends its Neighbor nodes 2 and 3 the simple message "1". Since no propagation delay is assumed, nodes 2 and 3 both receive the message "1" at the same time 0. Being a distributed system, both nodes begin performing the algorithm upon receipt of the update message. The activity at node 2 is first observed. A descriptive format is utilized in presenting the algorithm steps which are

performed at each ncde. The algorithm step label from Appendix A is included with each corresponding action.

Step 2.A; Node 2 deletes node 1 from its Update Transmission List. This eliminates the possiblilty of sending an update to a node from which one has already been received.

Step 2.B; The clock is read and the time of reception cf the update from rcde 1 over the link (1,2) is ncted. This time is then mapped into a Reception Weight. Since the time of reception was 0 and the mapping multiplier, C, equalled 1, the value cf the Update Reception Weight was 0.

Step 2.C; The Update Reception Weight is added tc the actual weight of the reverse link (2,1) resulting in the path distance. This path distance is compared to the previous path distance (which was initially infinite) with the smaller of the twc being selected. The tentative shortest path distance is then 0 + 2 = 2.

Step 2.D,E; The path distance is then mapped into a time value of 2. Node 2 will now tentatively schedule the transmission cf the update message "2" at time 2 to all nodes in its Update Transmissicn List. The message number indicates the transmitting nodes identity.

Node 2 completes this action at time 0 since no processing time is assumed. At node 3 a similar process takes place.

Step 2.A; Node 3 deletes node 1 from its Update Transmission List.

Step 2.B; The Update Reception Weight is computed to be 0 since the time of reception was 0.

Step 2.C; The Reception Weight is added to the reverse link weight with the result of 0 + 4 = 4 for the path distance.

Step 2.D,E; Node 3 schedules the tentative transmission of the message "3" to nodes 2 and 4 at time 4.

The network is then "quiet" until time 2 whereupon node 2 sends message "2" to nodes 3 and 4. Upon receipt of the message at time 2, node 4 performs the following:

Step 2.A; Node 4 deletes node 2 from its Update Transmission List.

Step 2.B; The Update Reception Weight of the message is computed to be 2 since the time of reception was 2.

Step 2.C; This weight is added to the reverse link weight with the result of 2 + 3 = 5 for the path distance.

Step 2.D; Node 4 tentatively schedules the transmission of message "4" tc node 3 at time 5.

At node 3 the following action is occurring:

Step 2.A; Node 3 deletes node 2 from its Update Transmission List leaving only node 4.

Step 2.B; The Update Reception Weight of the message is computed to be 2 since the time of reception was 2.

Step 2.C; The Reception Weight is added to this link weight with the result of 2 + 1 = 3 being the path distance. This path distance is then ccmpared with the previous value of 4. Since this path distance is shorter, the new tentative path distance becomes 3.

Step 2.D,E; Node 3 then reschedules the transmission of the message "3" to occur at time 3 when it will be sent to node 4.

Again the clock ticks until time 3 at which point ncde 3 sends Neighbor node 4 the message "3". Node 4 then takes the fcllcwing action:

Step 2.A; Node 4 deletes node 3 from its now empty Update Transmission List.

Step 2.B; The Update Reception Weight is computed to be 3 since the time of reception was 3.

Step 2.C; This Reception Weight is added to the reverse link weight resulting in 3 + 1 = 4 which is less than the previcus value of 5. The path distance is then updated to the new value of 4.

Since the Update Transmission List of node 4 is empty, additional update transmissions are not needed. At termination of the algorithm, the solution to the shortest path to node 1 for each node is:

Node 2: Shortest path (distance = 2) was via node 1.

Node 3: Shortest path (distance = 3) was via node 2.

Node 4: Shortest path (distance = 4) was via node 3.

From inspection of this simple network, the solution calculated by the Yen algorithm is seen to be correct. It is now shown that the algorithm arrives at a set of optimal shortest path distances from nodes J to the destination node K for any connected network.

During the course of the algorithm's operation at some time t, there exists a set of tentative path distances, F(J,K), which have corresponding time values, T[F(J,K)], greater than t. These path distances represent the tentative shortest paths from nodes J to destination node K using

the best paths computed up to that time. As time passes, the smallest of the tentative F(J,K)'s, say F(J*,K) will become permanently labeled because at time t = T[F(J*,K)] it becomes apparent that there is no other path from node J* to node K that has a shorter distance than F(J*,K). On one hand, the tentative F(J,K)'s become permanently labeled as time passes; and, on the other hand, whenever a F(J,K) becomes permanently labeled it is used to update other tentative F(J,K)'s. Therefore, at the termination of the algorithm, the F(J,K)'s thus obtained are the distances of the optimal shortest paths from nodes J to the destination node K.

A useful analogy to picture the operation of the algorithm is to visualize each node as having an alarm clock. Upon reception of the first update message "K", node J sets its alarm to go off at time t = T[F(J,K)]. If, prior to the alarm clock "buzzing", node J receives another message over a different link, then it compares the current alarm setting with the newly computed time. The smallest of the two time settings is adopted, which in the analogy equates to resetting the alarm clock to the new shorter time. When time t = T[F(J,K)] finally occurs, node J "wakes up" and sends its

own message "J" to all Neighbors from which it has not yet received a message. The terminology used in this description (i.e. alarm clock, setting, buzzer, and wake up) is used throughout the simulation program to aid in visualizing the various operations performed by the nodes for the algorithm.

B. MODIFICATIONS TO THE YEN ALGORITHM

As described in Appendix A, the Yen algorithm does not compensate for time delays caused from the transmission and processing of information and is not able to find simultaneously the shortest paths to many destination nodes. Prior to application within the simulation, both of these modifications were made. The solution to the problem of time delays is presented first.

1. Time Delay

The delay associated with transmitting information over a link between nodes within a digital communication network involves three factors:

1. The length of the transmitted message.
2. The propagation delay of the transmission medium.
3. The message processing time for the receiver.

If at time 0, node J begins sending the first bit of an N bit packet message to node K then at some later time the last bit of the message will have been sent. If R is the transmission rate in bits per second of the node then this time will be

$$Tu = N * R$$

The first bit of the transmitted message will reach the receiving node after a delay in propagation given by

$$Tp = L / V$$

where L is the physical length of the link in meters and V is the velocity of propagation for the medium in meters per second. The receiver then will have the entire message at time Tu + Td. The microprocessor must then perform the timing computations which will be completed in the processing time Tm. The total delay from the time a node begins sending a message to the completion of processing by the receiving node is

$$Td = Tu + Tp + Tm$$

The basic Yen algorithm functions on the principle of mapping the weight of a link into a time delay in the

transmission of an update message. The mapping function is linear based upon the constant C. When delays are present in a network without proper compensation, link weights are, in effect, changed which can result in erroneous best path calculations. It is important to bear in mind that the weight W(I,J) associated with link (I,J) is specified by a commodity which the network designer is attempting to minimize over all possible paths. Physical transmission and processing delays may have nothing to do with the selected commodity and thus their effect upon the algorithm's operation must be removed.

The nature of the problem is illustrated using the basic network given is Figure 3.2. In this network a time delay, Td, is associated with each link which is assumed to be identical for all links in the network. If node 1 originates an update message then node 3 will eventually receive this destination update over both links (3,2) and (3,1). Node 3 will then perform two separate calculations of the algorithm resulting in two time settings related to the transmission of its own update. The time setting from the update traveling over link (1,3) is

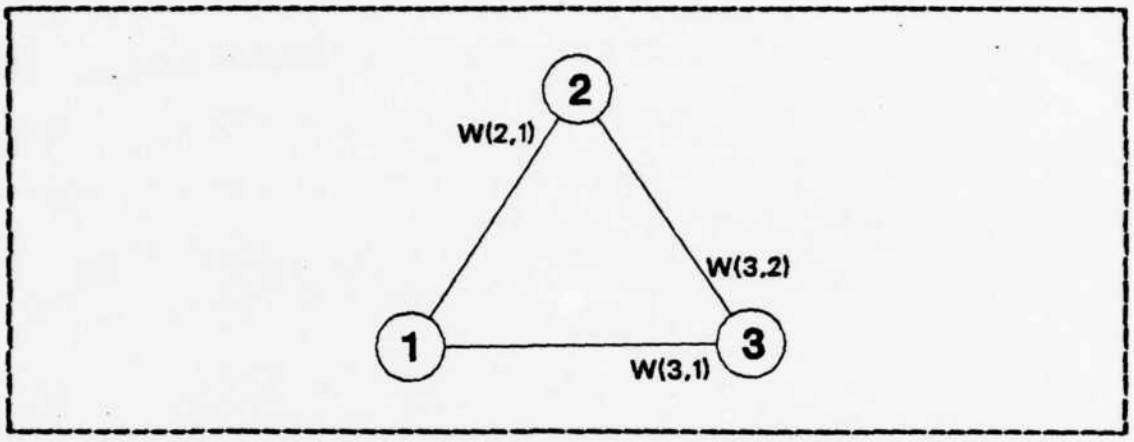


Figure 3.2 Transmission Delay Demonstration Network

$$T(3:1) = Td + \frac{W(3,1)}{C}$$

In the same way, the time setting from updates traveling over links (1,2) and (2,3) is given by

$$T(3:2:1) = 2Td + \frac{W(2,1) + W(3,2)}{C}$$

For a given selection of link weights, either path may be the best. Therefore, T(3:1) must be able to be greater than, less than or equal to T(3:2:1) based upon the link weights themselves and independant of transmission delays. The most direct means of removing these effects is to let each link weight be compensated individually such that

$$W(I,J) = W(I,J) - (Td * C)$$

This means, however, that the initial stipulation of having all link weights be greater than or equal to zero must now be modified to

$$W(I,J) \geq Td * C$$

In implementing this scheme there are two alternatives. The first is to establish link weights such that all

$$W(I,J) \geq 1$$

which would mean that

$$C = 1 / Td$$

Therefore Step 2.C of the algorithm would become

$$F(J,K) = \min[\, F(J,K) \,,\, W(J,L) + F(L,K) - 1 \,] .$$

The second alternative (which was actually used in the simulation) is to ensure that

$$W(I,J) \geq Td$$

which meant that

$$C = 1$$

Step 2.C then becomes

$$F(J,K) = \min[\, F(J,K) \,,\, W(J,L) + F(L,K) - Td \,]$$

The raising of the lower bound of allowable link weights prevents the algorithm from scheduling update transmissions in "negative" or past time.

Figure 3.3 depicts the network used to show the effects that time delays have upon the basic algorithm and

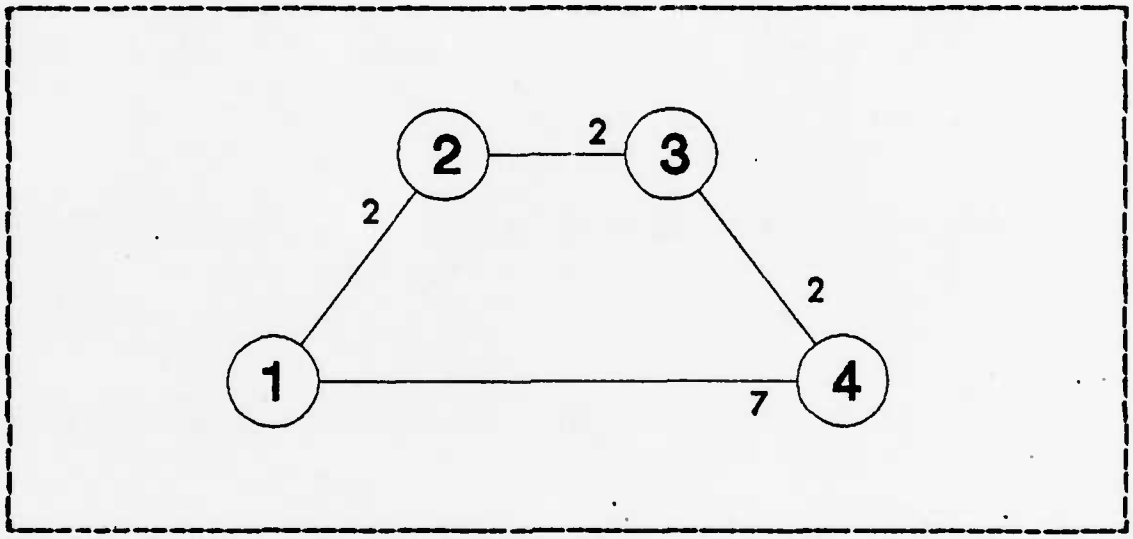


Figure 3.3 Time Delay Test Network

then the modified version. The time delay for each link is 1 second and C is set equal to 1. Link weights are labeled and the destination is node 1. The actions of both the basic and modified algorithms are summarized in Table I. By observation it is found that the best path from node 4 to node 1 is via node 3 which has a path distance of 6. Note, however, that node 4 using the unmodified algorithm arrived

TABLE I

Actions of Basic and Modified Algorithms

Time (secs)	Node Activity
	BASIC ALGORITHM
0	n1 xmts '1' to n2 and n4.
1	n2 rcvs '1', schedules xmt of '2' at time = 2 + 1 = 3.
	n4 rcvs '1', schedules xmt of '4' at time = 7 + 1 = 8.
3	n2 xmts '2' to n3.
4	n3 rcvs '2', schedules xmt of '3' at time = 2 + 4 = 6.
6	n3 xmts '3' to n4.
7	n4 rcvs '3', computes the min[8 , 2 + 7] = 8.
	MODIFIED ALGORITHM
0	n1 sends '1' to n2 and n4.
1	n2 gets '1', schedules xmt of '2' at time = 2 + 1 - 1 = 2.
	n4 gets '1', schedules xmt of '4' at time = 7 + 1 - 1 = 7.
2	n2 send '2' to n3.
3	n3 gets '2', schedules xmt of '3' at time = 2 + 3 - 1 = 4.
4	n3 sends '3' to n4.
5	n4 gets '3', computes the min[7 , 2 + 5 - 1] = 6.

at an incorrect solution. In contrast, the modified algorithm compensated for propagation delays and computed the correct path distances and best path next node at all nodes within the network.

In applying the modified algorithm, when all link weights are set to their minimum allowable values, the resulting routing table is a least hop one. This important result was used during the development of the link weight function in Chapter V. At this point, though, one can begin to see if link weights are related to traffic parameters in some way, that during periods of light activity (when weights are small) the modified algorithm will "automatically" produce least hop routing. As shown earlier, least hop routing is very effective when traffic intensity is light.

2. Multiple Destination Nodes

The second modification involves enabling the algorithm to handle simultaneously the best path calculations to many destination nodes. The simple message 'K' and 'J' must be changed to contain additional information in order to identify the destination node. One solution is that the update message contain both the originating and retransmitting nodes identity. This would uniquely identify the update message for the receiving node.

Along similar lines as multiple destination nodes is the need to distinguish consecutive updates originating from

the same node. In order to differentiate them, a sequence number can be included in the update message. The determination of the number of bits assigned to the sequence number would relate to the amount of differentiation required. If no more that M updates originating from the same node could be in transit at the same time within the network, the number of bits for the sequence number would be log M (using base two logarithm). Using this method, the sequence numbers would continuously cycle from 1 to M.

All network nodes must also have the additional capacity for keeping account of multiple update "alarm clock" sequences. Memory requirements include maintaining the clock setting and the corresponding Update Transmission List for each destination node update which is in progress. For very large networks this may be a considerable requirement. Alternate methods involving hierarchical network structures are considered later in this chapter as possible schemes to reduce these requirements.

C. COMPUTATIONAL EFFICIENCY

In the comparison of shortest path algorithms criterion which is often used for judging their performance is the calculation of upper bounds on the number of computational

steps each require. Computational steps refer to the additions, subtractions and comparisons used during the execution of the algorithm by the node's microprocessor. An additional criterion related to distributed algorithms involves the nature and frequency of information which is exchanged between nodes.

Corresponding to almost any shortest path algorithm, there exists some special network structure for which the algorithm is efficient. Dreyfus [Ref. 16] provides a comparison of the computational efficiency of many shortest path algorithms using a test network in which every pair of nodes is connected by a link. In interpreting the results it is considered that one algorithm is significantly superior to another when the computational bounds differ by the multiplicative factor involving N, the number of nodes. When computational bounds differ only by a constant then the selection between the algorithms must be based on other criterion. Dreyfus concludes that the Dijkstra centralized shortest path algorithm was most efficient (of the ones examined) requiring 1/2 N^3 additions and N^3 comparisons in calculating the shortest paths between all node pairs. The only decentralized algorithm tested was one by Abram and

Rhodes [Ref. 17] which fared much worse than Dijkstra's. The efficiency of the algorithms by Gallagher and Segall was not discussed in the references.

The Yen algorithm in determining the best paths from all nodes J to a destination node K in an N-node complete network requires as cost

$$(N-1) + (N-2) + \ldots + 1 = 1/2\ N^2 \text{ additions}$$

and the same number of comparisons to execute the N-1 iterations of Step 2.C of the algorithm. For calculating the best paths between all node pairs, the previous results are multiplied by N resulting in a requirement of $1/2\ N^3$ additions and $1/2\ N^3$ comparisons. The algorithm also requires at most $1/2\ N^3$ transmissions of the simple update messages in Step 1 and Step 2,E of the algorithm.

This result concerning the update transmissions can be used to find the upper bound on link utilization due to update message traffic in a connected N-node network. The maximum number of transmissions required to determine the best path from all nodes to a single destination node is 1/2 N. The proof can be shown using Step 2.A of the algorithm which states that after receiving a message, node J deletes the retransmitting node from its own Update Transmission

TABLE II

Computational Efficiency

Algorithm	Additions	Comparisons	Transmissions
Yen	$1/2\ N^3$	$1/2\ N^3$	$1/2\ N^3$
Abram	$1/2\ N^4$	$1/2\ N^4$	$1/2\ N^4$
Dijkstra	$1/2\ N^3$	N^3	----

List. This act eliminates the possibility of update looping upon transmission by node J. This means that for each pair of links (I,J) and (J,I) in a full duplex system, only one of them will carry an update transmission associated with a given destination node. The total number of transmissions in the network for a single destination update is then one half the number of links which for a fully connected network is approximately $1/2\ N^2$.

The results of the computational efficiencies of the three algorithms is given in Table II. Additional computational advantages of the Yen algorithm include:

1. The computational effort of the algorithm is proportional to the number of links in the network. Therefore the Yen algorithm requires fewer computations in sparse networks where there are fewer links.

2. Unlike Dijkstra's algorithm, the Yen algorithm does not have to scan for the minimum of all tentative best path distances in order to sort out the permanent best path; consequently, it saves computations.

3. Unlike the Abram and Rhodes algorithm, the Yen algorithm does not use a best path distance F(L,K) to update other tentative best path distances F(J,K) unless F(L,K) itself is permanent.

D. APPLICATION TO PACKET SWITCHED NETWORKS

In adapting the Yen algorithm to a packet-switched network, techniques must be developed to enable the algorithm to operate as "transparently" as possible within the network. Specific problems relating to burdening the node's microprocessor with routing calculations and excessive overhead from update transmissions have to be avoided. Fortunately, the computational efficiency of the Yen algorithm is very good and the only special hardware requirement is a clocking device which already exists in the microprocessor. The challenge, then, for implementing the algorithm is centered in the fundamental way in which the link weight information must propagate in the network.

For the Yen algorithm to function it is critical that the update messages be transmitted at the calculated time setting since the time of transmission is directly tied to the path distance. The capability for a node to be able to transmit an update over a link at a precise time is crucial to the algorithm's success.

Problems develop when at time t = T[F(J,K)], node J is to send update 'J' to nodes in its Update Transmission List and some links are busy transmitting message packets. Even if update packets are given highest priority and placed at the top of the link queues, the delays due to busy links may be too great.

There is another factor which concerns the desire to have updates propagate throughout the network as quickly as possible. Fast update propagation means more timely routing information upon which to base calculations. This can be accomplished by proper selection of the constant C in the algorithm. The propagation speed of updates through a network can be increased by letting C equal its maximum allowable value. The limitation on this upper value is set by the propagation delay associated with the links themselves. When C is large the time delays computed for the

transmission of update messages will be their shortest. Therefore delays due to updates waiting for message packet transmissions will result in errors with best path calculations being even more amplified. The solution to this problem lies in the configuration of the update packet itself.

To reduce the overhead from update traffic, the length of the update packet should be kept to a minimum. Fortunately, the Yen algorithm requirement on information between nodes is quite small. The update message in its basic form contains the identity of the destination and retransmitting nodes. The update also requires a preamble which identifies it as being an update packet. By selecting a specific form for the preamble, the answer to the problem of transmitting an update over a link which is busy sending a message packet is found.

1. Update Insertion Technique

While a message packet is being transmitted an update packet could be transmitted at its precise time if there was a means of inserting the update into the bit stream of the message packet. The receiving node must then be able to distinguish this inserted update from the message

packet, remove the update packet and then reassemble the message packet. Though scunding like quite a complex program for the node's microprocessor to handle, the implementaticn is straightforward.

Iet the preamble of the update packet be a specific bit pattern of length L which is called a reserved pattern or symbol. Hamming [Ref. 18] illustrates the concept of reserved symbols by the quotation marks in FORTRAN programming which are used to distinquish comments from actual operating ccde. The receiving node, knowing the reserved pattern, simple scans the incoming bit stream and upon detecting this pattern, removes these L bits and the following U bits which make up the entire update packet.

The problem is not completely solved for if by chance the reserved pattern occurs in a message not containing an update packet then the node would respond to the reserved pattern which is undesirable. The soluticn is to add an extra bit after the L bit reserved pattern with a 1 signifying an actual update packet and a 0 otherwise. The transmitting node is ncw required to observe the bit stream of outgoing message packets and when detecting the reserved pattern, insert a 0 as the following bit. If the reserved

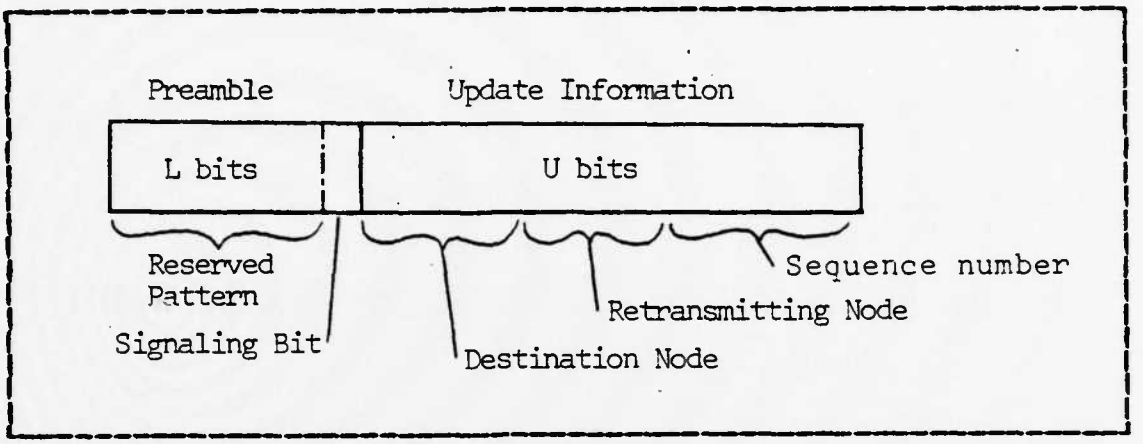


Figure 3.4 Update Packet Configuration

pattern happens to appear often in the course of message bit streams then a good deal of channel capacity will be wasted. Figure 3.4 shows how the components of the update packet are configured.

The probability of a reserved pattern occurring can be reduced by increasing its length L. However, as L increases, the total update packet length grows and more available channel capacity is used in update overhead. In examining the trade-offs involved in selecting the length of the pattern, the link utilization using the insertion technique is examined.

a. Overhead from Insertion Technique

The insertion technique "costs" L + 1 bits per update packet and 1 bit for each occurrence of the reserved

pattern in a message packet. Since the data transmission rate is known, the total link utilization due to this method is

$$UTIL[inserts] = [Nu(L + 1) + Nm(1)] / R$$

where Nu = Average number of updates transmitted per link per second.

Nm = Average number of occurrences of the reserved pattern in messages per link per second.

R = Data transmission rate (bits per second)

The upper bound on update link utilization can be directly computed from the knowledge of the number of nodes in the network and the update interval. The update interval corresponds to the average amount of time between consecutive updates being generated by a single node. If each of the N nodes in the network generates a destination update every T seconds then Nu will be

$$Nu = N / 2T$$

The next effect examined is that of the reserved pattern length upon link utilization caused from the additional signaling bit necessary whenever the pattern occurs in a message packet. In finding this, it is

necessary to determine the number of occurences, on the average, of the reserved pattern which appear in a message packet. The problem is approached by first modeling the message packet as a random binary sequence of length M. If it is assumed that the message is the same length as the reserved pattern then the probability of the pattern and the message being the same is

$$Prob[\text{pattern occurring}] = 1 / 2^{**}L$$

In the case where the message is not the same size as the reserved pattern then a binomial random variable is used with the following definitions:

- p = The probability of the reserved pattern occuring in a sequence of length L.
- n = The number of possible L length sequences in a message of length M.
- X = The number of times the reserved pattern occurs.
- P[X = k] = The probability that the reserved pattern occurs k times.

From the first case of equal length patterns and messages, the value of p is

$$p = 1 / 2^{**}L$$

The value of n can be found by adding one bit at a time tc a message packet which initially has length M equal to L and determining how many possible reserved pattern sequences could occur. If M equalled L + 1, then one pattern could cccur starting at either the first or second bit. However, it is cbvious that two patterns could not coexist. A message length of 2L wculd have to exist before two patterns could be found. This result is generalized such that

$$n = M / L$$

The existence of "cverlapping" reserved patterns is impossitle based upon the receiver's recognition technique for detecting reserved patterns. The receiver views L bits at a time, and upon finding a reserved pattern, looks at the following signaling bit and takes the appropriate action. In the case of no update packet present, the receiver strips off the signaling bit and continues to check the bit stream starting with the next message bit. This precludes the possibility of overlapping reserved patterns.

The probatlility of the reserved pattern occurring k times is then a binomial random variable in which

$$P[X = k] = \frac{n!}{k!(n - k)!} (p^{**}k) [(1 - p)^{**}(n - k)]$$

The expected number of occurrences is given by

$$E[X] = np$$

If the average number of message packets transmitted per link per second is P, then

$$Nm = \frac{PM}{L(2^{**}L)}$$

Substituting the values of Nu and Nm into the utilization formula yeilds

$$UTIL[inserts] = \frac{N(L+1)}{2TR} + \frac{PM}{LR(2^{**}L)}$$

This result indicates that the highest overhead will be due to the update packets themselves when a reserved packet is selected which is long enough so that the exponential term in the denominator becomes large.

E. A HIERARCHICAL VERSION

The need for reducing the size of routing tables and the overhead from node update transmissions can be met by modifying the Yen algorithm to be able to function in an hierarchical form. In presenting the modifications to the basic algorithm, the requirements necessary for a hierarchical version to function are identified.

When using a hierarchical scheme, the network is divided into regions of nodes; if additional levels are needed, these regions may in turn be grouped into clusters. The number of levels is in no way limited to two, depending to a large extent upon the total network size, but this illustrates the point. The terms "group" and "family" will be used to refer to the two respective levels which is in keeping with some of the prior work in this area [Ref. 19].

Routing table size is therefore reduced since individual nodes no longer have to keep entries on each node within the entire network. Nodes only keep "node" entries in their routing tables to nodes within their same group. To route messages to nodes outside their group, but within the same family, "group" entries are maintained so that each node knows the best path to groups within the family. Finally "family" entries are kept so that nodes know the routes to families outside their own.

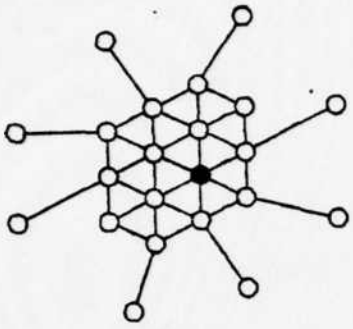
The Yen algorithm functions successfully because of the implicit ability of a node to send its destination update to its neighbor nodes at the same time. In this manner, the link weight information which is directly related to update transmission time is accurately disseminated. For the Yen

algorithm tc functicn in a hierarchical structure, then, demands that a group cf nodes be able to transmit its grcup destinaticn update tc all of its neighbor nodes at the same time. This, of course, applies to each hierarchical level. The key tc applying the basic algorithm in this structure is to develop a means whereby a grouping of individual nodes can act as a single 'super' node. By using a grcup "warning" message, adequate coordination can be accomplished so that this is possible.

1. Warning Messages

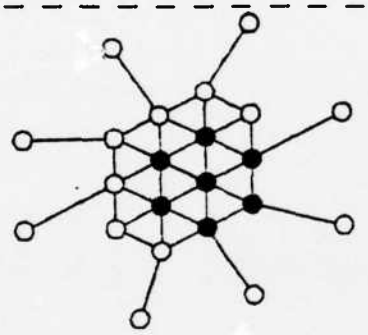
This warning message could originate from any node within the basic group. The warning contains information identifying its nature and the future time at which the group update message is to be sent. This future time (called the "firing time") would be far enough in the future as tc ensure that all nodes within the basic group had received the warning prior to firing time. Each node knows the identity of its neighbor nodes and therefore this warning would only be transmitted to nodes in the basic group.

The means of disseminating warning messages within the group could be by a simple flooding scheme. Upon



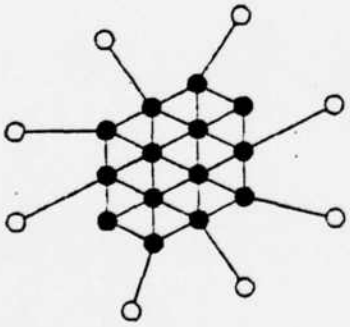
Time = 0.0 seconds

The "Leader" transmits the Group Warning Message to neighbor nodes (Firing Time is 1.0 seconds).
Transmission Delay = 0.1 secs



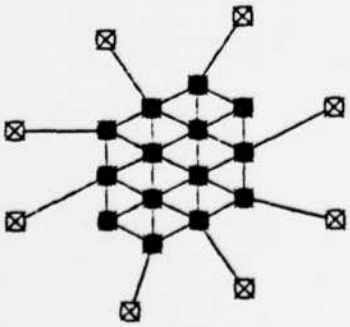
Time = 0.1 seconds

Neighbor nodes receive the warning message and retransmit it to their neighbors which are in the same basic group.



Time = 0.2 seconds

All nodes in the group are now "cocked" and ready to fire the group update at the given firing time.



Time = 1.0 seconds

Group "border" nodes fire the update to neighbor nodes which are not in the basic group.

Figure 3.5 Group Update Sequence

reception of the warning message, the node is "cocked" and ready to fire the update. When the firing time occurred, nodes in the group would only transmit the group update message to nodes outside of the basic group. Since all of these "border" nodes of the group would fire the update to outside nodes at the same time, the net effect would be as if a single "group-sized" node had originated the update message. Figure 3.5 depicts the actions that occur during this sequence of events.

The algorithm would not change except that the originator of the update would be the group identifier. For a two level hierarchy, updates would originate from nodes, groups or families and warning messages would be of two types, either group or family.

2. Originating Warning Messages

In using the warning message concept, a method of determining how they will be originated for the group is required. The key consideration in the selection of the method is whether a single node within a group should be responsible for originating the warnings or if this function is to be distributed amongst all nodes in the group. The first method looked at was a distributed scheme in an

attempt to minimize the use of centralized control functions.

This proposed method was based upon having each node originate its own group warning message on a regular basis. The time interval between messages could be a uniformly random amount with a prescribed average value, perhaps related to a multiple of the node update interval. An anticipated problem with this approach is the difficulty in maintaining acceptable "spacing" in time between warning messages which originate from many nodes. Situations could develop where all warning messages occur within a small time interval, followed by a relatively long period of time where none are originated. This "bunching" and "spreading" of update transmissions can cause poor routing information and degraded network performance.

A solution to this problem is to have a single node within a group be the originator of warning messages at any given time. With this method, the problem of poorly distributed group update messages in time is eliminated, but now the network is subject to the vulnerabilites which any form of centralized control introduce. If the node generating these group warnings fails, then the network will

suffer. Additicnally there is the difficulty in selecting which node is to be the "leader".

An algorithm is provided in Appendix B which gives a means cf determining the single node which will originate the group updates and a way in which "Successor" nodes can be chcsen to replace fallen leaders. It is based on the use cf a timer which is "set" when a node receives a warning message. The timer's setting is greater than the maximum interval between ccnsecutive warning messages plus the maximum propagation cf such messages throughout the group. If the ncde's timer runs out prior to the arrival of ancther warning message then the node concludes that the leader has failed. The action taken by the node is the originaticn cf its own "Successor" message which notifies other nodes that it is now competing for the functicn of leader. Provision is made such that orly one successor is selected if indeed the previcus leader had failed.

3. Synchronizing Unit Update Transmissions

A possible prcblem with the hierarchical versicn of the Yen algcrithm is the overhead associated with achieving simultaneous update transmissions by all nodes of a grcup or family. It was mentioned before that to disseminate the

warning message, a flooding technique may have to be used. As discussed in Chapter II, flooding can cause significant message duplication and excessive overhead. Another condition that exists is that the only nodes which actually transmit the group update message are those on borders which have neighbors outside the basic group. If these border nodes, then, knew on their own when to "fire" the update messages then a substantial reduction in overhead could be achieved.

The proposed solution to this problem is best illustrated by means of a musical analogy. A node can be likened to a musician in a band where the band is the basic group or family. Under the current warning message scheme the nodes are directed to fire the update at a specified time. This is similar to a band without sheet music being directed to play a song one note at a time. A more efficient method would be to provide each musician with his own copy of the song. Now the director (i.e. leader node) only has to ensure that the band members begin playing at the same time. He must also periodically wave his baton to ensure that the tempo is maintained.

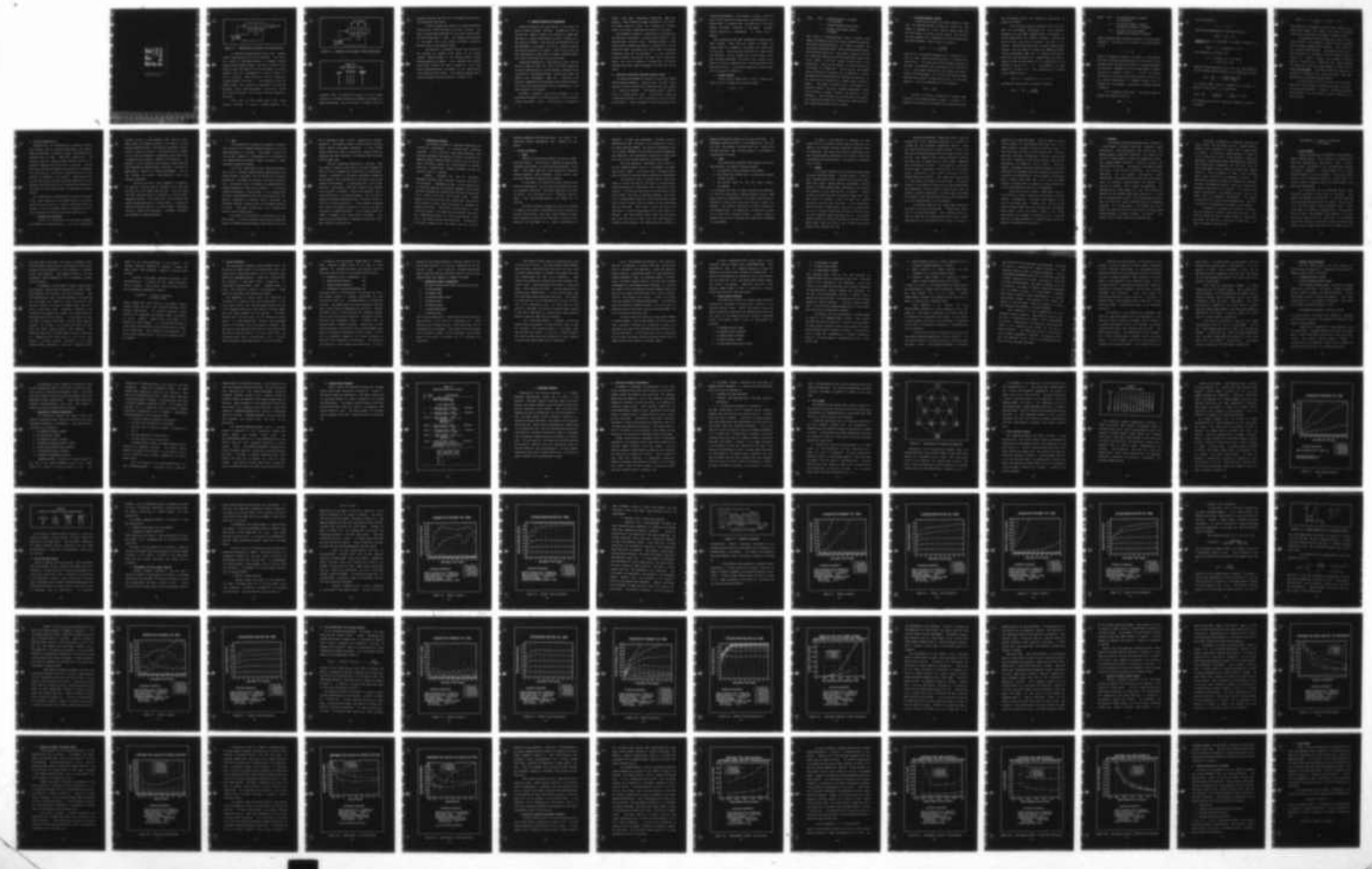
A simple pattern which each node within a group could follow would be setting a fixed interval between consecutive update generations. For example, if the resynchronization time was 15.8 seconds and the agreed upon time interval was 2 seconds, then nodes would transmit their group updates at times of 15.8, 17.8, 19.8 and so on. For many applications this simple scheme may be suitable. However, if highly repetitive transmissions are not desireable, then a pseudo-random sequence may be used.

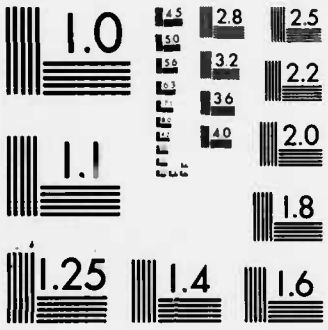
In order to generate a pseudo-random sequence of firing times for the unit update messages, a concept from spread spectrum communication systems was utilized. Such systems often employ feedback shift registers (FSR) to generate the pseudonoise (PN) binary sequences which are modulated by the data bits.

The configuration of a three stage FSR is shown in Figure 3.6. When clocked, a K-stage FSR produces a PN sequence of length 2**K - 1 before repeating [Ref. 20]. Even when using a small FSR, then, one can produce a relatively long sequence. Some Direct Spreading spread spectrum (DSSS) systems use the FSR in this manner.

AD-A139 147 APPLICATION OF A DISTRIBUTED ROUTING ALGORITHM TO A PACKET-SWITCHED COMMUNICATIONS NETWORK(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA R R LOGAN DEC 83 2/4

UNCLASSIFIED NPS62-83-060 F/G 17/2 NL





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

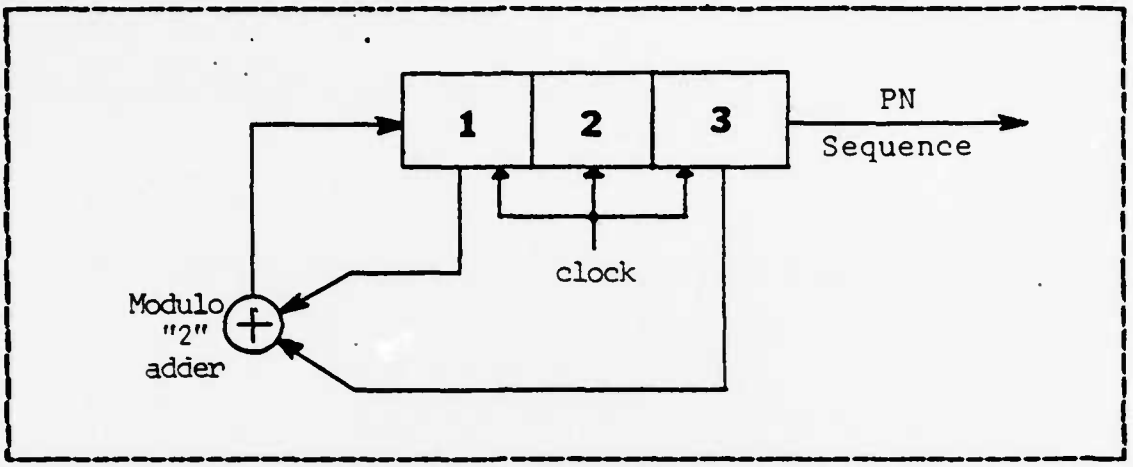


Figure 3.6 Feedback Shift Register: DSSS Configuration

To produce the pseudo-random firing times, however, another configuration of the FSR is proposed. This particular method is sometimes employed in Frequency Hopping spread spectrum (FHSS) systems. Note that in Figure 3.7 the output is from taps taken off of all FSR stages. The output sequence from this configuration is given in Table III. This K bit binary "time" value will then hop amongst all integers between 1 and 2**K - 1. The sequence of hops will appear random, but of course is predictable if one has the knowledge of the FSR configurement, the initial "value" which the FSR contains, and the start time and frequency of the clock.

Since the K bit time number takes on all values between 1 and 2**K - 1 during a complete sequence, its

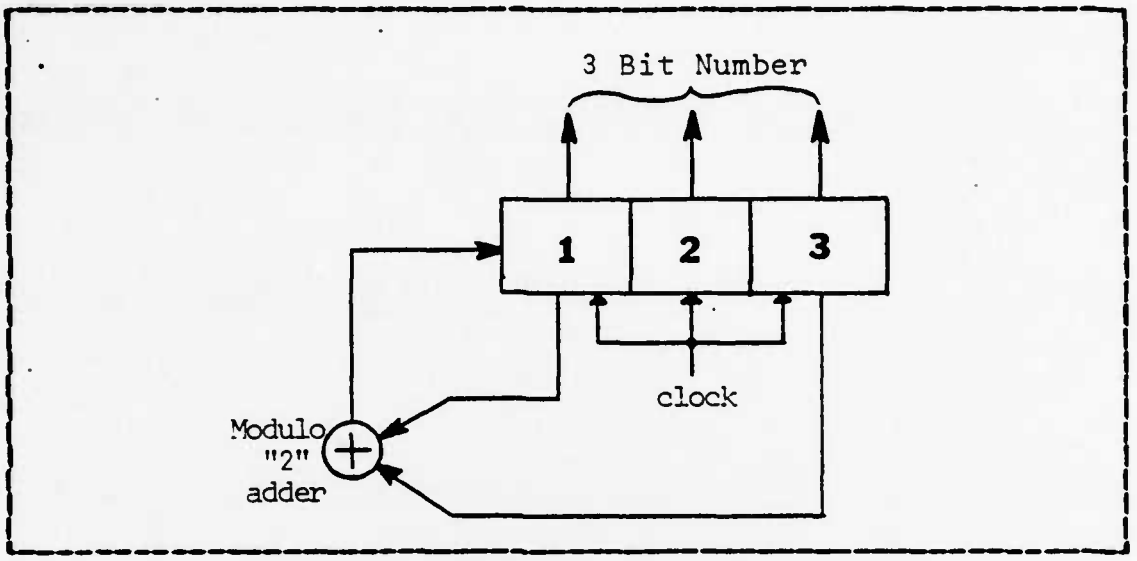


Figure 3.7 Feedback Shift Register: FHSS Configuration

TABLE III

Output Sequence of FSR

clock	FSR stage 1	2	3	number value
1	1	1	1	7
2	0	1	1	3
3	1	0	1	5
4	0	1	0	2
5	0	0	1	1
6	1	0	0	4
7	1	1	0	6
8	1	1	1	7

"average" value is simply the midpoint of this range. Therefcre, to produce a particular average time between unit update generations, cne could include a constant term or a

scaling multiplier such that the FSR numbers would map into the appropriate time range.

In use the nodes would receive the resynchronization time and then "clock" the FSR at the time of firing. The value received would correspond to the time interval before the next firing were to occur. The process would continue in this manner with a new firing time being calculated directly after a unit update is sent.

The reduction in overhead between this method and the previous one relates to the ratio of the average interval between updates to the average interval before resynchronization is needed. As an example if the average group update interval had been 1 second and if the time before resynchronization was needed for the pseudo-random time sequence was 20 seconds, the required overhead due to warning messages would be reduced by a factor of 20.

IV. NETWORK MODELING AND SIMULATION

Numerous techniques have been applied to the design and study of packet transport networks. Mathematical tools to include both models of queues and networks of queues have been used to predict the behavior of packet systems with a high degree of success. Many heuristic techniques have also been developed in the quest to find the optimum structure and operating methods of a system. Work of this nature has been accomplished in areas such as network topology design, link capacity selection and routing algorithms.

Analytical techniques at times cannot be used when the theory is unable to deal with some of the "real" properties of packet networks such as state dependent transition probabilities and correlations between interarrival times and service time requirements [Ref. 21]. Simulation then becomes a frequently used technique to gain useful insights, and in fact most major systems, prior to construction, were first studied in this manner.

Basically, simulation has two main purposes: a) the performance evaluation of network protocols that are analytically intractable, and b) the validation of analytical

models based upon simplifying assumptions [Ref. 21]. However, these purposes can only be achieved if the simulation model itself is valid, and techniques are required which can be used to guarantee the validity of the model.

This research involved an application of simulation for the purpose of evaluating a network routing protocol. The approach used to verify that a SIMSCRIPT program of 3500 lines actually modeled the system in question was to use an analytical model and compare the results when the program parameters were set to correspond. If the simulation produces results which agree with that of the analytical model, then one is inclined to believe that the results obtained when other program parameters are used will also be valid.

A. ANALYTICAL TECHNIQUES INVOLVING QUEUING THEORY

In developing mathematical methods for analyzing complex packet-switched networks, queuing theory has been widely applied with many results being developed that give insight to a variety of problems. The simplest queuing model is that of a single server queue. Single server queues are categorized according to their interarrival time and service distributions. These categories are often designated using

a Letter/letter/Number, such as M/M/1 or M/G/1, where the first letter denotes the interarrival time distribution, the second letter the service time distribution and the third element the number of servers in the system. Standard letters used are M - exponential, D - fixed and G - general.

Many formulations have been developed for single server queue systems. Those which are related to finding the average number of customers in the system are focused upon in this work. This particular statistic was chosen based upon selecting the one quantity from many elements which most accurately described the status of the network at any given time during a simulation run. The details behind this explanation relate to the actual simulation program itself and are presented in Chapter V.

1. Little's Result

Little's Result [Ref. 22] is a basic formula which states that for a steady state queuing process

$$E[n] = \lambda T$$

where $E[n]$ = expected number of customers in the system;

λ = average customer arrival rate;

T = average time customer spends in system.

This result is widely applicable since the formal proof makes no assumptions about the arrival process distribution, the service time distribution, the number of servers, nor the service discipline which may be first-come-first-serve, last-come-first-serve, etc. The system in question can also be a network of queues and the M/M/1 hypothesis doesn't have to be invoked, so that the theorem is quite broad in its application. Using Little's Result, an entire network can be enclosed in a "black box", and as long as the arrival rates are known and the network is in steady state, then by determining the average number of customers (or packets) in the system one can calculate the average transit time for a packet. This simple technique allows the comparison of simulation results with those obtained analytically using Little's Result in order to verify the "behavior" of the simulation. This technique was utilized during the simulation program test phases with results that agreed very well with those obtained using Little's Result.

2. Pollaczek-Khinchin Formula

A second formula useful in network analysis is based upon the Pollaczek-Khinchin formula for finding the mean number of messages in an M/G/1 queuing system [Ref. 23]. The M/G/1 system involves a general service time distribution which implies that messages may have non-exponentially distributed lengths. The P-K result is given by

$$E[n] = \rho + \rho^2 \frac{1 + Cb^2}{2(1 - \rho)}$$

where "ρ" is the traffic intensity and is defined as λ / μ with "λ" again being the arrival rate and "μ" denoting the service rate. Cb is the ratio of the standard deviation to the mean of the service time distribution and is defined as σ_b / μ. This formula shows that if two service time distributions have equal means, the one with the larger standard deviation will produce a longer waiting time. For the M/M/1 queue, Cb = 1 and the expression for E[n] reduces to

$$E[n] = \frac{\rho}{1 - \rho}$$

The use of the term "customer" and "service time" are used in illustrating queueing models; of course, in a communications network the customers are the message packets

and the service rate is the capacity of the channel in packets per second.

The P-K formula applies to message length statistics which are continuously distributed but a more realistic assumption is that these message lengths are discrete in nature, being made up of a number of packets. Studies have shown for both long (15 to 30 minutes) and short (1 to 2 minutes) holding-time connections in packet communications systems, that the number of packets per message are geometrically distributed [Ref. 25]. A geometric distribution means that the probability of a message being one packet in length is given by some number $p \leq 1$, the probability of two packets is pq with $q = 1 - p$, the probability of three packets is pq^2 and so on. The probability of a message being k packets long is

$$P[X = k] = p(q^{**}(k-1)) \quad , k = 1, 2, \ldots$$

The discrete form of the P-K formula is given as

$$E[I] = \frac{\mu_p}{2} + \frac{\sigma_b^2}{2(1 - \mu_p)}$$

where $E[L]$ = the average number of packets in the system;

μ_p = the average number of packets entering the system in D seconds;

σ_p^2 = the variance of the packets entering the system in D seconds.

These parameters refer to packet arrivals in a fixed service interval D. The value of D is related to the link capacity "C" by

$$C = 1 / D$$

which is in packets per second. The term μ_p is the same as the traffic intensity parameter since it is the ratio of the average traffic rate into the system to the system transmission capacity. Therefore ρ is the average number of packets arriving in D seconds or just μ_p. To find the arrival statistics of the packets entering the system (μ_p and σ_p^2), the Poisson distribution of message arrivals must be combined with the geometric distribution of message lengths in packets.

For a geometric distribution, the average number of packets per message is given by

$$E[X] = 1 / p$$

and the variance by

$$\sigma_x^2 = q / p^2 .$$

From Poisson message arrival statistics we have

$$E[M] = \sigma_m^2 = \lambda D$$

Therefore if P is the number of packets received in D seconds then

$$E[P] = \mu_p = E[X]E[M] = \lambda D / p.$$

The variance of P is found to be

$$\sigma_p^2 = (E[M]\sigma_x^2) + (\sigma_m^2 E^2[X])$$
$$= (\lambda D(1 + q)) / p^2$$

Replacing these expressions for μ_p and σ_p^2, the expression for the average number of packets in the system then becomes

$$E[I] = \frac{\lambda D}{2p} + \frac{(\lambda D(1 + q)) / p^2}{2(1 - \lambda D/p)}$$

As a check assume that all messages are one packet in length, in which case p = 1 and therefore

$$E[L] = \frac{\lambda D}{2} + \frac{\lambda D}{2(1 - \lambda D)} = \frac{\lambda D}{1 - \lambda D}(1 - \lambda D/2)$$

Now compare this with the continuous formulation using Cb = 0, whereby

$$E[N] = \rho + \rho^2 \frac{1}{2(1-\rho)} = \frac{\rho}{1-\rho}(1 - \rho/2)$$

Both results are equivalent since when all messages are of fixed length equal tc one packet, the number cf messages stored must be the same as the number of packets stored.

The importance of this formulation is that the variance cr spread in the number of packets arriving increases as the average message length increases. This means that for a fixed traffic intensity, the message arrival rate λ must decrease as 1/p increases to keep the average number of packets $\mu_p = \rho$ fixed. However, this increase in variance means an increase in the probability that longer messages will arrive and hence the average number of packets in the system must increase. This result proved very importart in studying the behavicr of the routing algorithm when using geometrically distributed message lengths.

Little's Result and the P-K formula were the primary analytical tools utilized in determining whether the simulation program was operating correctly. Statistics obtained during program runs were checked by direct applicatien of these fcrmulas.

B. NOTES ON SIMSCRIPT II.5

SIMSCRIPT II.5 is a discrete-event simulation programming language which is designed specifically for modeling systems whose state is deemed to change instantaneously at discrete points in time, rather than continuously. The discrete nature of packet-switched networks make this an excellent language for construction of the model. Some of the unique features and general characteristics of SIMSCRIPT II.5 which made it a particularly desirable choice for this research are examined in this section. These descriptions are aimed primarily at giving the reader unfamiliar with the language some insight into why the program was structured in the manner selected without having to assimilate the entire program.

As a convenience in understanding the following descriptions, terms appearing in ALL.CAPITAL letters with seperating periods are the actual variable names used within the program. In this way readers wishing to correlate the overview information with the actual program code may do so.

1. Entities and Attributes

An entity is a structured data item which represents some element of a modeled system. It is like a subscripted

variable in that it may represent more that one value. These values are termed attributes of the entity and when assigned specific values, they define a particular configuration or state of the entity. The important point, though, is that these attributes are referenced by name rather than by subscript which enhances readability and modeling. For example, one can create an entity called a PACKET which has such attributes as BEGIN.TIME and DEST.NODE. The value of BEGIN.TIME is referred to by BEGIN.TIME(PACKET), which is more readable than using a multidimensional array with numerous subscripts.

Entities also may be either permanent or temporary. All permanent entities must be "created" at the same time and thus their total number must be known. This fact makes this data structure ideal for modeling the network nodes. Temporary entities, on the other hand, may be created when needed and destroyed when no longer needed. By destroying temporary entities which are no longer needed, significant savings in memory can be achieved. Messages, therefore, were modeled as temporary entities.

2. Sets

The structuring of related data elements by means of sets proved to be an extremely powerful concept and was thus used extensively in the model. This single concept is by far the most important in understanding the development of the network model and provided a means to produce an "understandable" simulation program.

One of the major goals of this particular research was to write a "driver" packet-switched communications simulation program. This meant that the program had to function efficiently, be modular in structure for ease of modification, and above all else be conceptually understandable to others continuing in this area of research. In reviewing simulation programs by other authors, it is apparent that the use of multidimensional array data structures with their numerous subscripts in modeling network activities and components produces programs which are extremely difficult to follow. One simply becomes lost in the indexes when trying to trace program logic.

The answer to the problem for this author lay in the use of sets. It was soon apparent that having a background steeped in FORTRAN array structures caused inertia which had

to be overcome in order to find a better way of making a simulation. The results, though, of using the set structure method seem to have proved successful, in that another thesis student was able to utilize this network simulation program as the basic structure for modeling another routing protocol [Ref. 24].

Sets are organized collections of entities. Sets are similar to arrays in that each of the entity elements of which the set is composed may be identified and manipulated. However, in contrast to the static structuring imposed on array elements, the organization of entities in sets may be dynamic and changeable. An entity may own a set of other entities. In our model a LINK owns a QUEUE in which PACKETS can be "filed" or "removed". Sets can be searched to see if a particular entity is currently present, and may also be ordered automatically based upon such criteria as first-in-first-out or last-in-first-out. A final comment concerning sets is that the number of entities in a set at any time is contained in a program variable. In the previous case, the number of PACKETs in the QUEUE would be N.QUEUE. This feature was used extensively in gathering statistical information during a simulation run.

3. Statistical Analysis

The principal outputs of simulation experiments are statistical measurements. In reviewing other simulation programs, it was noted that often-times statements were scattered throughout the program to collect information with special summary routines written to print it out at the end. Aside from being tedious and time-consuming to produce, these data collection and analysis statements clutter the operating logic of a program.

However, SIMSCRIPT II.5 has two statements, ACCUMULATE and TALLY which virtually eliminate this kind of "book-keeping". ACCUMULATE was used in this program since it introduces time into the average, variance, and standard deviation calculations, weighting the collected observations by the length of time they have had each value. Another reason why array structures were avoided was because both these powerful routines would only perform on permanent or temporary entities. During the initial check-out of the simulation program, key statistics were obtained using both the ACCUMULATE procedure and written book-keeping statements. The results obtained were identical. Additionally, the program execution time was reduced when the built-in

statistic gathering functions were used. As a result, all applicable network measurements were obtained in this manner.

C. NETWORK COMPONENTS

1. Nodes

Nodes are the switching elements for the data communications network. The switching elements are specialized computers which handle the many functions performed by the node. Other common terms for a node include IMP (Interface Message Processors used in ARPANET), packet switch, data switching exchange and communication computer.

In this model, all nodes have unique identities and identical capabilities. In the simulation program each node is a permanent entity with attributes identifying its group and family. The node owns a list of "neighbor" nodes for which a full duplex communications link exists in its LINK.SET.

The clock information required for the Yen algorithm is modeled as the node's ALARM.CLOCK set which contains the routing algorithm data. Each time a node receives an update from a destination node for the first time, the steps of the algorithm are performed and an update transmission is

scheduled. To keep this infcrmation, the node creates a BUZZER entity whose attributes are the destination node of the update message (CAUSE), the tentative transmission time cf the update (SETTING), and the current Update Transmission List (BUZ.LIST) which stores the identities of those nodes from which it has already received a particular update.

The node maintains a XMT.RECORD which contains a list of update messages which it has received and subsequently sent onward. When "late" update messages arrive, then, the node can identify them by checking the XMT.RECORD.

The simulatien may be operated using either datagrams or virtual circuits as the means of routing message packets from source tc destination node. When using virtual circuits, the ncde must keep track of the circuits or paths which packets belonging to particular messages are taking. This is acccmplished by means of a VIRT.CKT.LIST which nodes keep in order to ensure that all packets of the same message take the same path. When virtual circuits are used in the simulatien, a route is established by the first packet of a message. This "trail-blazer" packet takes the then-current best path from node to ncde with each node creating a CIRCUIT fcr that message. The CIRCUIT has the attributes of

message identification number and the best path taken. All subsequent packets belonging to the message will then take that path. The CIRCUIT is "broken down" (i.e. removed from the VIRT.CKT.LIST) when the last packet of the message is transmitted by the node.

2. Links

Each node owns a LINK.SET which contains the set of LINK entities. The LINK's attributes include

1. The identity of the terminating node (TERM);
2. The current link STATUS which is either BUSY sending traffic or IDLE;
3. The current value of the link weight function (LK.WEIGHT);

It is apparent that some of these attributes are "physical" in that they model actual network features such as TERM while others are "descriptive" in the sense of associating a condition or state of an entity with the entity itself (i.e. STATUS). The power of the entity-attribute structure is very apparent here. By associating the condition or state of an entity with the entity itself, a simple method of keeping track and thus analysing very complicated processes is accomplished.

In using a set structure to describe the link connectivity of a network, a great degree of flexibility was added to the possible network topologies available within the simulation. The option also exists to dynamically alter that topology during a simulation run in order to model such occurences as new nodes entering the system or the destruction of links.

3. Queues

In an actual implementation of message buffer queues for packet-switched systems, the node itself is likely to have a single memory buffer designated as the packet queue. Under this scheme, packets awaiting transmission would be stored in a single memory buffer, regardless of the outgoing link selected as the best path for the packet. When an outgoing link which had been busy finally became idle, software would be required to scan the queue for packets destined to travel on that link next. In this simulation, rather than have a single packet queue for the entire node, the job of buffering packets was divided up such that each link had a seperate queue associated with it. The tradeoff between scanning the single node queue and dividing allotted memory space among the links is one of access time versus maximum buffer capacity per link.

Asscciating individual queues with their links was incorporated in the simulation primarily as a methcd of tracking the flcw of packets through the network. Using this mcdel means that when a packet arrives at a node, its best path is determined based upon the current routing table entries. The packet is then either transmitted over the best path link cr placed into the link's queue, depending upon the status of the link. In the event of a busy link and an already large queue, the best path for that packet when it finally gets to the "top" of the queue may have changed during the interval. The packet will then be sent to a ncde which is nc longer on the best path. The prcblem inherent to link queues is thus one of routing packets based upon infcrmation when the packet arrives and not when the packet is actually transmitted. Of course, when link queues are generally short the possible routing errors associated with this problem shculd not be substantial.

An alternate mcdel for the packet queue required at each node (which was not simulated) is based upon a variation cf the single ncde queue. When a packet arrives at a node, its best path is determined based upon current routing information and if the selected link is busy, the packet is

placed in the single node queue. When the packet is placed into the queue, it is not assigned a best path link, but is merely placed in the queue in first-in-first-out order. At some later time, the link to node J becomes idle. The software will then scan the queue for the first packet whose best path, according to the current routing table for its destination, is indeed the link to node J. Using this system, all packets within the queue are going to be routed based upon the "freshest" best path information available at the time a link becomes idle. By not freezing the best path which a queued packet must take at the time the packet arrives, we ensure that all packets will take the best path to their destination at the time of transmission.

Each link also has an update queue which is used in the simulation to model the priority status given to routing traffic. Conceptually, the update queue appears at the "top" of the packet queue. Updates, then, are sent before regular message traffic when both the update and the message packet queues of the link contain traffic. This double queue system also provided a convenient way to model the insertion of update packets into ongoing message packets by means of the reserved pattern method.

4. Messages

The simulation generated messages which consisted of a random number of packets based upon a selected length distribution. The distributions used were uniform (the maximum message length was selectable), and geometric (in which the average number of packets per message was picked). Each message PACKET contained a number of physical and descriptive attributes. The physical attributes included such things as the originator (ORG.NODE), the destination (DEST.NODE) and the message identification number (MSG.ID.NUM). The descriptive attributes were used specifically for gathering statistics about the packet's trip through the network. When a packet was generated, its LIFETIME was set ALIVE (equal to 1) and upon reaching the destination, the LIFETIME became DEAD (equal to 0). In this way the average packet transit time was determined by averaging the sum of all packet LIFETIME's using the ACCUMULATE procedure. In the same manner, the average time that a packet was queued during its trip was found by setting QSTAT "ALIVE" whenever the packet entered a link queue. Upon removal from the queue, QSTAT was pronounced DEAD.

The packet attributes of P.NAME and P.ARRIVAL are used in order that the arrival time of a message packet in transit can be modified in the event that an update packet is inserted. In order that message packets in transit not be in a limbo state as far as the simulation program was concerned, a "propagation queue" (PROP.QUEUE) was develcped. When packets were transmitted they were filed into the link's PROP.QUEUE and upon arrival to the next node, removed therefrom. This was required in order to keep accurate track of the identity of all messages in transit. The informaticn contained in P.NAME and P.ARRIVAL was related to the name and scheduled cccurence time of the event ccrresponding to the arrival of that packet. In this way, if an update was inserted, the scheduled arrival time of the packet could be lengthened accordingly. The details involving hcw the carcelling and rescheduling of the packet's arrival is accomplished is included in the program.

The length of the packet was described in terms of the time required to transmit a packet over one link. This time was selected by using a channel capacity of 20,000 bits per seccnd per link and a packet size of 1000 bits. The packet transmission time was then

PKT.XMN.TIME = 1000 bits / 20,000 bps
= 0.05 seconds .

5. Update Packets

Routing information was disseminated among nodes by means of UPDATE packets. The attributes of an UPDATE identified the destination as being either a node, group or family (CLASS) and whether or not the update was "regular" or "warning" (VARIETY). A regular update is the simple message described in the Yen algorithm. The warning update is used in the hierarchical version to enable nodes of a group or family to transmit their unit update message at the same time (FIRING.TIME).

The update also had an additional counter which indicated the number of hops which it had taken. This counter was employed in anticipation of a problem which can occur when hierarchical routing is utilized. Nodes residing on or near unit borders can have neighbors which, though quite close, are located in different units. Routing information to these close neighbors would normally be in terms of the best path to the unit, rather than to the specific node itself. Thus unit (vice nodal) routing information may result in much longer paths being taken between nodes that

are very close to each other but belong to different units. A more efficient routing should be achieved by having nodes maintain routing information on close neighbors, regardless of the unit boudaries. The proposed method on how this "boundary" routing information can be developed utilizes this hop counter.

By examining the HOP.COUNT of "node" updates which are received, precise routing information to "close" neighbors outside of the the basic unit can be developed. When this technique was used, the user parameter HOP.LIMIT was set to indicate the "depth" at which node updates were allowed to travel into groups or families which were not a part of their basic group. Therefore, whenever a node received such an update from outside of its basic group the HOP.COUNT would first be checked. If the packet had traveled over the specified HOP.LIMIT, then the update would be ignored. If, however, the HOP.COUNT was within the limit then the normal update procedure would commence. The receiving node would now possess special routing information to a neighboring node which was not in its group. This ability to maintain routing information to all close neighbor nodes regardless of their unit assignment should

assist in the total perfcrmance of the network. The benefits from this technique are expected to exceed the small routing table increments required to maintain these entries.

The size of the update message was selected to be relatively short, in keeping with the simple information which it contains. For an update packet of length 40 bits the correspcnding transmission time was

$$U.XMN.TIME = 40 \text{ bits} / 20{,}000 \text{ bps} = 0.00125 \text{ seconds}.$$

Update packets were given priority over regular message traffic through the use of the insertion technique and a separate update queue. The insertion technique allowed updates tc be transmitted when message traffic was already in transit. However, updates were not allowed to be inserted into other updates. If a link was busy sending an update packet then additional update packets would be placed in the update queue. Upon completion of transmission, all waiting update packets would be sent before message traffic was resumed.

D. NETWORK OPERATION

The transmission of message packets between nodes, the firing of an update packet at the precise time and all the other network activities required for the simulation were modeled by SIMSCRIPT "events". Events are routines which are scheduled to take place during a simulation. An initializing program called "Main", schedules events to occur, and during the course of the simulation, either events or subroutines can schedule more events.

The sequencing of events within the simulation is controlled by a simulation clock and an event queue. At any given moment during a run, the event queue contains the name of all scheduled events with their order in the queue dependent upon their scheduled time of occurence. The "Main" program loads the event queue with the scheduled starting events. At "Start Simulation" the first event scheduled (which is at the head of the event queue) is executed. This event may in turn schedule other events to take place either NOW (next in the event queue) or at some future time. The simulation stops when no further events are in the event queue, whereupon control passes back to the Main program.

In keeping with the initial design goal of a modularized, adaptable program which can be used in future research, all routines and events were written such that they were a part of one of the following four divisions:

1. Preparation Prior to Simulation (P)
2. Update Routing Protocol (U)
3. Message Packet Transport (M)
4. Evaluation of Network Performance (E)

The single letter at the beginning of each name was used in the computer program to indicate the division for each routine and event (i.e. E2.COLLECT.DATA is an Evaluation routine). The number after the letter designator indicates their relative order of appearance in the program. Routines and events within the same division are basically independent of those in the other divisions. This means that modifications and alterations within a division will not directly effect "outside" routines. Descriptions of each routine and event are provided in the program comments which detail the function and operation of each routine. In addition an alphabetic summary of all global variables is provided at the beginning of the program. The purpose of this section is not to repeat those details but to give an

cverview of the network operation and some insight intc the ever present question as tc why the simulation was written the way it was. It is hoped that through understanding "why" things were done as they were, follow on work can move forward rather than needing to reinvent the wheel.

1. Preparation Prior to Simulation

The seven routines which make up this divisicn are:

1. P1.BUILD.NETWORK
2. P2.CCNNECT.LINKS
3. P3.INITIAL.RCUTING.TABLE
4. P4.STATIC.EVENTS
5. P5.DYNAMIC.EVENTS
6. P6.PURGE.EVENT.QUEUE
7. P7.ZEROIZE.SETS

This divisicn performs all initializing functions which are required prior to the start of a simulation run. These functions can be broken down into three areas which are the construction of the network, the loading of the event queue with the starting events, and the "cleaning of the slate" prior to the start of the next run of a multiple run simulatcn.

The network is built using two routines which divide the task into two parts. The first routine, BUILD.NETWORK, takes the input data which lists the nodes and their group and family relationships and produces the program node identity numbers which are used in the simulation. These program node numbers are a single value which uniquely identifies a node, which otherwise would require a three part name. After creating the nodes, the link connectivity is produced using CONNECT.LINKS. In this routine duplex links are constructed between specified node pairs. Upon completion each node has its own LINK.SET which contains the identity of the terminating node. By using sets for this portion of the model, the ability to produce dynamic changes in the link connectivity of a system can be done simply by adding or deleting LINKs from the LINK.SET.

As an additional feature, a graphics routine was written which incorporated the same data set used in these two routines. The network figures in this report were produced from this routine. In this way, a visual confirmation could be made of all test networks as to their topology to ensure their correctness prior to simulation.

One of the problems anticipated in this simulation was that of producing an initial best path routing table prior to the start of the simulation. Rather than produce a routing table by some external means or simply assigning a randomly generated table, an alternate approach was used. Since the Update Routing Protocol routines could operate independently of the Message Packet Transport, the solution was simply to run the simulation without message traffic and with only one update per destination node, group or family. With all link weights set to unity, a least hop routing table was produced by INITIAL.ROUTING.TABLE. The resulting table was stored in a BEST.PATH matrix. As a time saver, an additional copy of this table was made (DUP.BEST.PATH) for use with multiple simulation runs.

The event queue was loaded with either STATIC.EVENTS or DYNAMIC EVENTS. Using STATIC.EVENTS, the initial routing table was fixed throughout the simulation by simply not scheduling any events from the Update Routing Protocol division to take place. On the other hand, when the simulation was run using the dynamic routing algorithm, events from the Update division were loaded in the event queue.

Tc stop a simulation after a given period cf time, the event queue is emptied using PURGE.EVENT.QUEUE. Upon completion cf this, all statistic gathering routines and dynamic set structures were cleared with ZEROIZE.SETS. To ensure the repeatability of simulation runs an added precaution was taken with all of the system random number generators used in the prcgram. These were reset with their starting "seed" values in order to ensure that multiple simulaticns would have identical inputs.

2. Update Routing Protocol

The connectien between the Message Packet Transport division and the Update Routing Protocol division is the BEST.PATH routing table. Events and routines within the Update divison are responsible for producing and maintaining this table. In this research the Yen algorithm is simulated within this division using a total of eight events and routines:

1. U1.GENERATE.UPDATE (EVENT)
2. U2.TRANSMIT.UPDATE (ROUTINE)
3. U3.INSERT.UPDATE (ROUTINE)
4. U4.RECEIVE.UPDATE (EVENT)
5. U5.LINK.WEIGHT.CALCULATION (EVENT)

6. U6.NCDE.WAKE.UP (EVENT)

7. U7.UNIT.FIRING (EVENT)

8. U8.ERASE.RECORD (EVENT)

The starting event used by the Yen algorithm is GENERATE.UPDATE which is scheduled for each node, group and family at the beginning of the simulation. During the course of the simulation each node will continue to generate its own destination update at regular time intervals. In the case of the hierarchical version, a "leader" node is selected in each unit which then generates the unit warning message at regular intervals. The time interval between node, group and family update messages are all user parameters which may be independently selected.

The updates are then transmitted to neighbor nodes by means of the TRANSMIT.UPDATE routine. This routine performs the critical function of ensuring that the effects of update traffic inserted within the bit stream of a propagating message packet is properly modeled. The routine is able to handle the transmission of an update differently based on the three possible circumstances by which a link may be busy:

1. Regular message traffic is being transmitted with no "inserted" updates presently in transit;
2. Update traffic is in transit which has not been "inserted" into a regular message packet;
3. "Inserted" update traffic is currently in transit within a regular message packet's bit stream.

In cases (1) and (3), the message packet in transit has to be "lengthened" to account for the additional update which has been inserted into its bit stream. Routine INSERT.UPDATE readjusts the message arrival time when an update is inserted into a message packet.

Upon reception of an update, the event RECEIVE.UPDATE performs the basic calculation of the Yen algorithm. The update is determined to be either "Regular" or "Warning" with appropriate action taken. In the case of a regular update, the value of the link weight is calculated and the time of transmission calculated in accordance with the Yen algorithm.

The user has the option of selecting one of the five link weight functions presented in the last chapter. If a link weight function is chosen which uses a window technique (as Method 5 did), a LINK.WEIGHT.CALCULATION is performed

each "window" of time throughout the simulation. The length of time for a window and the number of windows in a total averaging period are user controlled parameters. During a LINK.WEIGHT.CALCULATION, each link in the network would have a WEIGHT computed for the window time which is filed into the link's SET.OF.WEIGHTS. The number of WEIGHTS in the set is limited to the number of windows selected and therefore as a new WEIGHT is filed, the oldest WEIGHT is removed. This process provides a close approximation to a continuously sliding window. Upon reception of an update, the average of the link weight function is computed by taking the average of all WEIGHTs in the SET.OF.WEIGHTs.

Upon completion of these calculations the node schedules itself to "wake up" (NODE.WAKE.UP) when the best path time of transmission is to occur. This is accomplished by creating a BUZZER which has attributes identifying the update by its originating node (CAUSE) and the time of origin (BUZ.BEG). The BUZZER also contains the tentative time of update transmission (SETTING), the associated best path node (TEMP.BP) and the name of the NODE.WAKE.UP event which had been scheduled (BUZ.WAKE).

This last item was critical to the operation of the program. In order tc change the time of update transmission in case a "better" rcute occurs, the previously scheduled NODE.WAKE.UP event has to be cancelled and then rescheduled. In order to cancel a specific event residing in the event queue, it was required to have a specific "name". The act cf giving an event a name and then storing that name as an attribute of the entity invclved with that event was used throughout this simulation. In any truly dynamic and distributed simulaticn such as this one, the ability to properly cancel and reschedule events whose time of cccurence are subject tc change was critical to its successful cperation.

The identity cf the neighbor node (BUZ.NODE) from which the update was received is also maintained in a set (BUZ.LIST) which is owned by the BUZZER. This is similar to the Update Transmissicn List cf the algorithm, but instead cf deleting the nodes, this technique adds the nodes. When cther updates arrive, the ALARM.CLOCK is first checked to see if there already exists a BUZZER on this particular update. If one is found, then the new time "setting" is compared to the previcus BUZZER. If the previous setting is

still better, no further action is taken except that the identity of the node is added to the BUZ.LIST. If the new setting is better, then the BUZZER is reset and the NODE.WAKE.UP event rescheduled to the new time. Upon waking up, the node transmits the update to all nodes which are not listed in the BUZ.LIST.

When using the hierarchical version, the update received can be of a warning nature. Upon reception of a warning message, the node retransmits the warning using a flooding technique and then schedules its own unit update transmission using the event UNIT.FIRING. The event UNIT.FIRING is scheduled at the firing time of a regular group and family update message. The events UNIT.FIRING and NODE.WAKE.UP are similar in function except the latter refers to the unit update transmissions.

The node also maintains a record of all update retransmissions which is kept in the XMT.RECORD. In this way, "late" updates arriving at a node can be identifed and ignored. In order that this record not become excessively long, the event ERASE.RECORD is scheduled for each record after a set time interval. Upon execution, the outdated record is removed from the XMT.RECORD and destroyed.

3. Message Packet Transport

The two events falling into this category are involved with the generation, transmission and reception of message traffic within the network:

1. M1.GENERATE.MESSAGE (EVENT)
2. M2.RECEIVE.MESSAGE.PACKET (EVENT)

Event GENERATE.MESSAGE.PACKET produces new message traffic for the network. The traffic intensity (PPS.AVE) and the number of packets per message (AVE.PPM) are both controllable through simulation parameters. This event occurs each time a new message is generated. The average time between consecutive messages is then

MSG.GENERATION.INTERVAL = AVE.PPM / PPS.AVE

A Poisson arrival distribution was assumed and therefore this interval was used as the exponential interarrival time average between messages.

Packets were transmitted or queued depending upon the link status. A RECEIVE.MESSAGE.PACKET event was scheduled to occur PKT.XMN.TIME seconds after a packet was transmitted. This arrival time was subject, however, to variation due to the possible insertion of an UPDATE into the simulated "bit stream" of the propagating packet.

The selection of the source and destination nodes for each message was usually done using a uniform distribution so that all node pairs were equally likely. Provisions were made, though, so that the simulation could be run using unbalanced traffic distributions. Under these conditions, selected source-destination node pairs were chosen to produce non-uniform distributions of traffic.

4. Evaluation of Network Performance

The Evaluation routines and events collect and output all parameters which change for multiple runs and the accumulated statistical measurements. The seven routines and events are as follows:

1. E1.TRANSIENT.BLANKING (EVENT)
2. E2.COLLECT.DATA (EVENT)
3. E3.PARAMETER.LISTING (ROUTINE)
4. E4.NETWORK.PERFORMANCE.REPORT (EVENT)
5. E5.BEST.PATH.ROUTING.TABLE (ROUTINE)
6. E6.LINK.WEIGHT.MATRIX (ROUTINE)
7. E7.TRAFFIC.DISTRIBUTION (ROUTINE)

PARAMETER.LISTING provides the values assigned by the user to all program parameters which do not change during the course of multiple simulation runs. This

information is outputted prior to the start of the first simulation. During the simulation itself, the event COLLECT.DATA samples the status of the network at times which are controlled by the user. Under normal conditions, a particular number of samples is selected which then occur at regular intervals throughout the run. If 100 SAMPLES are selected for a 10 second run then the interval between consecutive samples would be 0.1 seconds. COLLECT.DATA takes measurements on the following items:

1. The total number of packets in the network.
2. The mean link utilization of the network.
3. The mean time it takes for a packet to complete its trip.
4. The mean queue size of the network.
5. The mean time per hop of each packet.

Collected data is stored in external memory and then retrieved by graphical routines. The plotted results enabled the status of the network to be analyzed as a function of simulation time and many examples of the results appear herein.

Used in conjunction with DATA.COLLECTION was the event TRANSIENT.BLANKING. Its purpose was to remove the

effects which the "start-up" behavior of the simulation had upon the statistics gathering routines. At the beginning of each simulation run, the network performance statistics displayed transient behavior which rapidly settled as traffic "filled" the network and the update process began to function. Being interested in the steady state condition of the network, the statistics taken during this period were removed by TRANSIENT.BLANKING which reset all statistical gathering routines. TRANSIENT.BLANKING for the simulation runs occurred five seconds after the start of the simulation.

The NETWORK.PERFORMANCE.REPORT gives a detailed summary of statistics on link utilization, message and update queues, and message packet trips. The BEST.PATH.ROUTING.TABLE and the LINK.WEIGHT.MATRIX give the current status of their respective quantities in matrix format. Lastly, the TRAFFIC.DISTRIBUTION summarizes the number of packets generated between each source-destination node pair, the number of packets completing their trips between the same and the average trip time for these packets. This routine is quite useful when analyzing the network performance using unbalanced traffic distributions.

5. Example Update Sequence

Table IV gives the simulation results for the same sample network used previously in Chapter II, Figure 3.1. The various events are described in the order of their time of occurrence during the update sequence of the single destination node. The results coincide exactly with those obtained previously when the steps of the algorithm were worked through by hand. The table consists of the actual output statements which were produced during the simulation run.

TABLE IV

Simulation Best Path Results

```
TIME   (NODE):                EVENT DESCRIPTION
----   ------      ----------------------------------------------

0.      ( 1) : NODE UPDATE ORIGINATED.
               SENT TO THE FOLLOWING NODES:
                                        2
                                        3
                 1  UPDATE GENERATED SO FAR.

0.      ( 2) : **NEW ALARM CLOCK**-- UPDATE( 1,  0.    ) FROM NODE  1.
                 SETTING TO GO OFF AT   2.0000.

0.      ( 3) : **NEW ALARM CLOCK**-- UPDATE( 1,  0.    ) FROM NODE  1.
                 SETTING TO GO OFF AT   4.0000.

2.0000  ( 2) : **WAKE UP SOUNDED** -- UPDATE( 1,  0.    )
                  BEST PATH FROM   2  TO   1   IS   1.
                       UPDATE SENT TO   3
                       UPDATE SENT TO   4

2.0000  ( 3) : ALARM CLOCK RESET - UPDATE( 1,  0.    ) FROM NODE  2.
               NOW SET TO GO OFF AT   3.0000.

2.0000  ( 4) : **NEW ALARM CLOCK**-- UPDATE( 1,  0.    ) FROM NODE  2.
                 SETTING TO GO OFF AT   5.0000.

3.0000  ( 3) : **WAKE UP SOUNDED** -- UPDATE( 1,  0.    )
                  BEST PATH FROM   3  TO   1   IS   2.
                       UPDATE SENT TO   4

3.0000  ( 4) : ALARM CLOCK RESET - UPDATE( 1,  0.    ) FROM NODE  3.
               NOW SET TO GO OFF AT   4.0000.
             ** RETRANSMISSION OF UPDATE NOT NEEDED **
             BEST PATH FROM   4  TO   1  IS   3.
```

```
                  **********************************
                  ******     BEST PATHS      ******
                  **********************************

                  FM-TO  N 1    N 2    N 3    N 4
                   N 1
                   N 2    1
                   N 3    2
                   N 4    3
```

V. SIMULATION RESULTS

Simulating the Yen routing algorithm in a packet-switched communications network was the focus of this research. The first phase of the study dealt with determining a link weight function which provided an viable estimate of the state of the network. When this particular function was used in conjunction with the Yen algorithm, a very efficient traffic routing procedure occurred for a variety of network conditions. The testing of the algorithm using this link weight function comprised the second phase of the simulation work. Here the algorithm was subjected to a multitude of different conditions in order to thoroughly test its operating characteristics and evaluate its performance. The last phase of the simulation involved repeating the process of phase two using the hierarchical version of the Yen algorithm which was developed in the last chapter. Comparisons could then be made between both the "nodal" or basic algorithm and the hierarchical version.

A. MEASURING ALGORITHM PERFORMANCE

In seeking to determine the performance of the algorithm, statistics were chosen which related to the number of packets within the network at any given time. The reasoning behind this is related in part to the analytical techniques presented in Chapter IV which are useful in validating the simulation model itself. Both Little's Result and the Pollaczek-Khinchin formula can give answers for the average number of packets within a network when the traffic distributions are known. Therefore by using those particular statistics, a means of validating the simulation model prior to testing the algorithm was possible. The second reason is due to the nature of the simulation "run" itself and how statistics are collected and evaluated within the program.

Perhaps a more familiar measurement used in performance evaluation of an algorithm is the mean packet delay and not the mean number of packets within the network. Little's Result intimately relates these two statistics and therefore the choice may not appear too critical. This is indeed the case under most conditions of performance evaluation. However, the lesser used quantity was chosen in this program for a reason which can best be seen by showing how statistics were gathered during a simulation run.

At any moment during a simulation run the number of message packets generated, in transit and having completed their trips can be described as:

1. X packets have been generated.
2. Y packets have completed their trip from source to destination.
3. X - Y packets are currently in the network.

The mean packet delay of all Y packets can be computed. However, this does not take into account those X - Y packets which are still in transit. One could, of course, approximate their delay based upon some scheme, but this may not be valid under some very important conditions. If for any reason the routing algorithm fails and congestion develops, it is possible that the mean delay of packets which have completed their trip will not change to reflect the current slow down very rapidly. However, if the number of packets in the system at a given time were used as the index, this condition could readily be noted. The mean packet delay was not used because at any moment during a simulation run, this factor accounts only for those packets which have completed their trip and not for ones which are currently in transit. By using the number of packets in the system at any given

time, the performance of the routing algorithm could accurately be determined. The better the algorithm performed, the fewer the number of packets in transit at any given time.

B. TEST NETWORK

The topology of the network used is given in Figure 5.1. The network has thirteen nodes and thirty full duplex links. In describing the characteristics of the network some terms familiar in graph theory are first defined.

A complete graph exists when all nodes are connected to all other nodes by a direct link. All properties of a complete graph are specified simply by indicating the number of nodes. A test network was chosen deliberately which was not complete in order that there would be multiple hop paths between some node pairs.

To indicate the relative "connectiveness" of a given graph, the link connectivity and node connectivity of a graph as a whole is specified. The link connectivity between two nodes is the minimum number of links that must be removed from the graph to disconnect them. The link connectivity of the graph as a whole is the minimum of the link connectivity of all pairs of nodes. For a complete N-node graph, the link connectivity is N - 1.

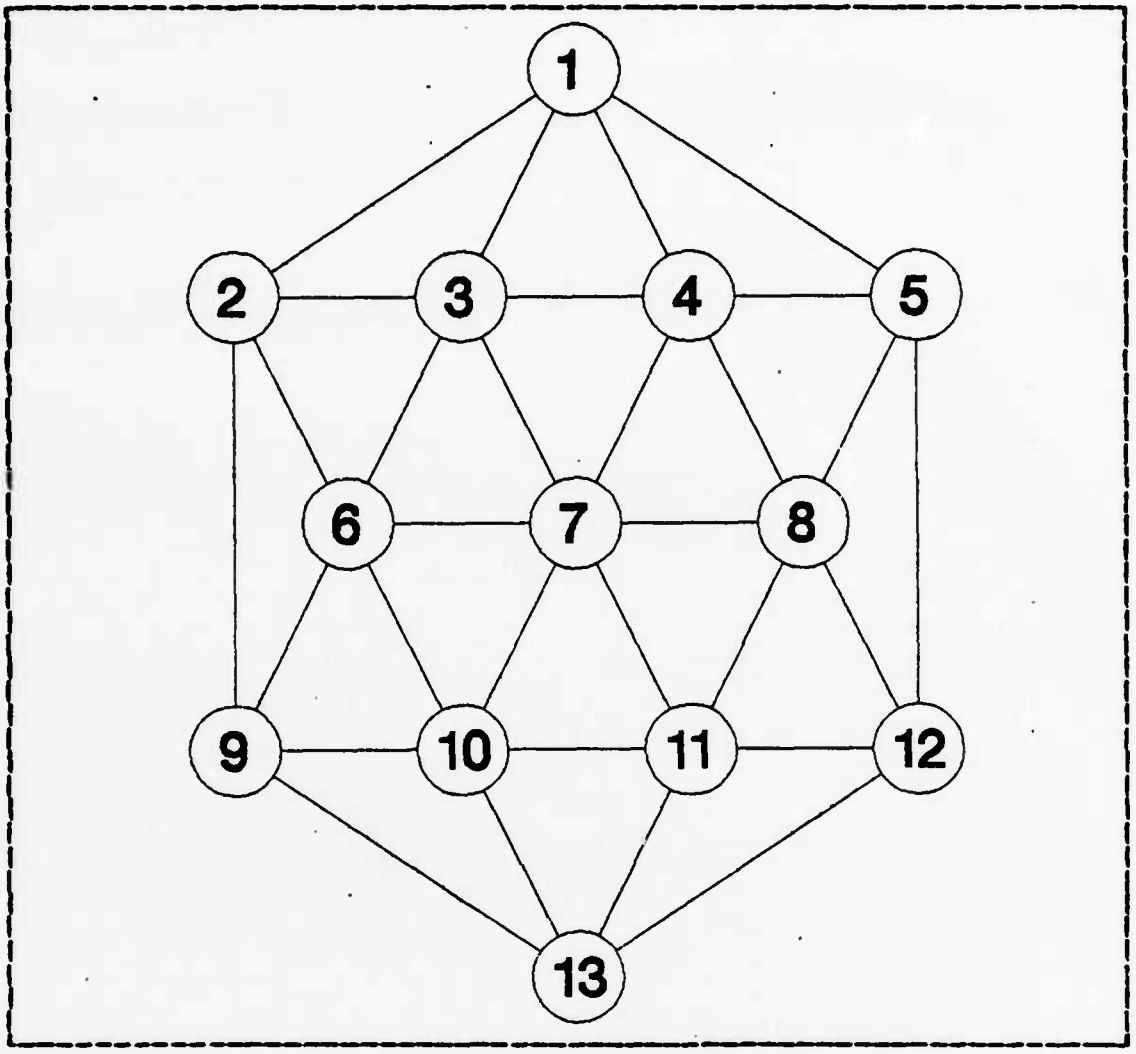


Figure 5.1 Test Network for Nodal Yen Algorithm

Analogous to the link connectivity between nodes, the node connectivity is the minimum number of other nodes whose removal will disconnect the two nodes. The node connectivity of the graph is the minimum taken over all node pairs. Since all nodes are directly connected in a complete N-node graph, its node connectivity is defined to be N - 1.

By inspection, it is found that the test network has a link connectivity of 4. This can be seen between nodes 1 and 13 which are completely seperated by the removal of links (1,2), (1,3), (1,4) and (1,5). The node connectivity is also 4 which can be seen by the isolation of node 2 by removing nodes 1, 3, 6 and 9. As a comparison, had this 13 node network been linked differently such that it formed a complete graph, both its link and node connectivities would have been 12. The difference between 12 and 4 gives an indication that this test network is lightly connected and that most paths between node pairs will involve multiple hops.

C. STATIC ROUTING RESULTS

The simulation program was first run using a fixed routing table. This initial routing table was determined by applying the Yen algorithm with all link weights set to the same value of 1. In this manner the best paths were those which required the fewest hops. The routing table which was produced is included in Table V. This particular result is in no way unique as there are numerous paths between node pairs, all requiring the same number of hops. The selection of this specific table was not, however, critical to the results obtained.

TABLE V

Least Hop Routing Table

FM/TO	1	2	3	4	5	6	7	8	9	10	11	12	13
1		2	3	4	5	3	3	4	2	3	4	5	5
2	1		3	1	1	6	3	3	9	6	1	9	9
3	1	2		4	4	6	7	4	2	7	7	4	2
4	1	3	3		5	3	7	8	3	8	8	5	5
5	1	1	1	4		1	8	8	12	12	8	12	12
6	2	2	3	7	7		7	7	9	10	7	9	9
7	4	3	3	4	4	6		8	6	10	11	11	10
8	5	4	4	4	5	7	7		12	11	11	12	12
9	2	2	2	10	13	6	6	13		13	13	13	13
10	7	9	6	7	13	6	7	7	9		11	11	13
11	12	13	7	7	8	7	7	8	10	10		12	13
12	5	13	5	5	5	8	11	8	13	11	11		13
13	9	9	9	12	12	9	11	12	9	10	11	12	

The simulation program automatically determines the least hop routing table without the designer having to initialize this matrix. This is accomplished by running the simulation without message packet traffic for a complete update cycle. The update cycle consists of each node, group and family originating its own destination update and is completed when no further update retransmissions are produced. Upon completion, each node has a best path routing table. This table is saved within the simulation program and utilized at the beginning of subsequent runs involving message traffic.

During static routing, this least hop table was fixed for the simulation run. Multiple runs were performed using different traffic levels by varying the rate at which message packets were generated. Source and destination nodes were selected in a uniform fashion such that each possible pairing had the same probability of selection. Each message was fixed to a length of one packet and the simulation run for a period of thirty seconds. Figure 5.2 presents the summary of the network performance.

The results indicate that the network had become "unstable" when the packet generation rate was 300 packets per second. The terms stable and unstable are used to describe the general state of traffic flow within the network. Given an unchanging rate of packets entering a system, a stable network will, after some transient start-up behavior, settle to a relatively constant number of packets in transit. In other words, the average number of packets in transit will remain stable during the simulation run. If, however, the network were unstable, the number of packets in transit would continue to increase with time. This "blowing up" can be caused from links becoming saturated and thus the packets entering the system no longer are able to leave at a steady rate.

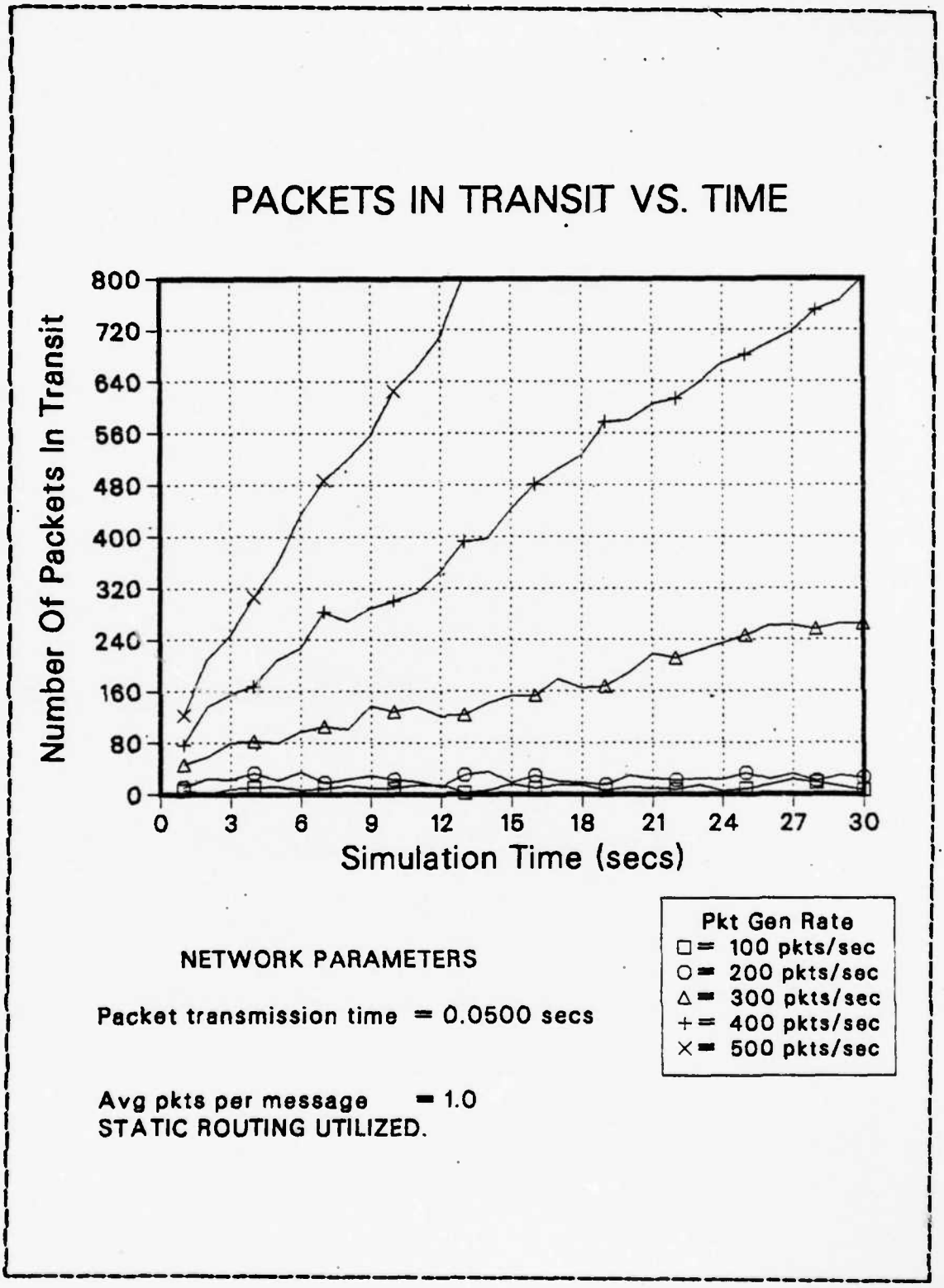


Figure 5.2 Static Routing Results

TABLE VI

Simulation Validation using Little's Result

λ (pkts/sec)	$\bar{T}$ (sec)	Simulation $\bar{E}(n)$ (pkts)	Little's $\bar{E}(n)$ (pkts)
100	0.01010	10.306	10.100
200	0.12108	24.250	24.216

For the lower levels of traffic intensity in which the network was stable, Little's Result was used to verify the operation of the model itself. Table VI gives a comparison of results obtained from the simulation run and using Little's Result in which the close agreement between both can be seen.

D. DYNAMIC ROUTING RESULTS

During the dynamic routing test phase, those parameters which would effect the Yen algorithm performance were first identified. The next step was to isolate one parameter at a time and by varying that value, establish its contribution to the algorithm. Finally, after arriving at a "working" set of parameters which provided acceptable performance, the network was subjected to a wide range of situations in order to thoroughly test its capabilities. Of particular

interest, was the algorithm's ability to handle wide ranges of traffic intensities, unbalanced traffic distributions, and timing errors due to queued updates and clock inaccuracies.

The primary parameters affecting the operation of the Yen algorithm are

1. Method of determining the Link Weights
2. Frequency of Update Generation

These points are not unique to this algorithm but are actually common to all implementations of distributed routing algorithms.

The object of this section was to develop a system of setting link weights which would minimize the average number of packets in the network. In doing so the mean packet delay and the average link queue size should also be minimized.

1. Determining the Link Weight Function

Previous research conducted at NPS has used a variety of methods for determining link weights. One scheme presented by Heritsch [Ref. 19] derived the weight by taking the average queue size over a period of time. Another method used by Tritchler [Ref. 27] was incorporated within a

Time Division Multiple Access packet radio environment. In this case the links were provided by radio transmissions. The weight of a link was chosen to be a function of both the attenuation of the radio link and the number of time "slots" being used on the link.

In minimizing the average number of packets within the system, we first investigated weighting methods which were related to the queue size of a link. This method appeared attractive due to its simplicity of implementation and the direct relationship between queue size and packet delay.

In all a total of five different link functions were eventually developed and tested. They were produced in a progressive fashion with the final method proving to be significantly better than the other four. In understanding the characteristics of each method and the rational which led to Method 5, the simplest of the link weight functions will be explained first.

a. Method 1: Queue-Size-Now

Method 1 assigns the weight to a link based upon the number of packets which are in the link queue at the time of updating. This method is referred to in the program as BY.QSIZE.NOW. The link function is given simply as

$$W(I,J) = Q(I,J)$$

where W(I,J) is the weight of link (I,J) and Q(I,J) is the size of link queue at the moment of updating. The simulation was run and the results are given in Figure 5.3. The results indicate that even with the higher levels of traffic intensity that the network was still stable. However, as shown in Figure 5.4, the average link utilization was approaching saturation for these high levels. This high utilization was due, in part, to packets taking more hops per trip than may be optimal. This is caused by the manner in which the link function determines the link weight.

Since the link weight function uses only the queue-size-now, the best paths which the Yen algorithm arrive at are those which avoid large queues. Therefore a packet may take a long trip through the network in its attempt to avoid high cost trips. These longer trips translate into higher link utilizations than would be obtained through a least hop scheme. By minimizing queue size, the average number of hops per packet trip increases and thus the link utilization increases.

Method 1 is sensitive only to what is happening in the network at the present moment. The next two methods

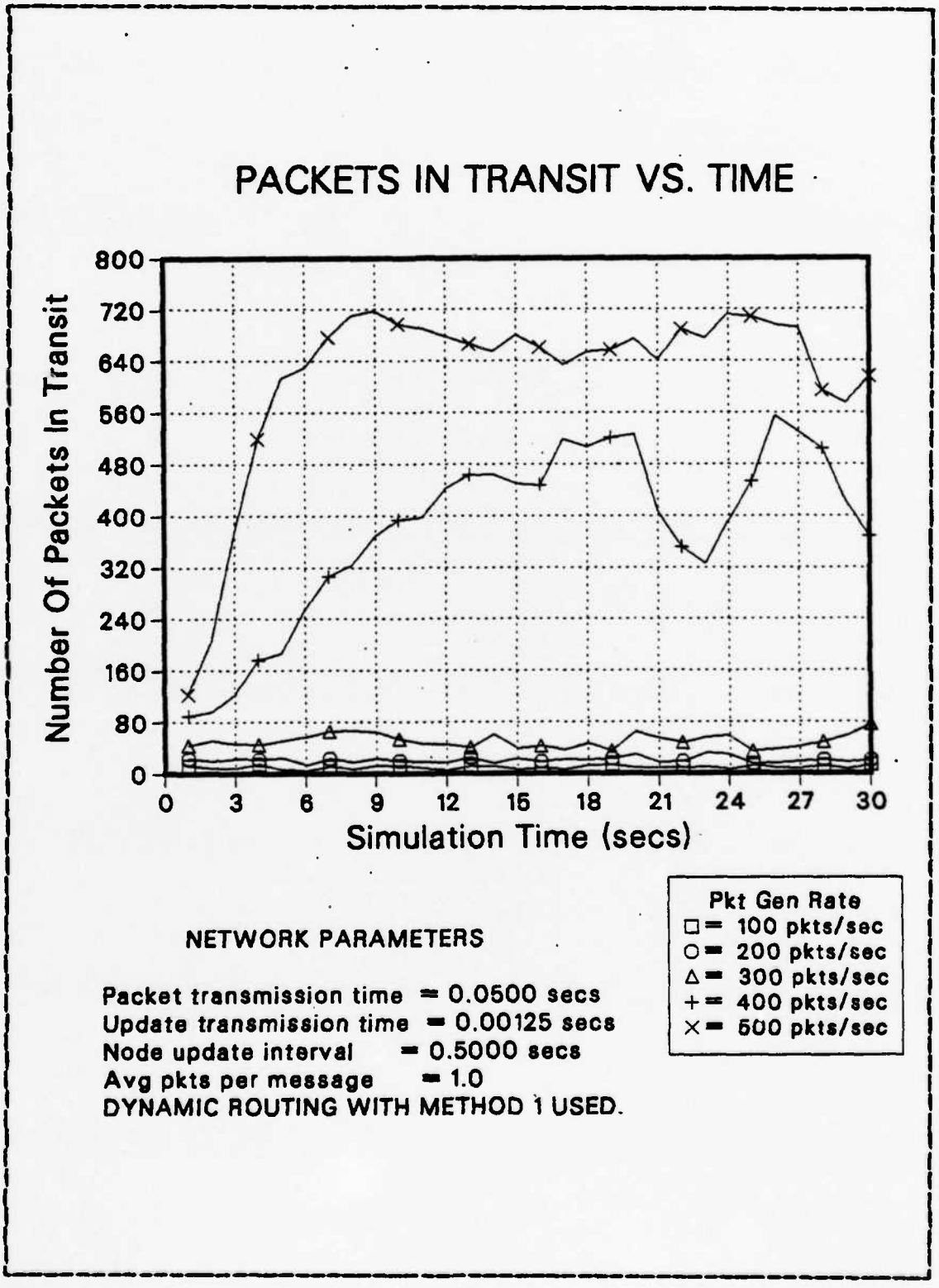


Figure 5.3 Method 1 Results

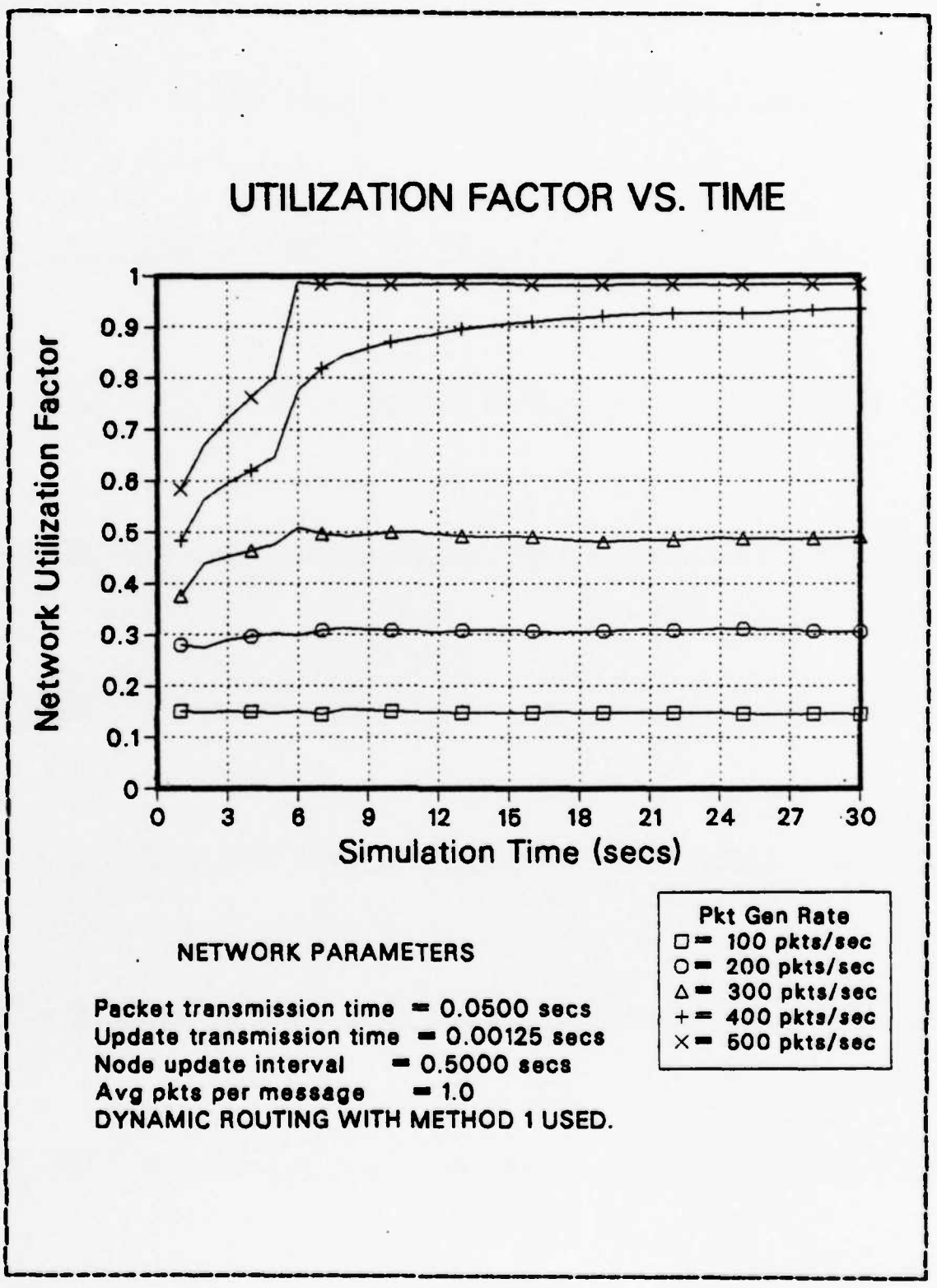


Figure 5.4 Method 1 Link Utilization

were developed to see if taking into account the past condition of the link queue would favorably effect the performance.

b. Methods 2 and 3: Queue Size Averaging

Methods 2 and 3 were developed to reduce the high utilization which appeared to be caused by link weights which were changing too rapidly in Method 1. Instead of looking just at the queue size at the time of updating, the average queue size over a past time interval was observed. By using an averaging scheme, it was felt that the best paths would not fluctuate as quickly and thus excessively long trips would be reduced. Both methods incorporated a sliding window technique over which the average queue size was evaluated (Figure 5.5). In each method a "sub-weight" was calculated for every window used in the entire period. The time duration of each window and the number of total windows used were both parameters in the simulation. The link weight was computed as the average of the sub-weight values. Methods 2 and 3 only differed in the manner in which the sub-weights were determined. Method 2 calculated the sub-weights as being the queue size at the start of the time window. This method is referred to in the program as

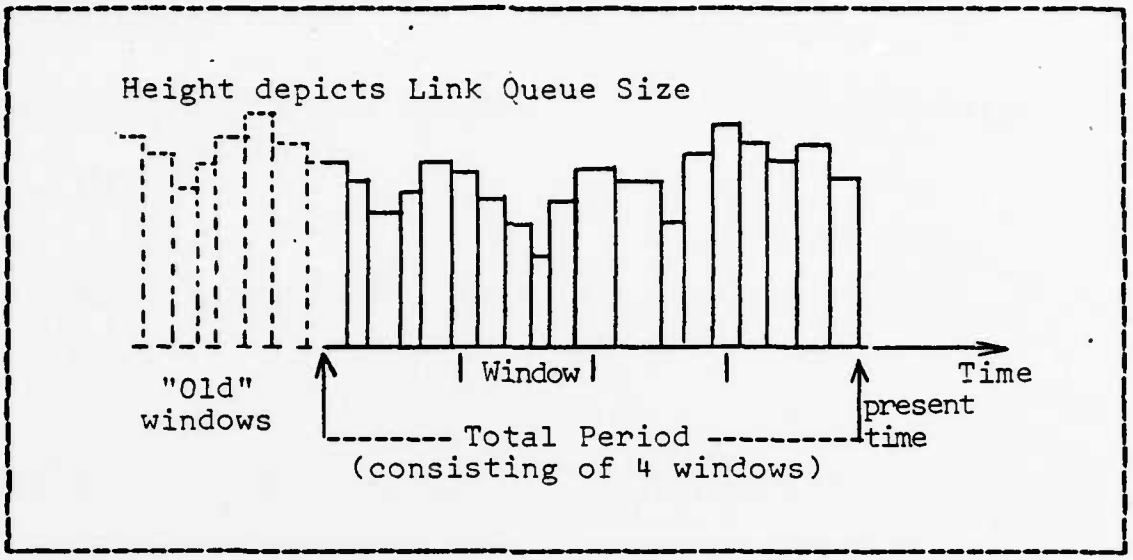


Figure 5.5 Windowing Technique

BY.PAST.QSIZE. Method 3 determines sub-weights as the average queue size for the time window and is referred to as BY.PAST.QAVG in the program. The results for simulation runs using Methods 2 and 3 are given in Figures 5.6 through 5.9

Both methods were unstable at moderate traffic intensities. It was found that as window size and the number of windows increased, the network performance deteriorated. The use of past average queue size does not appear to be a useful indication of the status of the network for use in best path determination.

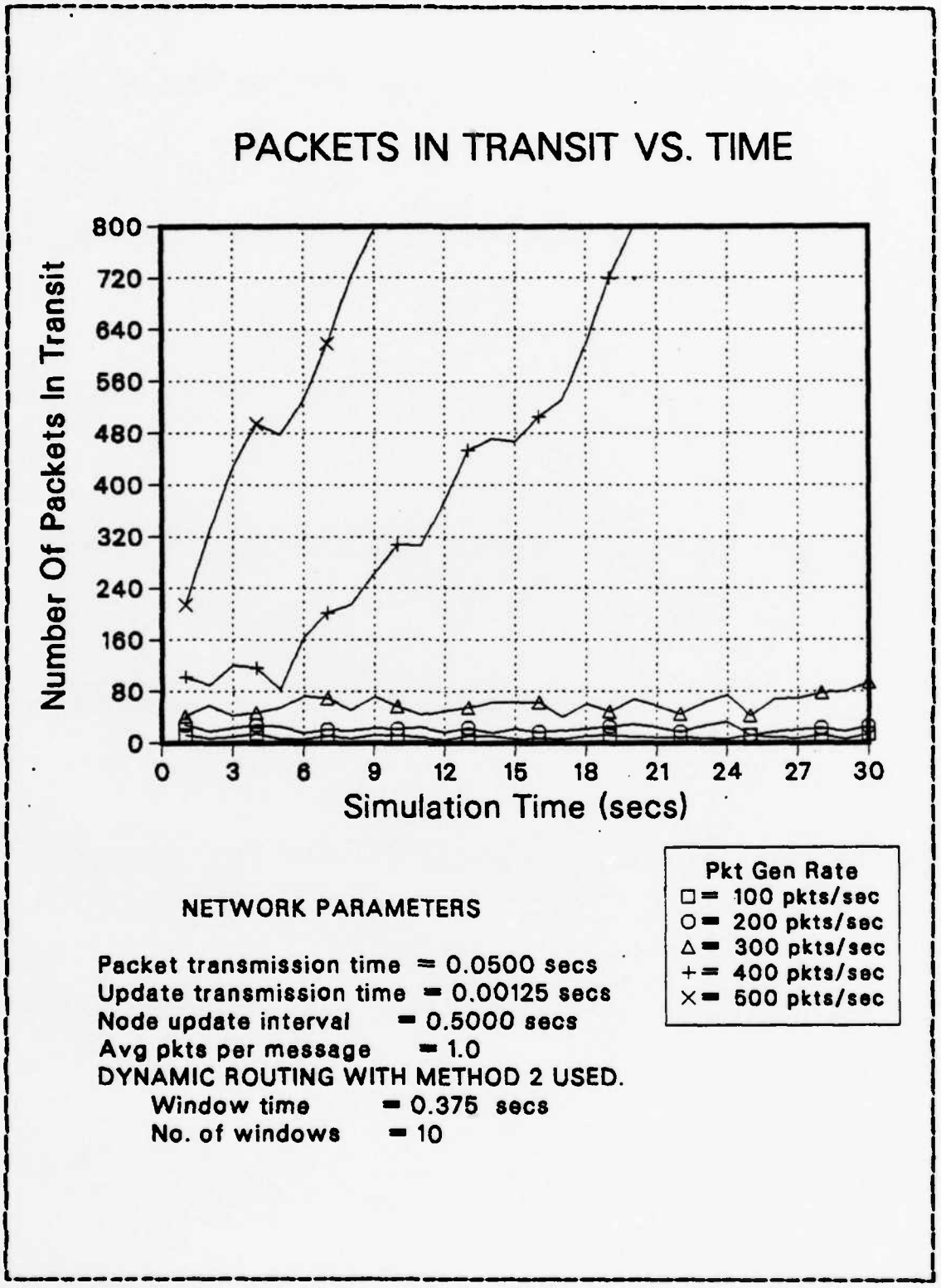


Figure 5.6 Method 2 Results

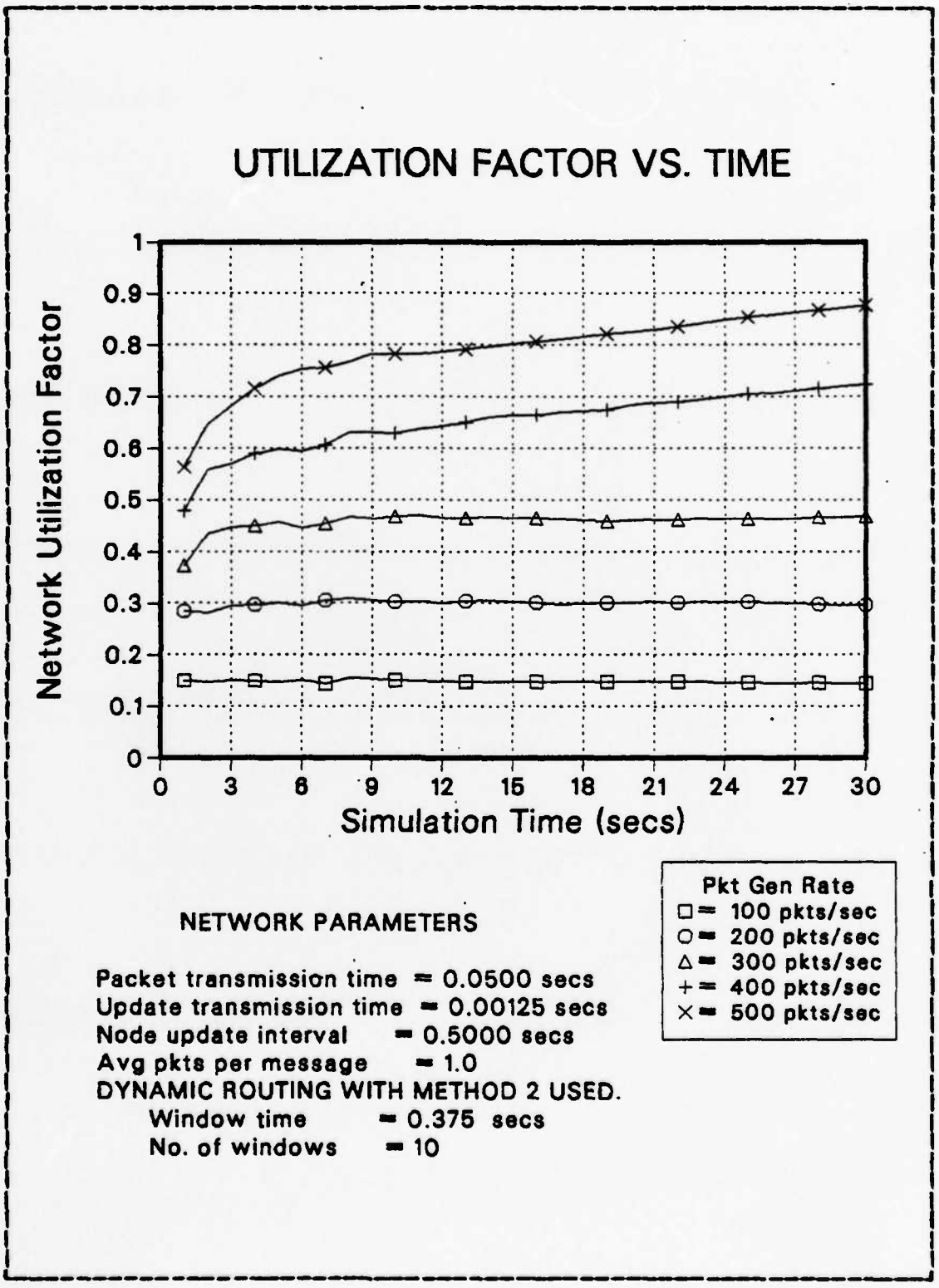


Figure 5.7 Method 2 Link Utilization

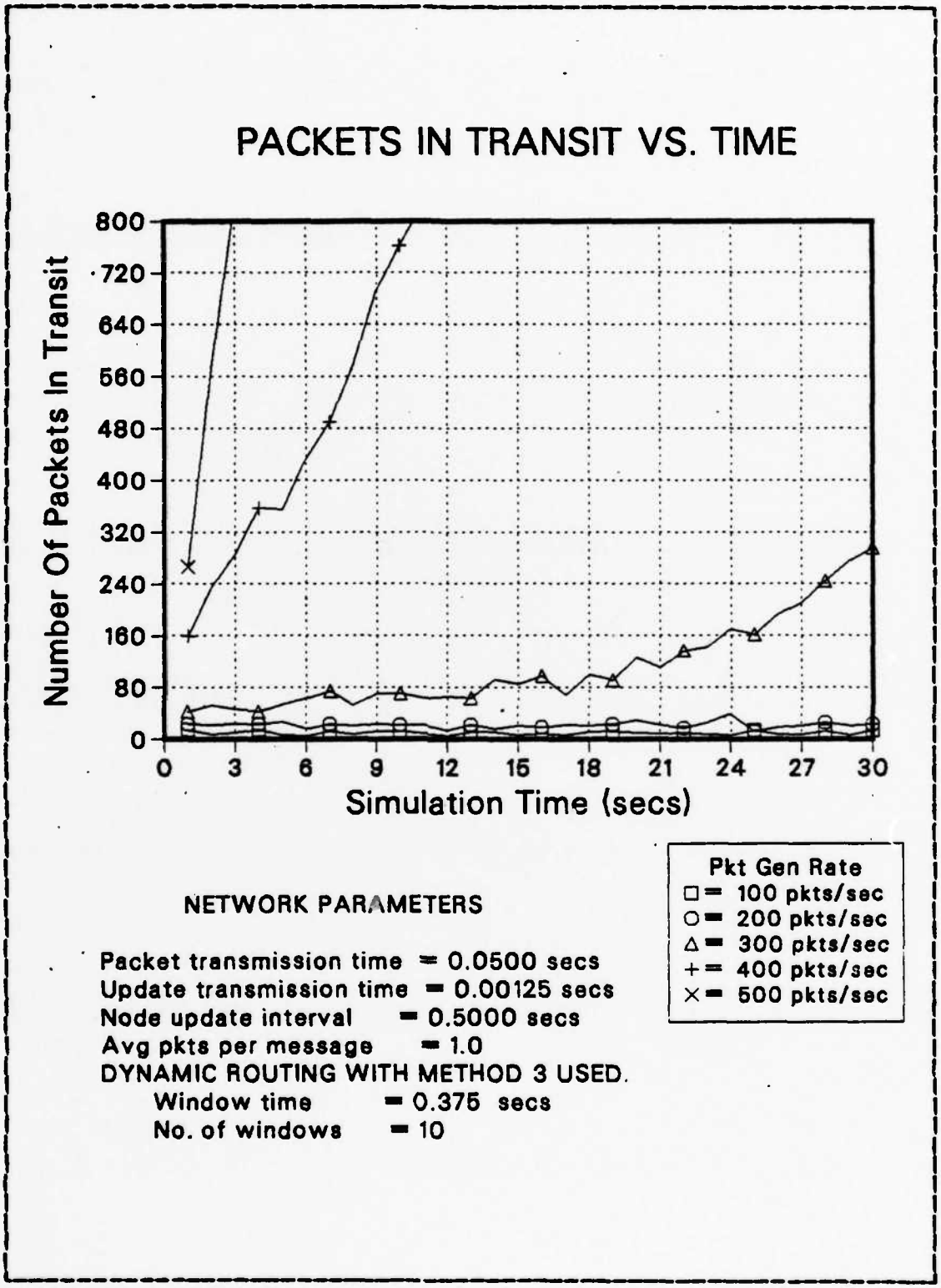
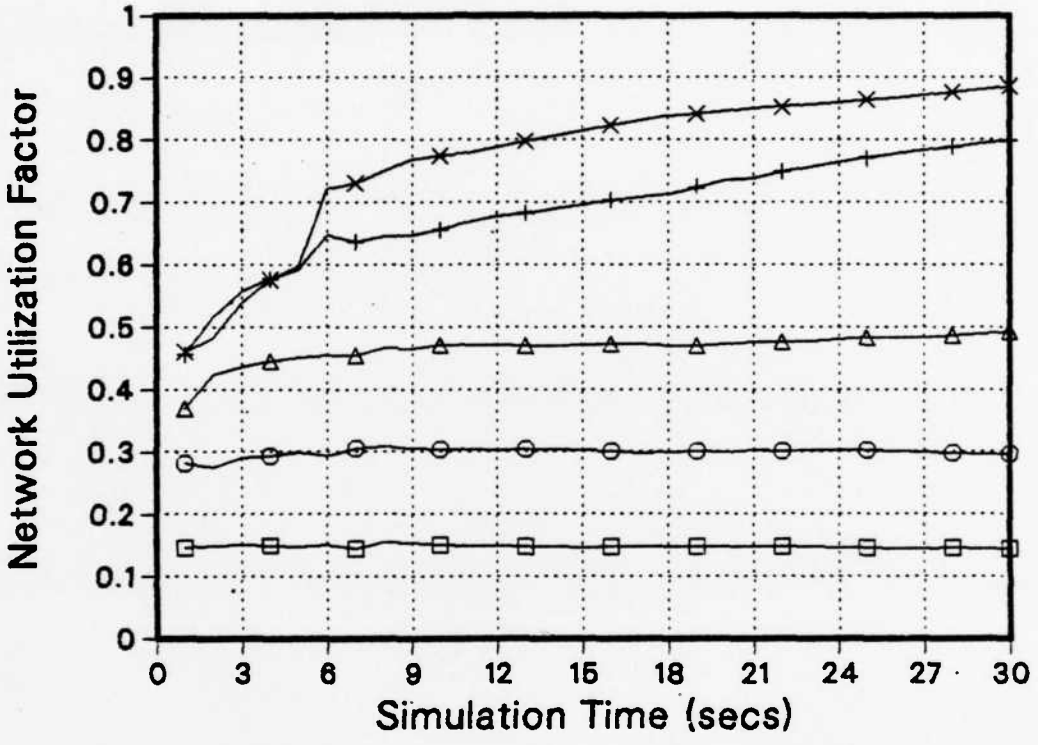


Figure 5.8 Method 3 Results



NETWORK PARAMETERS

Packet transmission time = 0.0500 secs
Update transmission time = 0.00125 secs
Node update interval = 0.5000 secs
Avg pkts per message = 1.0
DYNAMIC ROUTING WITH METHOD 3 USED.
Window time = 0.375 secs
No. of windows = 10

Figure 5.9 Method 3 Link Utilization

c. Method 4: Link Utilization

At this point, due to the lack of success from the queue averaging schemes, a link weight function was selected which would relate directly to the utilization of the link. Method 1, though stable, produced very high utilization. By using a criterion for minimization based upon link utilization it was hoped that stable performance could be achieved.

The utilization factor of a link is the ratio

$$\text{Utilization} = \frac{\text{Busy Time}}{\text{Busy Time} + \text{Idle Time}}$$

and is therefore a number between 0 (no traffic on link) and 1 (continuous traffic on link). Link utilization for an M/M/1 queue system is also related to the average queue size by

$$E[n] = \frac{\text{Util}}{1 - \text{Util}}$$

where E[n] is the average number of packets in the queue. A plot of this relationship is given in Figure 5.10. When the link utilization is below 0.5, the average number of packets in the link queue is less than 1. For utilization factors greater the 0.5, the number increases rapidly. This formula

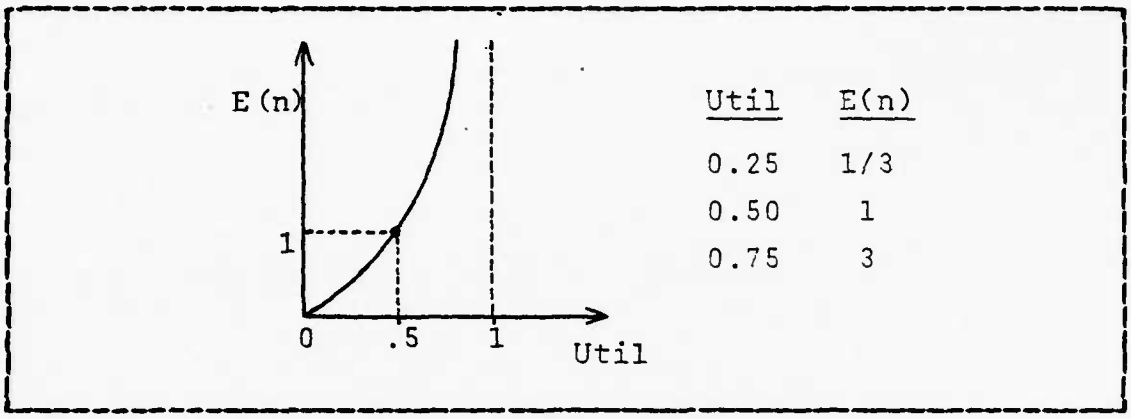


Figure 5.10 Average Queue Size vs. Utilization Factor

for the average number of packets as a function of utilization was used for the link weight calculation. However since this function had no upper bound, a maximum value was placed upon it for link utilizations greater than a particular value.

The formulation which became Method 4 has a link weight function of

$$W(I,J) = \begin{cases} 1 + \dfrac{Util}{1 - Util}, & 0 \leq Util < 0.95 \\ 21, & Util \geq 0.95 \end{cases}$$

The value of the link utilization is calculated using the same window scheme used in Method's 2 and 3. The value of the sub-weight is the utilization factor of the link over that window period. The sub-weights are averaged over the total number of past windows used.

Method 4 is the first scheme presented which has some of the hybrid features discussed in Chapter II. This link weight function behaves differently depending upon the link utilization within the network. When the network is lightly loaded such that link utilizations are less than 0.5, all link weights will have values between 1 and 2. This means that the unity term included in the link weight function will dominate the utilization factor term and thus the routing will be a least hop system. As seen previously, the least hop scheme is an excellent best path choice when the system is lightly loaded.

When traffic intensity increases such that link utilizations increase above 0.5, the link weight function will be dominated by this term and the best paths will be chosen to avoid highly utilized links. The addition of the unity term has in effect provided a stabilizing effect which keeps the link weights from churning or varying rapidly under light traffic conditions. Results of Method 4 simulations are given in Figure 5.11. This plot is quite similar to the Method 1 results. However, the average link utilization seen in Figure 5.12 is greatly reduced using Method 4.

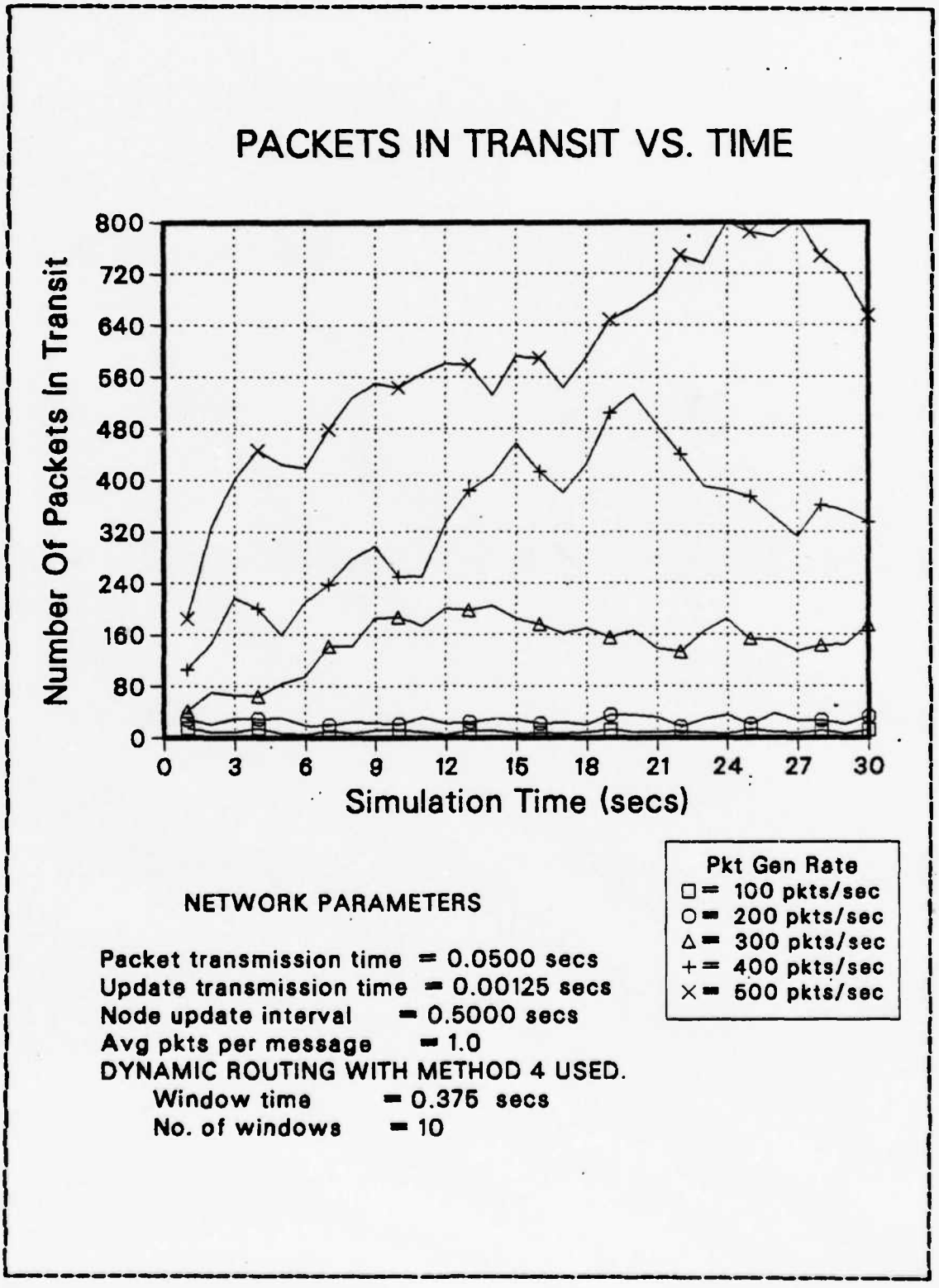


Figure 5.11 Method 4 Results

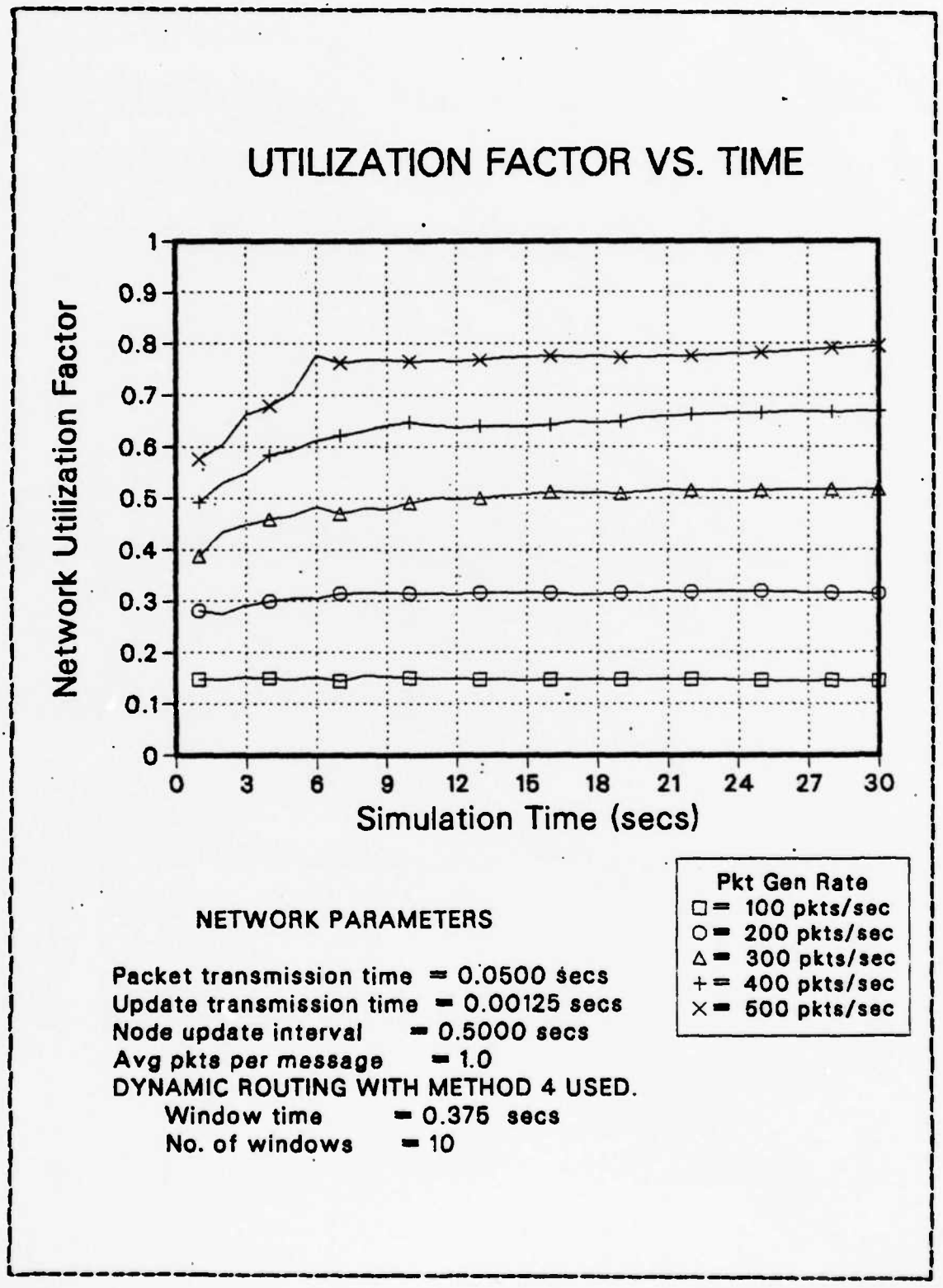


Figure 5.12 Method 4 Link Utilization

E. THE "COMBINATION" LINK WEIGHT FUNCTION

In the preceeding paragraphs, two schemes were found which provided stable results. Method 1 determined link weights based upon the queue size at the moment of updating while Method 4 calculated link weights as a function of average link utiization. Method 5 was developed as a linear combination of these two methods. The link weight function used in this "combination" scheme is

$$W(I,J) = [QU.FACT * Q(I,J)] + 1 + \frac{Util}{1 - Util}$$

QU.FACT is the Queue Scaling Factor used to observe the effect of changing the relative influence of the term associated with the queue-size-now to the utilization term. The same window operation was used in this formula as appeared in Method 4. In addition, the same upper bound was placed on this term as was given in Method 4.

The simulation results using Method 5, which is called the RHO.COMBINE method in the program, is given in Figures 5.13 through 5.16. QU.FACT equalled unity during these runs. Latter sections present the effects of variations in QU.FACT on algorithm performance. As seen from the plots, the result of combining these methods has vastly improved

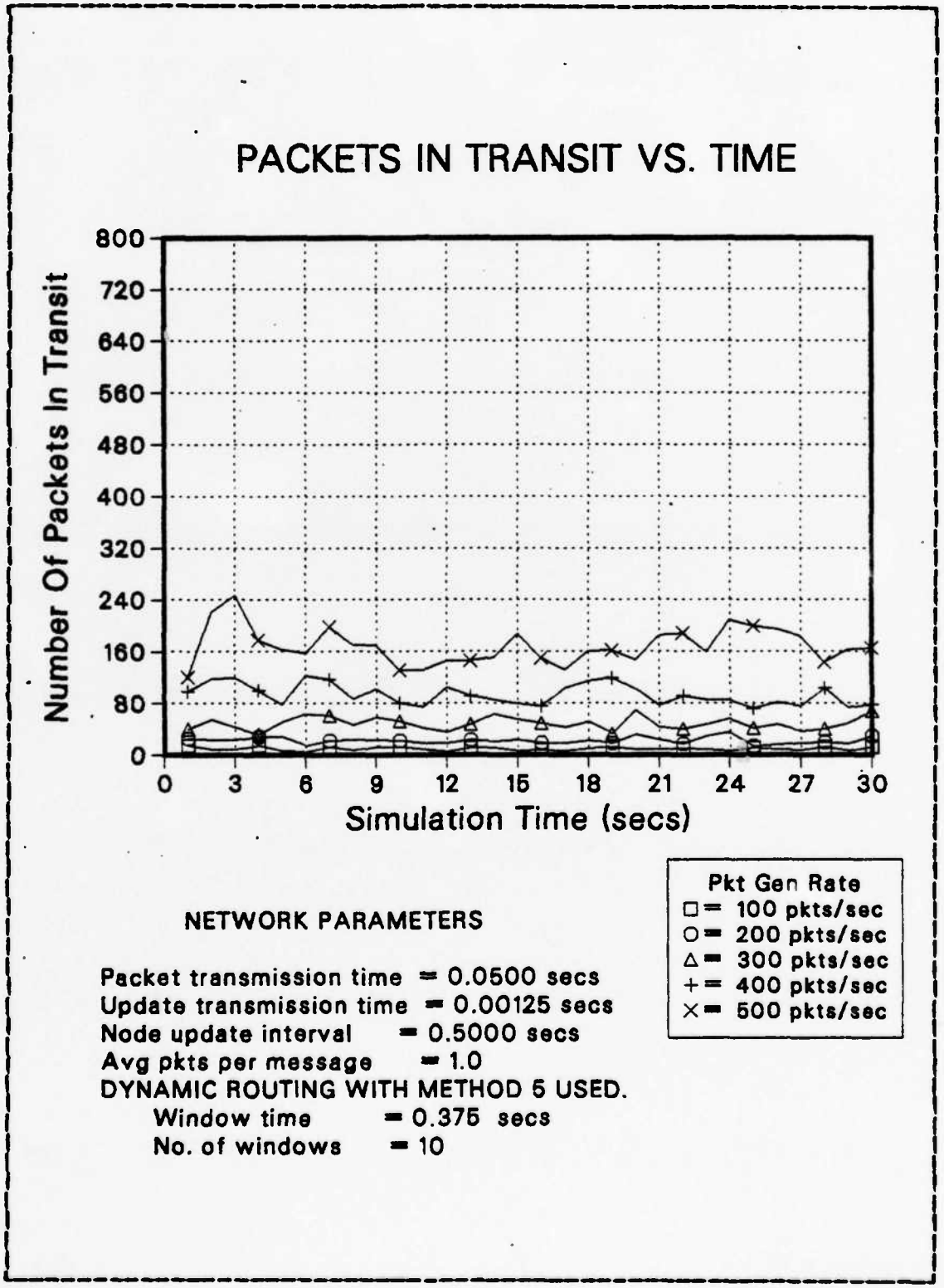


Figure 5.13 Method 5 Results I

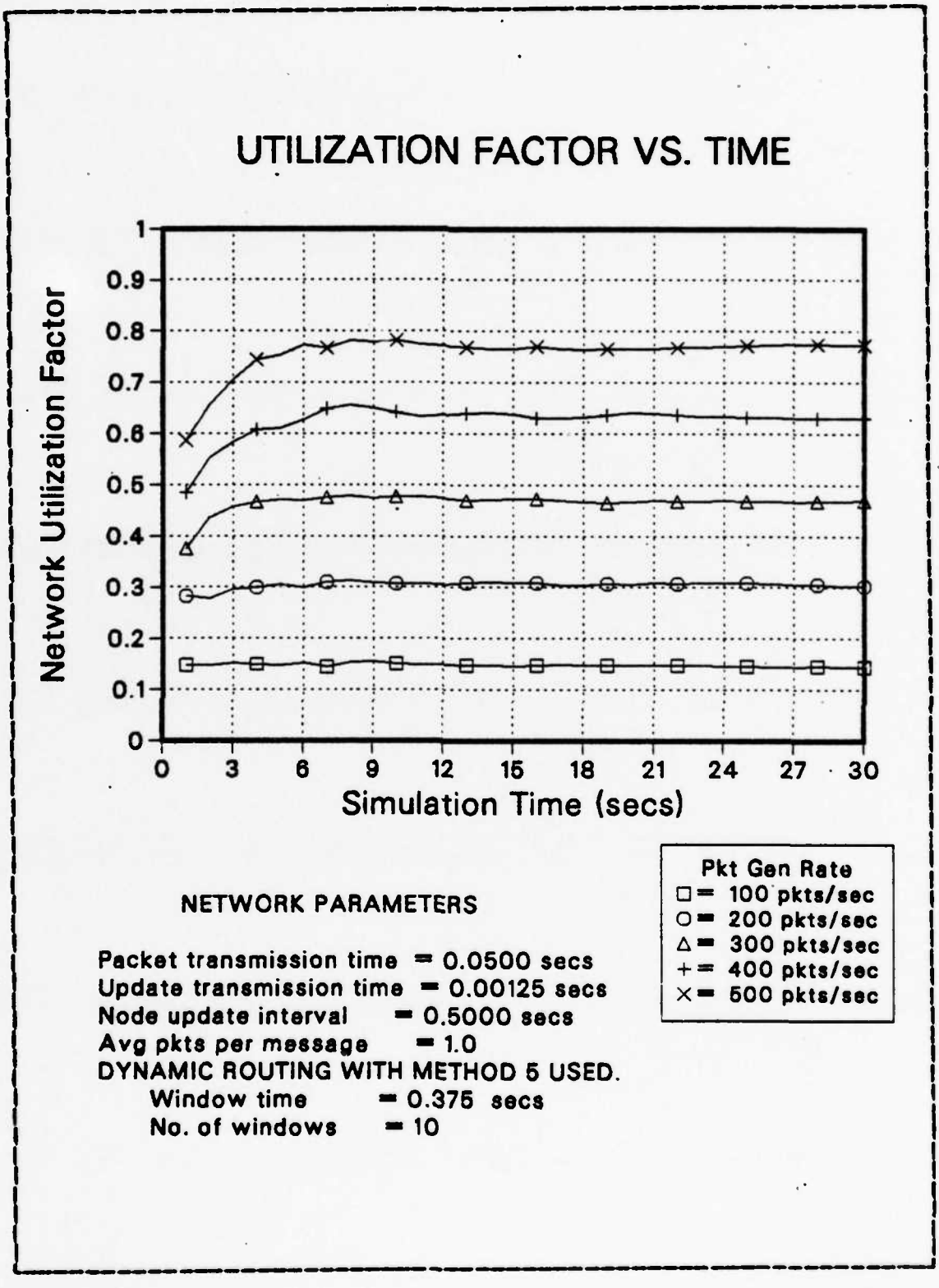


Figure 5.14 Method 5 Link Utilization I

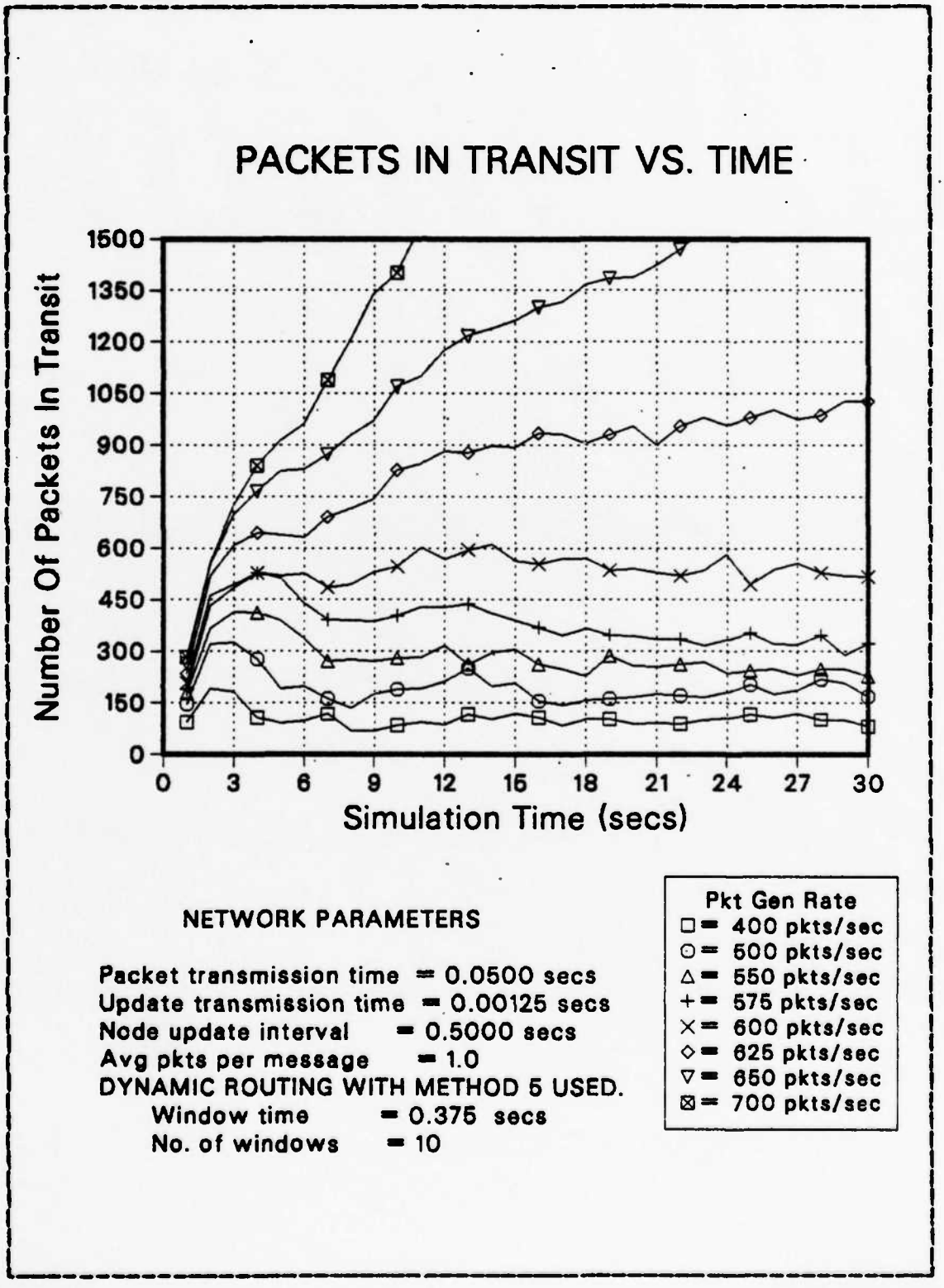


Figure 5.15 Method 5 Results II

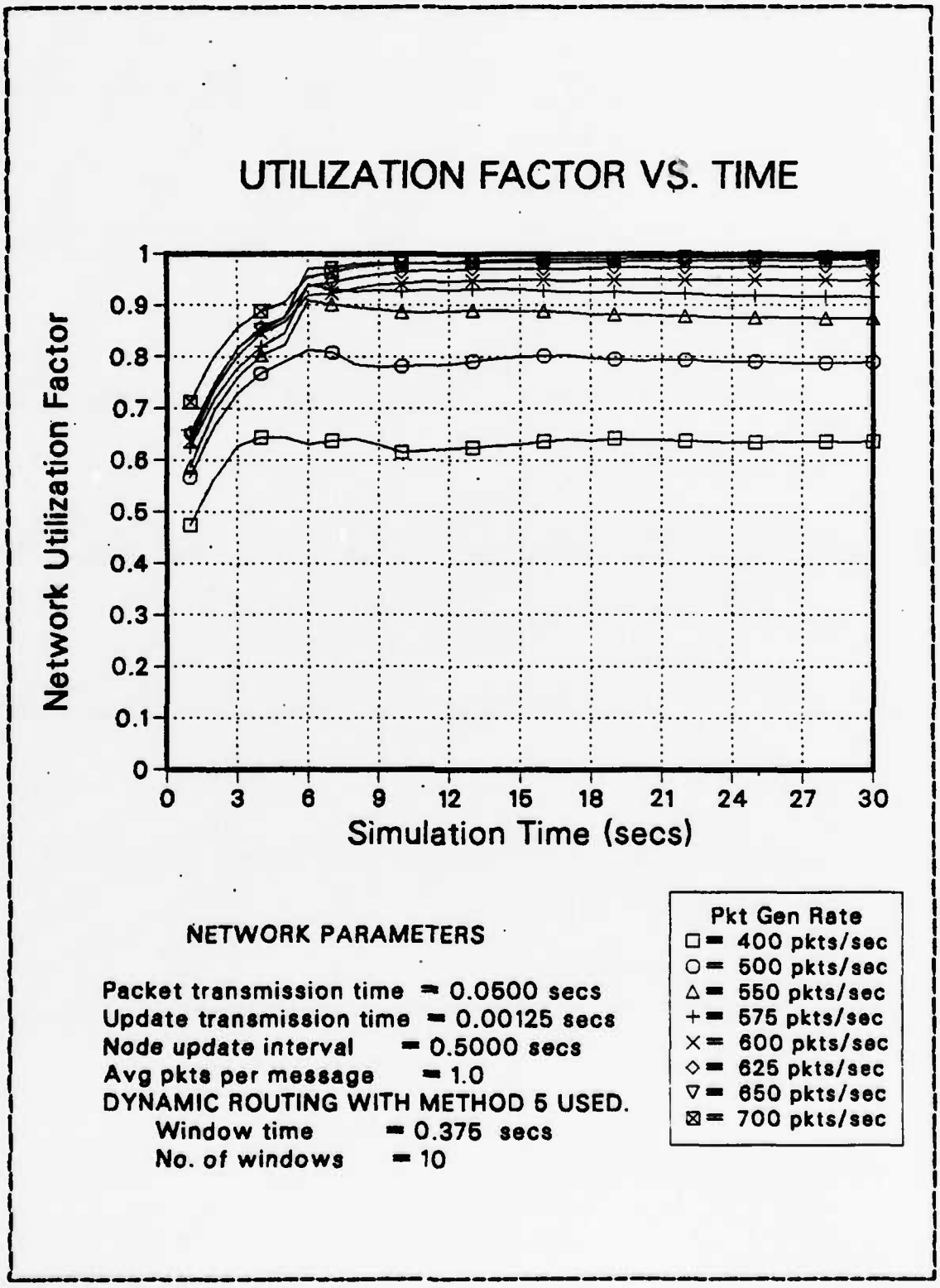


Figure 5.16 Method 5 Link Utilization II

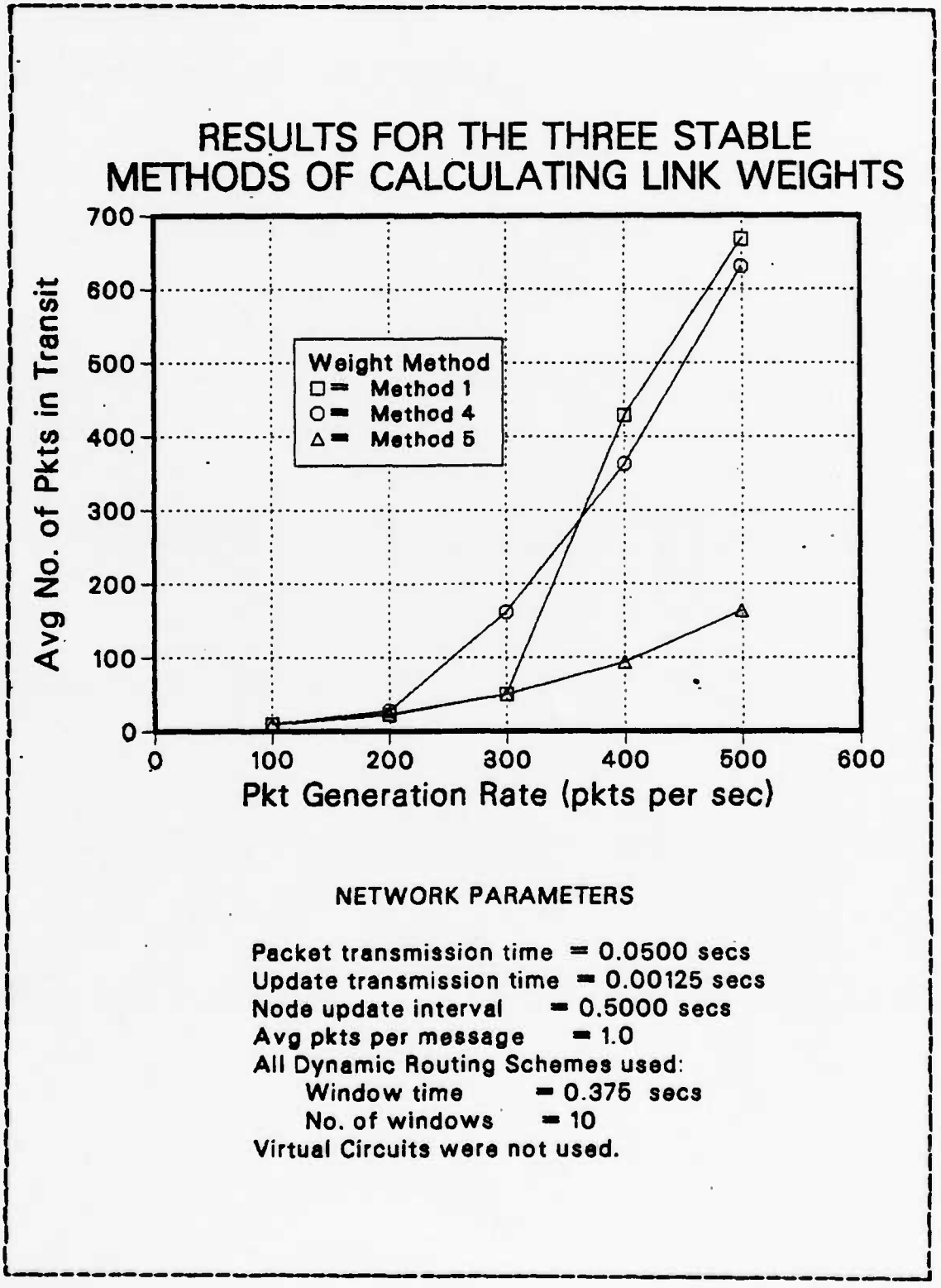


Figure 5.17 Link Weight Methods : Stable Performance

the perfcrmance of the algorithm. Figure 5.17 compares the performance of the three tested link weight functions which produced stable network performance. The dramatic improvement achieved with the final method can be readily seen. These results indicate that by linearly combining Methods 1 and 4, the best of both worlds in a link weight function is achieved.

The explanation as to why Method 5 is so successful seems to center about two ideas which correspond to the two terms of the function. The first is the link weight term involving the recent history of utilization. This factor can be viewed as providing as estimate of conditions at nodes in nearby areas of the network without actually receiving direct information from those nodes. This principle can best be understood through an example. Suppose that a link has been heavily utilized with a utilization factor for the current window period of 0.75. Additionally at the moment of best path calculation the link queue has just emptied its last packet. If the link weight function is based only upon the current queue size then the weight of the link would be minimal. However, the terminating node of the link may now be "backed up" with traffic due to the

previcus period cf high utilization. By incorporating link utilization into the weighting function, the conditions at the terminating nodes for the links can be estimated. This ability tc see a little way "over the horizcn" enables the routing algorithm which uses link utilization to steer traffic away frcm areas which have recently received high volumes cf traffic.

The seccnd idea, which adds "balance" to the previcus point, is that looking at only the link utilization does not take into account the immediate impact which traffic originating frcm a node has upon best path routing. When link weights are determined from utilization alone, situaticns can occur where links of similar usage can suddenly have very different queue sizes due to the origination of messages at the node. As an example, suppose that two outgoing links of a node have had utilization factcrs of 0.25 cver the current window period and presently their link queues are empty. At this point the node accepts 20 messages which it, being the originator, places intc the queue of cne of the links based upon the routing table. Now a length cf time propcrtional to the window period will pass before the utilizatien of this link becomes different enough

so as to cause a best path change. This time lag is due to the effect of using a calculation based upon an average over a period of time. By including the term related to queue-size-now in the link weight function of Method 5, the link weights can reflect the activity of a node as an originator of traffic.

Method 5 has some important distinctions from Chou's hybrid link weight function mentioned in Chapter II. Chou's quadratic function used only the queue size term. Method 5, on the other hand, employs the additional term related to the immediate past history of utilization of the links. It is this combination of terms which appears to give the algorithm such a useful estimation of the network status.

1. Results of Varying Window Period

The link utilization term used in the link weight function of Method 5 was calculated over an interval of time called the window period. For this discrete simulation, since updates occurred at random times, the window period was modeled such that during operation it appears as a continuously sliding time window. To accomplish this continuously sliding effect, the total window period was divided into a number of smaller sized windows. If ten

small windows were chosen in the previous example to model the one second window period, each small window would be 0.1 seconds long. Therefore every 0.1 seconds the link utilizations would have been computed at the node with only the 10 most recent windows being used during updating.

The effects of varying the window period is shown in Figure 5.18. The algorithm performance improved as the window period (which equalled the product of the number of windows and the window size) increased to a certain point after which lengthening the period did not achieve substantial performance improvements. When a smaller window was used, a greater number of windows were required to achieve the same performance as with larger windows. However, the total window period required was shorter than when using the larger windows. This is especially noticeable from the plot in the case where 13 windows of duration 0.125 seconds had the same average number of packets in transit as 4 windows of duration 0.675 seconds. The total window period for the shorter windows was 1.625 seconds while the longer windows had a total period of 2.7 seconds. For the remainder of the testing 10 windows of length 0.375 seconds were used resulting in a total window period of 3.75 seconds.

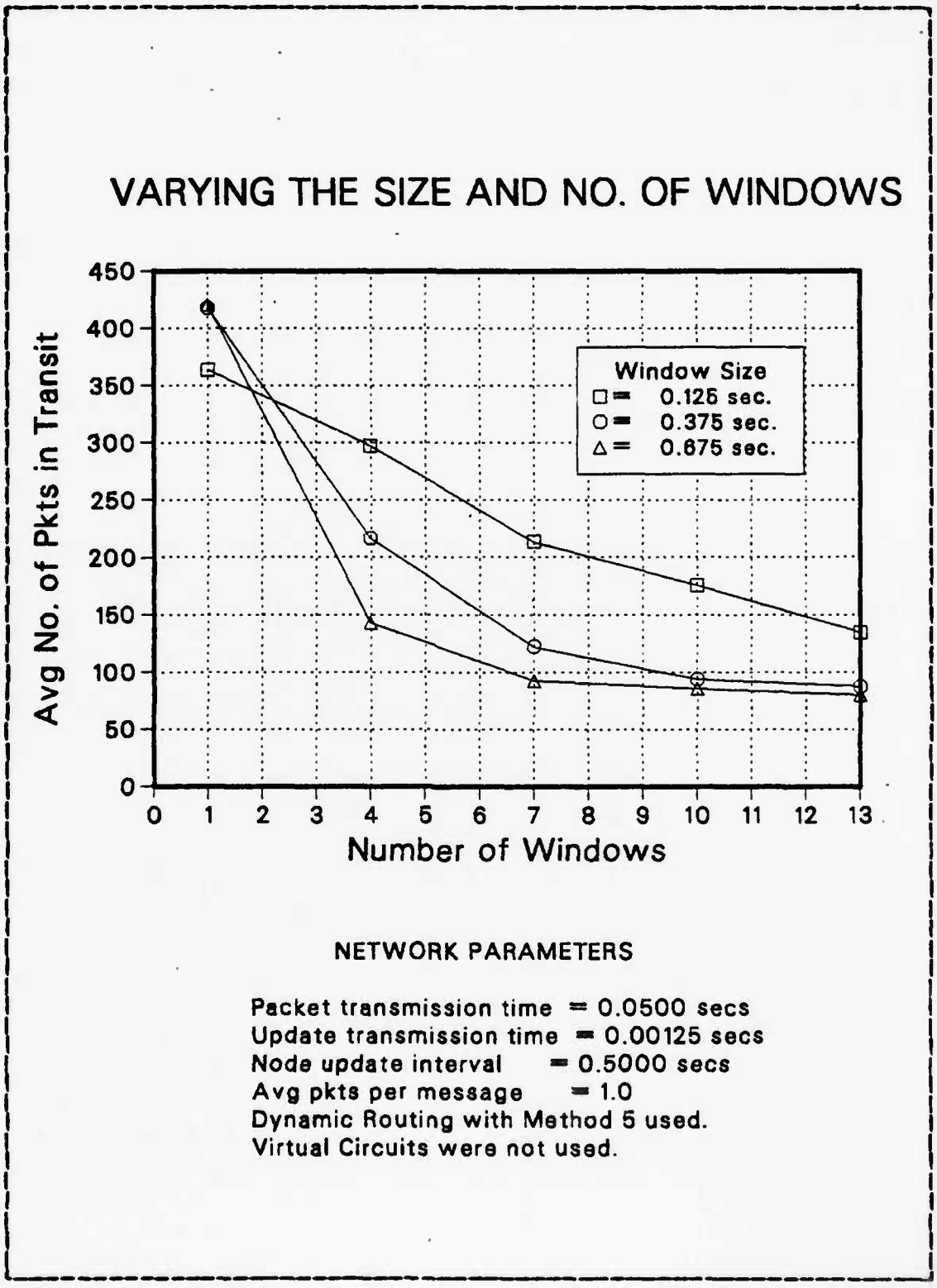


Figure 5.18 Varying the Window Period

2. Results of Varying the Queue Factor

The Queue Factor is the scaling multiplier for the queue-size-now term in Method 5. By varying this value, either the queue-size-now term or the utilization term can be emphasised. When the Queue Factor is set to zero the queue size term drops out resulting in simply Method 4. By the same token, as the Queue Factor is made large, a Method 1 link weight function occurs with the utilization term being "swamped" by the queue-size term.

Figure 5.19 shows the network performance of the test network when the Queue Factor was varied from 0 to 4. As expected the results with the Queue Factor equal to zero are identical to those of Method 4. Similarly as Queue Factor increases, the network performance begins to degrade to that obtained using Method 1. The interesting result is the very substantial improvement in performance achieved when the Queue Factor had values near unity.

This result provokes a question concerning the sensitivity of the Queue Factor term upon the Method 5 link weight function. Since Queue Factor provides the balance between the queue size and utilization term, its sensitivity was tested in the following way.

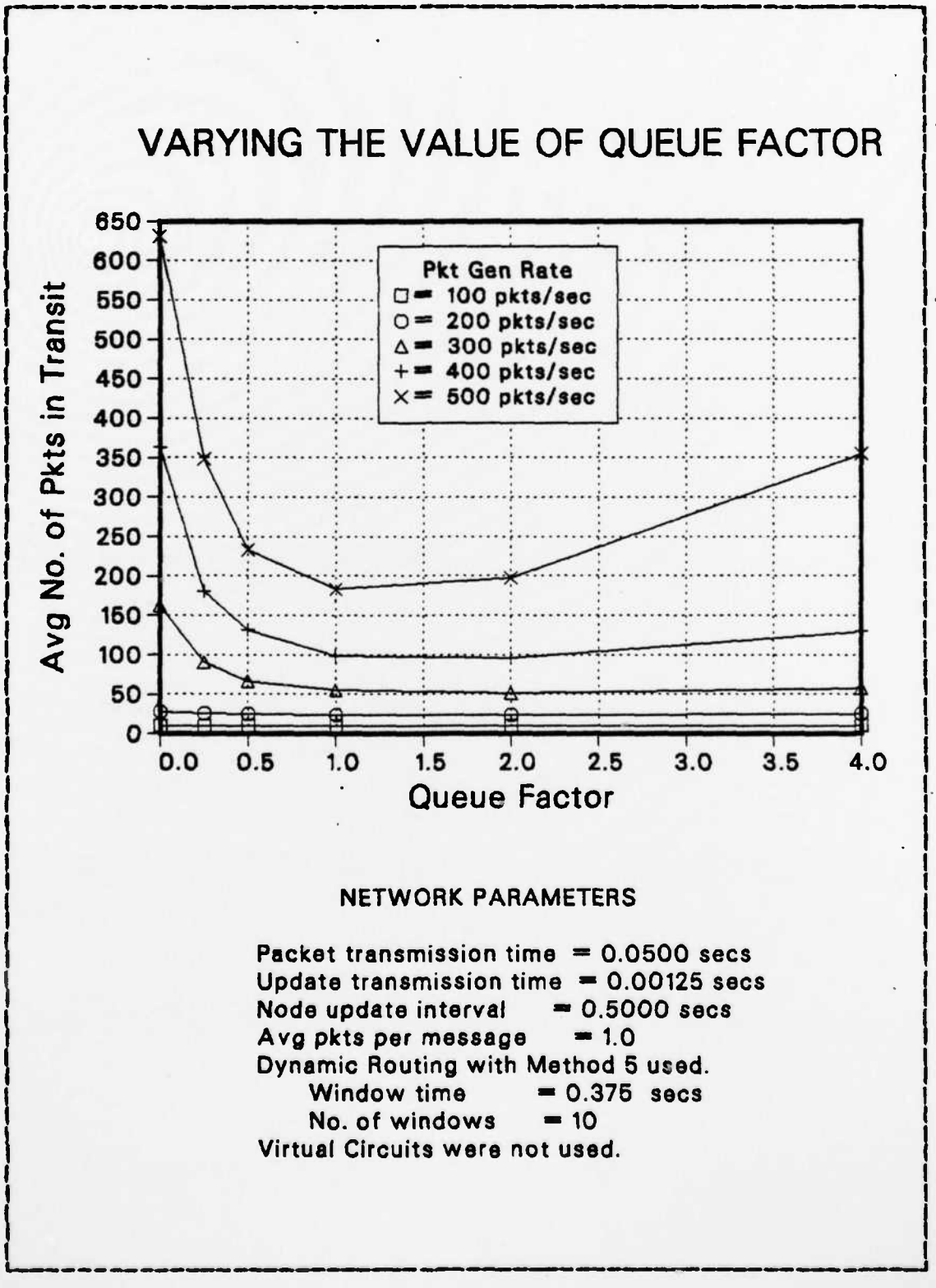


Figure 5.19 Varying the Queue Factor

If one were tc vary the length of a message packet, the same number of packets traveling over a link would cause different utilizatien factors. By the same token, if different packet sizes were used and the packet generatien rate intc the network was properly varied, one could maintain a particular average link utilization for simulation runs. This was the approach used in testing the sensitivity of the Queue Factor term. Three different packet sizes were used such that their transmissicn times were 0.0125, 0.0500 and 0.2000 seconds. The packet generation rate was set so that the average link utilization was the same for each packet size. In this manner the utilization term is held constant while the queue size term is allowed to take on different values.

The results when the average network link utilization was adjusted to 0.50 and 0.80 are given in Figures 5.20 and 5.21. For both these results the average number of packets within the network was normalized so that a single plot could show the effects. In the case where utilization was at 0.50 the network performance improved dramatically as the Queue Factor became greater than zero. The performance with the shorter packet of length 0.0125 continued to

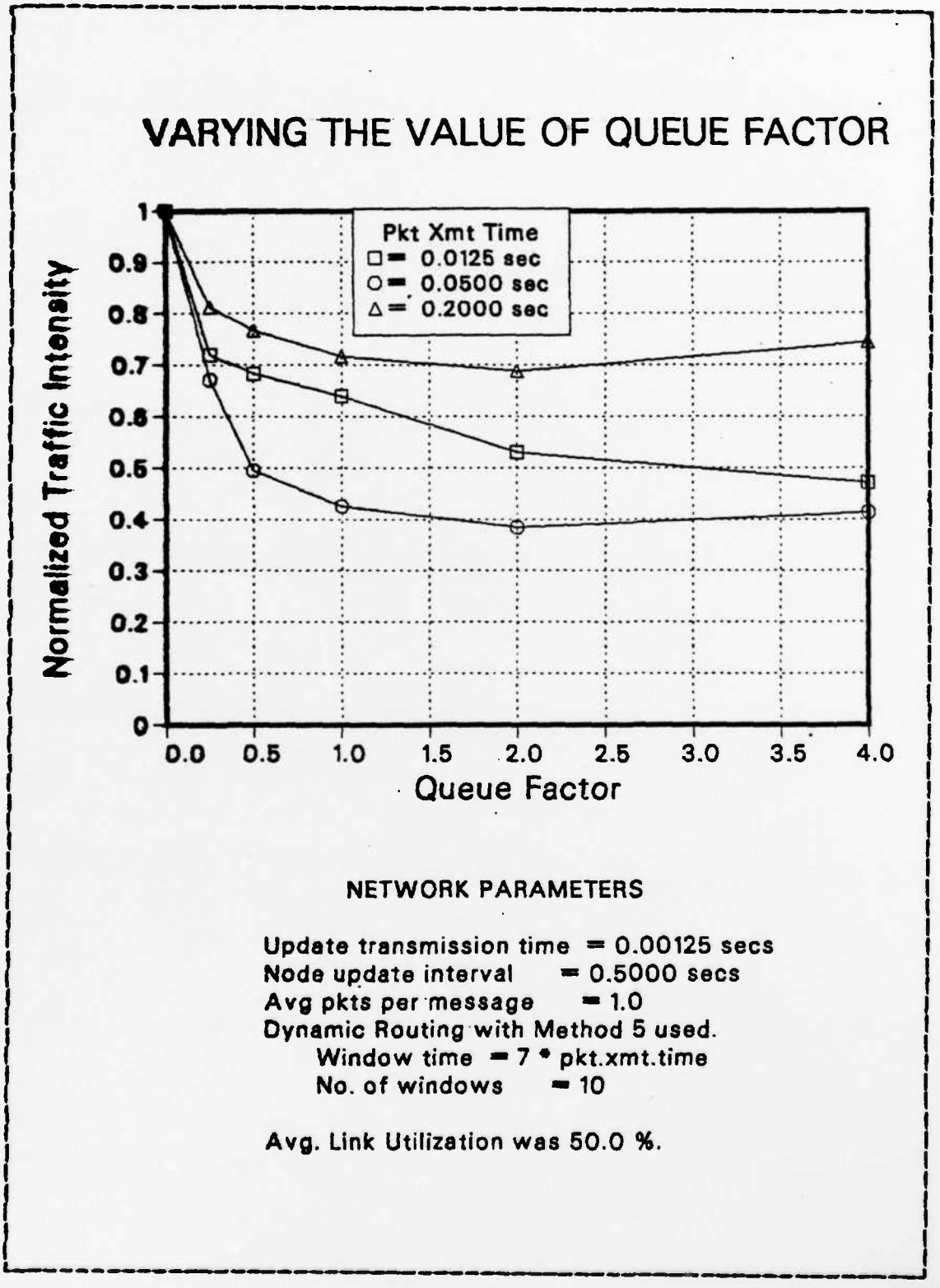
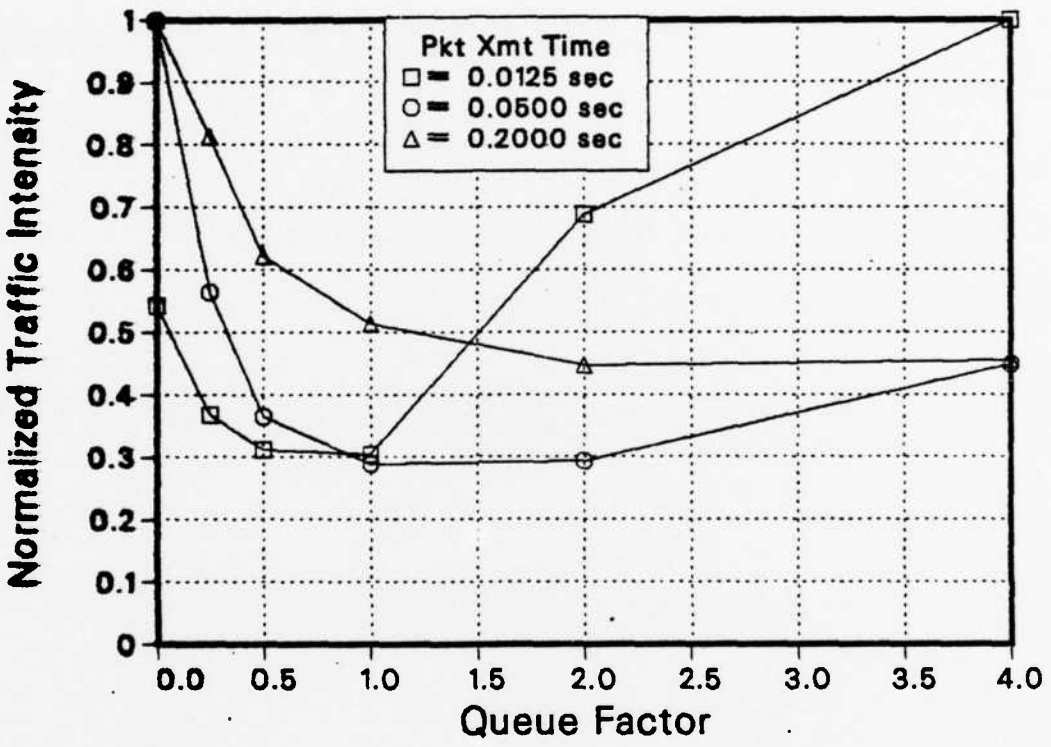


Figure 5.20 Queue Factor : 0.50 Utilization



NETWORK PARAMETERS

Update transmission time = 0.00125 secs
Node update interval = 0.5000 secs
Avg pkts per message = 1.0
Dynamic Routing with Method 5 used.
Window time = 7 * pkt.xmt.time
No. of windows = 10

Avg. Link Utilization was 80.0 %.

Figure 5.21 Queue Factor : 0.80 Utilization

improve yet was beginning to level off. The perfomance of the lcnger packets had both started to degrade as the Queue Factor reached a value of 4. When the utilization was increased to 0.80 the performance with all three packet sizes again improved as Queue Factor became greater than zero. Here it is apparent that the shorter packet performance degraded rapidly as the Queue Factor became greater that unity.

These results indicate that the Queue Factor can be changed to match a given network situation to a particular level cf performance. In addition, when link capacities are large in terms of the number of packets per second which can be transmitted over them, then the Queue Factor is best kept small so as not to cverpower the utilization term. In the remainder of the simulatien the Queue Factor was set to unity which produced satisfactcry performance for the packet transmissicn time of 0.0500 seconds.

3. Results cf Varying Node Update Interval

The node update interval is the time between consecutive updates being generated by a single node. The ncde update frequency is the inverse of this interval. It would appear that by reducing the time interval between updates,

the algorithm would perform more effectively since best paths would reflect changes in network conditions more rapidly. However problems occur when the increased link utilization due to update messages begins to interfere with normal message traffic.

As discussed before, the link utilization caused by update messages is a function of the number of nodes in the network, the update interval and the update transmission time. In the thirteen node test network where the update interval is 0.5 seconds and the update transmission time is 0.00125 seconds, the utilization per link due to this overhead is only 0.01625. Such a low value does not "load down" the network substantially enough to interfere with regular message transmissions. This is demonstrated in Figure 5.22, which indicates that performance improves as the frequency of updates increases. The size of an update packet, though, is relatively small being only 1/40th the size of a message packet. This small update is in keeping with the very simple information which the Yen algorithm requires and is by no means unrealistic. Since the link utilization is proportional to the number of nodes in the network, however, the performance of the network under conditions of higher overhead must still be evaluated.

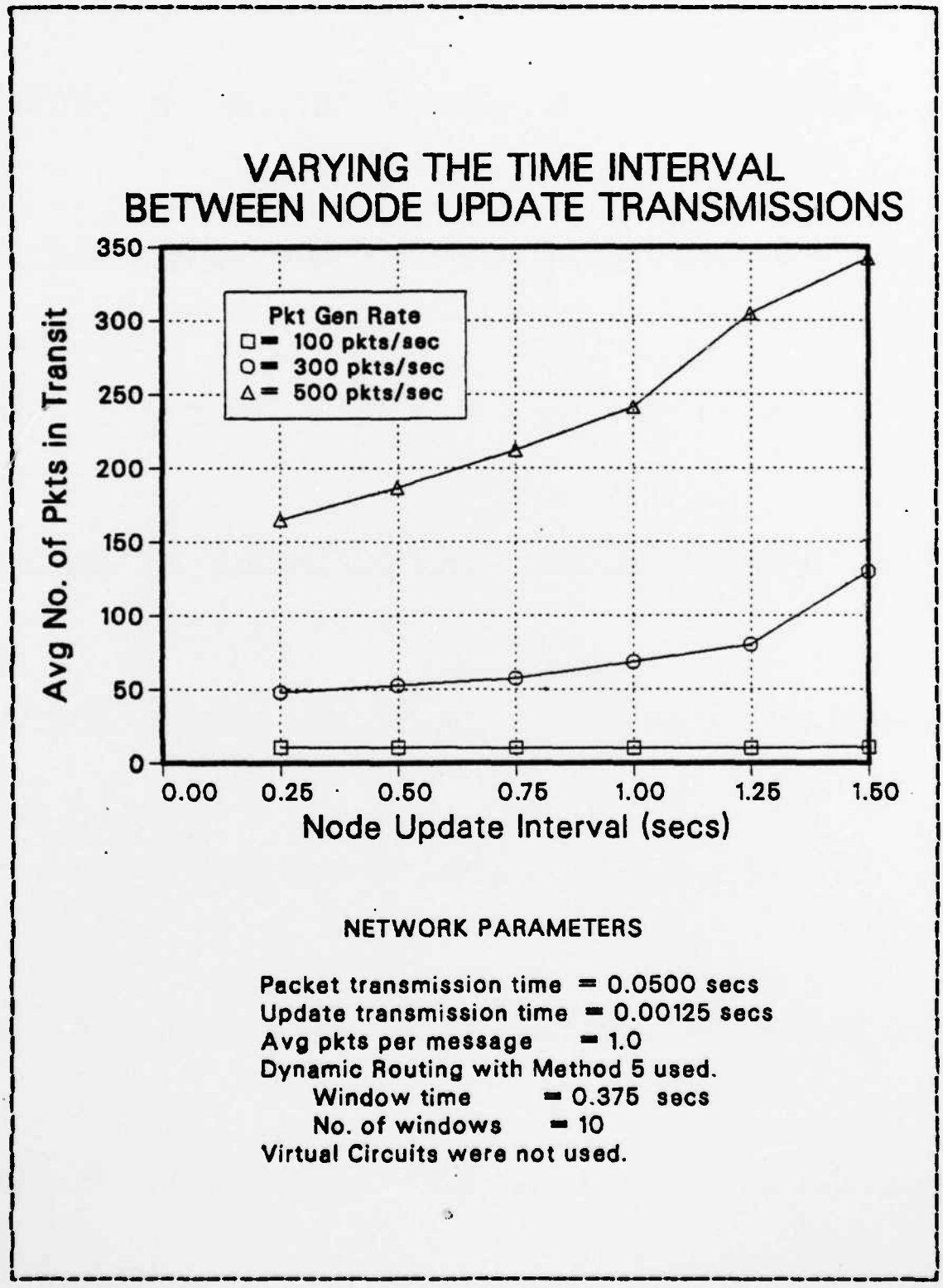


Figure 5.22 Node Update Interval : Low Overhead

One way to prcduce a situation where update overhead becomes significant is simply to increase the size of the network. However, due to the increased computing time this answer was not chosen initially. The solution selected was to increase the size cf the update packet by a factor cf 6. The new update transmission time was 0.00750 seconds and this packet was now only 1/15the the size of the message packet. Figure 5.23 gives the network performance using the larger update. For conditions of relatively high traffic intensity, the effects of the update transmissions substantially reduces the performance of the network. The average link utilization increases as the frequency of update transmissions increases. Figure 5.24 shows that when the packet generation was 500 packets per second and short update intervals were used, the entire network became saturated. The actual link utilization from overhead traffic is given in Figure 5.25. The calculated link utilization plotted in the figure was obtained using the relationship developed in Chapter III which stated that

$$\text{link Util[updates]} = (N / 2T)\,U.XMN.TIME$$

where N was the number of ncdes and T was the time interval between ccnsecutive update transmissions by a node. The

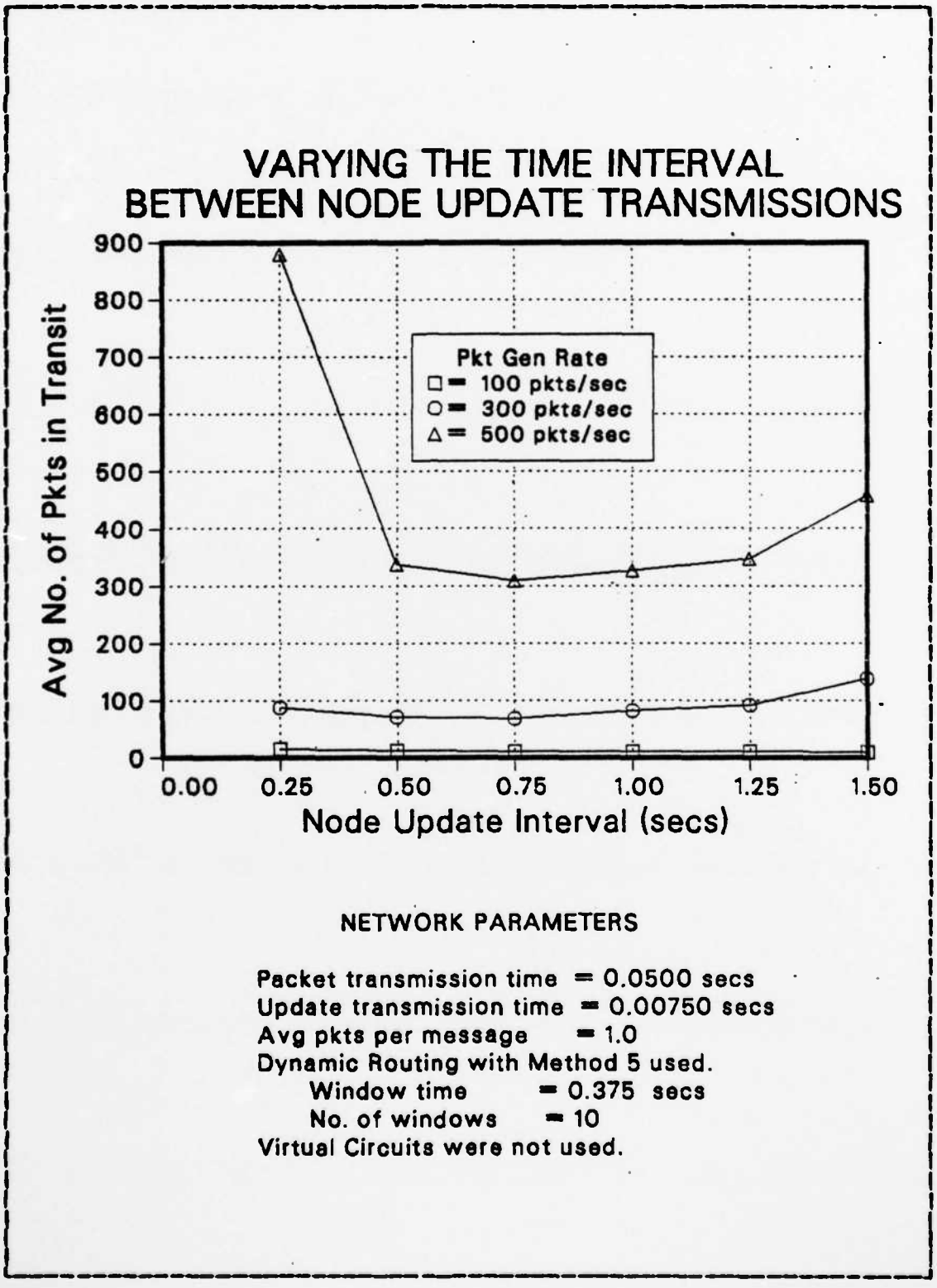


Figure 5.23 Node Update Interval : High Overhead

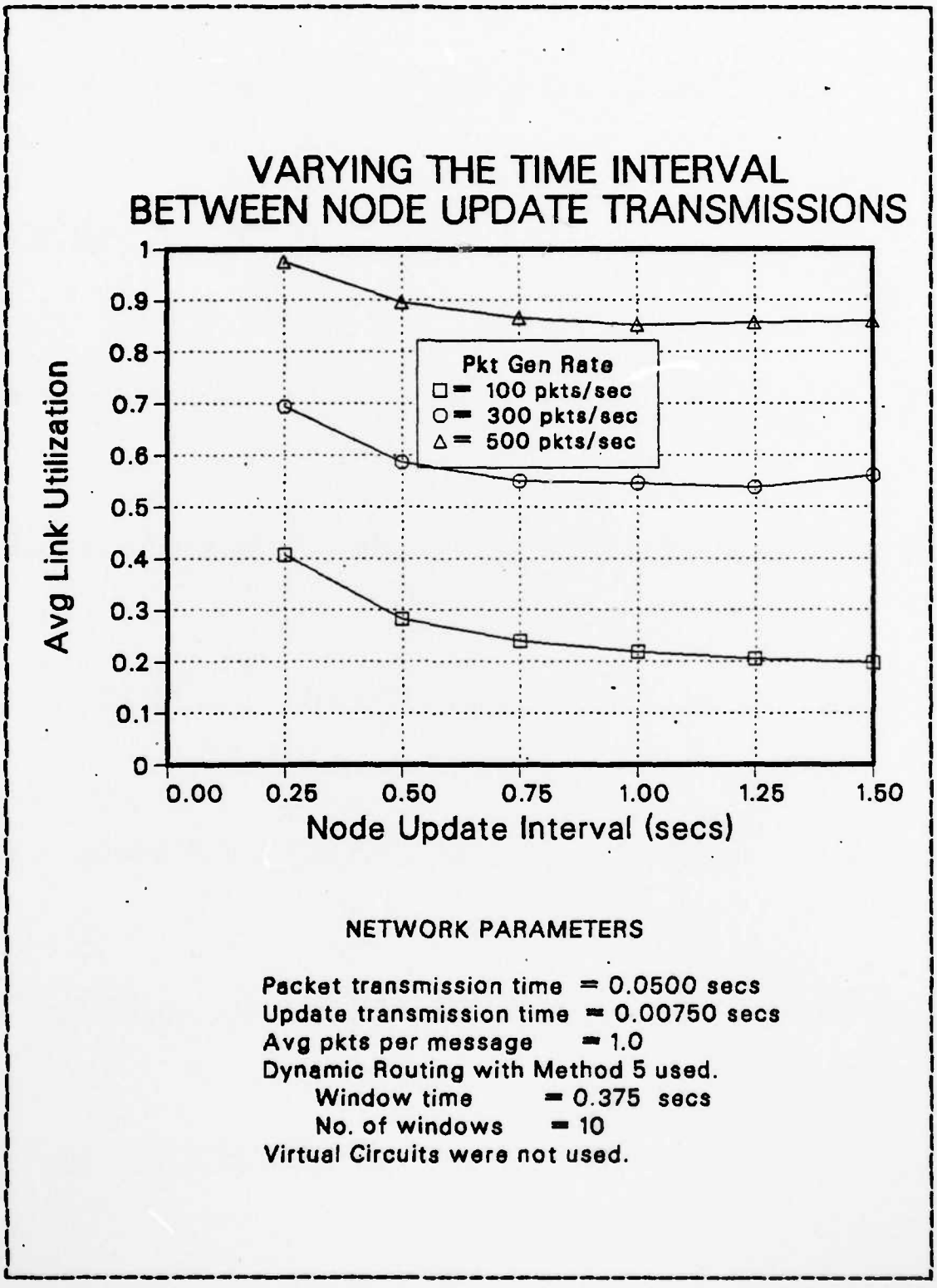


Figure 5.24 Node Update Interval : Total Link Utilization

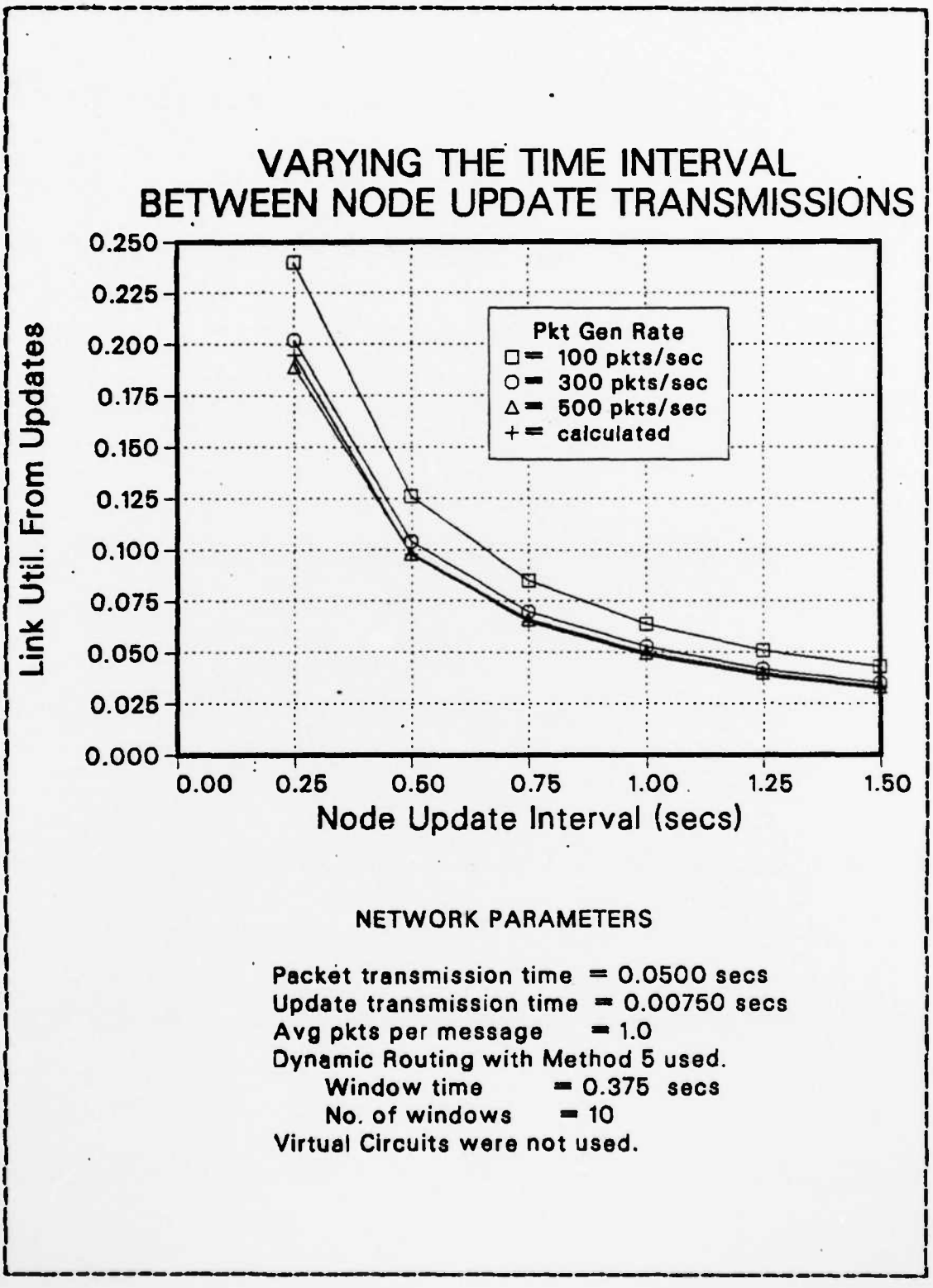


Figure 5.25 Node Update Interval : Update Link Utilization

agreement between the calculated value and the simulation results is very good. This not only provides another validation method for the simulation program itself but it also verifies an analytical method for estimating the utilization overhead for a network.

F. TESTING THE NODAL YEN ALGORITHM

The basic set of parameters for the nodal Yen algorithm have been established from the previous sections. The window period, Queue Factor and node update interval were selected based upon simulations using uniformly distributed traffic intensities and a highly symmetric network topology. The second phase of this research centers around the test and evaluation of the basic algorithm under a variety of different conditions. Specifically, the performance of the algorithm under the following conditions is sought:

1. Timing Errors
2. Geometrically Distributed Message Lengths.
3. Unbalanced Traffic Patterns.
4. Datagram and Virtual Circuits.

In this way, it can be determined whether or not routing algorithm parameters selected while using uniform traffic distributions will provide acceptable network performance under different circumstances.

1. Timing Errors

A major issue in determining the applicablility of these time-dependent routing protocols is how sensitive the best path selection is to timing errors in the arrival of update messages. It is obvious that if update messages do not arrive when they are supposed to that the receiving node will incorrectly calculate the Update Reception Weight. The question is how wrong the arrival time of the update can be before the best path selection becomes incorrect. Insight as to the magnitude of this timing error can be obtained by reviewing how the tentative time for update transmission is calculated.

The calculation of the update transmission time involved dividing the tentative shortest path distance by C.

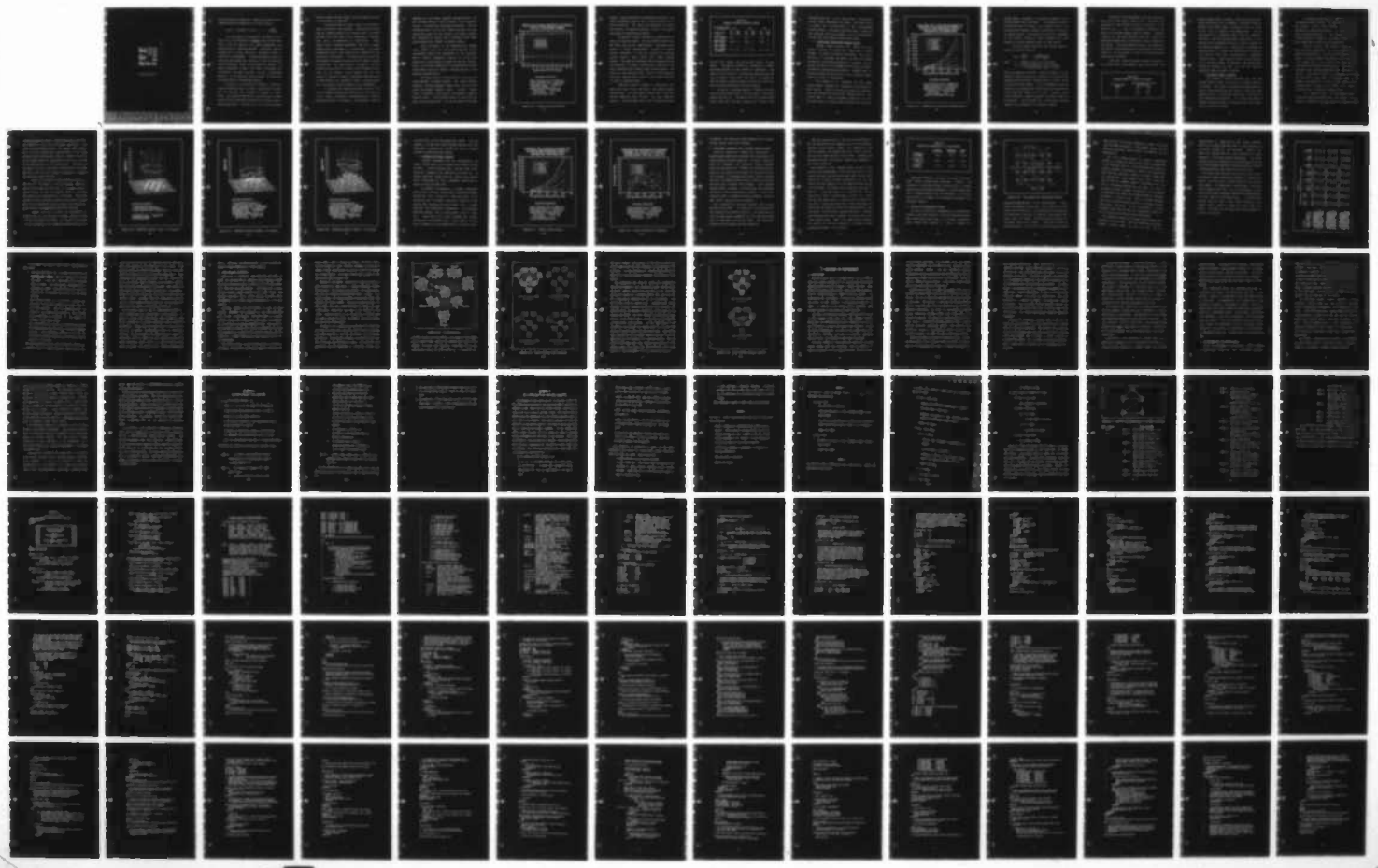
$$T[F(J,K)] = F(J,K) / C$$

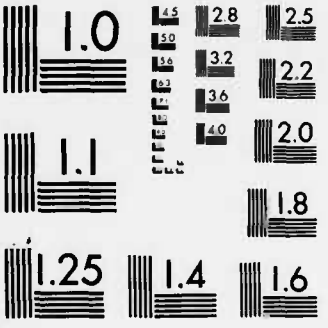
Since C is chosen to equal the inverse of the time required to transmit an update (U.XMN.TIME), this is equivalent to multiplying the path distance by the update transmission time.

$$T[F(J,K)] = F(J,K) * U.XMN.TIME$$

AD-A139 147 APPLICATION OF A DISTRIBUTED ROUTING ALGORITHM TO A PACKET-SWITCHED COMMUNICATIONS NETWORK(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA R R LOGAN DEC 83 3/4

UNCLASSIFIED NPS62-83-060 F/G 17/2 NL





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

The path distance includes the Reverse Link Weight which is computed using the combination link weight function.

$$W(I,J) = [QU.FACT * Q(I,J)] + 1 + \frac{Util}{1 - Util}$$

The link weight is constrained to be greater than or equal to one and therefore the minimum time interval after which a node can send its own update transmission upon receiving one is U.XMN.TIME seconds. Using this fact the maximum speed at which updates can "propagate" through a network can be determined. Maximum propagation speed will occur when link weights are minimum and thus the calculated update transmission delay is only U.XMN.TIME seconds. An update originating at time 0 will reach a node that is K hops distant (via least hop path) in (2K - 1) U.XMN.TIME seconds. If the path were one hop longer the additional time delay would be (2) U.XMN.TIME seconds.

The size of the timing error required to cause the incorrect selection within a uniform weight least hop scenario is twice the time required to transmit an update packet. This gives an indication of the magnitude of timing errors which the algorithm can function with without miscalculating best paths. The next step is to look at possible

sources of update arrival timing errors and their magnitude in relationship to U.XMN.TIME.

A possible source of timing errors is the microprocessor clock used by the algorithm. These errors originate from the discrete nature of the clock in which all events performed by the node (such as the transmission of messages) occur only at the beginning of a clock period. Errors develop when a computed transmission time for an update message falls in the middle of a clock period. The actual transmission time of that update will be delayed until the beginning of a new period. It follows that the shorter the clock period, the smaller the errors due to delays in update transmissions. The upper bound on the size of the clock interval would be the bit interval of the transmitted data. Since the bit interval is much smaller than the time required to transmit an update, performance degradation due to these errors is not expected to occur. Through simulation, the accuracy of this prediction can be determined.

By modeling the simulation program such that the time for message transmissions occurred only at the start of a clock period, the effects on performance due to finite clock periods could be analyzed. The simulation clock of

SIMSCRIPT is a real double precision variable (TIME.V) and has a discrete period of 10**-16 seconds. For all practical matters, the simulation time appears continuous. The data transmission rate for the test network was 20000 bits per second which meant the bit interval was 0.00005 seconds. The simulation was tested using "discrete" clocks whose periods ranged from 0.000001 to 0.000050 seconds.

From the results of the simulation runs (Figure 5.26), effects due to the clock periods selected did not substantially change the performance of the algorithm. Errors in update transmission times of up to one clock period are not large enough to change the best path calculations. The relative insensitivity of the algorithm to small random timing errors is indeed a positive feature in its operating characteristics.

The second source of timing errors in the arrival of update messages is queueing delays. The update insertion technique eliminates queueing delays which may have been caused from regular message traffic. However, updates cannot be inserted within updates and therefore queueing delays do occur. If two updates attempt to travel over the same link at the same time, one will be delayed U.XMN.TIME

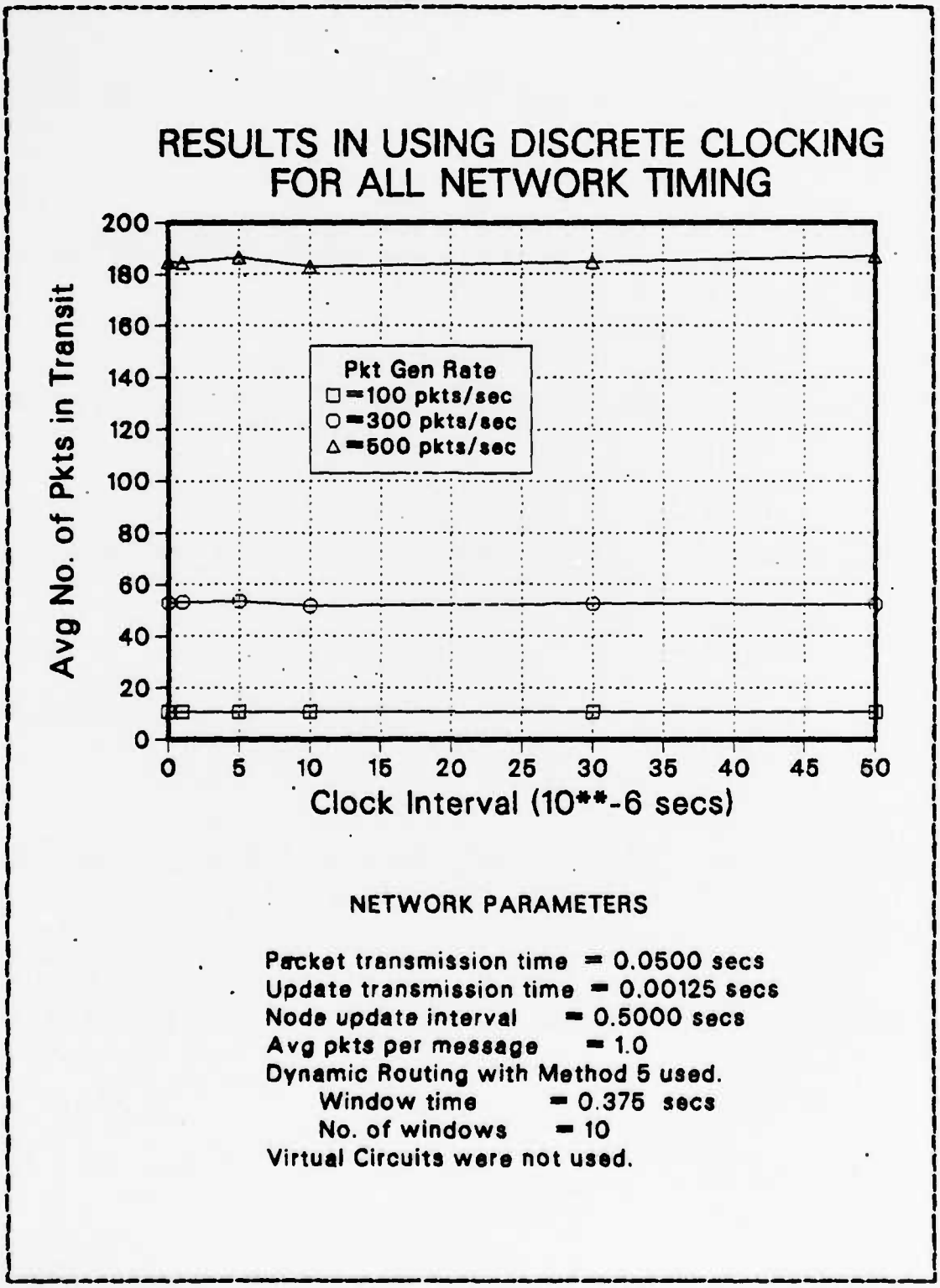


Figure 5.26 Varying the Clock Interval

seconds. Delays asscciated with queueing are related tc the update transmission time and thus can cause changes in best path calculations. The issue to be determined is whether the frequency of update "collisions" is such that network performance is substantially affected.

The effects cf update collisions are demonstrated by using twc simulation models. Both are identical except that cne versicn does not use update queues. This model allcws updates tc travel "transparently" through the netwcrk without collisicns. In fact this simpler (and earlier) versicn isclated update traffic from regular message traffic. The model was developed under the assumption that the probability of update collisions was small and total link utilization due to update traffic was also small. The second and more realistic model (which is used throughcut this work) employs update queues so that update collisicns do result in queueing delays.

Table VII gives the summary of results when both methcds were used on the test network with uniform traffic distributicns. The node update interval was 0.5 seccnds. Network parameters were identical for both simulation versions. The table uses wQ and woQ to indicate the final

TABLE VII

Effects of Update Queueing Delays

Pkt. Gen. Rate (pkts/sec) --->	100		200		400	
	wQ	woQ	wQ	woQ	wQ	woQ
Total Link Util.	.176	.155	.321	.305	.646	.633
Link Util. from Updates	.021	--	.019	--	.017	--
Mean Update Queue Size	.0002	--	.0001	--	.0001	--
Max. Update Queue Size	1	--	2	--	1	--
Avg. Number of Packets in Network	10.6	10.3	24.1	23.0	98.7	97.5

version (with Queue) and the earlier version (without Queue). The results indicate the difference in network performance between the two methods is negligible. This seems to be due in part to the very low values for both the mean and maximum update queue sizes during the simulation. Very few update collisions actually did occur and thus erroneous best path calculations were not prevalent.

During all remaining phases of simulation testing, after data was compiled using the final model, the earlier version was also run. The maximum size for an update queue during these runs was three packets with the mean value

remaining quite low. In all cases tested, the difference between network perfcrmances was slight. From these results it appears that the perfcrmance of this time-dependent routing protocol in networks exhibiting update delays from both "discrete" clocks and queueing is not only acceptable, but very near that obtained without these errors.

2. Geometrically Distributed Message Lengths

Message traffic entering the network has been modeled as having an arrival rate based upon a Poisson process. In using the Poisson distribution, the rate of arrival is "λ" messages per second and thus the average interarrival time between messages is $1/\lambda$ seconds. So far, though, the length of all messages has only been one packet. In operating packet-switched networks, messages occur in different lengths and must be split into a number of packet sized units prior to transmission.

From Chapter IV we considered the case where multiple packet messages had a geometric distribution for their lengths. Figure 5.27 gives the results of simulation runs using geometrically distributed message lengths. The average number of packets per message ranged from 1 tc 20. The results show for a fixed packet generation rate that the

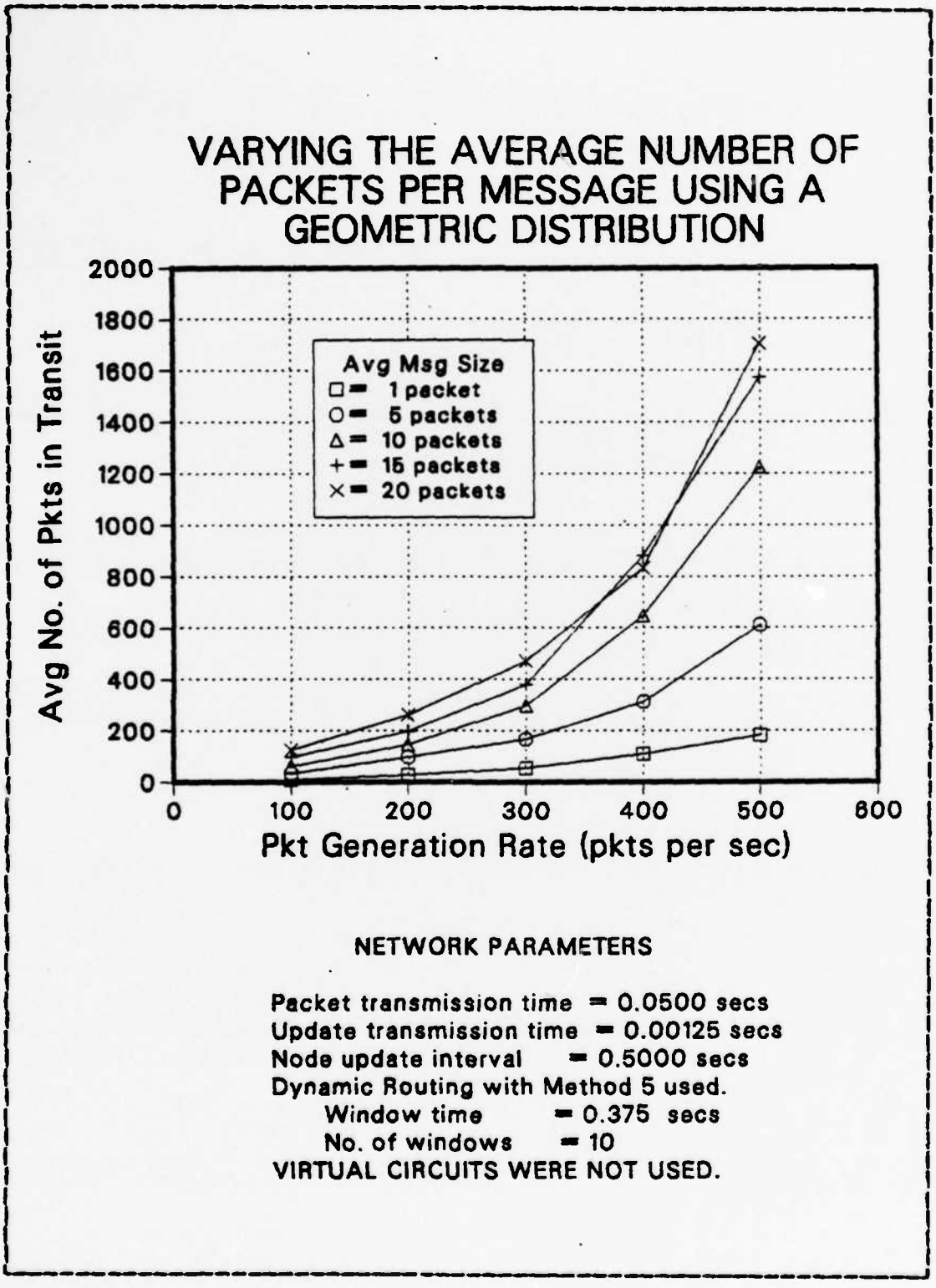


Figure 5.27 Geometrically Distributed Message Lengths

average number of packets in the network increased as the average message length increased. By applying the results obtained from the P-K formula, this increase is known to be characteristic whenever a geometric distribution is used for the message length. The increase is not due to a breakdown in the routing algorithm as might otherwise have been suspected had this analytical result not been applied.

The P-K formula stated that the average number of packets in an M/G/1 queue was

$$E[L] = \frac{\lambda D/p}{2} + \frac{(\lambda D/p^2)(2 - p)}{2(1 - \lambda D/p)}$$

where λ = Message arrival rate in pkts/sec
D = Packet transmission time in seconds
p = 1 / Average message length in packets

Since the simulation involves a network of queues, the P-K formula is used to analyze the behavior of the network when geometrically distributed message lengths are used rather than solve for E[L] exactly. This is permissable since the network is in an equilibrium state with no saturated links. Burke [Ref. 26] showed that in such a situation where traffic entering a system is Poisson, the output traffic is also Poisson. The P-K formula can thus be used to provide insight into the behavior of a network of queues.

The factor $\lambda D/p$ corresponds to the link utilization and as such is less than or equal to one. The second term containing p^2 will dominate the value of E[L] when λD is fixed and $p << 1$. E[L] will generally increase as the square of the average message length in packets. Therefore a doubling of the average message length will result in an approximate quadrupling of the average number of packets in the network. From the simulation results, the behavior of the network using fixed message arrival rates and varying values of p can be analyzed to see if this characteristic relationship is evident.

Table VIII shows the average number of packets in the network using fixed message rates while varying the

TABLE VIII

Average Number of Packets in Network

Avg Msg Length (pkts)	Message Generation Rate (msgs/sec) 10	20
5	14	37
10	55	164
15	124	392

average message length in packets. When the average message length doubled from 5 to 10 pkts/msg, the average number of packets in the network (E[n]) increased 3.93 and 4.43 times for the 10 and 20 msgs/sec rates respectively. This is approximately the quadrupling affect which was predicted. Similarily, when the message lengths tripled from 5 to 15 pkts/msg, E[n] increased 8.86 and 10.6 times as compared to the estimated increase of 9. Finally for the increase in message length of 1.5 times, the resulting E[n] increase was 2.25 and 2.39 times which corresponded to a calculated increase of 2.25. These results indicate that the behavior of the network of queues is reasonable when geometrically distributed messages are used and that the algorithm performance does not appear to degrade.

3. Unbalanced Traffic Patterns

All simulation results discussed thus far were obtained using a uniform distribution of traffic throughout the network. The uniform distribution was accomplished by picking source-destination node pairs at random for each message such that all node pairs within the network had an equal probability of selection. As a result the number of messages generated between all node pairs for each simulation run was about the same.

An unbalanced traffic pattern exists where the rate of message generation between node pairs differs throughout the network. Static routing methods, which have been designed for uniform distributions, are severely limited under unbalanced conditions. The ability of a dynamic routing algorithm to adjust to non-uniform conditions is of prime importance in evaluating its performance.

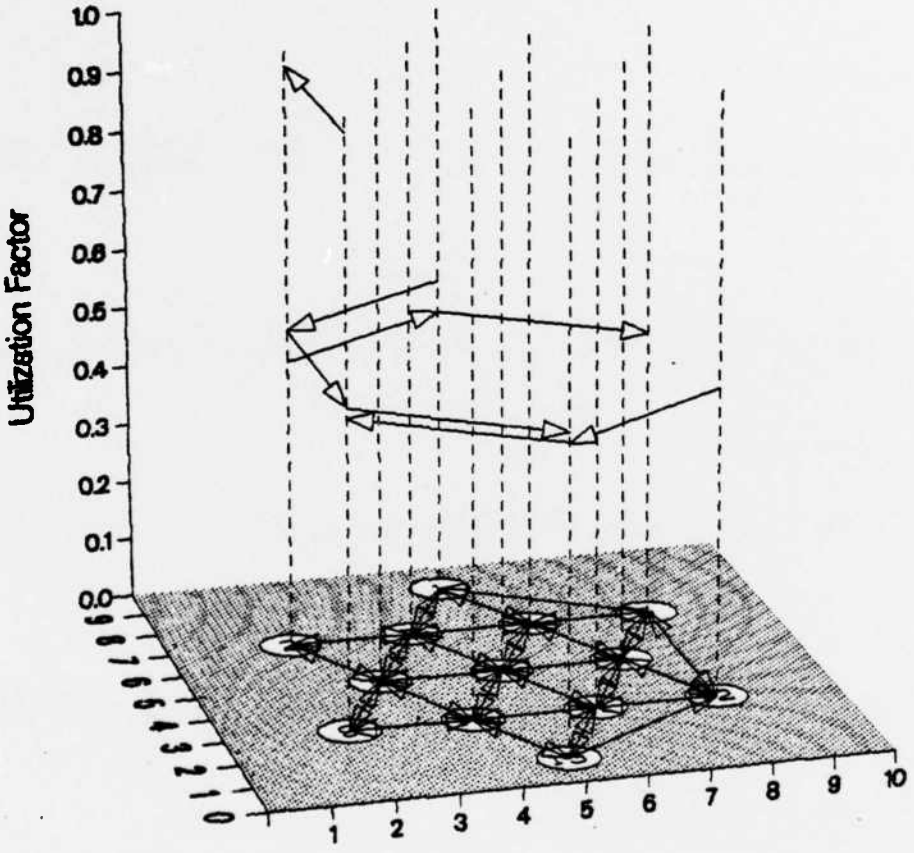
For this test the same thirteen node network was used with some modifications. All traffic generated for the network was dedicated to only three source-destination node pairs. The pairs selected were nodes 1 and 13, nodes 9 and 5 and nodes 12 and 2 with the first node being designated the source and the second the destination. These particular node pairs were selected based upon their locations in the network. They are all located on the perimeter with source and destination nodes being on opposite sides of the network. The purpose was to allow a wide number of criss-crossing paths to exist for the traffic in an attempt to vigorously test the algorithm's capabilities.

As a reference point, the unbalanced network was initially tested using static least hop routing. Prior to running the simulation one could compute the maximum traffic

intensity which the network could handle before saturating. The capacity of each link is 20 packets per second. With three source-destination pairs, the maximum packet generation rate (if seperate link paths existed for the three routes) would by 60 packets per second. However, the static routing table (Table V) indicated that link (9,2) was used in the best path for two of the node pairs. Therefore for the static run this link should saturate when the packet generation rate is only 30 packets per second or 10 packets per second for each node pair.

Figure 5.28 provides a three dimensional view of the network with the height of the arrows between nodes indicating the utilization of the link. It is evident that with static routing the lack of alternate paths was extremely detrimental to the performance of the network. When using dynamic routing, the traffic should be more broadly distributed over the network and thus the traffic load between node pairs could be increased before saturation occurred.

Figures 5.29 and 5.30 show the distribution of traffic using the Yen algorithm at packet generation rates of 50 and 100 packets per second. The highest link utilization for the network at 50 packets per second was only 0.43

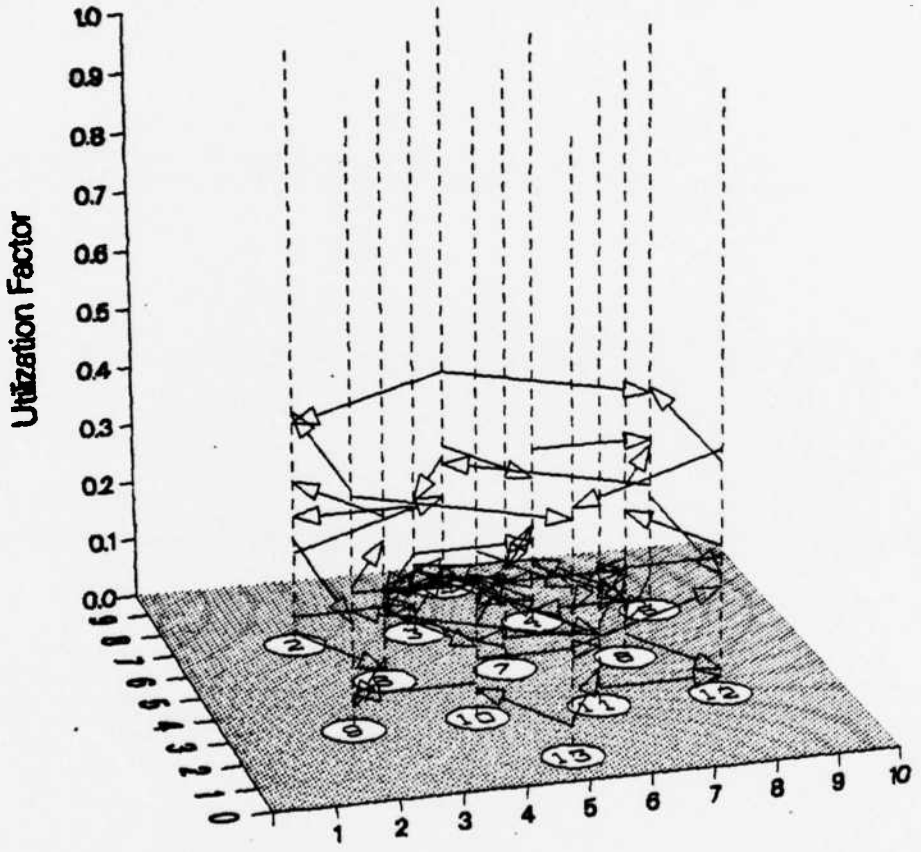


NETWORK PARAMETERS:

Source-Destination Node Pairs.....
Node 1 to 13, 9 to 5 and 12 to 2.
Packet transmission time = 0.0500 secs

Pkt generation rate = 30 pkts/sec
Avg pkts per message = 1.0
Static Routing used.

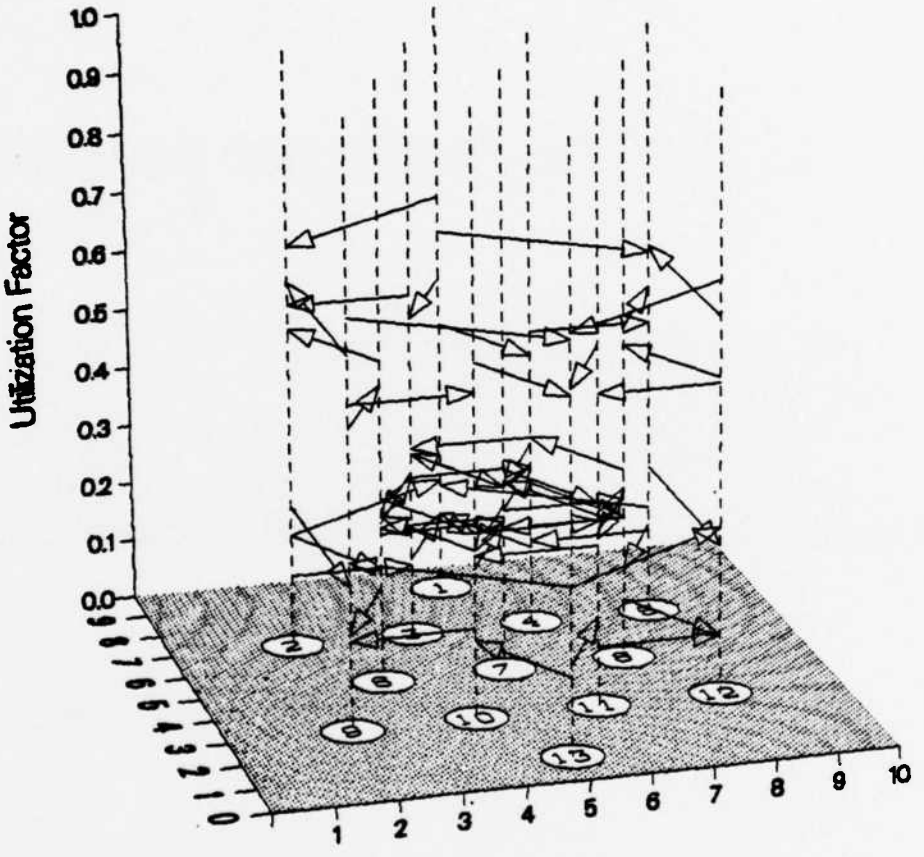
Figure 5.28 Unbalanced Traffic: Static - 30 pkts/sec



NETWORK PARAMETERS:

Source-Destination Node Pairs.....
Node 1 to 13, 9 to 5 and 12 to 2.
Packet transmission time = 0.0500 secs
Update transmission time = 0.00125 secs
Node update interval = 0.5000 secs
Pkt generation rate = 50 pkts/sec
Avg pkts per message = 1.0
Dynamic Routing with Method 5 used.
Window time = 0.375 secs
No. of windows = 10

Figure 5.29 Unbalanced Traffic: Dynamic - 50 pkts/sec



NETWORK PARAMETERS:

Source-Destination Node Pairs.....
Node 1 to 13, 9 to 5 and 12 to 2.
Packet transmission time = 0.0500 secs
Update transmission time = 0.00125 secs
Node update interval = 0.5000 secs
Pkt generation rate = 100 pkts/sec
Avg pkts per message = 1.0
Dynamic Routing with Method 5 used.
Window time = 0.375 secs
No. of windows = 10

Figure 5.30 Unbalanced Traffic: Dynamic - 100 pkts/sec

and still only 0.69 at 100 packets per second. From both figures it is clear that traffic between node pairs has been distributed throughout the network and thus the dramatic increase in performance was achieved.

4. Datagram and Virtual Circuits

Chapter I mentioned that the performance of any dynamic routing method will be reduced when virtual circuits are used. Virtual circuits "lock" the path taken by all packets of a message to that selected by the first one. If network conditions change after the initial "trail-blazer" packet establishes the route, subsequent packets may no longer be taking the best path.

In testing the degradation in network performance using virtual circuits, geometrically distributed message lengths were used. Figure 5.31 shows the outcome of the simulations with the results being very similar to those received when datagrams were used (Figure 5.27). However there was some degradation in performance, the percentage difference between the two methods being given in Figure 5.32. These results are highly dependent upon network topology and are presented only to give a feeling for the type of degradation in performance which is likely to occur.

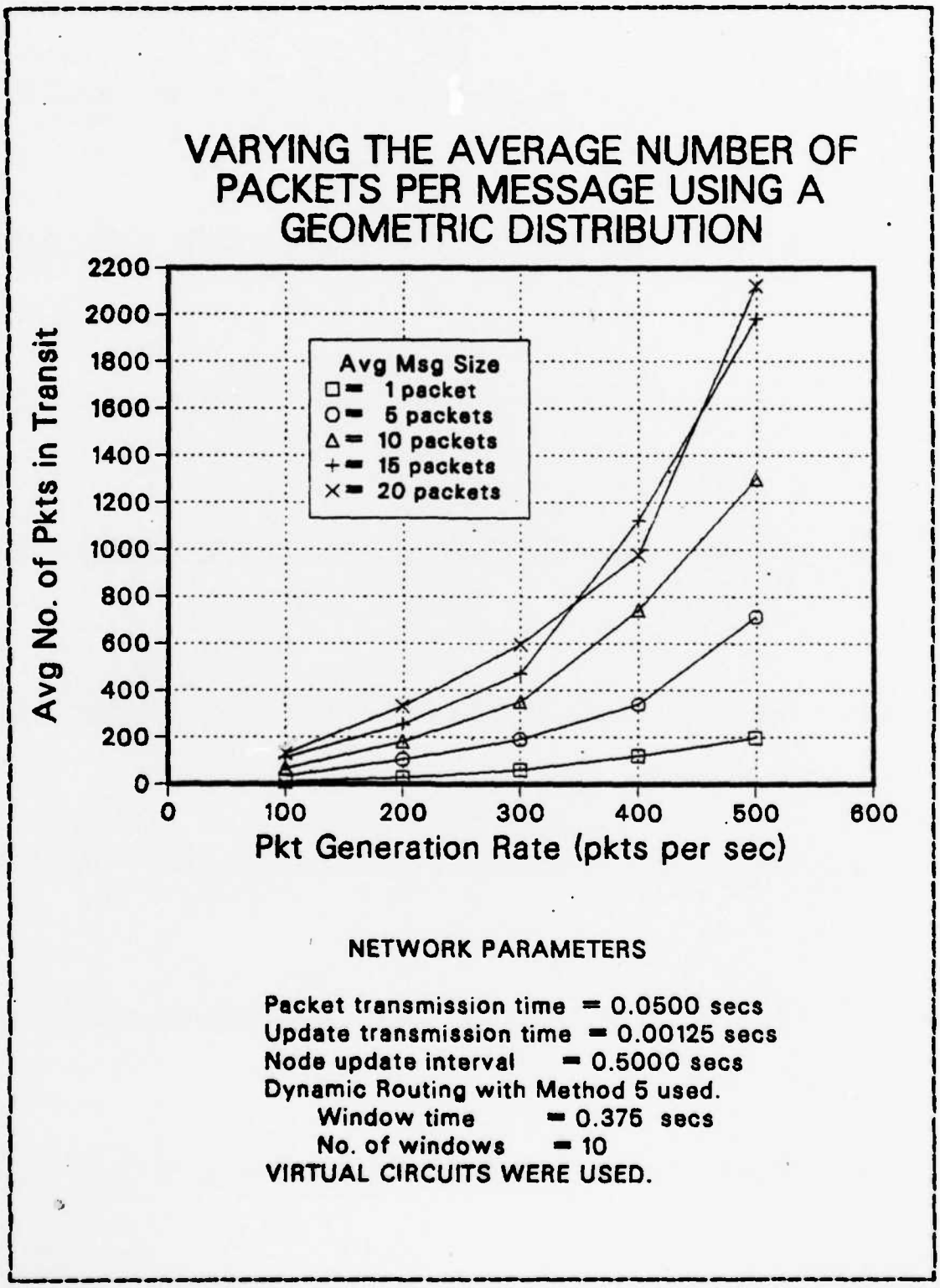


Figure 5.31 Virtual Circuit Results

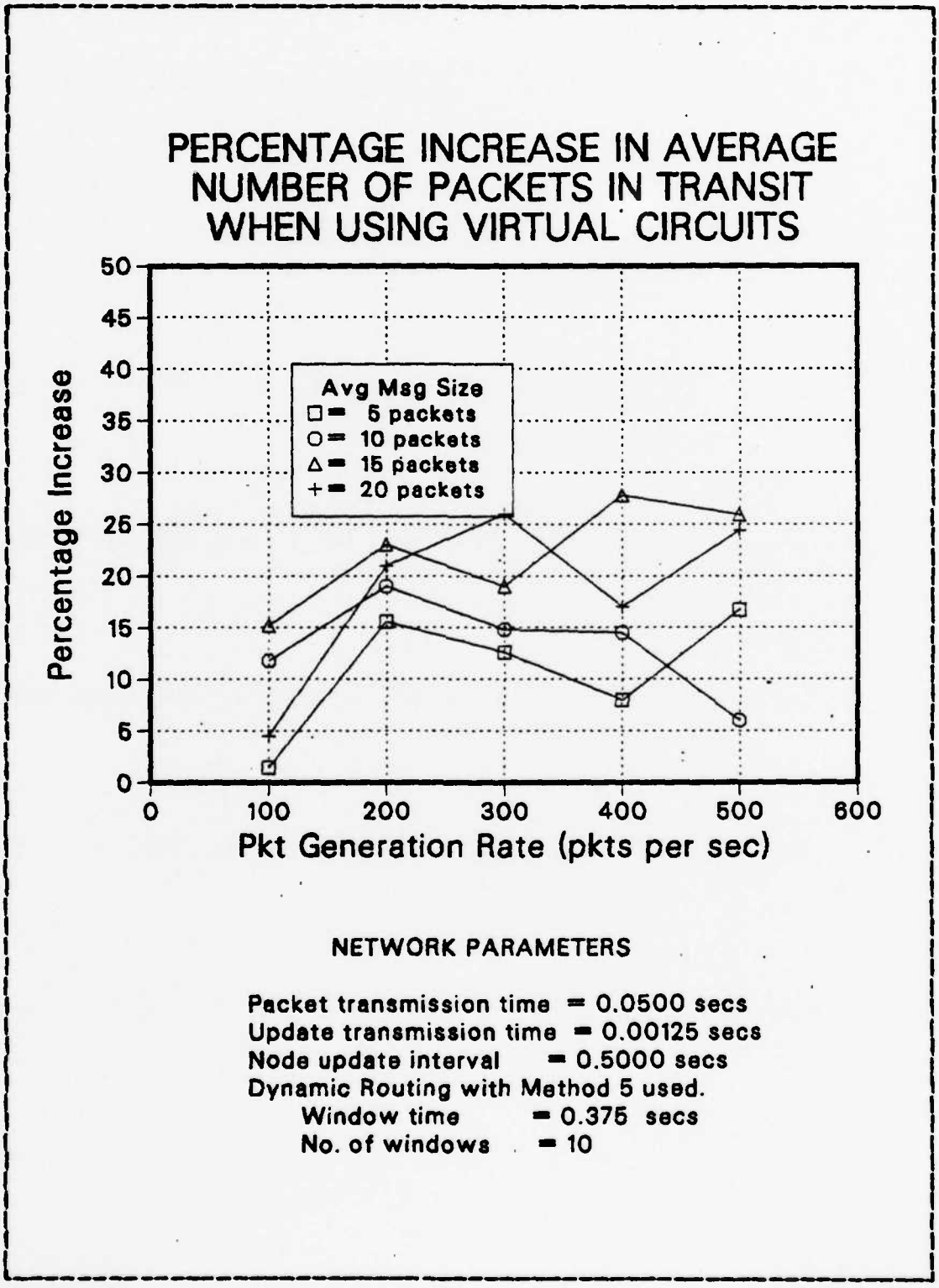


Figure 5.32 Virtual Circuits vs. Datagrams

As expected, the longer the average message size the poorer virtual circuits faired with datagrams.

G. PERFORMANCE COMPARISON WITH A ROUTING FRACTION SCHEME

While the research was underway for this project, a second project began which investigated the performance of a routing fraction protocol [Ref. 24]. In order to provide a basis for comparison, the simulation program presented herein was utilized as the "driver" for the second project. Because of the design of the program, the only major alterations required to accomodate the routing fraction protocol centered in the Update Routing Protocol events and routines. The other major divisions of the program, which concerned message traffic transport, data collection and simulation preparation, were not significantly changed. In this way a comparison study of different routing procedures using the same "driver" routines could be analyzed.

Upon completion of the second project, both routing protocols were tested. To ensure that the programs still only differed by the routing protocols, both were first run using the same network parameters and with fixed best paths from the same least hop routing table. The statistical results from both program runs were identical.

Next the programs were run such that the performance of the Yen algorithm could be compared to that of the optimally selected routing fraction method. The selection of the routing fractions was possible since the traffic distribution for the network was know to be uniform. Therefore the resulting multiple path routing strategy was "matched" to the network conditions.

The intensity of the incoming traffic was set at 335 pkts/sec which corresponded to the saturation point of the network when least hop routing had been used. The Yen simulation was run using the same parameters as in the last section with the exception that. the update interval was set to 0.20 seconds. The results of the comparison are given in Table IX. From these results it is evident that the Yen algorithm is able to perform almost as well as a static multiple path method which is operating on its "home turf". This term is used since within this static environment the routing fraction method can yield the optimal results. Thus the optimal routing fraction technique provides a performance yardstick with which to measure how well the Yen algorithm actually works. Again, from these initial findings, the ability of the Yen algorithm to approach optimal performance appears to be very good.

TABLE IX

Performance Comparison of the Algorithms

	Yen Algorithm	Routing Fraction	Least Hops
Avg Pkts in Network (in pkts)	59.6	50.3	102.3
Avg Pkt Trip Time (in secs)	0.173	0.148	0.301
Avg Queue Size (in pkts)	0.44	0.30	1.20

Additional simulation runs were done using the unbalanced traffic patterns used previously. However, the inability of the static routing fraction method to adjust caused severe performance degradation. We have already seen that the dynamic nature of the Yen algorithm, on the other hand, allows it to perform in an unbalanced environment with acceptable results.

B. TESTING THE HIERARCHICAL VERSION

In order to evaluate the peformance of the hierarchical version of the basic algorithm which was developed earlier, some modifications were made to the network parameters. These changes were required in order to display the

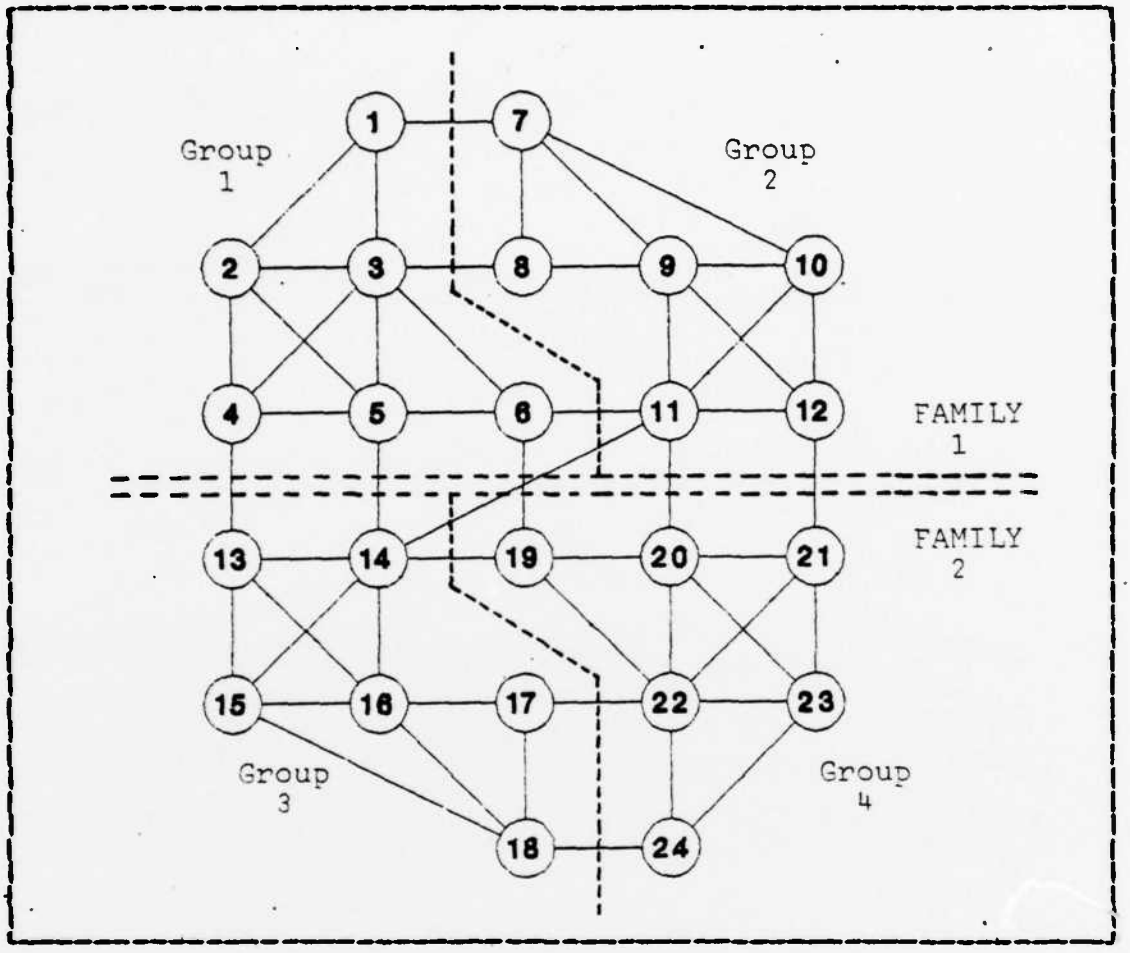


Figure 5.33 Test Network for Hierarchical Version

operating characteristics of the new algorithm. First a larger test network was used which had 24 nodes and 52 full duplex links (Figure 5.33). The hierarchical structuring of this network was such that two families were created (each with a total of 12 nodes) and each family was divided into two groups (each with 6 nodes). Though this network is relatively small, it was found that running larger networks

using the IBM 3033 was very expensive in terms of both CPU time and memory allocation. This network, though, with one additional parameter change proved to be adequate for the demonstration.

One of the primary reasons for using a hierarchical scheme is to reduce the overhead from update transmissions. It has been shown that the link utilization from updates is directly related to the number of nodes in the network when using the nodal algorithm. However, in order to "load down" the network from update packets which have a relatively short (1/40th of a message packet) transmission time, the size of the network required was too large. In order to increase the "cost" of update transmissions for the purpose of comparison, the length of the update packet was therefore increased by a factor of six. This is the same technique used in the previous section when the the interval between node updates was varied. It was now possible to run the simulation using both the nodal and group/family version of the Yen algorithm and evaluate their performances.

Two methods for generating the unit warning message were developed for the program. The first method was the single "leader" technique where one node per unit has the

responsibility of generating the unit warning messages. Initially the scheme where each node in a unit generated its own warning messages was also simulated. However, this method was discarded when the poor distribution of update messages in time caused degraded performance. The second technique which was simulated was the synchronized firing of unit update messages whereupon the warning message contained a resynchronization time instead of the firing time.

The results from simulation runs using the nodal and the two warning message hierarchical schemes are summarized in Table X. The table lists the nodal version of the algorithm under "NODE", the hierarchical version using firing time warning messages under "NS" (for non-synchronized) and the version using warning messages containing the resynchronization time under "S" (for synchronized). All hierarchical versions were run using a node update interval of 0.25 seconds while a variety of group and family times were chosen. Two nodal runs were made using update intervals of 0.25 and 0.50 seconds.

TABLE X

Performance of Hierarchical Version

Version	NODE	NODE	S	NS	S	NS	S	NS
Node Interval	0.25	0.50	0.25	0.25	0.25	0.25	0.25	0.25
Group Interval			0.25	0.25	0.50	0.50	1.00	1.00
Family Interval			0.25	0.25	1.00	1.00	4.00	4.00
Packet Generation rate = 150 pkts/sec								
Average Total Link Utilization	.647	.408	.381	.419	.378	.391	.371	.375
Average Update Link Utilization	.394	.211	.144	.177	.137	.149	.131	.137
Average Number of Packets in System	75.8	31.5	53.7	59.0	53.1	55.5	53.6	70.8
Average Packet Trip Length (hops)	2.73	2.60	3.19	3.24	3.25	3.28	3.23	3.21
Packet Generation rate = 250 pkts/sec								
Average Total Link Utilization	.779	.537	.564	.594	.550	.566	.555	.560
Average Update Link Utilization	.372	.196	.132	.167	.126	.140	.122	.128
Average Number of Packets in System	250.	81.4	215.	207.	191.	189.	437.	369.
Average Packet Trip Length (hops)	2.74	2.74	3.44	3.37	3.41	3.41	3.52	3.45
Packet Generation rate = 350 pkts/sec								
Average Total Link Utilization	.853	.678	.719	.747	.715	.731	.697	.705
Average Update Link Utilization	.354	.186	.129	.164	.122	.136	.118	.124
Average Number of Packets in System	869.	210.	772.	830.	562.	644.	1086.	1078.
Average Packet Trip Length (hops)	2.65	2.78	3.46	3.40	3.42	3.42	3.45	3.47

In analyzing the results some overall characteristics are noted:

1. The link utilization due to update transmissions was substantially reduced when the hierarchical methods were used. In the case where node update intervals were the same, the hierarchical methods showed a reduction in update link utilization by a factor of three (0.125 as compared with 0.375 for the nodal scheme).
2. The average length of a trip which a packet took using the hierarchical schemes was longer than with the nodal version. This is consistent with the nature of all hierarchical methods in which reduced update overhead is achieved at the expense of slightly longer packet trips.
3. The synchronized version of the hierarchical methods performed better than the firing time warning message version. The improvement appears to be from the reduction in overhead which was approximately twenty percent in this case.

The best performance using the hierarchical methods was not obtained when the update intervals were shortest but

rather when the groups and the families were not generating their unit updates at the same rate as the nodes. This occurred using the synchronized method with the group and family intervals at 0.5 and 1.0 seconds respectively. This performance was dramatically better than the node version which was clearly being congested with overhead traffic. Even though the nodal version packet trips were about 0.75 hops shorter than the hierarchical versions, the burdening effect of high link utilization from update traffic was the dominant factor. However, when the nodal version was run using a larger update interval such that the overhead link utilization was halved, its performance improved.

The benefits from the use of hierarchical methods in large networks have been demonstrated in this section. It should be noted that incorporating such schemes in very large networks is not merely convenient but an absolute requirement. Recall that the formula for the link utilization due to update transmissions could be calculated directly. Since the size of the update packet is small but relatively fixed in length and the update interval must remain reasonably short, there is a finite network size before the links will saturate due to update transmissions

alone. Therefore the implementation of the hierarchical version becomes mandatory in a large network.

I. LARGE NETWORK SIMULATION

The choice of relatively small networks (25 nodes or less) for the majority of simulation work was based primarily upon limitations which the IBM 3033 had upon job execution time and available memory. Simulating the routing protocol involved large amounts of both these items. The maximum time limit for a single job was one hour which corresponded to a thirty second simulation of a 25 node network.

However, during the last quarter of work on this project, SIMSCRIPT II.5 (Release 4.2) was made available with the VAX 11/780 machine. The VAX 11/780 is a virtual address computer with a large available memory (4 gigabytes) and no limitation on program execution time. The similarities between the VAX and IBM versions of SIMSCRIPT were such that transporting the simulation between machines was not difficult. Program modifications were centered only in the I/O processes.

The initial design of the "large" network consisted of 288 nodes. Hierarchical structuring produced 3 families (96

nodes each), with 8 groups per family (12 nodes each). Initial simulation runs, however, exceeded the available memory of even this machine and so the network was reduced to a single family. The final network had 96 nodes and 202 full duplex links (Figure 5.34).

Testing of the dynamic hierarchical routing protocol in this network was made using a highly unbalanced traffic distribution. The majority (95 percent) of traffic generated for the network originated from Group 1 nodes. In a similar manner, the same percentage of generated traffic was destined for Group 4 nodes. From the network topology, it can be seen that this traffic distribution means that most traffic must travel from one side of the network to the other. Numerous routes can be visualized to include flows of traffic through the upper two groups, the center group, and the bottom three groups.

In order to get a performance comparison, the simulation was first run statically using the least hop routing scheme. The packet generation rate for the network was 100 packets per second. This traffic intensity was selected such that link saturation was beginning to occur. The results of the simulation run are presented in a pictoral rather than

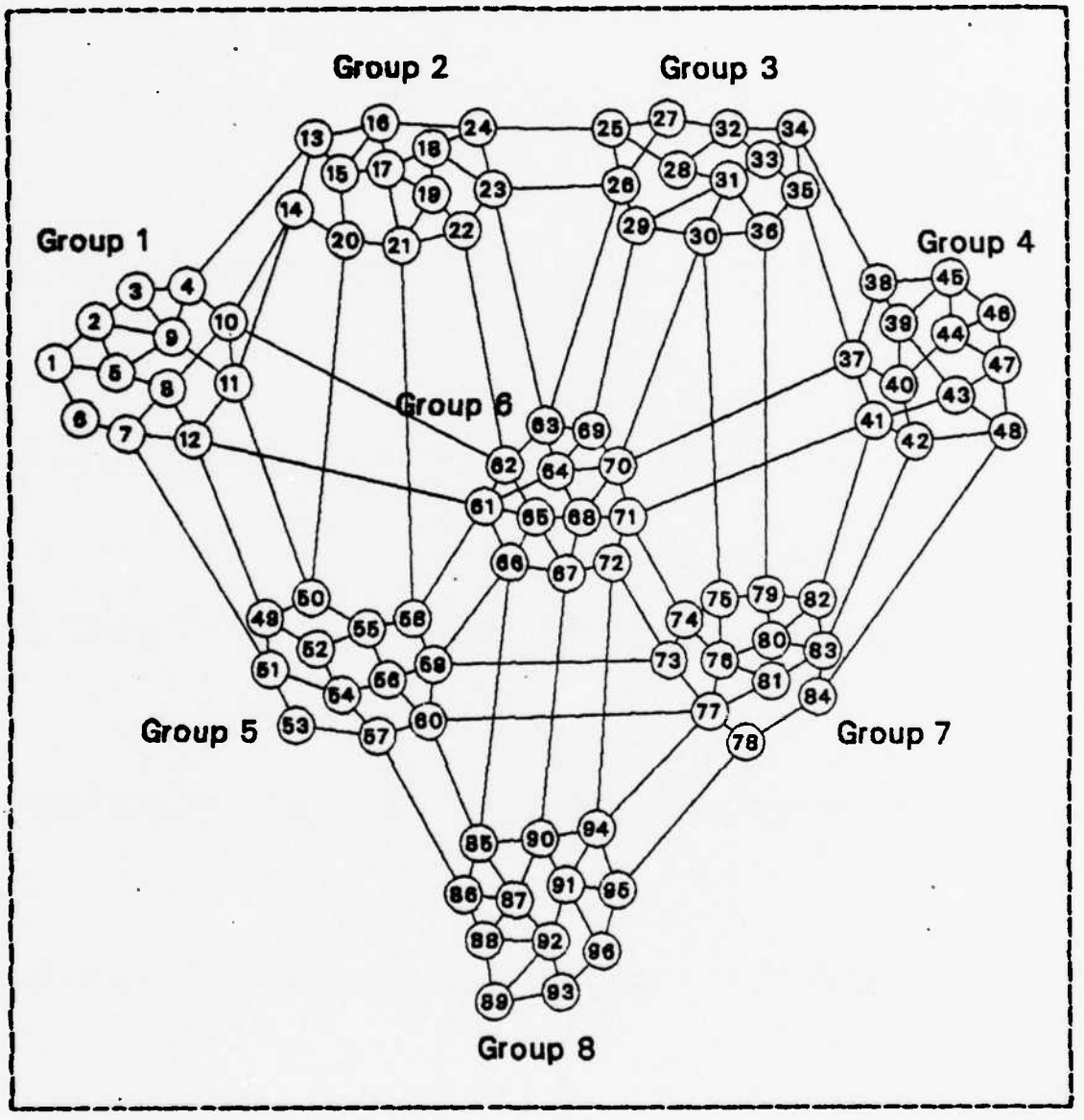


Figure 5.34 96 Node Network

tablular form so that the overall link usage of the network can be clearly seen. Figure 5.35 presents four views of the network with each view depicting those links which had utilizations within the range specified. As expected with

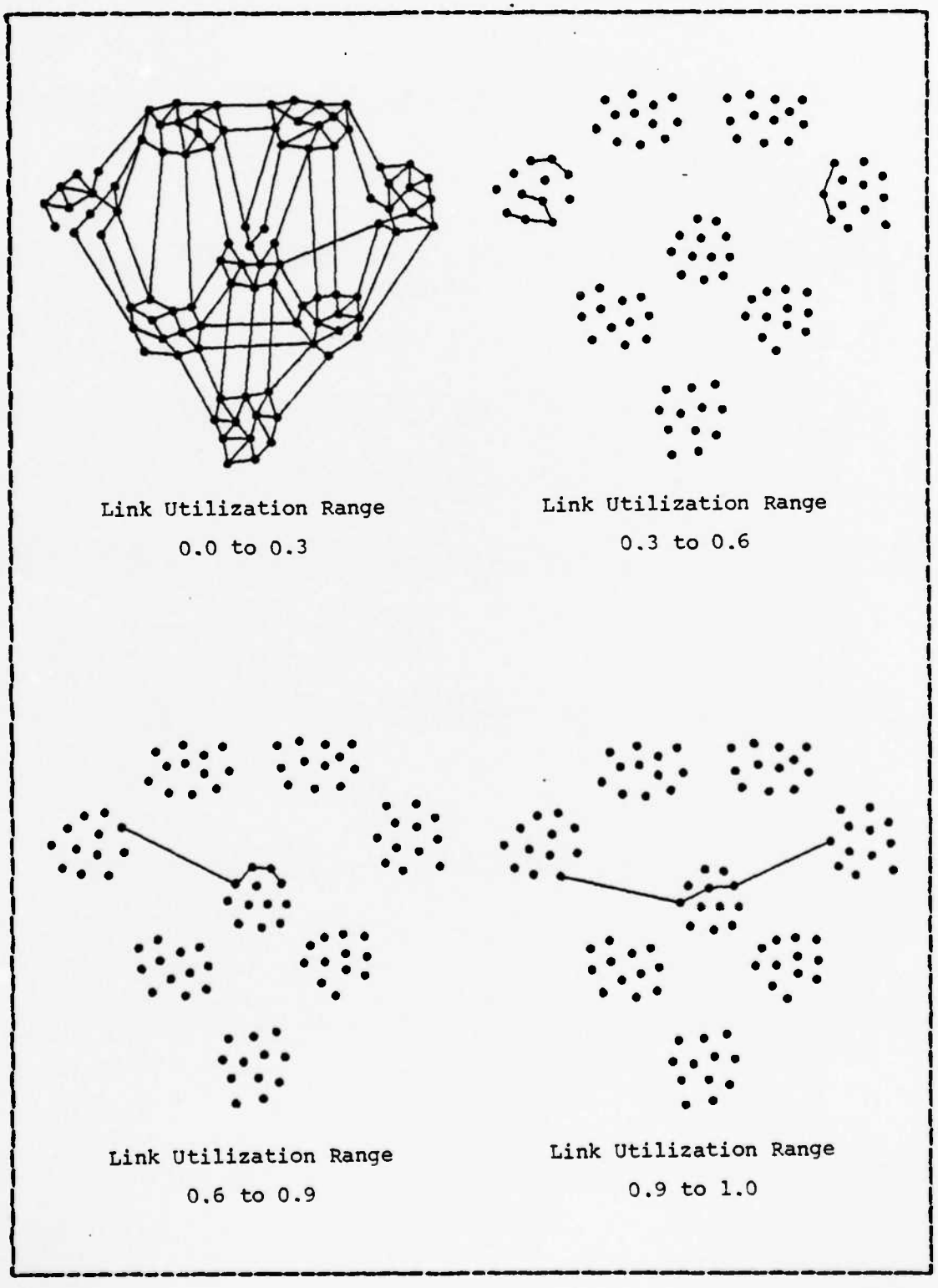
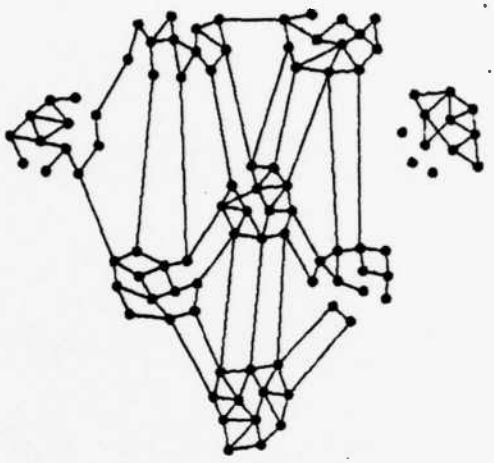


Figure 5.35 Large Network: Static Routing

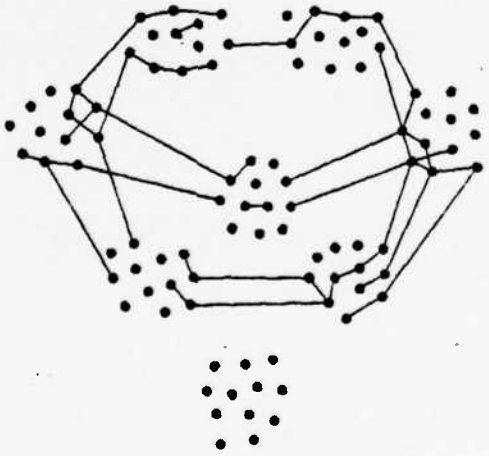
the static scheme, the majcrity of the links received very little usage while the links belonging to the best path were saturated. At the ccnclusion cf the ten second simulation only 727 of the 1030 packets generated had completed their trips.

The simulation was then run using the hierarchical distributed algorithm with synchronized unit update transmissions. Update packet size was kept at the size used previcusly (1/40th of a message packet) and both the node and group update interval was set to 0.5 seconds. The results are given in Figure 5.36. Network views corresponding tc link utilizations greater that 0.6 were not required since the highest utilization of any link in the network was 0.586. The view showing the utilization range from 0.3 to 0.6 clearly shows the ablility of the algorithm to effectively route traffic. At the conclusion of the simulatien the network was stable with 917 of the 1030 generated packets having completed their trip. The average link utilization due to update traffic was only 0.026 and therefore did not interfere with message traffic. Had the nodal versicn been used, the average link utilization from update traffic alone would been approximately 1 which again points cut the requirement for the hierarchical version.



Link Utilization Range

0.0 to 0.3



Link Utilization Range

0.3 to 0.6

Figure 5.36 Large Network: Dynamic Routing

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The path which lead to the development of the combination link weight function and the hierarchical distributed routing procedure was by no means a "least hop" one. Many detours were taken while enroute which were time consuming, but not without benefit in terms of derived insight. In summarizing the many points of conclusion obtained during the course of this project, a chronological approach is used. In addition to the actual results obtained, some lessons learned by this author are also included.

Initially the simulation program model was developed on a much simpler scale than that which appears herein. Modularity and understandability were emphasised so that as the model grew in realism (and became more complex), it did not become unmanageable. In doing so, the impact that modifications to the model had upon the simulation operation were not lost. This process is best seen by example.

An early version of the simulation model did not use update queues under the assumption that the effects of update collisions would be negligible upon the performance

of the routing algorithm. As the model grew, the update insertion technique was included with its resulting update packet queueing delays. The new simulation results confirmed an earlier speculation that queued updates did not dramatically change the best path calculations. However, only by working from the simple to the more complex model could these conclusions have been made.

Another goal during the model development stage was in finding techniques which could be used to validate the simulation. The use of analytical methods proved very effective in this area. The key was in matching the network parameters to situations which could be duplicated using the mathematical techniques. In doing so the results obtained during simulation closely agreed with those using the analytical methods. The simulation operation was also verified using a painstaking process whereby every network process involving individual packets was printed out. Therefore, prior to the simulation being run using the routing protocols, the model had been convincingly tested.

After establishing a workable simulation model, it soon became apparent that a dynamic routing protocol is only as good as the information upon which it makes the estimate of

the network conditions. The combination link weight function which was finally developed appears to provide an accurate and responsive measurement upon which the routing decision can be made. A somewhat unexpected characteristic of the combination method was the relative insensitivity of its parameters upon the algorithm's performance. The Queue Factor term demonstrated very acceptable performance over a wide range of network parameters. In fact, the modified Yen algorithm seemed to be hard to "break down" even when subjected to a variety of chaotic and unbalanced traffic conditions. The robustness of the modified version was further shown when during conditions of rather gross timing limitations, the network performance remained basically unchanged.

It should be emphasised, though, that it was not the intent of this research to find the "optimal" set of parameters for the routing protocol. The effects that one parameter had upon another were interrelated and thus the problem would have become one of finding a global optimum in N-dimensional space. The vastness of this space precluded such a search and so a working set of parameters was found instead.

During the development of the nodal version of the Yen algorithm, a simple formula was derived which solved for the overhead link utilization due to update transmissions from known network parameters. It was this result which helped to provide the motivation for the development of a hierarchical version. The nodal version, though extremely capable, was limited to networks of a particular size due to the detrimental effects which the overhead traffic produced. The maximum network size was a function of a number of parameters. For the parameter values used in this work, the limiting network size was approximately 100 nodes.

The hierarchical scheme which utilized synchronization warning messages produced the least overhead and the best performance of those tested. The feasibility of this method was demonstrated using the large network of 100 nodes. The real "savings" from this method over the non-synchronized hierarchical scheme came from the substantial reduction in unit warning messages which were passed using the flooding technique.

The ability of the algorithm to function with little performance degradation in an environment of hardware clock limitations and delays due to queued updates was another

positive result. It appears that in general, these random timing inaccuracies were negligable compared to the magnitude of the link weight function calculations. The hybrid characteristics of the combination function also assisted in damping small weight variations which may be caused by these delays.

The limited work which was performed using the VAX 11/780 with the nearly 100 node network came late in the project. However, the results from these tests were central to this work. By demonstrating the workability of the protocol on a complex, non-symmetric network, some of the doubts which haunt small, symmetric simulations are removed. During the development stages, small network simulations were required in order that the mechanics of the algorithm could be verified. Once the hierarchical version was developed, though, the requirement to test it on a large network became the central issue in establishing its credibility. In successfully doing so, a major goal of the project was met.

B. RECOMMENDATIONS FOR FURTHER STUDY

During the development of this project, several courses of action which future work might take became apparent.

Some of these areas require only further simulation with the program in Appendix C, while others require the production of additional programs. The area mentioned first, since it was partially begun during the final phase of this project, is that of large network simulations. Studies could look at the selection of node, group and family update intervals in order to achieve improved performance. Initial results tend to show that larger units did not have to update as often as smaller ones, but this was not conclusive.

The simulation program already includes a routine designed to allows nodes near unit borders to maintain routing information on neighboring nodes belonging to other units. This information would enable packets destined for these neighboring nodes to take the best path to the node itself vice the best path to the group or family of the node. Using the large network, it would be possible to run comparison simulations while varying the "depth" within the unit borders for which node routing table entries would be maintained.

The topology remained fixed throughout all simulations run during this project. The program, though, is capable of modeling a dynamically changing network topology. Link

connectivity can be altered simply by removing or adding links to the node's link set. The addition or deletion of nodes would require a bit more planning since nodes, as permanent entities, are created at the same time. However, the destruction or joining of nodes to the network could still be handled by the deactivation or activation of links. Using this technique, a wide variety of scenarios can develop ranging from static network topologies that experience random node and link failures to completely mobile ones used to represent a packet radio network.

The "Successor" algorithm of Appendix B could be simulated in this dynamic environment. This algorithm could be used in conjunction with work concerned with the "start-up" phase of a packet network. The whole area of network activation is very important to the feasibility of a distributed network and cannot be overlooked in the design considerations.

This work has focused on the application of a distributed routing protocol within a packet-switched communications network. It may be of interest to study the applicability of this protocol in a packet-radio network. Packet radio networks exhibit some unique qualities which

might make the use of a time-dependent routing protocol particularly difficult.

During this work, performance comparisons were done with a routing fraction protocol. The Yen algorithm produces a single best path node for each destination and not a set of routing fractions. Future work may look at developing a set of alternative paths from the Yen algorithm by keeping track of the tentative update transmission times. Relationships may be developed, for example, between the "second best" update transmission time and the selection of alternate paths.

In conclusion, this thesis was a preliminary investigation into the application of the Yen routing algorithm within a packet-switched communications network. The initial development from this study was a modified version of the algorithm which exhibited hybrid operating characteristics when used in conjunction with the proposed combination link weight function. The final algorithm development was a hierarchical version utilizing synchronized unit update transmissions which allowed the protocol to operate in large networks.

APPENDIX A

THE YEN SHORTEST PATH ALGORITHM

In an N-node directed network, let

{I}, I = 1 to N, be the nodes of the network.

(I,J) be the link connecting node I to node J.

W(I,J) ≥ 0 be the weight of(I,J).

F(I,K) be the distance of the tentative shortest path from node I to node K.

T[F(I,K)] be the finite length of time defined to represent the corresponding value of F(I,K).

Let C be the constant such that C = F(I,K)/T[F(I,K)].

Initially, all F(I,K)'s and T[F(I,K)]'s are set to ∞.

STEP 1: At time 0, the destination node K sends each of its Neighbor(out) nodes J a simple message "K".

STEP 2: On receiving a message "K", each node J must:

A. Label the node that has just sent

the message node L and delete node L from its Update Transmission List.

B. Read the clock and let T[F(L,K)] equal the time it reads from the clock and let F(L,K) = C * T[F(L,K)]. F(L,K) is the Update Reception Weight.

C. Update F(J,K) by F(J,K) = min[F(J,K) , W(J,L) + F(L,K)]. F(J,K) is the Tentative Shortest Path Distarce and W(J,L) is the Reverse Link Weight.

D. Let T[F(J,K)] = F(J,K) / C. T[F(J,K)] is the Tentative Time of Update Transmission.

E. At time T[F(J,K)], node J sends the message "J" to Neighbors on its Update Transmission List.

STEP 3: Repeat STEP 2 until time t*, where t* is a predetermined constant larger that any possible T[F(J,K)].

At the termination of the algorithm, each node J has the following solution to the shortest path to destination node K:

1. The distance of the optimal shortest path from node J to the destination node K, which is given by F(J,K), and
2. The identity of the next node on the shortest path from node J to node K, which is indicated by the node which generated the final F(J,K).

APPENDIX B

THE "SUCCESSOR" NODE SELECTION ALGORITHM

The purpose of this algorithm is to provide a means for the hierarchical version of the Yen algorithm to continue to operate in the event that the current Leader node within a basic unit (i.e. group or family) fails. The Leader node is responsible for generating the unit warning message which is then sent to all members. This warning message contains the identity of the Leader node and the time at which all nodes are to send the unit update message. If the Leader node fails, then the unit will no longer be able to provide update information for the rest of the network. This same algorithm is also applicable to network "start-up" periods when the units themselves are forming and Leader nodes must be selected. The following definitions are given which will be utilized in the algorithm.

In an N-node directed network, let:

I, J, K,..., L be nodes belonging to a basic group of the network. A group was chosen for this description yet this is applicable to any level unit.

Each node have a clocking device called a Timer which, upon setting, will time out at t = T(max).

T(max) be greater than any time interval between consecutive group warning messages being received by any node in the group.

GW[I] be the Group Warning message originating from node I.

T(grp) be the average time between generation of consecutive group warning messages by the leader node.

S[I,T(I)] be the Successor message containing the identity of node I and the time T(I) when the Successor message was originated.

The algorithm is presented in terms of the three different events which can occur during network operation resulting from its usage. These events are

1. A node receives a group warning message.
2. A node "times out". This means that the node has not received a group warning message within T(max) seconds. Each node has a re-settable timer which it uses to determine this.

3. A node receives a successor message. Successor messages are sent by nodes which have "timed cut" and are now competing for the function of being the group Leader.

The algorithms related to each event are as follows.

EVENT 1

At time t, node I receives GW[L] from node K and performs the following:

Node I carries out standard group update procedures. This involves retransmitting the group warning message and then scheduling its own group update message to be "fired" at the specified firing time given in the warning message.

Cancels the scheduled transmission of GW[I] if it had been scheduled.

Sets $S[I, T(I)] = S[I, \infty]$.

Resets its Timer.

EVENT 2

At time t*, node J's Timer "times out". Node J then performs the following:

If $T(J) = \infty$,

Let $S[J,T(J)] = S[J,t^*]$;

Transmit $S[J,T(J)]$ to all neighbor nodes within same group;

Schedule the transmission of GW[J] to all neighbor nodes within same group at time $t = t^* + T(max)$;

Reset the Timer.

If $T(J) \neq \infty$,

Transmit $S[J,T(J)]$ to all neighbor nodes within same group;

Reset the Timer.

EVENT 3

At time t**, node J receives $S[I,T(I)]$ from node K. Node J then performs the following:

If node J has already "timed out" and transmitted S[J,T(J)] then,

If T(I) < T(J) then,

Cancel the scheduled transmission of GW[J];

Let S[J,T(J)] = S[I,T(I)];

Transmit S[J,T(J)] to all neighbor nodes within the same group except node K;

Reset the Timer.

If T(I) = T(J) then,

If I < J then,

Cancel the scheduled transmission of GW[J];

Let S[J,T(J)] = S[I,T(I)];

Transmit S[J,T(J)] to all neighbor nodes within the same group except node K;

Reset the Timer.

If I > J then,

No action is taken.

If T(I) > T(J) then,

No action is taken.

If node J has not "timed cut" then,

If T(I) < T(J) then,

Let S[J,T(J)] = S[I,T(I)];

If T(I) = T(J) then,

If I < J then,

Let S[J,T(J)] = S[I,T(I)].

If I > J then,

No action is taken.

If T(I) > T(J) then,

No action is taken.

Figure B.1 shows the network which is used to illustrate how this algorithm performs in the event that the leader node fails. The current leader is node 1 as the network activities commence. The time delay associated with each link is 0.1 seconds with the average time between consecutive group warning messages being 2 seconds. T(max) is 4 seconds which is greater than the normal interval of time between receptions of the group warning message by any node in the unit.

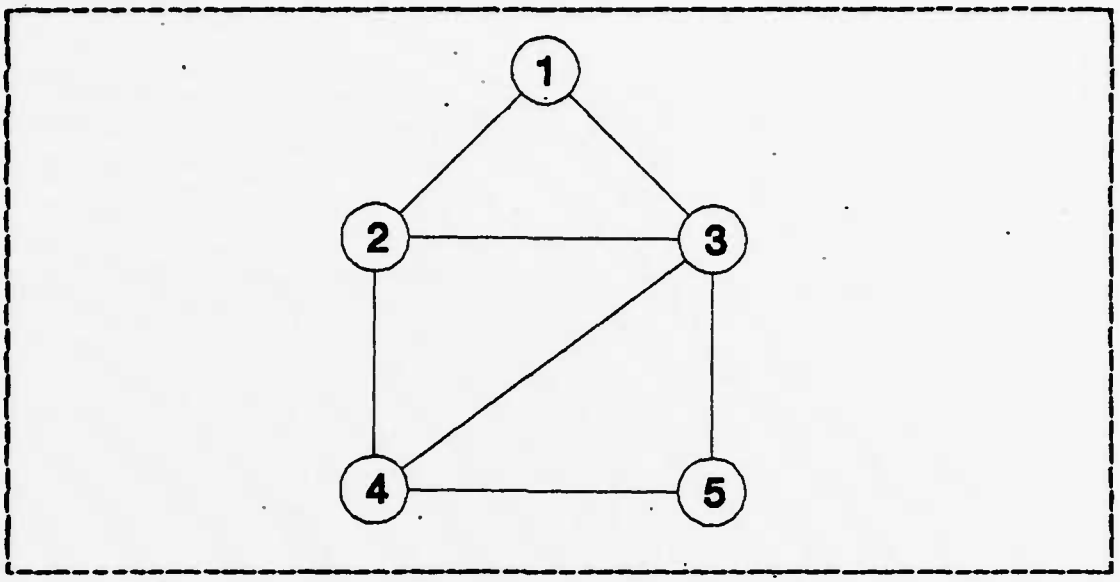


Figure B.1 Network for Demonstrating Successor Algorithm

TIME (SECS)		NODE ACTIVITY
0.0	Ncde 1:	Xmts GW(1) to Nodes 2 and 3.
0.1	Ncde 2:	Rcvs GW(1) from Node 1.
		Sets Timer to 0.1 + 4 = 4.1.
		Xmts GW(1) to Nodes 3 and 4.
	Ncde 3:	Rcvs GW(1) from Node 1.
		Sets Timer to 0.1 + 4 = 4.1.
		Xmts GW(1) to Nodes 2, 4, 5.
0.2	Node 4:	Rcvs GW(1) from Nodes 2 and 4.
		Sets Timer to 0.2 + 4 = 4.2.
		Xmts GW(1) to Node 5.
	Node 5:	Rcvs GW(1) from Node 3.
		Sets Timer to 0.2 + 4 = 4.2.
		Xmts GW(1) to Node 4.
0.3	Node 4:	Ignores GW(1) from Node 5.
	Ncde 5:	Ignores GW(1) from Node 4.

1.0	Ncde 1:	"Fails" as Group Leader.
4.1	Ncde 2:	Timer "Times Out". Xmts S(2 , 4.1) to Nodes 3, 4. Schedules GW(2) xmt at 4.1 + 2 = 6.1. Resets Timer for 4.1 + 4 = 8.1.
	Ncde 3:	Timer "Times Out". Xmts S(3 , 4.1) to Nodes 2, 4, 5. Schedules GW(3) at 4.1 + 2 = 6.1. Resets Timer for 4.1 + 4 = 8.1.
4.2	Ncde 2:	Rcvs S(3 , 4.1) from Node 3. No action is taken.
	Ncde 3:	Rcvs S(2 , 4.1) from Node 2. Cancels GW(3). Xmts S(2 , 4.1) to Nodes 4, 5. Sets Timer for 4.2 + 4 = 8.2.
	Ncde 4:	Timer "Times Out". Rcvs S(2 , 4.1) and S(3 , 4.1). Xmts S(2 , 4.1) to Node 5. Sets Timer for 4.2 + 4 = 8.2.
	Ncde 5:	Timer "Times Out". Rcvs S(3 , 4.1) from Node 3. Xmts S(3 , 4.1) to Node 4. Sets Timer for 4.2 + 4 = 8.2.
4.3	Ncde 4:	Rcvs S(3 , 4.1) from Node 5. No action is taken.
	Ncde 5:	Rcvs S(2 , 4.1) from Node 4. Identity of Node 2 < Ncde 3. Xmts S(2 , 4.1) to Node 3.

		Sets Timer for 4.3 + 4 = 8.3.
6.1	Node 2:	Xmts GW(2) to Nodes 3 and 4.
6.2	Node 3:	Rcvs GW(2) from Node 2. Resets Timer for 6.2 + 4 = 10.2. Xmts GW(2) to Nodes 4 and 5.
	Node 4:	Rcvs GW(2) from Node 2. Resets Timer for 6.2 + 4 = 10.2. Xmts GW(2) to Nodes 3 and 5.
6.3	Node 3:	Ignores GW(3) from Node 4.
	Node 4:	Ignores GW(3) from Node 3.
	Node 5:	Rcvs GW(3) from Nodes 3 and 4. Resets Timer for 6.3 + 4 = 10.3.

Upon conclusion of this example, Node 2 had become the new leader. The algorithm operates in such a manner that nodes closest to the Leader will compete for that function if the leader fails. This occurs because nodes closest to the Leader receive the warning messages first and thus "time out" first in the event that a warning is not received.

APPENDIX C

SIMULATION PROGRAM

```
// EXEC SIM25CLG,REGION.GO=4096K,PARM.GO='MAP,SIZE=760K'
//SYSPRINT DD SYSOUT=A
//SIM.SYSIN DD *
PREAMBLE
**
**       ************************************************************
**       ******                                                ******
**       ******          PACKET SWITCHED NETWORK             ******
**       ******            SIMULATION PROGRAM                ******
**       ******                                              ******
**       ******              UTILIZING A                     ******
**       ******          DYNAMIC ROUTING PROTOCOL            ******
**       ******                BASED ON                      ******
**       ******                                              ******
**       ******       THE YEN SHORTEST PATH ALGORITHM        ******
**       ******          BY  ROBERT R. LOGAN                 ******
**       ******                                              ******
**       ************************************************************
**
NORMALLY MODE IS INTEGER
GENERATE LIST ROUTINES
**
**
PERMANENT ENTITIES.....
**
    EVERY NODE HAS A GROUP, A FAMILY, A XMT.PERCENT, A RCV.PERCENT,
               A OPT.RCV,
               OWNS A LINK.SET, A XMT.RECORD, A ALARM.CLOCK,
                    A VIRT.CKT.LIST
          DEFINE XMT.PERCENT, RCV.PERCENT AS REAL VARIABLES
**
**
TEMPORARY ENTITIES.......
**
    EVERY UPDATE HAS A ORIGIN, A BEGIN.TIME, A FIRING.TIME,
                       A CLASS, A FM.NODE, A TO.NODE, A HOPCNT,
                       A VARIETY,
                  MAY BELONG TO A UQUEUE, A UPROP.QUEUE
          DEFINE BEGIN.TIME, FIRING.TIME AS REAL VARIABLES
**
    EVERY PACKET HAS A ORG.NODE, A BEG.TIME, A HOP.COUNT, A DEST.NODE,
                       A RELAY.NODE, A NEXT.NODE, A MSG.ID.NUM, A PKT.ID,
                       A PKT.SUM, A LIFETIME, A QSTAT, A P.NAME,
                       A P.ARRIVAL,
                  MAY BELONG TO A QUEUE, A PROP.QUEUE
          DEFINE BEG.TIME, P.ARRIVAL AS REAL VARIABLES
**
```

```
    EVERY LINK HAS A STATUS, A TERM, A QTEST, A QUESIZ, A LK.WEIGHT,
               A UPCOUNT, A PTCOUNT,
               OWNS A QUEUE, A PROP.QUEUE, A SET.OF.WEIGHTS,
                    A UQUEUE, A UPROP.QUEUE,
               MAY BELONG TO A LINK.SET
          DEFINE LK.WEIGHT AS A REAL VARIABLE
..
    EVERY WEIGHT HAS A WT.SIZE,
                MAY BELONG TO A SET.OF.WEIGHTS
          DEFINE WT.SIZE AS A REAL VARIABLE
..
    EVERY CIRCUIT HAS A CKT.ID, A VIRT.BP,
                 MAY BELONG TO A VIRT.CKT.LIST
..
    EVERY RECORD HAS A REC.ORIGIN, A REC.TIME,
                MAY BELONG TO A XMT.RECORD
          DEFINE REC.TIME  AS A REAL VARIABLE
..
    EVERY BUZZER HAS A SETTING, A CAUSE, A TEMP.BP, A BUZ.BEG,
                    A BUZ.WAKE,
                OWNS A BUZ.LIST,
                MAY BELONG TO A ALARM.CLOCK
          DEFINE SETTING, BUZ.BEG AS REAL VARIABLES
..
    EVERY BUZ.NODE HAS AN ID.NUMBER,
                  MAY BELONG TO A BUZ.LIST
..
..
EVENT NOTICES INCLUDE U5.LINK.WEIGHT.CALCULATION, E1.TRANSIENT.BLANKING,
    E2.COLLECT.DATA, E4.NETWORK.PERFORMANCE.REPORT
..
    EVERY U1.GENERATE.UPDATE HAS A SENDING.NODE, A UP.CLASS
..
    EVERY U4.RECEIVE.UPDATE HAS A UP.MESSAGE
..
    EVERY U6.NODE.WAKE.UP HAS A WK.NODE, A WK.MSG, A BPATH, A HOPNUM
..
    EVERY U7.UNIT.FIRING HAS A PF.NODE, A PF.CLASS, A PF.ORIGIN,
                              A PF.BEGIN
              DEFINE PF.BEGIN AS A REAL VARIABLE
..
    EVERY U8.ERASE.RECORD HAS A ERA.NODE, A ERA.RECORD
..
    EVERY M1.GENERATE.MESSAGE HAS A T.MESSAGE
..
    EVERY M2.RECEIVE.MESSAGE.PACKET HAS A PA.CKET
..
    PRIORITY ORDER IS U4.RECEIVE.UPDATE, M2.RECEIVE.MESSAGE.PACKET,
         M1.GENERATE.MESSAGE, U1.GENERATE.UPDATE,
         U7.UNIT.FIRING, U6.NODE.WAKE.UP, U8.ERASE.RECORD,
```

```
        E2.COLLECT.DATA, U5.LINK.WEIGHT.CALCULATION,
        E1.TRANSIENT.BLANKING, E4.NETWORK.PERFORMANCE.REPORT
**
**  DEFINE QUEUE AS A SET RANKED BY HIGH HOP.COUNT
**
DEFINE   NOD.TIME,    BRACKET,    U.XMN.TIME,    PKT.XMN.TIME,
         GP.FIRE,     GRP.TIME,   FM.FIRE,    FAM.TIME,    DELTA,
         MSG.GENERATION.INTERVAL,  TIME.LIMIT,  ERA.PERIOD,
         INC.PPM,     END.PPM,  PPS.BEG,    PPS.INC,     PPS.END,
         PPS.AVE,     AVE.PPM,    MAX.PPM,  XMT.TOTAL,    QU.TOTAL,
         BEG.PPM,     RCV.PCNT,   TRNS.PCNT, LNK.NOD.RATIO,
         MAX.LINKS.PER.NODE,      MAX.GRP.NODES,  XCOUNT, WT.TIME,
         MAX.FAM.NODES,  IN.GROUP,   IN.FAMILY,  TRANSIENT,
         QU.FACT, BIT.INTERVAL, NO.STEPS, NUL.TRANSIENT, NO.FINISH
                                                  AS REAL VARIABLES
**
**
DEFINE   TRAF.LIMIT,   UP.COUNT,   HOP.LIMIT,   LINKS, IPPM, IMSG,
         INIT.BP,  NUM.REPORTS.REQUESTED, TEST, SAMPLES, NUMRUNS,
         NEW.MSG.TOTAL,  NEW.PKT.TOTAL,  COMP.TRIP.PKT, SYS.PACKETS,
         HOP.TOTAL,  NODE.FACTOR,  PRNT,  FMLYS,  GRPS,  NGFS,
         NUM.OF.WEIGHTS,  MANNER.OF.WEIGHTING, STEP.BY.WEIGHTS,
         USE.VIRT.CKTS, DISCRETE.TIMER, U.COUNT, STATIC.BP,
         BY.GEOM.DIST, GEOMETRIC, STEP.BY.NOD.TIME, SYNCH.FIRING
                                                  AS INTEGER VARIABLES
**
ACCUMULATE QUE.PKT.TIME AS THE SUM OF QSTAT
ACCUMULATE TRANSIT.TIME AS THE SUM OF LIFETIME
ACCUMULATE LK.MEAN AS THE MEAN OF STATUS
ACCUMULATE QU.MAX AS THE MAXIMUM, QU.MEAN AS THE MEAN, QU.DEV
        AS THE STD.DEV OF N.QUEUE
ACCUMULATE UP.MAX AS THE MAXIMUM, UP.MEAN AS THE MEAN, UP.DEV
        AS THE STD.DEV OF N.UQUEUE
ACCUMULATE AVG.WEIGHT AS THE MEAN OF QUESIZ
ACCUMULATE UP.RHO AS THE MEAN OF N.UPROP.QUEUE
ACCUMULATE RHO AS THE MEAN OF N.PROP.QUEUE
ACCUMULATE AV.SYS.PACKETS AS THE MEAN OF SYS.PACKETS
**
DEFINE   OUTQ        TO MEAN      0
DEFINE   INQ         TO MEAN      1
DEFINE   DEAD        TO MEAN      0
DEFINE   ALIVE       TO MEAN      1
DEFINE   IDLE        TO MEAN      0
DEFINE   BUSY        TO MEAN      1
DEFINE   REGULAR     TO MEAN      1
DEFINE   WARNING     TO MEAN      2
DEFINE   NOD.MSG     TO MEAN      1
DEFINE   GRP.MSG     TO MEAN      2
DEFINE   FAM.MSG     TO MEAN      3
DEFINE   NONE        TO MEAN      0
```

```
DEFINE   BY.QSIZE.NOW   TO MEAN    1
DEFINE   BY.PAST.QSIZE  TO MEAN    2
DEFINE   BY.PAST.QAVG   TO MEAN    3
DEFINE   BY.RHO.WEIGHT  TO MEAN    4
DEFINE   BY.COMBINE     TO MEAN    5
''
DEFINE   BEST.PATH      AS A 2-DIMENSIONAL ARRAY
DEFINE   DUP.BEST.PATH  AS A 2-DIMENSIONAL ARRAY
DEFINE   FAM.OF.GRP     AS A 1-DIMENSIONAL ARRAY
DEFINE   BORN.PKTS      AS A 2-DIMENSIONAL ARRAY
DEFINE   GONE.PKTS      AS A 2-DIMENSIONAL ARRAY
DEFINE   JUMP.TOTAL     AS A 2-DIMENSIONAL ARRAY
DEFINE   LENGTH.TRIP    AS A 2-DIMENSIONAL REAL ARRAY
''
END ''OF PREAMBLE
''  ××××××
```

```
''            1.6  P6.PURGE.EVENT.QUEUE (ROUTINE)
''            1.7  P7.ZEROIZE.SETS (ROUTINE)
''
''       2.   *UPDATE* ROUTING PROTOCOL (U)
''
''            2.1  U1.GENERATE.UPDATE (EVENT)
''            2.2  U2.TRANSMIT.UPDATE (ROUTINE)
''            2.3  U3.INSERT.UPDATE (ROUTINE)
''            2.4  U4.RECEIVE.UPDATE (EVENT)
''            2.5  U5.LINK.WEIGHT.CALCULATION (EVENT)
''            2.6  U6.NODE.WAKE.UP (EVENT)
''            2.7  U7.UNIT.FIRING (EVENT)
''            2.8  U8.ERASE.RECORD (EVENT)
''
''       3.   *MESSAGE* PACKET TRANSPORT (M)
''
''            3.1  M1.GENERATE.MESSAGE (EVENT)
''            3.2  M2.RECEIVE.MESSAGE.PACKET (EVENT)
''
''       4.   *EVALUATION* OF NETWORK PERFORMANCE (E)
''
''            4.1  E1.TRANSIENT.BLANKING (EVENT)
''            4.2  E2.COLLECT.DATA (EVENT)
''            4.3  E3.PARAMETER.LISTING (ROUTINE)
''            4.4  E4.NETWORK.PERFORMANCE.REPORT (EVENT)
''            4.5  E5.BEST.PATH.ROUTING.TABLE (ROUTINE)
''            4.6  E6.LINK.WEIGHT.MATRIX (ROUTINE)
''            4.7  E7.TRAFFIC.DISTRIBUTION (ROUTINE)
''
'' GLOBAL VARIABLE DESCRIPTION: (ALPHABETICAL ORDER)
''
''       AVE.PPM          PACKETS PER MESSAGE (AVERAGE VALUE)
''       BEG.PPM          PACKETS PER MESSAGE (MINIMUM VALUE)
''       BIT.INTERVAL     THE INTERVAL (IN SECONDS) OF THE DISCRETE CLOCK
''                        WHICH MAY BE USED IN THE SIMULATION.
''       BRACKET          THE TIME WINDOW IN WHICH A NODE MAY
''                        GENERATE AN UPDATE MESSAGE.
''                        BRACKET WILL EQUAL .1 * NOD.TIME.
''       DELTA            A VALUE (0 < DELTA < 1) WHICH DICTATES THE
''                        MULTIPLICATION FACTOR USED IN WEIGHTING THE
''                        LINKS WHEN A NODE IS COMPUTING THE SETTING
''                        TIME FOR THE "ALARM CLOCK" ASSOCIATED WITH AN
''                        UPDATE MESSAGE.
''       ERA.PERIOD       THE TIME LENGTH FOR WHICH A NODE MAINTAINS ITS
''                        RECORDS OF PREVIOUS UPDATE MESSAGE RETRANSMITS.
''       FAM.TIME         SAME AS NOD.TIME EXCEPT CONCERNS FAMILY
''                        UPDATES BEING GENERATED BY A SINGLE NODE.
''       FM.FIRE          SAME AS GR.FIRE EXCEPT CONCERNS FAMILY MESSAGE.
''                        LET FM.FIRE   =    MAX.FAM.NODES * U.XMN.TIME.
''       FMLYS            THE NUMBER OF FAMILIES.
```

```
''      GP.FIRE          THE TIME FROM ORIGINATION OF A GROUP UPDATE
''                       WARNING MESSAGE BY A NODE, AT WHICH ALL NODES
''                       IN THE SAME GROUP SIMULTANEOUSLY TRANSMIT THE
''                       GROUP UPDATE MESSAGE.  THIS TIME ALLOWS FOR
''                       THE WARNING MESSAGE TO PROPAGATE TO ALL NODES
''                       IN THE GROUP.
''                       LET GP.FIRE   =    MAX.GRP.NODES * U.XMN.TIME.
''      GRPS             THE NUMBER OF GROUPS.
''      GRP.TIME         SAME AS NOD.TIME EXCEPT CONCERNS GROUP
''                       UPDATES BEING GENERATED BY A SINGLE NODE.
''      HOP.LIMIT        THE MAXIMUM NUMBER OF HOPS WHICH AN UPDATE
''                       MESSAGE CAN TRAVEL WITHIN GROUPS OR FAMILIES
''                       NOT WITHIN THE ORIGINATOR'S BASIC GROUP/FAMILY.
''                       (I.E. USED FOR ESTABLISHING BORDER NODE PATHS).
''      IN.GROUP         THE PERCENTAGE OF GENERATED MESSAGES THAT WILL
''                       NOT LEAVE THE BASIC GROUP.
''      IN.FAMILY        THE PERCENTAGE OF GENERATED MESSAGES THAT WILL
''                       NOT LEAVE THE BASIC FAMILY.
''      LNK.NODE.RATIO THE RATIO OF LINKS TO NODES IN THE NETWORK.
''      MAX.FAM.NODES  MAXIMUM NUMBER OF NODES PER FAMILY.
''      MAX.GRP.NODES  MAXIMUM NUMBER OF NODES PER GROUP.
''      MAX.LINKS.PER.NODE   THE MAXIMUM NUMBER OF LINKS FOR A NODE
''                       IN THE NETWORK.
''      MAX.PPM          PACKETS PER MESSAGE (MAXIMUM VALUE)
''      NGFS             (# OF NODES) + (# OF GROUPS) + (# OF FAMILIES)
''      NODE.FACTOR      A NORMALIZATION FACTOR WHICH ALLOWS NETWORKS OF
''                       DIFFERENT SIZES TO BE COMPARED USING THE
''                       AVERAGE NUMBER OF PACKETS/SECOND/NODE AS A
''                       CONSTANT MEASURE OF NETWORK LOADING.FOR
''                       EXAMPLE, IF A 5 NODE NETWORK IS USED AS THE
''                       COMPARISON BASE THEN A 10 NODE NETWORK HAS A
''                       NODE.FACTOR OF 2.
''      NOD.TIME         THE MINIMUM LENGTH OF TIME BETWEEN
''                       CONSECUTIVE NODE UPDATES BEING GENERATED
''                       BY A SINGLE NODE.
''      NUL.TRANSIENT    THE TIME WHEN TRANSIENT EFFECTS ARE ERASED FROM
''                       THE ACCUMULATED VARIABLES.
''      NUM.OF.WEIGHTS THE NUMBER OF MOST RECENT WEIGHTS CONSIDERED
''                       IN THE COMPUTATION OF THE LINK WEIGHT.
''      NUM.REPORTS.REQUESTED    THE NUMBER OF NETWORK STATUS REPORTS
''                       REQUESTED DURING A SIMULATION RUN.
''      PKT.XMN.TIME   TIME FOR TRANSMISSION OF A  PACKET
''                       OVER A LINK BETWEEN TWO NODES.
''      PPS.BEG          PACKETS PER SECOND (MINIMUM VALUE)
''      PPS.INC            "     "     "   (INCREMENT  " )
''      PPS.END            "     "     "   (MAXIMUM    " )
''      PPS.AVE            "     "     "   (AVERAGE    " )
''      PRNT             AN INTEGER WHICH CONTROLS THE LEVEL OF
''                       DIAGNOSTIC PRINTING.
''         0 ===>        NETWORK TOPOLOGY + INITIAL DATA + NET.REPORTS
```

```
''          1 ===> 0 =  TRACES ALL PACKETS
''          2 ===> 1 =  ANNOUNCES ALL UPDATE 'WAKE UPS' WITH BEST PATHS
''          3 ===> 2 =  ANNOUNCES ORIGINATION, RELAY/ARRIVAL OF
''                      ALL UPDATE MESSAGES= ERASURE OF XMT.RECORDS
''      QU.FACT         WHEN USING THE WEIGHTING METHOD 'COMBINE' WE
''                      USE A COMBINATION OF THE QUEUE SIZE NOW AND
''                      THE UTILIZATION FACTOR.  THE MAX VALUE OF THE
''                      UTILIZATION FACTOR TERM IS 20. THEREFORE WE
''                      USE QU.FACT AS A SCALING FACTOR FOR THE VALUE
''                      OF THE QUEUE SIZE NOW TERM.
''      SAMPLES         NUMBER OF DATA POINTS REQUESTED PER RUN.
''      TIME.LIMIT      THE LENGTH OF TIME FOR WHICH THE SIMULATION
''                      LASTS.
''      TRAF.LIMIT      THE MAXIMUM NUMBER OF MESSAGES GENERATED
''                      DURING A SIMULATION RUN.
''      WT.TIME         TIME INTERVAL FOR WHICH LINK WEIGHTS ARE FOUND.
''      U.XMN.TIME      TIME FOR TRANSMISSION OF AN UPDATE
''                      OVER A LINK BETWEEN TWO NODES.
''.....
''.....     BEGINNING OF USER CONTROLLED PARAMETERS      .............
''.....
''
''  FIXED NETWORK VALUES...
''       NOT SUBJECT TO CHANGE WITHIN THE SIMULATION.
''
LET PKT.XMN.TIME   =    0.05000
LET U.XMN.TIME     =    0.00100
LET BIT.INTERVAL   =    0.000001
''.....
''
''  VARIABLE PARAMETERS...
''       SUBJECT TO CHANGE WITHIN THE SIMULATION.
''
LET NOD.TIME       =    0.500
LET GRP.TIME       =    1.000
LET FAM.TIME       =    2.000
LET DELTA          =    0.00100
LET ERA.PERIOD     =    1.000
LET HOP.LIMIT      =    0
LET IN.GROUP       =    0.000
LET IN.FAMILY      =    0.000
LET NODE.FACTOR    =    1
''.....
''
''  SIMULATION RUN PARAMETERS...
''
LET TIME.LIMIT     =    30.000
LET TRAF.LIMIT     =    100000
LET NUL.TRANSIENT  =    5.0
''.....
```

```
''
''  REPORTING AND DATA COLLECTION PARAMETERS...
''
LET SAMPLES                 =    30
LET NUM.REPORTS.REQUESTED   =     1
LET PRNT                    =     0
''
''.....................................................................
''
''  -----------------   TEST ROUTING   ---------------------------------
''
''  TEST    : (ALIVE/DEAD) SET WHEN THE BEST PATHS TO A GIVEN NODE ARE
''            DESIRED.  THE SIMULATION STOPS AFTER THESE PATHS ARE FOUND.
''
LET TEST     =    DEAD
LET TST.NODE =      1
''
''  -----------------   STATIC ROUTING   -------------------------------
''
''  STATIC.BP : (ALIVE/DEAD)  SET WHEN THE SIMULATION IS TO BE RUN
''              USING STATIC ROUTING BASED UPON THE INITIAL BEST PATHS
''              OBTAINED VIA THE INITIAL LEAST HOPS METHOD.
''
LET STATIC.BP  =  DEAD
''
''  -----------------   DYNAMIC ROUTING   ------------------------------
''
''  MANNER OF WEIGHTING MAY BE :  (1) BY.QSIZE.NOW
''                                (2) BY.PAST.QSIZE
''                                (3) BY.PAST.QAVG
''                                (4) BY.RHO.WEIGHT
''                                (5) BY.COMBINE
''
LET MANNER.OF.WEIGHTING  =  BY.COMBINE
LET QU.FACT              =  1.00
LET WT.TIME              =  PKT.XMN.TIME * 7.5
LET NUM.OF.WEIGHTS       =  10
''
''  ------  MULTIPLE SIMULATION RUN PARAMETERS  ----------------------
''
''  IF NONE OF THE SPECIAL OPTIONS BELOW ARE SELECTED THEN THE PROGRAM
''  IS RUN WITH THE DEFAULT MULTIPLE RUNS INVOLVING VARYING THE
''  PACKETS PER SECOND AND THE PACKETS PER MESSAGE FOR THE NETWORK.
''
''  OPTION 1:      VARYING THE NUMBER OF WEIGHTS
''
LET STEP.BY.WEIGHTS =  DEAD
''  THE STARTING VALUE IS GIVEN BY THE VARIABLE NUM.OF.WEIGHTS.
LET WT.STEPS        =  3
LET WT.FINISH       =  13
```

```
''
''  OPTION 2:       VARYING THE TIME BETWEEN NODE UPDATES.
''
LET ST.EP.BY.NOD.TIME =  DEAD
''  THE STARTING VALUE IS GIVEN BY THE VARIABLE NOD.TIME.
LET ND.STEPS          =  0.25
LET ND.FINISH         =  1.5
''
''  ----------------  VIRTUAL CIRCUITS  ------------------------------
''
''  THE SIMULATION MAY BE RUN USING VIRTUAL CIRCUITS (WHEREBY ALL
''  PACKETS BELONGING TO THE SAME MESSAGE TAKE THE SAME ROUTE) OR
''  NOT (WHEREBY PACKETS TRAVEL ACCORDING TO CURRENT BEST PATH
''  INFORMATION AND PACKETS BELONGING TO THE SAME MESSAGE MAY
''  INDEED TAKE DIFFERENT ROUTES).
''
LET USE.VIRT.CKTS = DEAD
''
''  ----------------  DISCRETE TIMER  --------------------------------
''
''  THIS OPTION ALLOWS THE SIMULATION TO BE RUN USING A DISCRETE
''  CLOCK SYSTEM WHICH ACCOUNTS FOR THE SYNCHRONIZATION PRESENT
''  IN THE NETWORK.  BASICALLY A NODE IS ONLY ALLOWED TO TRANSMIT
''  EITHER UPDATE OR MESSAGE PACKETS AT SPECIFIC DISCRETE POINTS
''  IN TIME.  ALL NODES ARE "SYNCHRONIZED" AT THE START OF THE
''  SIMULATION AND THEREAFTER THE CLOCK "PERIOD" IS SET EQUAL TO
''  THE TIME IT TAKES FOR AN UPDATE PACKET TO BE SENT.  THUS ALL
''  PACKETS WILL BE TRANSMITTED AT TIMES EQUAL TO N * U.XMN.TIME
''  (WHERE N IS AN INTEGER).
''
LET DISCRETE.TIMER = DEAD
''
''  ------   SYNCHRONIZED GROUP/FAMILY UPDATE XMTS  ------------------
''
''  THE SIMULATION CAN BE RUN UNDER THE SYSTEM WHERE GROUP AND FAMILY
''  UPDATE MESSAGES ARE SIMULTANEOUSLY SENT BY BORDER NODES OF THE BASIC
''  UNIT TO 'OUTSIDE' NODES.  THIS IS BASED UPON A SYNCHRONIZED SCHEDULE
''  OF UPDATE "FIRINGS" WHICH DO NOT REQUIRE THE WARNING MESSAGES THAT
''  NOTIFY OF FUTURE UPDATE TIMES UNDER THE STANDARD SCHEME.
''  THIS GREATLY REDUCES THE UPDATE OVERHEAD DUE TO 'FLOODING' OF
''  WARNING MESSAGES WITHIN THE BASIC UNIT.
''
LET SYNCH.FIRING = DEAD
''
''  ----------------   TRAFFIC PARAMETERS  ----------------------------
''
LET PPS.BEG    =   100.  *   REAL.F(NODE.FACTOR)
LET PPS.INC    =   100.  *   REAL.F(NODE.FACTOR)
LET PPS.END    =   500.  *   REAL.F(NODE.FACTOR)
''
```

```
**  THE SIMULATION MAY BE RUN USING EITHER A UNIFORM OR A GEOMETRIC
**  DISTRIBTION FOR THE NUMBER OF PACKETS PER MESSAGE.  IF A UNIFORM
**  DISTRIBUTION IS SELECTED THEN THE PPM VARIABLE BELOW IS THE
**  MAXIMUM VALUE FOR THE UNIFORM DISTRIBUTION OF MESSAGE LENGTHS
**  (I.E. THE MESSAGE LENGTHS WILL VARY UNIFORMLY FROM 1 TO PPM PACKETS
**  IN LENGTH).  THEREFORE THE AVERAGE MESSAGE LENGTH WILL BE
**  (PPM + 1)/2 PACKETS LONG.  IF THE GEOMETRIC DISTRIBUTION IS
**  SELECTED THEN THE PPM VARIABLE CORRESPONDS TO THE AVERAGE LENGTH
**  OF A MESSAGE IN PACKETS.
**
LET BY.GEOM.DIST = ALIVE
**
LET BEG.PPM     =     5.
LET INC.PPM     =     5.
LET END.PPM     =    20.
**
**.....................................................................
**.............     END OF USER CONTROLLED PARAMETERS      .............
**.....................................................................
**
**  COMPUTE THE NUMBER OF RUNS WHICH WILL BE MADE.
**
LET IPPM = 1
LET XCOUNT = BEG.PPM
'RUN.PPM'
LET XCOUNT = XCOUNT + INC.PPM
IF XCOUNT <=  END.PPM,
    LET IPPM = IPPM + 1
    GO RUN.PPM
REGARDLESS
LET IMSG = 1
LET XCOUNT = PPS.BEG
'RUN.MSG'
LET XCOUNT = XCOUNT + PPS.INC
IF XCOUNT <= PPS.END,
    LET IMSG = IMSG + 1
    GO RUN.MSG
REGARDLESS
LET NUMRUNS     =  IPPM * IMSG
**
IF STEP.BY.WEIGHTS = ALIVE,
    LET NUMRUNS = 1
    LET XCOUNT = NUM.OF.WEIGHTS
    'WT.HERE'
    LET XCOUNT = XCOUNT + WT.STEPS
    IF XCOUNT > WT.FINISH,
         GO WT.END
    REGARDLESS
    LET NUMRUNS = NUMRUNS + 1
    GO WT.HERE
```

```
    'WT.END'
REGARDLESS
**
IF ST.EP.BY.NOD.TIME = ALIVE,
    LET NUMRUNS = 1
    LET XCOUNT = NOD.TIME
    'ND.HERE'
    LET XCOUNT = XCOUNT + ND.STEPS
    IF XCOUNT > ND.FINISH,
        GO ND.END
    REGARDLESS
    LET NUMRUNS = NUMRUNS + 1
    GO ND.HERE
    'ND.END'
REGARDLESS
**
USE UNIT 8 FOR OUTPUT
    WRITE NUMRUNS AS I 5
USE UNIT 6 FOR OUTPUT
**
PERFORM P1.BUILD.NETWORK
RELEASE P1.BUILD.NETWORK
**
**  THE INITIAL VALUES FOR THE FIRST SIMULATION RUN ARE COMPUTED.
**
LET GP.FIRE    =    MAX.GRP.NODES * U.XMN.TIME
LET FM.FIRE    =    MAX.FAM.NODES * U.XMN.TIME
LET BRACKET    =    NOD.TIME  *  0.1
**
LET PPS.AVE = PPS.BEG
LET MAX.PPM = BEG.PPM
LET AVE.PPM = (1.0 + MAX.PPM) / 2.
IF BY.GEOM.DIST = ALIVE,
    LET AVE.PPM = MAX.PPM
REGARDLESS
LET MSG.GENERATION.INTERVAL = AVE.PPM / PPS.AVE
**
** TEST PROCEDURE
IF TEST  =  ALIVE,
    LET TIME.LIMIT  =   100.0
    LET INIT.BP = ALIVE
    LET DELTA = 1.0
    LET U.XMN.TIME = 0.0
    SCHEDULE A U1.GENERATE.UPDATE GIVEN  TST.NODE, NOD.MSG AT 0.0
    GO RUN.IT
REGARDLESS
**
LET INIT.BP = ALIVE
**
PERFORM P3.INITIAL.ROUTING.TABLE
```

```
''
GO RUN.IT
''
'BEGIN.SIMULATION'
''
''  INITIALIZE THE BEST PATH MATRIX.
''
FOR I = 1 TO NGFS, DO
    FOR J = 1 TO NGFS, DO
        LET BEST.PATH(I,J) = DUP.BEST.PATH(I,J)
    LOOP
LOOP
''
'SET.UP'
''
''  RECORD PLOT INFORMATION PRIOR TO DATA COLLECTION.
''
USE UNIT 8 FOR OUTPUT
WRITE SAMPLES, N.NODE, LINKS, DELTA, PPS.AVE, AVE.PPM,
    NOD.TIME, IN.GROUP, IN.FAMILY, STATIC.BP,
    MANNER.OF.WEIGHTING, NUM.OF.WEIGHTS, WT.TIME,
    U.XMN.TIME, PKT.XMN.TIME, BIT.INTERVAL, PR.COLLISION,
    USE.VIRT.CKTS, DISCRETE.TIMER, BY.COLLISION, BY.GEOM.DIST
    AS /,B 1,3 I 5,3 D(10,6),/,B 1,3 D(10,3),1 I 5,/,B 1,
    2 I 5,1 D(10,6),/,B 1,4 D(10,6),4 I 3
USE UNIT 6 FOR OUTPUT
''
''  RUNNING THE SIMULATION USING STATIC BEST PATHS
''
IF STATIC.BP = ALIVE
    PERFORM P4.STATIC.EVENTS
    GO RUN.IT
REGARDLESS
''
''
PERFORM P5.DYNAMIC.EVENTS
''
'RUN.IT'
''
START SIMULATION
''
IF TEST = ALIVE,
    PERFORM E5.BEST.PATH.ROUTING.TABLE
    GO END.IT
REGARDLESS
''
IF INIT.BP = ALIVE
    PERFORM E5.BEST.PATH.ROUTING.TABLE
    PERFORM P6.PURGE.EVENT.QUEUE
    PERFORM P7.ZEROIZE.SETS
```

```
    LET  PRNT  =  0
    LET INIT.BP = DEAD
    PERFORM E3.PARAMETER.LISTING
    GO SET.UP
REGARDLESS
''
PERFORM P7.ZEROIZE.SETS
''
''  THE SIMULATION MAY BE RUN USING A DIFFERENT NUMBER OF WINDOWS FOR
''  COMPUTING THE WEIGHTS OF THE LINKS.  IN THIS WAY ONE CAN SEE THE
''  EFFECTS OF VARYING WEIGHT LENGTHS IN THE OPERATION OF THE SYSTEM.
''
IF STEP.BY.WEIGHTS = ALIVE,
''
    LET NUM.OF.WEIGHTS = NUM.OF.WEIGHTS + WT.STEPS
    IF NUM.OF.WEIGHTS > WT.FINISH,
         GO END.IT
    REGARDLESS
    GO BEGIN.SIMULATION
REGARDLESS
''
''  THE SIMULATION MAY BE RUN USING A DIFFERENT TIME BETWEEN
''  NODE UPDATES.  IN THIS WAY ONE CAN SEE THE EFFECTS OF CHANGING
''  THE 'SPEED' AT WHICH THE NETWORK PRODUCES ROUTING INFORMATION.
''
IF ST.EP.BY.NOD.TIME = ALIVE,
''
    LET NOD.TIME = NOD.TIME + ND.STEPS
    IF NOD.TIME > ND.FINISH,
         GO END.IT
    REGARDLESS
    GO BEGIN.SIMULATION
REGARDLESS
''
''  THE SIMULATION CAN BE REPEATED USING A CHANGING MESSAGE
''  GENERATION INTERVAL CONTROLLED BY AN INCREMENTAL SUM.  IN THIS
''  MANNER THE NETWORK CAN BE TESTED UNDER DIFFERENT TRAFFIC LOAD
''  CONDITIONS WHILE HOLDING CONSTANT THE OTHER NETWORK PARAMETERS.
''
LET PPS.AVE = PPS.AVE + PPS.INC
''
IF PPS.AVE <= PPS.END,
''
    LET MSG.GENERATION.INTERVAL = AVE.PPM / PPS.AVE
    GO BEGIN.SIMULATION
REGARDLESS
''
''  AFTER THE SIMULATION IS PERFORMED USING DIFFERENT MESSAGE
''  GENERATION INTERVALS, THE SIMULATION CAN BE RUN AGAIN USING
''  A NEW VALUE FOR EITHER THE MAXIMUM PACKETS PER MESSAGE ALLOWED
```

```
**  (IF A UNIFORM DISTRIBUTION FOR MESSAGE LENGTHS IS USED) OR A
**  NEW VALUE FOR THE AVERAGE MESSAGE LENGTH (IF A GEOMETRIC
**  DISTRIBUTION FOR MESSAGE LENGTHS IS USED).
**
LET PPS.AVE = PPS.BEG
LET MAX.PPM = MAX.PPM + INC.PPM
**
IF MAX.PPM <= END.PPM,
**
    LET AVE.PPM = (MAX.PPM + 1.0) / 2.
**
    IF BY.GEOM.DIST = ALIVE,
        LET AVE.PPM = MAX.PPM
    REGARDLESS
    LET MSG.GENERATION.INTERVAL = AVE.PPM / PPS.AVE
    GO BEGIN.SIMULATION
REGARDLESS
'END.IT'
STOP
**
END "OF MAIN
**  ********************************************************************************
**
ROUTINE FOR P1.BUILD.NETWORK
**
**  THIS ROUTINE INITIALIZES ALL THE VARIABLES WHICH ARE INVOLVED
**  IN THE ACTUAL CONSTRUCTION OF THE NETWORK.
**
**  THIS ROUTINE IS CALLED FROM 'MAIN'.
DEFINE GRPCOUNT, FAMCOUNT  AS REAL VARIABLES
**
READ N.NODE
**
START NEW PAGE
SKIP 5 LINES
PRINT 6 LINES AS FOLLOWS
        **************************************************************
        ********************* NETWORK TOPOLOGY ***********************
        **************************************************************

        NODE      TRANSMIT     RECEIVE     GROUP     FAMILY      OPTIONAL
         NO.       FACTOR       FACTOR    (PGM =)    (PGM =)     RECEIVER
**
SKIP 1 LINE
CREATE EVERY NODE
FOR EVERY NODE
 READ XMT.PERCENT(NODE), RCV.PERCENT(NODE), GROUP(NODE), FAMILY(NODE),
    OPT.RCV(NODE)
**
**     TRNS.PCNT AND RCV.PCNT ARE THE SUM OF TRANSMIT AND RECEIVE FACTORS.
```

```
''  THESE VALUES ARE USED WHEN UNBALANCED TRAFFIC PATTERNS ARE DESIRED.
''    GROUP NUMBERS ARE ADDED TO N.NODE TO GET PROGRAM GROUP NUMBERS.
''  FAMILY NUMBERS ARE ADDED TO N.NODE + THE HIGHEST GROUP NUMBER TO GET
''  THE PROGRAM FAMILY NUMBER.
''    THE OPTIONAL RECEIVER FOR EACH NODE IS A DEDICATED RECEIVING NODE
''  FOR ALL TRAFFIC TRANSMITTED BY THAT NODE.  FOR EXAMPLE IF THE
''  OPTIONAL RECEIVER OF NODE 3 IS NODE 7 THEN ANY TRAFFIC ORIGINATING
''  FROM NODE 3 WILL BE DESTINED FOR NODE 7.  IN THIS WAY SPECIFIC
''  IMBALANCED LOAD CONDITIONS CAN BE SIMULATED.  IF NO OPTIONAL
''  RECEIVER IS STATED THEN THE VALUE IS ZERO.
''
''  WITHIN THE SIMULATION GROUPS AND FAMILIES ARE HANDLED AS IF
''     THEY WERE SUPER-NODES.  A USEFUL ANALOGY IS TO
''     ENVISION MANY SUB-NODES WITHIN A GROUP OR FAMILY SUPER-
''     NODE.  ACCESS TO THE SUB-NODES IS CONTROLLED BY THE
''     THE SUPER-NODE'S ADDRESS.
''
LET GRPS       =    NONE
LET FMLYS      =    NONE
LET NGFS       =    NONE
LET GRPCOUNT   =    0.0
LET FAMCOUNT   =    0.0
''
FOR EVERY NODE, DO
    LET TRNS.PCNT = TRNS.PCNT + XMT.PERCENT(NODE)
    LET RCV.PCNT = RCV.PCNT + RCV.PERCENT(NODE)
    IF GRPS < GROUP(NODE)
        LET GRPS = GROUP(NODE)
    REGARDLESS
''
''  SET PROGRAM GRP NUM
''
    LET GROUP(NODE) = GROUP(NODE) + N.NODE
LOOP
''
RESERVE FAM.OF.GRP(*) AS (GRPS + N.NODE + 25)
''
FOR EVERY NODE, DO
    IF FMLYS < FAMILY(NODE)
        LET FMLYS = FAMILY(NODE)
    REGARDLESS
''
''  SET PROGRAM FAM NUM
''
    LET FAMILY(NODE) = N.NODE + GRPS + FAMILY(NODE)
    LET FAM.OF.GRP (GROUP(NODE)) = FAMILY(NODE)
LOOP
''
LET NGFS = N.NODE + GRPS + FMLYS
RESERVE BEST.PATH(*,*) AS NGFS BY NGFS
```

```
RESERVE DUP.BEST.PATH(*,*) AS NGFS BY NGFS
**
**  THE FOLLOWING ARRAYS ARE USED TO HOLD INFORMATION CONCERNING
**  THE STATISTICS ON THE TRAFFIC BETWEEN NODE PAIRS.
**
RESERVE BORN.PKTS(*,*) AS N.NODE BY N.NODE
RESERVE GONE.PKTS(*,*) AS N.NODE BY N.NODE
RESERVE JUMP.TOTAL(*,*) AS N.NODE BY N.NODE
RESERVE LENGTH.TRIP(*,*) AS N.NODE BY N.NODE
**
FOR EVERY NODE, DO
    PRINT 1 LINE WITH NODE, XMT.PERCENT(NODE), RCV.PERCENT(NODE),
         (GROUP(NODE) - N.NODE), GROUP(NODE),
         (FAMILY(NODE) - N.NODE - GRPS) AND FAMILY(NODE),
         OPT.RCV(NODE)  AS FOLLOWS
         **        **.***      **.***      **(**)    **(**)        ***
    SKIP 1 LINE
LOOP
SKIP 1 LINE
**
**  COMPUTING THE MAXIMUM NUMBER OF NODES PER GRP AND NODES PER FAM.
**
LET MAX.GRP.NODES  =    0.0
LET MAX.FAM.NODES  =    0.0
**
FOR I = N.NODE + 1 TO N.NODE + GRPS, DO
    FOR EACH NODE, WITH GROUP(NODE) = I, DO
        COMPUTE GRPCOUNT AS THE NUMBER OF GROUP(NODE)
    LOOP
    IF MAX.GRP.NODES < GRPCOUNT,
        LET MAX.GRP.NODES  =  GRPCOUNT
    REGARDLESS
LOOP
**
FOR J = N.NODE + GRPS + 1 TO NGFS, DO
    FOR EACH NODE, WITH FAMILY(NODE) = J, DO
        COMPUTE FAMCOUNT AS THE NUMBER OF FAMILY(NODE)
    LOOP
    IF MAX.FAM.NODES < FAMCOUNT,
        LET MAX.FAM.NODES  =  FAMCOUNT
    REGARDLESS
LOOP
**
**
PERFORM P2.CONNECT.LINKS
RELEASE P2.CONNECT.LINKS
**
PERFORM E6.LINK.WEIGHT.MATRIX
**
RETURN
```

```
END ''OF P1.BUILD.NETWORK
'' ***********************************************************************
''
ROUTINE FOR P2.CONNECT.LINKS
''
''  THIS ROUTINE HANDLES THE ASSIGNMENT OF LINKS FOR EACH NODE
''  (I.E. THE ESTABLISHMENT OF THE BASIC NETWORK TOPOLOGY).
''  THIS ASSIGNMENT IS ACCOMPLISHED THROUGH THE USE OF SETS.
''  EACH NODE OWNS A LINK SET WHICH CONTAINS THE IDENTITY OF
''  ALL NEIGHBOR NODES.
''
''       VARIABLE DESCRIPTION:
''            LINKS          TOTAL NUMBER OF LINKS FOR THE NETWORK
''                           (ALL LINKS ARE FULL-DUPLEX)
''
DEFINE FIRST.NODE, SECOND.NODE, QFST, QSEC AS VARIABLES
''
READ LINKS
''
LET LNK.NOD.RATIO = REAL.F(2*LINKS)/REAL.F(N.NODE)
''
FOR I = 1 TO LINKS DO
    READ FIRST.NODE, SECOND.NODE, QFST, QSEC
    CREATE A LINK
         LET STATUS(LINK) = IDLE
         LET TEAM(LINK) = SECOND.NODE
         LET QTEST(LINK) = QFST
         LET UPCOUNT(LINK) = NONE
         FILE LINK IN LINK.SET(FIRST.NODE)
    CREATE A LINK
         LET STATUS(LINK) = IDLE
         LET TEAM(LINK) = FIRST.NODE
         LET QTEST(LINK) = QSEC
         LET UPCOUNT(LINK) = NONE
         FILE LINK IN LINK.SET(SECOND.NODE)
LOOP
''
LET MAX.LINKS.PER.NODE  =    0.0
''
FOR EACH NODE, DO
    IF MAX.LINKS.PER.NODE < REAL.F(N.LINK.SET(NODE))
         LET MAX.LINKS.PER.NODE = REAL.F(N.LINK.SET(NODE))
    REGARDLESS
LOOP
RETURN
END ''OF P2.CONNECT.LINKS
'' ***********************************************************************
''
ROUTINE FOR P3.INITIAL.ROUTING.TABLE
''
```

```
    REGARDLESS
    IF FAMILY(NODE) IS NOT EQUAL TO SAME.FAMILY,
''
''      HAVE ONLY ONE NODE PER FAMILY GENERATE A FAMILY UPDATE.
''
        LET NOD.WAIT  =  UNIFORM.F(0.0,BRACKET,6)
        SCHEDULE A U1.GENERATE.UPDATE GIVEN NODE, FAM.MSG IN
             NOD.WAIT UNITS
''
        LET SAME.FAMILY = FAMILY(NODE)
''
    REGARDLESS
    'TRY.AGAIN'
''
LOOP
''
RETURN
END '"OF P3.INITIAL.ROUTING.TABLE
'' ********************************************************************************
''
ROUTINE FOR P4.STATIC.EVENTS
''
''  THIS ROUTINE IS USED FOR RUNNING THE SIMULATION USING STATIC
''  ROUTING AS DETERMINED BY THE INITIAL BEST PATH MATRIX DETERMINED
''  PRIOR TO THE START OF THE RUN.
''
''  PACKET MESSAGES ARE GENERATED USING AN EXPONENTIAL INTERARRIVAL
''  RATE.
''
SCHEDULE A M1.GENERATE.MESSAGE GIVEN REGULAR IN
    EXPONENTIAL.F(MSG.GENERATION.INTERVAL,8) UNITS
''
''  DATA IS COLLECTED AT REGULAR TIME INTERVALS.
''
SCHEDULE A E2.COLLECT.DATA IN (TIME.LIMIT/REAL.F(SAMPLES)) UNITS
''
''  REPORT DATA SUMMARIES MAY BE REQUESTED AT THE USER'S DESCRETION.
''
SCHEDULE A E4.NETWORK.PERFORMANCE.REPORT IN
    (TIME.LIMIT/REAL.F(NUM.REPORTS.REQUESTED)) UNITS
''
''  THE TRANSIENT EFFECT OF THE NETWORK IS ELIMINATED AFTER 10 SECONDS.
''
SCHEDULE A E1.TRANSIENT.BLANKING IN  NUL.TRANSIENT UNITS
''
RETURN
END '"OF P4.STATIC.EVENTS
'' ********************************************************************************
''
ROUTINE FOR P5.DYNAMIC.EVENTS
```

```
**  THIS ROUTINE SCHEDULES EXACTLY ONE UPDATE MESSAGE FOR EACH NODE,
**  GROUP, AND FAMILY WITHIN THE NETWORK.  IN THIS WAY, THE INITIAL
**  BEST PATH MATRIX CAN BE OBTAINED BY RUNNING THE SIMULATION
**  (WITHOUT ADDITIONAL TRAFFIC BEING GENERATED).  DURING SUBSEQUENT
**  RUNS, THIS BEST PATH MATRIX IS USED AS THE INITIAL ROUTING
**  CONDITION FOR THE NETWORK.
**
DEFINE SAME.GROUP, SAME.FAMILY  AS VARIABLES
DEFINE NOD.WAIT, GRP.WAIT, FAM.WAIT AS REAL VARIABLE
**
LET SAME.GROUP = NONE
LET SAME.FAMILY = NONE
LET GRP.WAIT  =  UNIFORM.F(0.0,BRACKET,2)
LET FAM.WAIT  =  UNIFORM.F(0.0,BRACKET,3)
**
SKIP 2 LINES
FOR EACH NODE, DO
**
    LET NOD.WAIT  =  UNIFORM.F(0.0,BRACKET,6)
    SCHEDULE A U1.GENERATE.UPDATE GIVEN NODE, NOD.MSG IN
         NOD.WAIT UNITS
**
    IF GRPS = 1,
         GO TRY.FAMILIES
    REGARDLESS
    IF SYNCH.FIRING = ALIVE,
         SCHEDULE A U7.UNIT.FIRING GIVEN NODE, GRP.MSG, GROUP(NODE),
              GRP.WAIT IN GRP.WAIT UNITS
         GO TRY.FAMILIES
    REGARDLESS
    IF GROUP(NODE) IS NOT EQUAL TO SAME.GROUP,
**
**       HAVE ONLY ONE NODE PER GROUP GENERATE A GROUP UPDATE MESSAGE.
**
         LET NOD.WAIT  =  UNIFORM.F(0.0,BRACKET,6)
         SCHEDULE A U1.GENERATE.UPDATE GIVEN NODE, GRP.MSG IN
              NOD.WAIT UNITS
**
         LET SAME.GROUP = GROUP(NODE)
**
    REGARDLESS
    'TRY.FAMILIES'
**
    IF FMLYS = 1,
         GO TRY.AGAIN
    REGARDLESS
    IF SYNCH.FIRING = ALIVE,
         SCHEDULE A U7.UNIT.FIRING GIVEN NODE, FAM.MSG, FAMILY(NODE),
              FAM.WAIT IN FAM.WAIT UNITS
         GO TRY.AGAIN
```

```
**
**  THIS ROUTINE SETS THE "STAGE" FOR THE START OF THE SIMULATION
**  BY SCHEDULING ALL INITIAL EVENTS.
**
DEFINE NN.TIME, GG.TIME, FF.TIME, GRP.WAIT, FAM.WAIT AS REAL VARIABLES
DEFINE SAME.GROUP, SAME.FAMILY  AS VARIABLES
**
LET SAME.GROUP = NONE
LET SAME.FAMILY = NONE
LET GRP.WAIT   =  UNIFORM.F (0.0,GRP.TIME,2)
LET FAM.WAIT   =  UNIFORM.F (0.0,FAM.TIME,3)
**
FOR EACH NODE, DO
**
    LET NN.TIME = UNIFORM.F (0.0,NOD.TIME,2)
    LET GG.TIME = UNIFORM.F (0.0,NOD.TIME,7)
    LET FF.TIME = UNIFORM.F (0.0,NOD.TIME,9)
**
    IF DISCRETE.TIMER = ALIVE,
        LET NN.TIME = (TRUNC.F (NN.TIME/BIT.INTERVAL) * BIT.INTERVAL)
            + U.XMN.TIME
        LET GG.TIME = (TRUNC.F (GG.TIME/BIT.INTERVAL) * BIT.INTERVAL)
            + U.XMN.TIME
        LET FF.TIME = (TRUNC.F (FF.TIME/BIT.INTERVAL) * BIT.INTERVAL)
            + U.XMN.TIME
    REGARDLESS
**
    SCHEDULE A U1.GENERATE.UPDATE GIVEN NODE, NOD.MSG IN
        NN.TIME UNITS
**
    IF GRPS = 1,
        GO TRY.FAMILIES
    REGARDLESS
    IF SYNCM.FIRING = ALIVE,
        SCHEDULE A U7.UNIT.FIRING GIVEN NODE, GRP.MSG, GROUP(NODE),
            GRP.WAIT IN GRP.WAIT UNITS
        GO TRY.FAMILIES
    REGARDLESS
    IF GROUP(NODE) IS NOT EQUAL TO SAME.GROUP
**
**      HAVE ONLY ONE NODE PER GROUP BE THE GENERATOR OF
**      GROUP WARNING MESSAGES.
**
        SCHEDULE A U1.GENERATE.UPDATE GIVEN NODE, GRP.MSG IN
            GG.TIME UNITS
**
        LET SAME.GROUP = GROUP(NODE)
**
    REGARDLESS
    'TRY.FAMILIES'
```

```
''
    IF FMLYS = 1,
         GO TRY.AGAIN
    REGARDLESS
    IF SYNCH.FIRING = ALIVE,
         SCHEDULE A U7.UNIT.FIRING GIVEN NODE, FAM.MSG, FAMILY(NODE),
              FAM.WAIT IN FAM.WAIT UNITS  .
         GO TRY.AGAIN
    REGARDLESS
    IF FAMILY(NODE) IS NOT EQUAL TO SAME.FAMILY,
''
''       HAVE ONLY ONE NODE PER FAMILY BE THE GENERATOR OF
''       FAMILY WARNING MESSAGES.
''
         SCHEDULE A U1.GENERATE.UPDATE GIVEN NODE, FAM.MSG IN
              FF.TIME UNITS
''
         LET SAME.FAMILY = FAMILY(NODE)
''
    REGARDLESS
    'TRY.AGAIN'
''
LOOP
''
''  PACKET MESSAGES ARE GENERATED USING AN EXPONENTIAL INTERARRIVAL
''  RATE.
''
SCHEDULE A M1.GENERATE.MESSAGE GIVEN REGULAR IN
    EXPONENTIAL.F(MSG.GENERATION.INTERVAL,8) UNITS
''
''  DATA IS COLLECTED AT REGULAR TIME INTERVALS.
''
SCHEDULE A E2.COLLECT.DATA IN (TIME.LIMIT/REAL.F(SAMPLES)) UNITS
''
''  REPORT DATA SUMMARIES MAY BE REQUESTED AT THE USER'S DESCRETION.
''
SCHEDULE A E4.NETWORK.PERFORMANCE.REPORT IN
    (TIME.LIMIT/REAL.F(NUM.REPORTS.REQUESTED)) UNITS
''
''  LINK WEIGHTS ARE CALCULATED AT REGULAR TIME INTERVALS.
''
SCHEDULE A U5.LINK.WEIGHT.CALCULATION IN WT.TIME UNITS
''
''  THE TRANSIENT EFFECT OF THE NETWORK IS ELIMINATED AFTER 10 SECONDS.
''
SCHEDULE A E1.TRANSIENT.BLANKING IN  NUL.TRANSIENT UNITS
''
RETURN
END ""OF PS.DYNAMIC.EVENTS
''  ************************************************************
```

```
''
ROUTINE FOR P6.PURGE.EVENT.QUEUE
''
''   THE FOLLOWING SUBROUTINE CANCELS AND/OR DESTROYS ALL ENTITIES
''       AND EVENTS WHICH REMAIN IN THE TIMING ROUTINE AFTER
''       TIME.V = TIME.LIMIT. SINCE THIS ACTION EMPTIES THE
''       TIMING ROUTINE, CONTROL OF THE PROGRAM IS RETURNED TO THE
''       STATEMENT FOLLOWING 'START SIMULATION' IN THE "'MAIN"
''       PROGRAM.
''
DEFINE ITEM1, ITEM2, ITEM3  AS VARIABLES
''
''  PRINT 1 LINE WITH TIME.V AS FOLLOWS
'' -******** DESTRUCTION RUN AT ****.******** SECONDS  ************
''
FOR EACH U6.NODE.WAKE.UP IN EV.S(I.U6.NODE.WAKE.UP), DO
  CANCEL THE U6.NODE.WAKE.UP
  DESTROY THE U6.NODE.WAKE.UP
LOOP
FOR EACH U8.ERASE.RECORD IN EV.S(I.U8.ERASE.RECORD), DO
  CANCEL THE U8.ERASE.RECORD
  DESTROY THE U8.ERASE.RECORD
LOOP
FOR EACH U1.GENERATE.UPDATE IN EV.S(I.U1.GENERATE.UPDATE), DO
  CANCEL THE U1.GENERATE.UPDATE
  DESTROY THE U1.GENERATE.UPDATE
LOOP
FOR EACH U7.UNIT.FIRING IN EV.S(I.U7.UNIT.FIRING), DO
  CANCEL THE U7.UNIT.FIRING
  DESTROY THE U7.UNIT.FIRING
LOOP
FOR EACH U4.RECEIVE.UPDATE IN EV.S(I.U4.RECEIVE.UPDATE), DO
  CANCEL THE U4.RECEIVE.UPDATE
  DESTROY THE U4.RECEIVE.UPDATE
LOOP
FOR EACH M2.RECEIVE.MESSAGE.PACKET IN
  EV.S(I.M2.RECEIVE.MESSAGE.PACKET), DO
  CANCEL THE M2.RECEIVE.MESSAGE.PACKET
  DESTROY THE M2.RECEIVE.MESSAGE.PACKET
LOOP
FOR EACH M1.GENERATE.MESSAGE IN EV.S(I.M1.GENERATE.MESSAGE), DO
  CANCEL THE M1.GENERATE.MESSAGE
  DESTROY THE M1.GENERATE.MESSAGE
LOOP
FOR EACH E4.NETWORK.PERFORMANCE.REPORT IN
  EV.S(I.E4.NETWORK.PERFORMANCE.REPORT), DO
  CANCEL THE E4.NETWORK.PERFORMANCE.REPORT
  DESTROY THE E4.NETWORK.PERFORMANCE.REPORT
LOOP
FOR EACH E2.COLLECT.DATA IN EV.S(I.E2.COLLECT.DATA), DO
```

```
  CANCEL THE E2.COLLECT.DATA
  DESTROY THE E2.COLLECT.DATA
LOOP
FOR EACH U5.LINK.WEIGHT.CALCULATION IN
  EV.S(I.U5.LINK.WEIGHT.CALCULATION), DO
  CANCEL THE U5.LINK.WEIGHT.CALCULATION
  DESTROY THE U5.LINK.WEIGHT.CALCULATION
LOOP
FOR EACH E1.TRANSIENT.BLANKING IN EV.S(I.E1.TRANSIENT.BLANKING), DO
  CANCEL THE E1.TRANSIENT.BLANKING
  DESTROY THE E1.TRANSIENT.BLANKING
LOOP
''
RETURN
END ''OF P6.PURGE.EVENT.QUEUE
''  ********************************************************************************
''
ROUTINE FOR P7.ZEROIZE.SETS
''
''  THIS ROUTINE CLEARS ALL SETS OF THEIR MEMBERS AND ZEROIZES THE
''  TIMING ROUTINE FOR THE SIMULATION WHICH ALLOWS FOR MULTIPLE RUNS.
''
DEFINE ITEM1, ITEM2, ITEM3, I, J  AS VARIABLES
''
LET TIME.V    =    0.0
''
FOR EVERY NODE, DO
''
    FOR EACH ITEM1 IN VIRT.CKT.LIST(NODE), DO
        REMOVE ITEM1 FROM VIRT.CKT.LIST(NODE)
        DESTROY CIRCUIT CALLED ITEM1
    LOOP
    FOR EACH ITEM1 IN XMT.RECORD(NODE), DO
        REMOVE ITEM1 FROM XMT.RECORD(NODE)
        DESTROY RECORD CALLED ITEM1
    LOOP
    FOR EACH ITEM1 IN ALARM.CLOCK(NODE), DO
        REMOVE ITEM1 FROM ALARM.CLOCK(NODE)
        FOR EACH ITEM2 IN BUZ.LIST(ITEM1), DO
            REMOVE ITEM2 FROM BUZ.LIST(ITEM1)
            DESTROY BUZ.NODE CALLED ITEM2
        LOOP
        DESTROY BUZZER CALLED ITEM1
    LOOP
    FOR EACH ITEM1 IN LINK.SET(NODE), DO
        FOR EACH ITEM2 IN QUEUE(ITEM1), DO
            REMOVE ITEM2 FROM QUEUE(ITEM1)
            RESET THE TOTALS OF LIFETIME(ITEM2) AND QSTAT(ITEM2)
            DESTROY PACKET CALLED ITEM2
        LOOP
```

```
            FOR EACH ITEM2 IN UQUEUE(ITEM1), DO
                REMOVE ITEM2 FROM UQUEUE(ITEM1)
                DESTROY UPDATE CALLED ITEM2
            LOOP
            LET STATUS(ITEM1)   = IDLE
            LET UPCOUNT(ITEM1)  = NONE
            LET PTCOUNT(ITEM1)  = NONE
            RESET THE TOTALS OF STATUS(ITEM1), N.QUEUE(ITEM1),
                N.UPROP.QUEUE(ITEM1), N.UQUEUE(ITEM1), N.PROP.QUEUE(ITEM1)
            FOR EACH ITEM2 IN PROP.QUEUE(ITEM1), DO
                REMOVE ITEM2 FROM PROP.QUEUE(ITEM1)
                RESET THE TOTALS OF LIFETIME(ITEM2) AND QSTAT(ITEM2)
                DESTROY PACKET CALLED ITEM2
            LOOP
            FOR EACH ITEM2 IN UPROP.QUEUE(ITEM1), DO
                REMOVE ITEM2 FROM UPROP.QUEUE(ITEM1)
                DESTROY UPDATE CALLED ITEM2
            LOOP
            FOR EACH ITEM2 IN SET.OF.WEIGHTS(ITEM1), DO
                REMOVE ITEM2 FROM SET.OF.WEIGHTS(ITEM1)
                DESTROY WEIGHT CALLED ITEM2
            LOOP
            RESET THE TOTALS OF QUESIZ(ITEM1)
        LOOP
    LOOP
    **
    FOR I = 1 TO N.NODE, DO
        FOR J = 1 TO N.NODE, DO
            LET BORN.PKTS(I,J) = 0
            LET GONE.PKTS(I,J) = 0
            LET JUMP.TOTAL(I,J) = 0
            LET LENGTH.TRIP(I,J) = 0.0
        LOOP
    LOOP
    LET    COMP.TRIP.PKT  =    0
    LET    NEW.MSG.TOTAL  =    0
    LET    NEW.PKT.TOTAL  =    0
    LET    SYS.PACKETS    =    0
    LET    U.COUNT        =    0
    RESET THE TOTALS OF SYS.PACKETS
    LET    HOP.TOTAL      =    0
    LET    XMT.TOTAL      =    0.0
    LET    QU.TOTAL       =    0.0
    **
    **  RESET THE STARTING VALUES FOR THE RANDOM NUMBER GENERATOR.
    LET SEED.V(1) =  2116429302
    LET SEED.V(2) =   683743814
    LET SEED.V(3) =   964393174
    LET SEED.V(4) =  1217426631
    LET SEED.V(5) =   618433579
```

```
LET SEED.V(6) =  1157240309
LET SEED.V(7) =    15726055
LET SEED.V(8) =    48108509
LET SEED.V(9) =  1797920909
LET SEED.V(10) =   477424540
RETURN
END ''OF P7.ZEROIZE.SETS
''  ××××××
```

```
        LET ORIGIN(UPDATE)      = THIS.NODE
        LET FM.NODE(UPDATE)     = THIS.NODE
        LET VARIETY(UPDATE)     = REGULAR
        LET HOPCNT(UPDATE)      = NONE
        LET TO.NODE(UPDATE)     = TERM(LINK)
''
    PERFORM U2.TRANSMIT.UPDATE GIVEN UPDATE, LINK
''
LOOP
''
''  RESCHEDULE ANOTHER UPDATE FOR THIS NODE AT A LATER TIME
''  UNLESS THIS IS THE INITIALIZATION PHASE FOR DETERMINING
''  THE BEST PATHS BETWEEN NODES.
''
IF INIT.BP = DEAD,
''
    LET NEW.INC = UNIFORM.F(NOD.TIME,NOD.TIME + BRACKET,3)
    IF DISCRETE.TIMER = ALIVE,
        LET NEW.INC = (TRUNC.F(NEW.INC/BIT.INTERVAL) * BIT.INTERVAL)
            + U.XMN.TIME
    REGARDLESS
    SCHEDULE A U1.GENERATE.UPDATE GIVEN THIS.NODE, THE.CLASS
        IN NEW.INC UNITS
REGARDLESS
''
GO FILE.THE.UPDATE
''
''  .   .   .   .   .   .   .   .   .   .   .   .   .   .
'GRP.REQUESTED'
''
''  A GROUP WARNING MSG WILL BE SENT TO ALL NODES IN THE SAME GROUP
''  AS THE ORIGINATOR.  THIS WARNING MSG WILL INCLUDE A FUTURE TIME
''  (FIRING.TIME) WHEN ALL NODES WITHIN THE GROUP WILL SEND THE GROUP
''  UPDATE MESSAGE AT THE SAME TIME.
''
''  SCHEDULE THIS NODE TO TRANSMIT ITS OWN REGULAR GROUP UPDATE
''  AT THE COMPUTED FIRING TIME.  THIS MESSAGE IS SENT ONLY TO
''  NEIGHBOR NODES WHICH ARE NOT IN THE SAME GROUP AS THIS NODE.
''
IF PRNT >= 3,
SKIP 1 LINE
PRINT 3 LINES WITH TIME.V, THIS.NODE, GROUP(THIS.NODE),
    TIME.V + GP.FIRE AS FOLLOWS
***.***** (**) : GROUP UPDATE WARNING ORIGINATED.
                 GROUP (**) SET TO XMT UPDATE AT ***.*****.
                 WARNING SENT TO THE FOLLOWING NODES:
REGARDLESS
''
SCHEDULE A U7.UNIT.FIRING GIVEN THIS.NODE, THE.CLASS, GROUP(THIS.NODE),
    TIME.V AT TIME.V + GP.FIRE
```

```
''
FOR EACH LINK IN LINK.SET(THIS.NODE) WITH GROUP(THIS.NODE) =
    GROUP(TERM(LINK)), DO
''
    IF PRNT >= 3,
    PRINT 1 LINE WITH TERM(LINK) AS FOLLOWS
                                                    **
    REGARDLESS
    CREATE AN UPDATE
         LET BEGIN.TIME(UPDATE)  = TIME.V
         LET CLASS(UPDATE)       = THE.CLASS
         LET ORIGIN(UPDATE)      = GROUP(THIS.NODE)
         LET FM.NODE(UPDATE)     = THIS.NODE
         LET VARIETY(UPDATE)     = WARNING
         LET HOPCNT(UPDATE)      = NONE
         LET TO.NODE(UPDATE)     = TERM(LINK)
         LET FIRING.TIME(UPDATE) = TIME.V + GP.FIRE
         IF DISCRETE.TIMER = ALIVE,
              LET FIRING.TIME(UPDATE) =
              (TRUNC.F(FIRING.TIME(UPDATE)/BIT.INTERVAL) * BIT.INTERVAL)
                   + U.XMN.TIME
         REGARDLESS
''
    PERFORM U2.TRANSMIT.UPDATE GIVEN UPDATE, LINK
''
LOOP
''
''  RESCHEDULE ANOTHER GROUP UPDATE FOR THIS NODE AT A FUTURE TIME
''
IF INIT.BP = DEAD,
''
    LET NEW.INC = UNIFORM.F(GRP.TIME,GRP.TIME + BRACKET,5)
    IF DISCRETE.TIMER = ALIVE,
         LET NEW.INC = (TRUNC.F(NEW.INC/BIT.INTERVAL) * BIT.INTERVAL)
              + U.XMN.TIME
    REGARDLESS
    SCHEDULE A U1.GENERATE,UPDATE GIVEN THIS.NODE, THE.CLASS
         IN NEW.INC UNITS
REGARDLESS
''
GO FILE.THE.UPDATE
''
''  .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
'FAM.REQUESTED'
''
''  A SIMILAR OPERATION IS PERFORMED FOR THE FAMILY AS WAS DONE WITH
''  THE PREVIOUS SECTION CONCERNING THE GROUP WARNING.
''
''
''  SCHEDULE THIS NODE TO TRANSMIT ITS OWN REGULAR FAMILY UPDATE
```

```
**  AT THE COMPUTED FIRING TIME.  THIS MESSAGE IS SENT ONLY TO
**  NEIGHBOR NODES WHICH ARE NOT IN THE SAME FAMILY AS THIS NODE.
**
IF PRNT >= 3,
SKIP 1 LINE
PRINT 3 LINES WITH TIME.V, THIS.NODE, FAMILY(THIS.NODE),
    TIME.V + FM.FIRE AS FOLLOWS
####.##### (##) : FAMILY UPDATE WARNING ORIGINATED.
                  FAMILY(##) SET TO XMT UPDATE AT ####.#####.
                  WARNING SENT TO THE FOLLOWING NODES:
REGARDLESS
**
SCHEDULE A U7.UNIT.FIRING GIVEN THIS.NODE, THE.CLASS, FAMILY(THIS.NODE),
    TIME.V AT TIME.V + FM.FIRE
**
**
FOR EACH LINK IN LINK.SET(THIS.NODE) WITH FAMILY(THIS.NODE) =
    FAMILY(TERM(LINK)), DO
**
    IF PRNT >= 3,
    PRINT 1 LINE WITH TERM(LINK) AS FOLLOWS
                                                        ##
    REGARDLESS
    CREATE AN UPDATE
         LET BEGIN.TIME(UPDATE)  = TIME.V
         LET CLASS(UPDATE)       = THE.CLASS
         LET ORIGIN(UPDATE)      = FAMILY(THIS.NODE)
         LET FM.NODE(UPDATE)     = THIS.NODE
         LET VARIETY(UPDATE)     = WARNING
         LET HOPCNT(UPDATE)      = NONE
         LET TO.NODE(UPDATE)     = TERM(LINK)
         LET FIRING.TIME(UPDATE) = TIME.V + FM.FIRE
         IF DISCRETE.TIMER = ALIVE,
              LET FIRING.TIME(UPDATE) =
              (TRUNC.F(FIRING.TIME(UPDATE)/BIT.INTERVAL) * BIT.INTERVAL)
                   + U.XMN.TIME
         REGARDLESS
**
    PERFORM U2.TRANSMIT.UPDATE GIVEN UPDATE, LINK
LOOP
**
**  RESCHEDULE A FAMILY UPDATE FOR THIS NODE AT A FUTURE TIME.
**
IF INIT.BP = DEAD,
**
    LET NEW.INC = UNIFORM.F(FAM.TIME,FAM.TIME + BRACKET,8)
    IF DISCRETE.TIMER = ALIVE,
         LET NEW.INC = (TRUNC.F(NEW.INC/BIT.INTERVAL) * BIT.INTERVAL)
              + U.XMN.TIME
    REGARDLESS
```

```
    SCHEDULE A U1.GENERATE.UPDATE GIVEN THIS.NODE, THE.CLASS
         IN NEW.INC UNITS
REGARDLESS
''
''  .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
'FILE.THE.UPDATE'
''
LET U.COUNT = U.COUNT + 1
''
IF PRNT >= 3,
PRINT 1 LINE WITH U.COUNT AS FOLLOWS
              ******  UPDATES GENERATED SO FAR.
REGARDLESS
''
RETURN
''
END ''OF U1.GENERATE.UPDATE
''  **********************************************************************
''
ROUTINE FOR U2.TRANSMIT.UPDATE GIVEN X.UPDATE, X.LINK
''
''  THIS ROUTINE PERFORMS THE ROLE OF TRANSMISSION OF THE UPDATE
''  OVER THE SELECTED LINK.  DEPENDING UPON THE STATUS OF THE LINK
''  (IDLE OR BUSY) AND THE TYPE OF TRAFFIC WHICH THE LINK IS
''  CURRENTLY TRANSMITTING, DIFFERENT ACTION IS TAKEN.
''
IF STATUS(X.LINK) = IDLE,
''
''  THE UPDATE MAY BE SENT.
    SCHEDULE AN U4.RECEIVE.UPDATE GIVEN X.UPDATE IN U.XMN.TIME UNITS
    LET STATUS(X.LINK) = BUSY
    FILE X.UPDATE IN UPROP.QUEUE(X.LINK)
ELSE
''
''  THE LINK IS BUSY UNDER ONE OF THREE CONDITIONS:
''       (1)  REGULAR MESSAGE PACKET TRAFFIC IS BEING TRANSMITTED WITH
''            NO 'INSERTED' UPDATES CURRENTLY IN TRANSIT.
''       (2)  UPDATE TRAFFIC IS IN TRANSIT WHICH HAS NOT BEEN 'INSERTED'
''            INTO A REGULAR MESSAGE PACKET DATA STREAM.
''       (3)  'INSERTED' UPDATE TRAFFIC IS CURRENTLY IN TRANSIT WITHIN
''            A RUGULAR MESSAGE PACKET'S DATA STREAM.
''
''  CONDITION (1)
''
    IF UPROP.QUEUE(X.LINK) IS EMPTY,
         SCHEDULE AN U4.RECEIVE.UPDATE GIVEN X.UPDATE IN U.XMN.TIME UNITS
         FILE X.UPDATE IN UPROP.QUEUE(X.LINK)
         PERFORM U3.INSERT.UPDATE GIVEN X.LINK
         RETURN
    REGARDLESS
```

```
''
''  CONDITION (2)
''
    IF PROP.QUEUE(X.LINK) IS EMPTY,
         FILE X.UPDATE IN UQUEUE(X.LINK)
         RETURN
    REGARDLESS
''
''  CONDITION (3)
''
    FILE X.UPDATE IN UQUEUE(X.LINK)
    PERFORM U3.INSERT.UPDATE GIVEN X.LINK
REGARDLESS
RETURN
END ''OF U2.TRANSMIT.UPDATE
'' ********************************************************************************
''
ROUTINE FOR U3.INSERT.UPDATE GIVEN ID.LINK
''
''  THIS ROUTINE MODIFIES THE ARRIVAL TIME OF A MESSAGE PACKET
''  CAUSED WHEN AN UPDATE IS INSERTED INTO IT'S BIT STREAM.
''
DEFINE PKT, THE.NAME AS VARIABLES
''
REMOVE FIRST PKT FROM PROP.QUEUE(ID.LINK)
''
''  THE EVENT OF THE ARRIVAL OF THE PACKET IS NOW DELAYED BY THE
''  AMOUNT OF TIME IT TAKES FOR AN UPDATE TO BE TRANSMITTED.
''
CANCEL M2.RECEIVE.MESSAGE.PACKET CALLED P.NAME(PKT)
DESTROY M2.RECEIVE.MESSAGE.PACKET CALLED P.NAME(PKT)
''
RESCHEDULE A M2.RECEIVE.MESSAGE.PACKET CALLED THE.NAME GIVEN PKT AT
    P.ARRIVAL(PKT) + U.XMN.TIME
''
LET P.ARRIVAL(PKT) = P.ARRIVAL(PKT) + U.XMN.TIME
LET P.NAME(PKT) = THE.NAME
''
FILE PKT IN PROP.QUEUE(ID.LINK)
RETURN
END ''OF U3.INSERT.UPDATE
'' ********************************************************************************
''
EVENT U4.RECEIVE.UPDATE GIVEN UP.MESSAGE
''
''  THIS EVENT PERFORMS THE PRIMARY FUNCTION OF THE YEN ALGORITHM.
''  WHEN THE UPDATE ARRIVES THE CALCULATIONS OF THE ALGORITHM TAKE
''  PLACE WITH THE APPROPRIATE SCHEDULING OF THE FUTURE UPDATE
''  TRANSMISSION OCCURRING.
''
```

```
DEFINE MSG, THIS.NODE, PAST.NODE, KIND,  OCCURANCE, THE.WAKE,
    NO.PCOUNT, THE.LINK, THE.BUZZ.U.MSG, Q.UP.MSG, PAC.KET, THE.NAME
    AS INTEGER VARIABLES
DEFINE DELAY.TIME, THIS.LINK.WEIGHT  AS REAL VARIABLES
''
LET MSG = UP.MESSAGE
LET KIND      = VARIETY(MSG)
LET THIS.NODE = TO.NODE(MSG)
LET PAST.NODE = FM.NODE(MSG)
LET NO.PCOUNT = HOPCNT(MSG) + 1
''
''  THE UPDATE HAS BEEN TRAVELING DOWN THE LINK AND IS KEPT TRACK OF BY
''  MEANS OF A UPROP.QUEUE.  THE UPDATE HAS ARRIVED AND CAN NOW BE
''  REMOVED FROM THAT QUEUE.
''  EACH LINK CONTAINS A COUNT OF THE NUMBER OF UPDATE MESSAGES WHICH
''  HAVE TRAVELED OVER IT.  IN THIS WAY, WE CAN SEE THE PROPORTION OF
''  TOTAL TRAFFIC OVER EACH LINK WHICH IS RELATED TO ROUTING INFO.
''
FOR EACH LINK IN LINK.SET(PAST.NODE) WITH TERM(LINK) = THIS.NODE,
    FIND THE.LINK = LINK,
IF FOUND,
    REMOVE THE FIRST U.MSG FROM UPROP.QUEUE(THE.LINK)
    LET UPCOUNT(THE.LINK) = UPCOUNT(THE.LINK) + 1
REGARDLESS
''
''  EVERY NODE MAINTAINS A RECORD OF UPDATE RETRANSMISSIONS WHICH
''  IT HAS PRODUCED.  IF THE NODE HAS PREVIOUSLY SENT A RETRANS OF
''  THIS CURRENT UPDATE THEN THE UPDATE IS IGNORED AND NO ACTION TAKEN.
''
FOR EACH RECORD IN XMT.RECORD(THIS.NODE) WITH ORIGIN(MSG) =
    REC.ORIGIN(RECORD) AND BEGIN.TIME(MSG) = REC.TIME(RECORD),
    FIND THE FIRST CASE,
IF FOUND,
''
    IF PRNT >= 3,
    SKIP 1 LINE
    PRINT 1 LINE WITH TIME.V, THIS.NODE, ORIGIN(MSG),BEGIN.TIME(MSG),
  PAST.NODE AS FOLLOWS
***.*****  (**) : 'LATE' UPDATE(**,***,*****) FROM NODE **.
    REGARDLESS
''
    DESTROY UPDATE CALLED U.MSG
    GO CHECK.THE.QUEUES
REGARDLESS
''
''  DETERMINE WHETHER REGULAR OR WARNING UPDATE AND TAKE THE
''  THE APPROPRIATE ACTION.
''
GO TO REG.MSG, WARN.MSG PER KIND
''
```

```
**  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
'REG.MSG'
**
**  DETERMINE WHETHER NODE, GROUP OR FAMILY AND TAKE APPROPRIATE ACTION.
**
GO TO NOD.REQUESTED, GRP.REQUESTED, FAM.REQUESTED PER CLASS(MSG)
**
**  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
'NOD.REQUESTED'
**
**  WE MUST DETERMINE IF THIS NODE UPDATE MSG ORIGINATED FROM A NODE
**  IN THIS NODE'S GROUP.  IF NOT, A BORDER CHECK IS RUN.  IF THE
**  BORDER CHECK PROVES NEGATIVE, NO ACTION IS TAKEN.
**
IF FAMILY(THIS.NODE) = FAMILY(ORIGIN(MSG)) AND
    GROUP(THIS.NODE) = GROUP(ORIGIN(MSG)),
**
    GO CALC.THE.DELAY
REGARDLESS
**
**  RUN A BORDER CHECK
**
IF HO.PCOUNT <= HOP.LIMIT,
    IF PANT >= 3,
    PRINT 1 LINE AS FOLLOWS
    *****  NODE BORDER INCIDENT  *****
    REGARDLESS
**
    GO CALC.THE.DELAY
REGARDLESS
IF PANT >= 3,
SKIP 1 LINE
PRINT 1 LINE WITH TIME.V, THIS.NODE, ORIGIN(MSG), BEGIN.TIME(MSG)
    AS FOLLOWS
###.#####  (##)  :  NODE UPDATE(##,###.#####)  ARRIVED.  **NOT IN RANGE**
REGARDLESS
DESTROY UPDATE CALLED U.MSG
GO CHECK.THE.QUEUES
**
**  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
'GRP.REQUESTED'
**
**  IF THE GROUP UPDATE IS FROM A NODE WITHIN THIS NODE'S GROUP
**  THEN NO ACTION IS TAKEN.
**
IF GROUP(THIS.NODE) = ORIGIN(MSG),
    DESTROY UPDATE CALLED U.MSG
    GO CHECK.THE.QUEUES
REGARDLESS
**
```

```
''  IF THE GROUP UPDATE IS FROM A NODE FROM A DIFFERENT FAMILY
''  THEN A BORDER CHECK IS PERFORMED.  IF BORDER CHECK IS NEGATIVE
''  THEN NO ACTION IS TAKEN.
''
IF FAMILY(THIS.NODE) = FAM.OF.GRP(ORIGIN(MSG)),
    GO CALC.THE.DELAY
REGARDLESS
''
''  PERFORM BORDER CHECK
''
IF NO.PCOUNT <= HOP.LIMIT,
    IF PRNT >= 3,
    PRINT 1 LINE AS FOLLOWS
    *****  GROUP BORDER INCIDENT  *****
    REGARDLESS
''
    GO CALC.THE.DELAY
REGARDLESS
IF PRNT >= 3,
SKIP 1 LINE
PRINT 1 LINE WITH TIME.V, THIS.NODE, ORIGIN(MSG), BEGIN.TIME(MSG)
    AS FOLLOWS
###.#### (##) : GROUP UPDATE(##,###.####) RECEIVED. **NOT IN RANGE**
REGARDLESS
DESTROY UPDATE CALLED U.MSG
GO CHECK.THE.QUEUES
''
''  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
'FAM.REQUESTED'
''
IF FAMILY(THIS.NODE) = ORIGIN(MSG),
    IF PRNT >= 3,
    SKIP 1 LINE
    PRINT 1 LINE WITH TIME.V, THIS.NODE, ORIGIN(MSG), BEGIN.TIME(MSG)
    AS FOLLOWS
###.#### (##) : FAMILY UPDATE(##,###.####) RECEIVED. **NOT IN RANGE**
    REGARDLESS
    DESTROY UPDATE CALLED U.MSG
    GO CHECK.THE.QUEUES
REGARDLESS
''
''  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
'CALC.THE.DELAY'
''
''  THE "WEIGHT" OF A LINK IS DETERMINED BY THE EVENT
''  "CALC.LINK.WEIGHT" IN WHICH FIVE METHODS ARE POSSIBLE.
''
LET THIS.LINK.WEIGHT = 0.0
''
FOR EACH LINK IN LINK.SET(THIS.NODE) WITH
```

```
    TERM(LINK)=PAST.NODE, FIND THE.LINK= LINK,
IF FOUND,
''
    IF TEST = ALIVE,
        LET THIS.LINK.WEIGHT = REAL.F(QTEST(THE.LINK))
        GO CONT.TEST
    REGARDLESS
''
    IF MANNER.OF.WEIGHTING = BY.QSIZE.NOW,
        LET THIS.LINK.WEIGHT = REAL.F(N.QUEUE(THE.LINK))
        GO CONT.TEST
    REGARDLESS
''
    IF MANNER.OF.WEIGHTING = BY.COMBINE,
        LET THIS.LINK.WEIGHT = (QU.FACT * REAL.F(N.QUEUE(THE.LINK)))
            + LK.WEIGHT(THE.LINK)
        GO CONT.TEST
    REGARDLESS
''
'' LINK WEIGHTING SCHEMES USING VARIOUS METHODS ARE DETERMINED
'' USING THE EVENT 'CALC.THE.DELAY'.
''
    LET THIS.LINK.WEIGHT = LK.WEIGHT(THE.LINK)
''
    'CONT.TEST'
''
REGARDLESS
''
'' DETERMINE THE DELAY TIME ASSOCIATED WITH THAT LINK.
''
LET DELAY.TIME = TIME.V + U.XMN.TIME + (DELTA * THIS.LINK.WEIGHT)
''
'' CHECK IF THIS DELAY IS LESS THAN PREVIOUS DELAY.
'' THE PREVIOUS DELAYS ARE MAINTAINED IN THE NODE'S ALARM CLOCK SET.
''
FOR EACH BUZZER IN ALARM.CLOCK(THIS.NODE) WITH CAUSE(BUZZER) =
    ORIGIN(MSG) AND BUZ.BEG(BUZZER) = BEGIN.TIME(MSG),
    FIND THE.BUZZ = BUZZER,
IF FOUND,
''
'' FILE THE ID OF THE NODE FROM WHICH THIS UPDATE WAS RECEIVED
'' INTO THE LIST HELD BY THE NODE.
    CREATE A BUZ.NODE
    LET ID.NUMBER(BUZ.NODE) = PAST.NODE
    FILE BUZ.NODE IN BUZ.LIST(THE.BUZZ)
''
'' CHECK IF THIS NEW DELAY IS FASTER THEN THE PREVIOUS BEST DELAY.
''
    IF DELAY.TIME < SETTING(THE.BUZZ),
''
```

```
          CANCEL U6.NODE.WAKE.UP CALLED BUZ.WAKE(THE.BUZZ)
          DESTROY U6.NODE.WAKE.UP CALLED BUZ.WAKE(THE.BUZZ)
**
**        PUT IN NEW FASTEST ALARM TIME
**
          LET SETTING(THE.BUZZ) = DELAY.TIME
          LET TEMP.BP(THE.BUZZ) = PAST.NODE
**
          IF PRNT >= 3,
          SKIP 1 LINE
          PRINT 2 LINES WITH TIME.V, THIS.NODE, ORIGIN(MSG),
               BEGIN.TIME(MSG), PAST.NODE, SETTING(THE.BUZZ)
               AS FOLLOWS
####.#### (##) : ALARM CLOCK RESET - UPDATE(##,###.####) FROM NODE ##.
                  NOW SET TO GO OFF AT ###.####.
          REGARDLESS
**
**        CHECK TO SEE IF WE HAVE ALREADY RECEIVED
**        UPDATES FROM ALL LINKS. IF THIS BE THE CASE THEN
**        A BEST PATH CAN BE DETERMINED WITHOUT HAVING TO
**        SEND OUT A RETRANSMISSION OF THE UPDATE.
**
          IF N.LINK.SET(THIS.NODE) = N.BUZ.LIST(THE.BUZZ),
               IF BEST.PATH(THIS.NODE, ORIGIN(MSG)) NE PAST.NODE,
                    IF PRNT =1,
                         SKIP 1 LINE
                         PRINT 2 LINES WITH TIME.V, THIS.NODE,
                              THIS.NODE, ORIGIN(MSG), PAST.NODE,
                              BEST.PATH(THIS.NODE,ORIGIN(MSG))
                              AS FOLLOWS
####.#### (##) : NEW BEST PATH  -  NO RETRANSMISSION NEEDED.
                  FROM  ##  TO  ##  IS NOW  ##.   (WAS  ##)
                    REGARDLESS
               REGARDLESS
               LET BEST.PATH(THIS.NODE,ORIGIN(MSG)) = PAST.NODE
**
               IF PRNT >= 2,
               PRINT 2 LINES WITH THIS.NODE, ORIGIN(MSG),PAST.NODE
                    AS FOLLOWS
               ** RETRANSMISSION OF UPDATE NOT NEEDED **
               BEST PATH FROM  ##  TO  ##  IS  ##.
               REGARDLESS
**
**             DESTROY ALL BUZZER INFORMATION FROM THE
**             ALARM CLOCK OF THE NODE.
**
               FOR EACH OCCURANCE OF BUZ.LIST(THE.BUZZ), DO
                    REMOVE OCCURANCE FROM BUZ.LIST(THE.BUZZ)
                    DESTROY BUZ.NODE CALLED OCCURANCE
               LOOP
```

```
''
            REMOVE THE.BUZZ FROM ALARM.CLOCK(THIS.NODE)
            DESTROY BUZZER CALLED THE.BUZZ
''
            DESTROY UPDATE CALLED U.MSG
            GO CHECK.THE.QUEUES
        REGARDLESS
''
''      RESCHEDULE THE NEW WAKE UP AT THE NEW SHORTER DELAY TIME.
''
        RESCHEDULE A U6.NODE.WAKE.UP CALLED THE.WAKE GIVEN
            THIS.NODE, MSG, TEMP.BP(THE.BUZZ), NO.PCOUNT
            AT SETTING(THE.BUZZ)
''
        LET BUZ.WAKE(THE.BUZZ) = THE.WAKE
''
        GO CHECK.THE.QUEUES
    REGARDLESS
    DESTROY UPDATE CALLED U.MSG
    GO CHECK.THE.QUEUES
REGARDLESS
''
''  THIS CASE CORRESPONDS TO THE FIRST TIME THIS NODE HAS RECEIVED
''  AN UPDATE ORIGINATION FROM THAT NODE.
''
CREATE A BUZZER
LET SETTING(BUZZER)  = DELAY.TIME
LET CAUSE(BUZZER)    = ORIGIN(MSG)
LET BUZ.BEG(BUZZER)  = BEGIN.TIME(MSG)
LET TEMP.BP(BUZZER)  = PAST.NODE
''
IF PRNT >= 3,
SKIP 1 LINE
PRINT 2 LINES WITH TIME.V, THIS.NODE, ORIGIN(MSG), BEGIN.TIME(MSG),
    PAST.NODE, SETTING(BUZZER) AS FOLLOWS
xxx.xxxx (xx) : **NEW ALARM CLOCK**-- UPDATE(xx,xxx.xxxx) FROM NODE xx.
                SETTING TO GO OFF AT xxx.xxxx.
REGARDLESS
''
CREATE A BUZ.NODE
LET ID.NUMBER(BUZ.NODE) = PAST.NODE
''
''  SET THE ALARM FOR THIS UPDATE TO GO OFF AFTER THE DELAY TIME IS
''  UP.  THIS IS CONDITIONED ON THE BASIS THAT A SHORTER DELAY BY
''  ANOTHER PATH IS NOT FOUND.
''
''  EACH WAKE UP EVENT IS GIVEN A UNIQUE NAME FOR EASE OF RESCHEDULING.
''
SCHEDULE A U6.NODE.WAKE.UP CALLED THE.WAKE GIVEN
    THIS.NODE, MSG, PAST.NODE, NO.PCOUNT AT SETTING(BUZZER)
```

```
''
LET BUZ.WAKE(BUZZER) = THE.WAKE
''
FILE BUZ.NODE IN BUZ.LIST(BUZZER)
FILE BUZZER IN ALARM.CLOCK(THIS.NODE)
''
GO CHECK.THE.QUEUES
''
''  .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
'WARN.MSG'
''
''
''  DETERMINE IF THIS IS A GROUP OR FAMILY WARNING AND TAKE PROPER ACTION.
GO TO ERROR, GRP.WARNING, FAM.WARNING PER CLASS(MSG)
''
'ERROR'
''  INVALID NODE WARNING MESSAGE. NO ACTION TAKEN.
    DESTROY UPDATE CALLED U.MSG
    GO CHECK.THE.QUEUES
''
''  .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
'GRP.WARNING'
''
''  CHECK IF GROUP IS VALID.
''
IF GROUP(THIS.NODE) NE ORIGIN(MSG),
    DESTROY UPDATE CALLED U.MSG
    GO CHECK.THE.QUEUES
REGARDLESS
''
''  KEEP A RECORD OF TRANSMISSIONS.
''
CREATE A RECORD
LET REC.ORIGIN(RECORD) = ORIGIN(MSG)
LET REC.TIME(RECORD) = BEGIN.TIME(MSG)
FILE RECORD IN XMT.RECORD(THIS.NODE)
''
''  LOCAL NODE RECORDS ARE MAINTAINED FOR ONLY A SET PERIOD OF TIME.
''
IF INIT.BP = DEAD,
SCHEDULE A U8.ERASE.RECORD GIVEN THIS.NODE, RECORD IN ERA.PERIOD UNITS
REGARDLESS
''
''  SCHEDULE THE RETRANSMISSION OF THE WARNING GROUP UPDATE TO
''  ALL NEIGHBOR NODES WITHIN THIS GROUP.
''
FOR EACH LINK IN LINK.SET(THIS.NODE) WITH GROUP(THIS.NODE) =
    GROUP(TERM(LINK)) AND TERM(LINK) NE PAST.NODE, DO
''
    CREATE AN UPDATE
```

```
        LET BEGIN.TIME(UPDATE)  = BEGIN.TIME(MSG)
        LET CLASS(UPDATE)       = CLASS(MSG)
        LET ORIGIN(UPDATE)      = ORIGIN(MSG)
        LET FM.NODE(UPDATE)     = THIS.NODE
        LET VARIETY(UPDATE)     = VARIETY(MSG)
        LET HOPCNT(UPDATE)      = HOPCNT(MSG) + 1
        LET TO.NODE(UPDATE)     = TERM(LINK)
        LET FIRING.TIME(UPDATE) = FIRING.TIME(MSG)
''
    PERFORM U2.TRANSMIT.UPDATE GIVEN UPDATE, LINK
LOOP
''
''  SCHEDULE THE TRANSMISSION OF A REGULAR GROUP UPDATE MESSAGE
''  TO ALL NEIGHBOR NODES WHICH ARE NOT IN THIS NODE'S GROUP.
''
IF PRNT >= 3,
SKIP 1 LINE
PRINT 2 LINES WITH TIME.V, THIS.NODE, GROUP(THIS.NODE),
    FIRING.TIME(MSG) AS FOLLOWS
***.***** (**) : GROUP UPDATE MESSAGE WARNING ORIGINATED.
                 GROUP(**) SET TO XMT UPDATE AT ***.*****.
REGARDLESS
''
SCHEDULE A U7.UNIT.FIRING GIVEN THIS.NODE, CLASS(MSG), ORIGIN(MSG),
    BEGIN.TIME(MSG) AT FIRING.TIME(MSG)
DESTROY UPDATE CALLED U.MSG
GO CHECK.THE.QUEUES
''
''  .   .   .   .   .   .   .   .   .   .   .   .   .   .
'FAM.WARNING'
''
''  CHECK IF FAMILY IS VALID.
''
IF FAMILY(THIS.NODE) NE ORIGIN(MSG)
    DESTROY UPDATE CALLED U.MSG
    GO CHECK.THE.QUEUES
REGARDLESS
''
''  SCHEDULE THE RETRANSMISSION OF THE WARNING FAMILY UPDATE TO
''  ALL NEIGHBOR NODES WITHIN THIS FAMILY.
''
''  KEEP A RECORD OF TRANSMISSIONS.
''
CREATE A RECORD
LET REC.ORIGIN(RECORD) = ORIGIN(MSG)
LET REC.TIME(RECORD) = BEGIN.TIME(MSG)
FILE RECORD IN XMT.RECORD(THIS.NODE)
''
''  LOCAL NODE RECORDS ARE MAINTAINED FOR ONLY A SET PERIOD OF TIME.
''
```

```
IF INIT.BP = DEAD,
SCHEDULE A UB.ERASE.RECORD GIVEN THIS.NODE, RECORD IN ERA.PERIOD UNITS
REGARDLESS
''
FOR EACH LINK IN LINK.SET(THIS.NODE) WITH FAMILY(THIS.NODE) =
    FAMILY(TERM(LINK)) AND TERM(LINK) NE PAST.NODE, DO
''
    CREATE AN UPDATE
         LET BEGIN.TIME(UPDATE)  = BEGIN.TIME(MSG)
         LET CLASS(UPDATE)       = CLASS(MSG)
         LET ORIGIN(UPDATE)      = ORIGIN(MSG)
         LET FM.NODE(UPDATE)     = THIS.NODE
         LET VARIETY(UPDATE)     = VARIETY(MSG)
         LET HOPCNT(UPDATE)      = HOPCNT(MSG) + 1
         LET TO.NODE(UPDATE)     = TERM(LINK)
         LET FIRING.TIME(UPDATE) = FIRING.TIME(MSG)
''
    PERFORM U2.TRANSMIT.UPDATE GIVEN UPDATE, LINK
LOOP
''
''  SCHEDULE THE TRANSMISSION OF A REGULAR FAMILY UPDATE MESSAGE
''  TO ALL NEIGHBOR NODES WHICH ARE NOT IN THIS NODE'S FAMILY.
''
IF PRNT >= 3,
SKIP 1 LINE
PRINT 2 LINES WITH TIME.V, THIS.NODE, FAMILY(THIS.NODE),
    FIRING.TIME(MSG) AS FOLLOWS
***.**** (**) : FAMILY UPDATE MESSAGE WARNING ORIGINATED.
                FAMILY(**) SET TO XMT UPDATE AT ***.****.
REGARDLESS
''
SCHEDULE A U7.UNIT.FIRING GIVEN THIS.NODE, CLASS(MSG), ORIGIN(MSG),
    BEGIN.TIME(MSG) AT FIRING.TIME(MSG)
DESTROY UPDATE CALLED U.MSG
''
'CHECK.THE.QUEUES'
''
FOR EACH LINK IN LINK.SET(PAST.NODE) WITH TERM(LINK) = THIS.NODE,
    FIND THE.LINK = LINK,
IF FOUND,
''
''  FIRST CHECK TO SEE IF THERE IS ADDITIONAL UPDATE TRAFFIC WAITING
''  TO BE SENT.
''
    IF UQUEUE(THE.LINK) IS NOT EMPTY,
         REMOVE FIRST Q.UP.MSG FROM UQUEUE(THE.LINK)
         FILE Q.UP.MSG IN UPROP.QUEUE(THE.LINK)
         SCHEDULE AN U4.RECEIVE.UPDATE GIVEN Q.UP.MSG IN U.XMN.TIME UNITS
''
         IF PROP.QUEUE(THE.LINK) IS NOT EMPTY,
```

```
**
**            THEN THE UPDATE WHICH JUST ARRIVED WAS INSERTED INTO
**            THE BIT STREAM OF A MESSAGE PACKET IN TRANSIT.  THEREFORE
**            THE NEXT UPDATE TO BE SENT WILL ALSO BE INSERTED.
**
              PERFORM U3.INSERT.UPDATE GIVEN THE.LINK
          REGARDLESS
      ELSE
**
**        THE UPDATE QUEUE IS EMPTY SO WE WILL CHECK FOR MESSAGE PACKET
**        TRAFFIC WAITING TO BE SENT.
**
          IF PROP.QUEUE(THE.LINK) IS NOT EMPTY,
**
**            THERE IS CURRENTLY MESSAGE TRAFFIC BEING SENT OVER THIS
**            LINK AND NO FURTHER ACTION IS NECESSARY.
              RETURN
          REGARDLESS
**
**        THERE IS NO TRAFFIC IN PROGRESS SO WE WILL CHECK THE MESSAGE
**        QUEUE TO SEE IF TRAFFIC IS WAITING TO BE SENT.
          IF QUEUE(THE.LINK) IS NOT EMPTY,
              REMOVE FIRST PAC.KET FROM QUEUE(THE.LINK)
              LET QUESIZ(THE.LINK) = N.QUEUE(THE.LINK)
              SCHEDULE A M2.RECEIVE.MESSAGE.PACKET CALLED
                   THE.NAME GIVEN PAC.KET AT TIME.V + PKT.XMN.TIME
              LET P.ARRIVAL(PAC.KET) = TIME.V + PKT.XMN.TIME
              LET P.NAME(PAC.KET)    = THE.NAME
              FILE PAC.KET IN PROP.QUEUE(THE.LINK)
              RETURN
          REGARDLESS
**
**        BOTH QUEUES ARE EMPTY AND NO MESSAGES ARE IN TRANSIT.
          LET STATUS(THE.LINK) = IDLE
      REGARDLESS
  REGARDLESS
  RETURN
**
  END **OF U4.RECEIVE.UPDATE
**********************************************************************
**
  EVENT U5.LINK.WEIGHT.CALCULATION
**
**  THIS EVENT DETERMINES THE POSITIVE WEIGHT ASSIGNED TO EVERY LINK
**  FROM WHICH THE YEN ALGORITHM DETERMINES THE BEST PATH ROUTING
**  BETWEEN NODES.  FIVE METHODS ARE DETERMINED IN THIS ROUTINE AND
**  THE USER SPECIFIES WHICH IS ACTUALLY UTILIZED DURING THE
**  SIMULATION.
**
  DEFINE NUM, WEI.GHT AS INTEGER VARIABLES
```

```
DEFINE WT.AVG AS REAL VARIABLES
''
LET NUM = NUM.OF.WEIGHTS
''
IF MANNER.OF.WEIGHTING = BY.QSIZE.NOW,
''
''  THIS METHOD INVOLVES USING THE INSTANTANEOUS QUEUE LENGTHS FOR
''  FINDING THE LINK WEIGHT.  THIS METHOD MAKES NO FURTHER USE
''  OF THIS ROUTINE.
    GO NO.MORE
REGARDLESS
''
FOR EACH NODE, DO
    FOR EACH LINK IN LINK.SET(NODE), DO
''
        CREATE A WEIGHT
''
        GO TO NO.MORE, USE.PAST.QSIZE, USE.PAST.QAVG,
             USE.RHO.WEIGHT, USE.RHO.WEIGHT PER MANNER.OF.WEIGHTING
''
        'USE.PAST.QSIZE'
''
''      THIS METHOD OF WEIGHTING IS BASED UPON TAKING THE AVERAGE OF
''      'N' QUEUE SAMPLES TAKEN DURING A "WINDOW" OF FIXED LENGTH.
''      THIS WINDOW 'SLIDES' ALONG IN TIME SINCE ONLY THE MOST RECENT
''      SAMPLES ARE USED IN DETERMINING THE AVERAGE.
''
        LET WT.SIZE(WEIGHT) = N.QUEUE(LINK)
        GO AROUND.AGAIN
''
        'USE.PAST.QAVG'
''
''      THIS METHOD USES THE SAME SLIDING WINDOW TECHNIQUE EXCEPT
''      INSTEAD OF TAKING A 'SNAPSHOT' SAMPLE OF THE QUEUE SIZE AT
''      REGULAR TIME INTERVALS, THIS USES THE AVERAGE QUEUE SIZE OVER
''      THE TIME INTERVAL.
''
        LET WT.SIZE(WEIGHT)  = AVG.WEIGHT(LINK)
        RESET TOTALS OF QUESIZ(LINK)
        GO AROUND.AGAIN
''
        'USE.RHO.WEIGHT'
''
''      THIS METHOD USES A COMBINATION OF WEIGHTING FUNCTIONS; ONE
''      BASED UPON THE LINK UTILIZATION FACTOR DURING THE PAST TIME
''      WINDOW AND THE SECOND A LEAST HOPS SYSTEM.  WHEN THE LINK
''      UTILIZATION FACTOR IS LESS THEN .5 THE BEST PATHS WILL BE
''      DETERMINED PRIMARILY BY A LEAST HOPS SCHEME.  HOWEVER IF THE
''      UTILIZATION FACTOR IS GREATER THAN .5 THE LINK WEIGHT WILL BE
''      PREDOMINANTLY A FUNCTION OF THAT FACTOR.
```

```
''        A SECOND METHOD (BY.COMBINE) MAKES USE OF THIS TERM BUT IN
''        ADDITION IT ALSO USES THE SIZE OF THE QUEUE.  IN THIS
''        METHOD BOTH THE AVERAGE LINK USAGE (THE RHO TERM) AND THE
''        CURRENT TRANSIENT BEHAVIOR OF THE LINK (THE QUEUE SIZE TERM)
''        ARE TAKEN INTO CONSIDERATION.
''
          IF RHO(LINK) > .95,
               LET WT.SIZE(WEIGHT) =  20.0
               GO SOME.MORE
          REGARDLESS
          LET WT.SIZE(WEIGHT) = 1. + (RHO(LINK)/(1. - RHO(LINK)))
          'SOME.MORE'
          RESET TOTALS OF N.PROP.QUEUE(LINK)
''
          'AROUND.AGAIN'
''
''        ONLY THE PAST 'NUM' OF SAMPLES ARE CONSIDERED.
          IF N.SET.OF.WEIGHTS(LINK) >= NUM,
''
               REMOVE THE FIRST WEIGHT FROM SET.OF.WEIGHTS(LINK)
               DESTROY WEIGHT CALLED WEIGHT
          REGARDLESS
''
          FILE WEIGHT IN SET.OF.WEIGHTS(LINK)
''
          FOR EACH WEIGHT IN SET.OF.WEIGHTS(LINK), COMPUTE WT.AVG
               AS THE MEAN OF WT.SIZE(WEIGHT)
          LET LK.WEIGHT(LINK) = WT.AVG
     LOOP
LOOP
''
SCHEDULE A US.LINK.WEIGHT.CALCULATION IN WT.TIME UNITS
''
'NO.MORE'
RETURN
END ''OF US.LINK.WEIGHT.CALCULATION
''  ************************************************************************
''
EVENT U6.NODE.WAKE.UP GIVEN WK.NODE, WK.MSG, BPATH, HOPNUM
''
''  THIS EVENT SCHEDULES THE TRANSMISSION OF AN UPDATE AT THE MOMENT
''  THE NODE'S ALARM CLOCK GOES OFF FOR THAT UPDATE MESSAGE.
''
DEFINE THIS.NODE, MSG, BESTPATH, HOP.NUM, THE.BUZZ, FLAG
     AS VARIABLES
DEFINE NEW.TIME AS A REAL VARIABLE
''
LET THIS.NODE = WK.NODE
LET MSG = WK.MSG
LET BESTPATH = BPATH
```