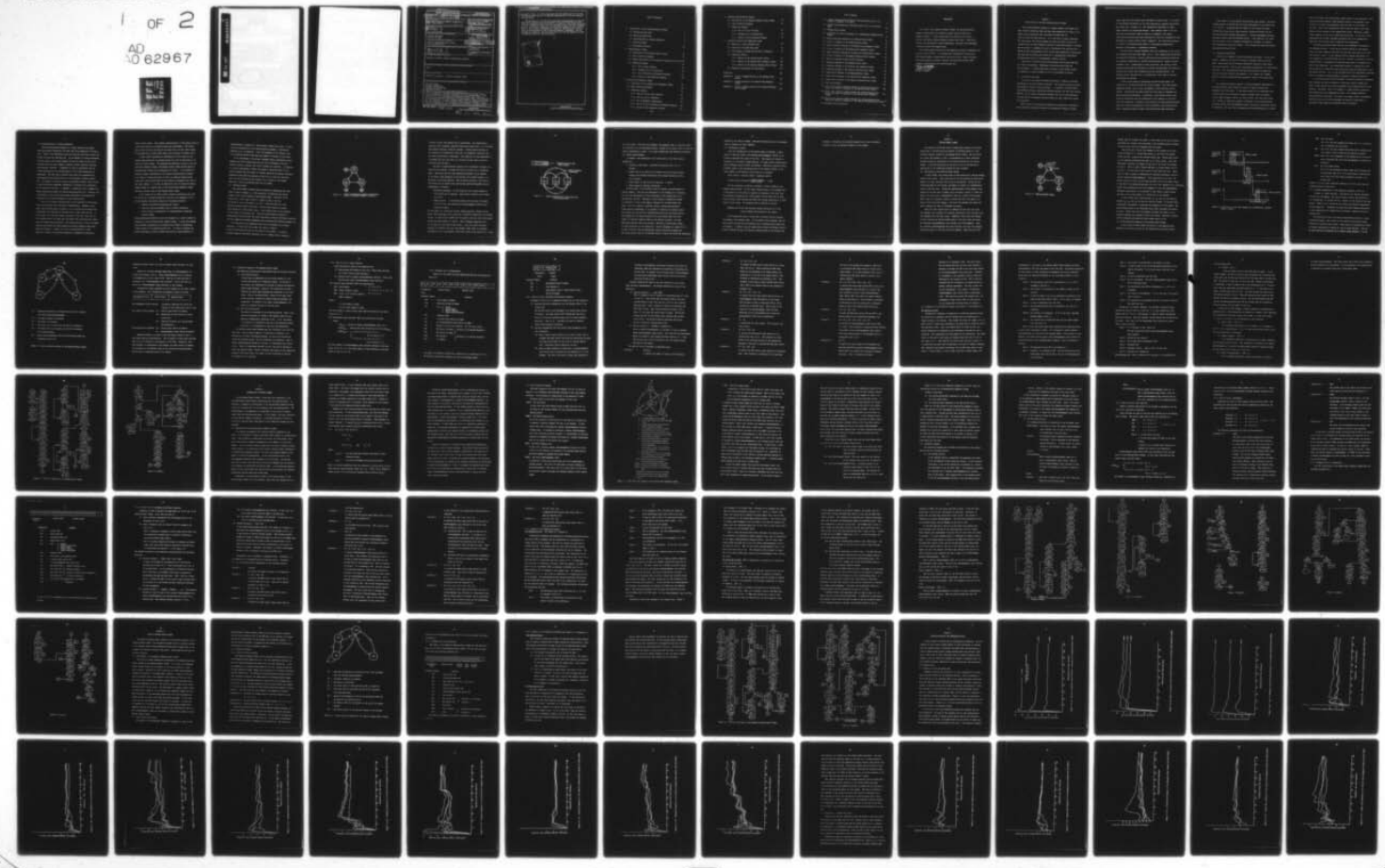
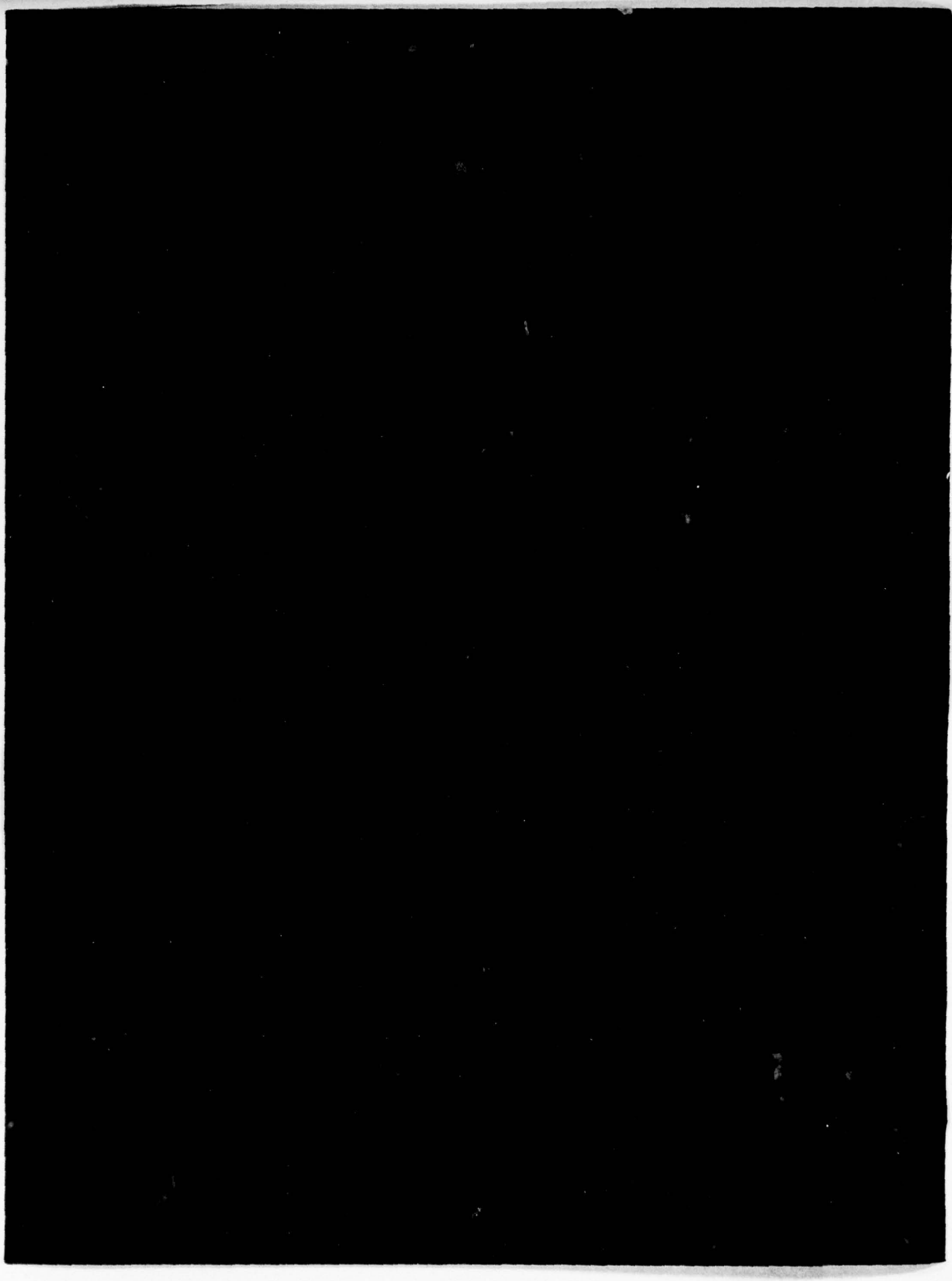
AD-A062 967 SYRACUSE UNIV N Y F/G 9/2
DATA SYNCHRONIZATION SCHEMES FOR MULTIPLE COPIED DATA BASES.(U)
DEC 78 C LEE F30602-75-C-0121
UNCLASSIFIED RADC-TR-78-240 NL

1 OF 2
AD A062967





UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(18) 1. REPORT NUMBER: RADC-TR-78-240 (19) TR-78-240 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): (16) DATA SYNCHRONIZATION SCHEMES FOR MULTIPLE COPIED DATA BASES

5. TYPE OF REPORT & PERIOD COVERED: (9) Final Technical Report / Jan 78 - Jun 78

6. PERFORMING ORG. REPORT NUMBER: N/A

7. AUTHOR(s): (10) Chin-Hwa Lee

8. CONTRACT OR GRANT NUMBER(s): (15) F30602-75-C-0121

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Syracuse University, Syracuse NY 13210 — 339 600

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, (16) 5599, 55971503

11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISCP), Griffiss AFB NY 13441

12. REPORT DATE: (11) December 1978 — (17) 15

13. NUMBER OF PAGES: 134

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same — (12) 143 p.

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

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)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Richard A. Metzger (ISCP)

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

Data Processing
Computer Programming
Programs and Programming
Information Theory

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

In the network environment with distributed multiple-copied files, a lockout mechanism is required to guarantee the data synchronization. File access requests from geographically dispersed computer nodes have to be coordinated to maintain consistency of multiple-copied files. The advantages of a multiple-copied file in a loosely coupled computer communication network are its reliability to partial network failures and good response to real time file manipulation. In this report, the fundamental problems relating to the lockout synchronization of a multiple-copied file are presented from a

DD FORM 1 JAN 73 1473 — 339 600

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

79 01 04 012

new point of view. It can be shown that the file lockout state of the distributed multiple copies is an inherent characteristic of the file manipulation operations.

The objective of this work was to determine the performance of data synchronization techniques in the maintenance of distributed multiple-copied files in a computer network using G.P.S.S. simulation. The system visualized is a set of identical copies of a file residing in nodes that form a computer network. Each file is supervised by a Local File Manager (L.F.M.) which is under a control scheme to maintain the congruency and consistency of these files by synchronizing the file access and updating information. This is achieved by locking out the file copies once an L.F.M. was granted the right of updating a file for a particular user.

A

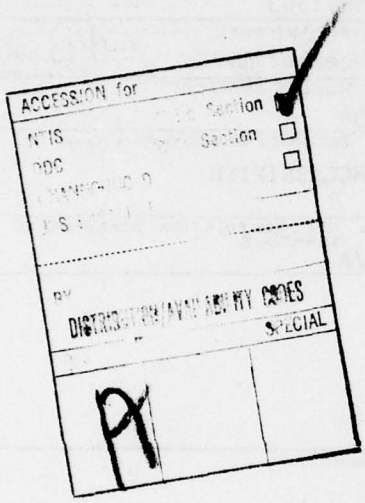


Table of Contents

List of Figures

EVALUATION

This effort investigated alternate schemes for synchronizing the update of files which are located on several distinct hosts on a computer network. It was assumed that the individual files could receive simultaneous update requests from multiple users. Theoretical analysis was performed on various approaches, and models were developed to obtain relative performance data.

This effort applies directly to the Command and Control Information Processing Thrust of TPO-3. It provides basic technology in the area of distributed computer system control and can be used as input to several distributed data base efforts recently initiated under Project 2530 "Computer System Reliability and Survivability".

Richard A Metzger

RICHARD A. METZGER
Project Engineer

Chapter 1

Introduction of the Data Synchronization Problem

Data synchronization problem in a single computer environment had been studied intensively some time ago using semaphores by Dijkstra [12]. There is the possibility that concurrent processes may intend to access the same data set. If in the lack of lockout mechanism for file access, one process changes the data set based on the old information that has been changed by another process, unrecoverable problems may be created. A semaphore is used to synchronize the concurrent processes so that exclusiveness of file access among themselves is guaranteed. The same type of problem also exists for a geographically distributed multiple-copied file in a heterogeneous computer network.

The nature of the file access exclusiveness and multiple updatings of a file is considered. Advantages of multiple-copied files are also discussed. A simple study model proposed in this chapter allows a comparison of control schemes on a set of performance criterion.

1.1 Distributed Data Base

In the age of so-called information explosion, computer networking technology is now being widely recognized. The original motivation behind networking is aimed at resource sharing. It generally included expensive hardware sharing, specialized software sharing, and data base sharing. However, in the trend of decreasing hardware cost due to new technology, the data base or information sharing becomes the most significant reason for networking.

When the need to facilitate transferring of appropriate information across partially connected networks of heterogeneous computers became

clear, many ideas and studies were initiated in recent years. As a result of evolutionary procedures, it is often desirable to integrate distributed data bases into a network [1]. The conventional approaches to handle these problems are centralized methods. Some computer nodes in the network have an equivalent right of control as compared to the others. Centralized approaches have been disfavored by their poor dependability and potential of causing network traffic congestion. The distributed approach to control of distributed data bases is a challenging problem which deserves a large amount of fundamental research.

Over the years, networks designers have been interested in the reality of a Network Operating system (NOS) [2,3,4]. User communication primitives, job migration primitives, control primitives, and data migration primitives are essential components in a Network Operating system. Despite the many problems in the implementation of other primitives, the reality of data migration alone means the existence of global (network-wide) file directory, partial file access, data translation, and data synchronization. The research effort described here is concentrated to the study of the data synchronization problem.

In the initial stage of integrating distributed data bases, the problem is to conceive and study the control scheme. Once some limited capability exists, there is also an emphasis on data security and protection. Distributed data base allows the local owner to implement any desired protection mechanism. In addition, a distributed control mechanism can efficiently use the communication link capacity to avoid network flow congestion. Therefore, the objectives of using distributed data bases and control mechanisms are to have reliability, fast access performance, optimizing usage of links, and localized data base protection.

Many aspects of a distributed data base have been studied. The partitioning problem of distributed files was investigated by the pioneer work of W. Chu [5]. He assumed that the number of file copies was known, queries were routed to all files, queries' pattern were known, and the arrivals followed a Poisson distribution. A linear programming technique was used to resolve the partitioning problem. V.K.M. Whitney, R.G. Casey, and K.D. Levin later on relaxed or extended the technique to consider the communication system as a whole. This included both data and program allocation problems (6,7,8).

1.2. Multiple-Copied Files

There are two methods to setup distributed data bases. In the first method, segments of a file are located at different nodes of the network. Data access will be directed through the networks to the correct node for processing. This is the conventional configuration of a distributed data base. The second method allows multiple-copies of the same file located at several nodes of the networks. P.R. Johnson, R.H. Thomas, and W.R. Sutherland mentioned these methods in some of their works [9,10 11].

In a dispersed computer network, frequency bandwidth requirements of the links between network nodes are chosen to support satisfactory thoughput and packet delay. It has been pointed out by researchers that extensive efforts are required to design a network which can support various high thoughput traffic, low delay traffic, and real time traffic [3]. Whenever a high level protocol is designed for an existing packet switching system, the time-bandwidth product required by the protocol should be minimized. The real time file manipulation and modification of a remote

file in a foreign node usually puts a great strain to the capacities of the store and forward networks. With multiple copies in the networks a user is able to retrieve and manipulate a file copy that is the closest to the user. User's file manipulation traffic does not need to load large amounts of the capacity of the communication links. Therefore, communication capacity of the link is used more efficiently. At the end of file manipulation activities, updating information has to be transferred to nodes with the same file copy to maintain file consistency.

Distributed multiple-copied file has the advantages of reliability and redundant backups. In the case of partial failure of the computer nodes, the other computer nodes with the same file copy still can provide file access to users. If the computer host has enough distributed control capability, a multiple-copied file in a network is resilient to partial network failures. Reliability of important data bases in a computer network is very important. The conventional approach to upgrade the reliability requires a standby data base system which does not participate in normal network activity. But, in the multiple-copied file environment mentioned above, no expensive standby system is required to achieve reliability.

In a multiple-copied file environment, control and updating information flows through the links according to a high level control scheme (protocol), those traffic are already enciphered by the previous states of the protocol. Therefore, there is no danger of a large volume of sensitive file flowing about the communication links. The resultant security should be better than before. Of course, low volume traffic, reliability, and better security in this environment are gained in compromise for redundant storage capacities required across the network.

1.3 Synchronization or Lockout mechanism

Data synchronization problems in a single computer environment have been studied intensively for some time using semaphores by Dijkstra [12]. There is the possibility that more than one concurrent process may intend to access the same data set. In the absence of lockout mechanisms for file access, one process changes the data set based on the old information that has been changed by another process, therefore creating unrecoverable problems. A semaphore is used to synchronize the concurrent processes so that exclusiveness of file access among them is guaranteed. The same type of problem also exists for geographically distributed multiple-copied files in a heterogeneous computer network [9]. In addition to the file access exclusiveness problem, there is also a need for multiple updatings mechanisms to guarantee the consistency of all copies of the file. J. Rothnie, P. Bernstein, and N. Goodman proposed serialization method to achieve synchronization [13]. However, synchronization of file access and multiple updatings in the network are achieved by the lockout mechanisms in this reported work.

There is an inherent characteristic in the operation of file access which can be either file evaluation or modification. Whenever a user gains the exclusive right to access a file, he should be protected from interference of activities on the same file by other users [15,16]. In other words, once the file access is granted to one user, that file is in a lockout state. No other user can access that file until the current user has finished his activities. The lockout state of a file in a single computer environment has been studied and handled elegantly using semaphores by Dijkstra. Lockout of a file is synchronized by a semaphore for a number of concurrent processes to guarantee the exclusive

right of file access. This inherent characteristic of the lockout state of a file also exists in a multiple-copied file environment. The difference is that not only one copy in the local host, but also other copies of the same file at remote hosts need to be protected in lockout state.

In this report prevention of interference of file access by the lockout synchronization of multiple-copied file will be described in details for three schemes. The modeling and simulation results of the network-wide semaphore scheme, and hopping permit scheme and the adaptive hopping permit scheme will be discussed [17,18,19]. It is possible to provide a general description of the lockout synchronization schemes. As a result of taking this point of view, an adaptive hopping permit scheme was conceived and shown to have superior performance than that of the other schemes. It is easy to demonstrate that the adaptive hopping permit scheme is a special case of the network-wide semaphore scheme and also a special case of the hopping permit scheme.

In our search for an ideal control scheme to synchronize the lockout states of a multiple-copied file, the local file managers (L.F.M.) of the computer node should possess the following attributes.

* involves distributed processing and control.
* uses only limited available local or global information.
* can be either a deterministic or nondeterministic (adaptive) control scheme.

Having distributed identical local file managers in a control scheme is essential to survive from partial network outages. A local file manager with available information can coordinate with others to synchronize lockout states of the multiple-copied file. In order to optimize the system performance criteria, either deterministic (time invariant) or

nondeterministic (adaptive or time variant) scheme can be used. To minimize the vulnerability of the synchronization schemes, a centralized approach cannot be employed. Local file managers at the computer node of the network should be identical and immune to failures of each other.

It is intriguing to the network designers which distributed control schemes can achieve these objectives and which performance criterion should be used to evaluate the control schemes (or protocol). In addition, it is also interesting to reveal the change of characteristics of the control schemes when the number of computer nodes in the network increases. Simulation using GPSS was employed in this work to study the characteristics of the proposed control schemes and to conceive a new scheme which has a better performance than the old scheme.

1.4 Simulation Model

The network model in which control schemes are implemented and simulated is shown in Figure 1.1. For simplicity in comparing the control schemes, only a file X exists in this network model. Each host in the network has a local copy of file X. Access to file X can only be granted to one of the users p, q, and r. While one user at a host gains the access right of the file X, all the other copies at foreign hosts should be in the lockout state. This network model can be expanded to four nodes or five nodes, etc. The data synchronization problem is complicated by the fact that the user's processes located at different computers can only communicate between themselves with stochastically delayed messages (4).

It is assumed in our simulation model that the networks are strongly connected. In other words we assume that there is always a virtual link between each pair of the nodes in the network. In reality a network might be partially connected, but at a higher level of protocol,

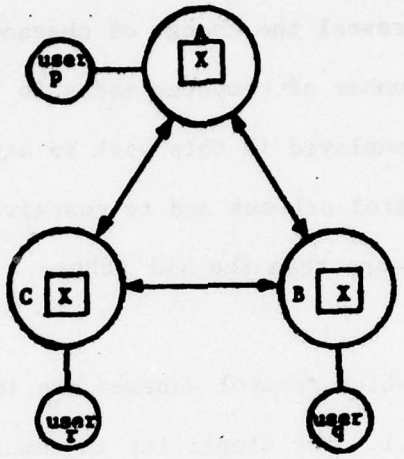


Figure 1.1 **Lockout synchronization problem of multi-copied file X in a network of computer A, B and C.**

a direct virtual link usually can be established. The communication link will have stochastic uniformly distributed delay from .1 to .9 second. Full-duplex idealistic links are assumed. Because characteristics of the protocol are studied here, data rate (or bandwidth) limitation has not been incorporated in this model. For simplicity in the simulation it is assumed that only one file X is involved and each node on the networks has only one user.

The lockout synchronization problem can be considered in Figure 1.2, where all the requests to access the file are forced into a conceptual queue. This queue may not exist physically anywhere in the network. If the queue forms somewhere in the network, this scheme becomes a undesirable centralized approach. The activities related to this file in the network can be divided into alternating synchronizing-phase and updating-phase in sequence.

* Synchronizing-phase: In this phase one file access request in the network will be granted. Some control traffics occur during this phase.
* Updating-phase: In this phase granted user process will modify its local copy, and updating and acknowledgement traffic occur during this phase.

By some means in a particular control scheme granting a request is resolved, then updatings of the local copy followed by updating of the foreign copies. In this work we assume that the service time for local file manipulation has fixed period of 1 second. This stringent assumption will be set aside for the study in the future. After the local file manipulation is finished, the local file manager (LFM) sends out updating information to all the computer nodes with copies of the same file X. Later

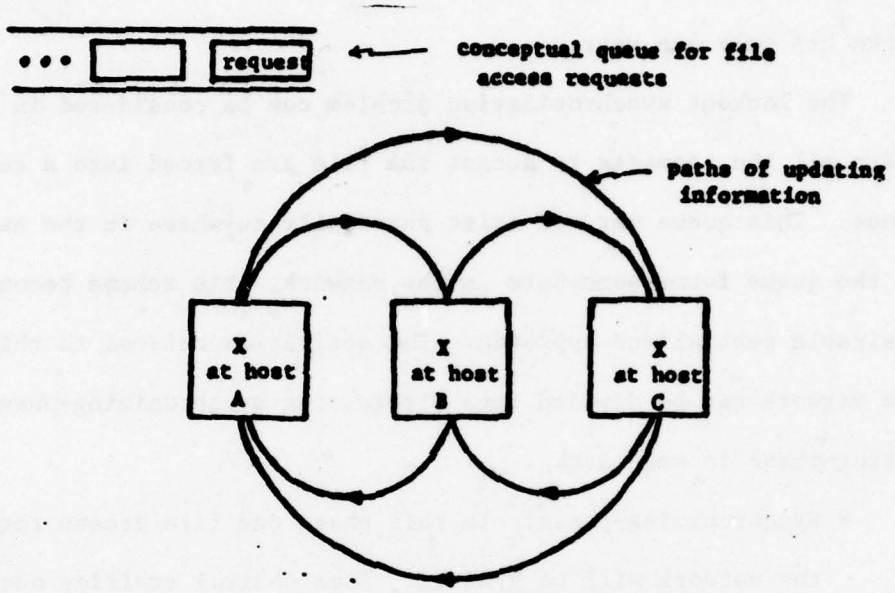


Figure 1.2 Lockout synchronization of multi-copied file in the network model.

on in the report, the local file manager, the computer node, or the node itself are referred to in an intermixed fashion, because in our model, there is only one control mechanism at a node. It is also assumed that the updating information is a fixed length message.

In summary, the assumptions in the simulations of the three control schemes are,

* stochastic link delay: uniformly distributed from .1 to .9 seconds.
* Single user at the node of the network with file access request: requests are Poisson distributed with average interarrival time of 20 seconds.
* fixed length of time for local updating: 1 second.
* fixed length of updating information.

In this study, G.P.S.S./360 was used to simulate the performance of control schemes. Not only the performance of the schemes are of interest, but also the dependencies of the performance on the network size is of considerable interest. Therefore, clever design of simulation program is essential to avoid finite memory limitation of the main-frame computer. G.P.S.S. supports a powerful indirect addressing capability. Using indirect addressing, it is possible to simulate the network activities using only two sections; transmitting section and receiving section. The program size of these sections do not grow rapidly when the total number of simulated computer nodes in the network increase. For each additional program, cards are inserted, therefore, leaving more storage for data collection in the simulation. Another advantage of using G.P.S.S. is that it allows close correspondence between simulation programs and the actual system. This way it becomes easier to debug and verify the simulation

program of the control schemes. Detailed descriptions about the programs will be presented in later chapters.

1.5 Performance Criteria

Once the configuration of the network model is decided, a fixed amount of updating traffic and updating acknowledgement traffic is involved to maintain the copies of the file. This amount of traffic is invariant to the control scheme employed. An ideal control scheme should have as minimal control traffic as possible. Therefore, control-traffic-overhead, which is the percentage ratio of the control traffic to the total traffic in the network, can be used as a criterion.

Total traffic = control traffic + updating traffic

$$\text{Control-traffic-overhead} \triangleq \frac{\text{control traffic}}{\text{total traffic}} \times 100\%$$

The next important criterion to evaluate a control scheme is the average response-time. For our simple network model, it is assumed that a fixed amount of one second is required in updating of the local file. The updating information has a fixed length of data which will be simulated by update begin message and update end message separated by a fixed period of one second. The response time is defined as follows:

Response-time $\triangleq$ the time interval between initiation of a file access request and granting of the request.

In the simulation study, a fixed load of Poisson arrival requests is assumed at the computer node. As the network size increases, the control-traffic-overhead and the response-time in handling these loads vary accordingly. A superior control scheme should impose the minimal control-traffic-overhead and have the shortest response-time as the network size

increases. Therefore, the variation sensitivity of the performance criterion is also an important measure of the schemes.

Chapter 2

HOPPING PERMIT SCHEME

The simplest distributed control scheme which achieves the desired objectives of synchronizing file updates of multiple-copies in a distributed computer network is the Hopping Permit Scheme. The activities of a Local File Manager (L.F.M.) is represented by a State Transition Diagram following a description of the basic operation of the control scheme. A detailed discussion of the use of G.P.S.S. entities and a description of the simulation program are also included.

2.1 Description of the Hopping Permit Scheme

In this scheme, file access right is associated with a special message packet called permit. Since only one host in the network can possess the permit at a time, exclusive file access is guaranteed. During the synchronizing-phase of this scheme, the permit is passed in a deterministic route to the next host. During the updating-phase of this scheme, local request at the host with a permit can be serviced. Figure 2.1 shows computer A, B, and C with a deterministic path for a permit in the scheme. When a user p at computer A wants to access the file X, he makes a request to his local file manager. The local file manager will grant the user's request only when it possesses the permit.

After the short modification is completed by the user, the local file manager then transmits the updating information to all foreign local file managers with the same copies. Meanwhile, this local file manager at host A blocks any further modification by the local users and waits for updating acknowledgements from foreign local file managers. After all updating acknowledgements have been received, the local file manager passes the permit to the next local file manager. When the local file

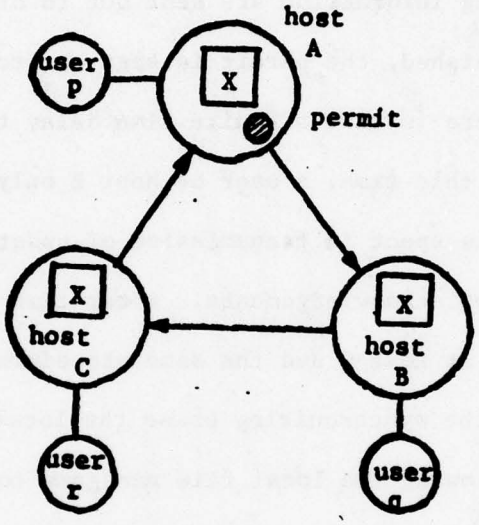


Figure 2.1 Hopping Permit Scheme.

manager does not possess the permit, it will queue up the local requests and modify the resident copy according to the incoming updating information from other local file managers with the permit.

The activities of local file managers in the network can be described in Figure 2.2. During time period t_A, the local file manager allows the user to modify the resident master copy. During time period $t_{\underline{A}}$, the updating information are sent out to other copies. After the updating is finished, the permit is sent out to the local file manager at host B. There is also a finite time delay t_1 before permit arrives at host B. At this time, a user at host B only reads the file. Therefore, no time is spent in transmission of updating information and waiting for updating acknowledgements. After time period t_2 is over, the permit arrives at host C and the same procedure starts over again. Since at the end of the synchronizing phase the local file managers are coordinated, rather than allowing the local file managers to start the open-bidding procedure autonomously, it is possible to keep them in lockout states and pass a file access permit to a second node in the network.

In this scheme, file access periods (t_A, t_B, and t_C) and updating periods ($t_{\underline{A}}$, $t_{\underline{B}}$, and $t_{\underline{C}}$) can have an arbitrary length. No precise clock information is necessary to guarantee the integrity of this scheme. The permit being passed by the associated hosts helps coordinate the events in order. Unless the local file manager possesses the permit, no ineffective messages are allowed to enter the network. Therefore, network traffic overhead associated with this scheme should be low.

2.2 State Transition Diagram of the Hopping Permit Scheme

In the course of events under the control of the Hopping Permit Scheme an L.F.M. may be in any of the following four distinct states,

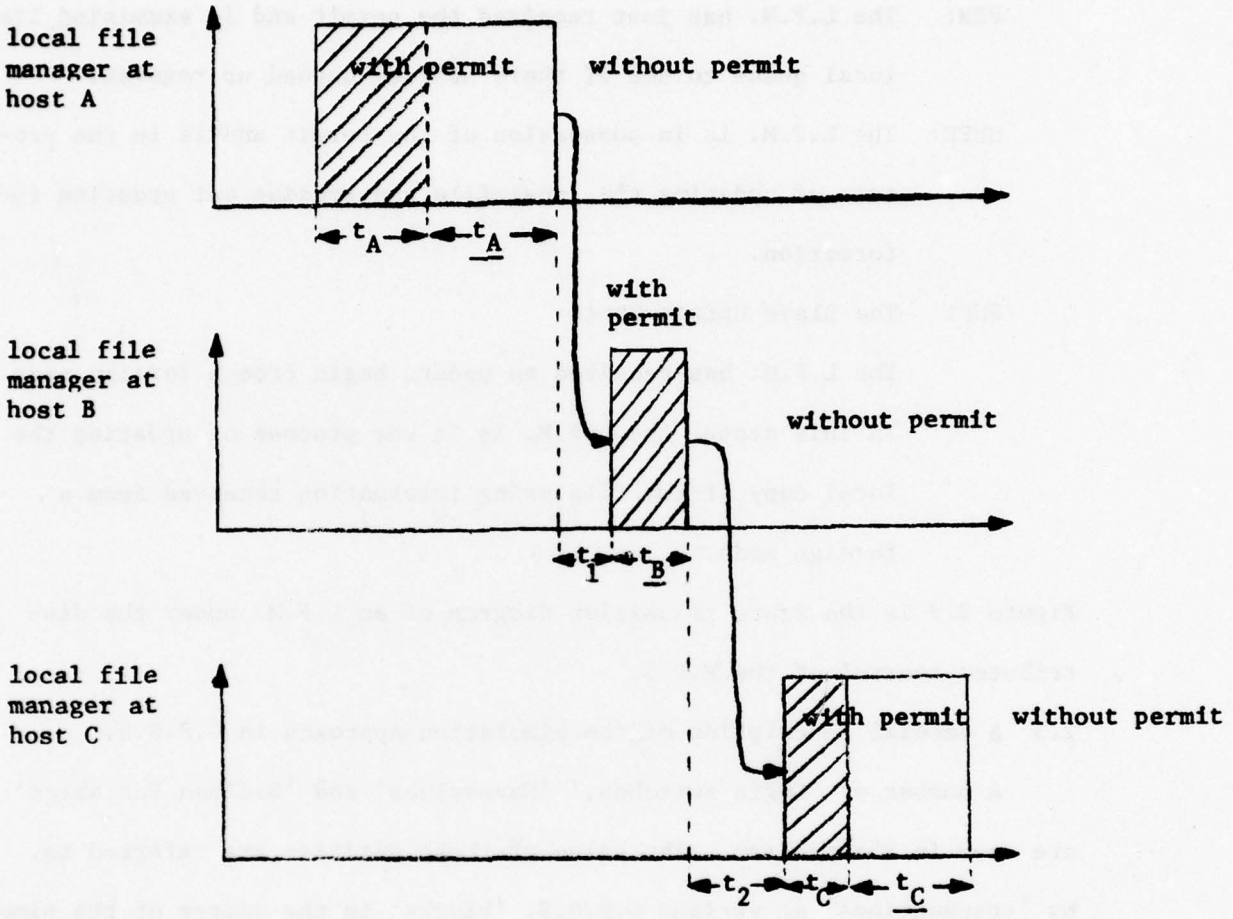


Figure 2.2 Activities of local file manager in a asynchronous Hopping Permit Scheme.

IDLE: The Idle State

The L.F.M. does not possess the permit nor is it in the process of updating its local file.

PER: The L.F.M. has just received the permit and is examining its local queue to see if there are any queued up request.

MUPD: The L.F.M. is in possession of the permit and is in the process of updating the local file and sending out updating information.

SUP: The Slave Update State

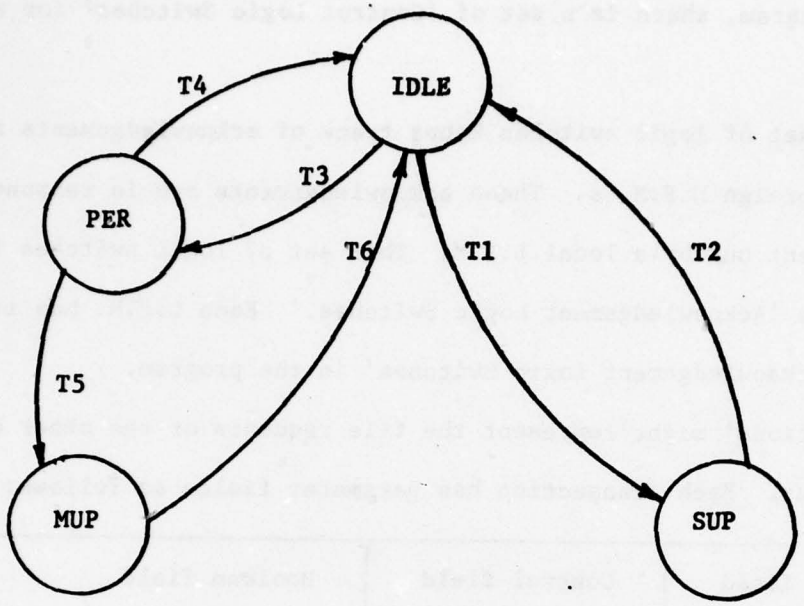
The L.F.M. has received an update begin from a foreign node. In this state, the L.F.M. is in the process of updating the local copy of the file using information received from a foreign node.

Figure 2.3 is the State transition diagram of an L.F.M. under the distributed control of the H.P.S.

2.3 A General Description of the Simulation Approach in G.P.S.S.

A number of 'Logic Switches,' 'Savevalues' and 'Boolean Variables' are used in the program. The value of these entities are referred to, by 'transactions' at various G.P.S.S. 'blocks' in the course of the simulation. These G.P.S.S. blocks in turn control the flow of transactions in the simulation. 'Variables' are used to achieve indirect addressing so that minimization of program size is possible. Details will be explained later.

The activities of the L.F.M. may be described as states in a state transition diagram. The L.F.M.'s in the course of activities transit through well defined stable states. These states are signalled by values or boolean combination of values of a set of Logic Switches. This set of logic switches is referred to as 'Control Logic Switches.' In the



T1 : Updating information is received and the L.F.M. proceeds into the foreign updating phase.

T2 : An update complete is received.

T3 : The permit is received.

T4 : The local queue is empty and the permit is passed on.

T5 : The local queue is non-empty and the L.F.M. proceeds into updating phase.

T6 : Update Acknowledgements from all participating nodes are received by the L.F.M.

Figure 2.3 State transition diagram of the Hopping Permit Scheme.

simulation program, there is a set of 'Control Logic Switches' for each L.F.M.

Another set of logic switches keeps track of acknowledgements received from foreign L.F.M.'s. These acknowledgements are in response to messages sent out by a local L.F.M. This set of logic switches is referred to as 'Acknowledgement Logic Switches.' Each L.F.M. has its own set of 'Acknowledgement Logic Switches' in the program.

'Transactions' might represent the file requests or the other communication messages. Each transaction has parameter fields as follows:

Information field	Control field	Boolean field

The information field contains information regarding the source destination of the transaction, and its type.

The control field contains
(1) Control Logic Switch ID numbers
(2) information for the control of transaction flow.
(3) temporary locations for storing indexing information.

The boolean field contains
(1) Control Logic Switch ID numbers
(2) Acknowledgement Logic Switch ID numbers

'Boolean Variables' are used to test the binary values of two or more logic switches simultaneously. The ID numbers of these logic switches that are to be referred to are entered in this field. Whenever a 'Boolean Variable' is referenced in the program, the logic switch ID numbers are obtained from the Boolean field of the transaction being processed. Details will be explained later in the chapter.

2.4 Simulation Program of the Hopping Permit Scheme

The simulation program may be partitioned into two distinct sections.

(1) The Sending Section:

Transactions are generated as file access requests in this section and queued up in respective queues. On receipt of the permit, the transaction is allowed to proceed and generate Update begin messages. Copies of the transaction are made, labelled as update begin messages, and dispatched to all other nodes. After a one second delay, copies of the same transaction are made, labelled as updated complete messages, and dispatched. On receipt of all update acknowledgements, the permit is released and sent to the next node.

(2) The Receiving Section:

The permit is contained in the receiving section. When a node receives the permit, it checks to see whether there are any queued up transactions (requests) or not. If so, the permit is retained to be released at the end of the updating session, otherwise, it is dispatched to the next node immediately.

An incoming update begin message sets the receiving L.F.M. into the SUP state. When an update complete transaction is received, it is labelled as an update acknowledge message, the source and destination fields are switched around, and the transaction is dispatched. When an update acknowledgement message is received, an acknowledge logic switch is set. When all acknowledge switches are set, the node is allowed to leave the updating phase. Before a detailed description of the simulation program is discussed later, the reader is first introduced to the use of various G.P.S.S. entities.

2.4.1 Use of G.P.S.S. Logic Switches

Logic Switches are used in the simulation for:

(1) Representing the states of the L.F.M. These logic switches are termed 'Control Logic Switches.'

(2) keeping track of update acknowledgements received. These logic switches are termed 'Acknowledgement Logic Switches.'

The 'Control Logic Switches' ID#'s are specified as:

SUP: Slave Update : $(i - 1) \times 3 + 1$

PER: Permit present in node : $(i - 1) \times 3 + 2$ $i = 1,2....N$

MUP: L.F.M. is in critical session : $(i - 1) \times 3 + 3$

(Master Update)

where:

i is the number of nodes

N is the total number of nodes

The total number of state control logic switches defined in the simulation is $3 \times N$

The 'Acknowledge Logic Switches' ID#'s are specified as follows:

where:

ACK_{jtoi}: denotes an update acknowledgement from j to i.

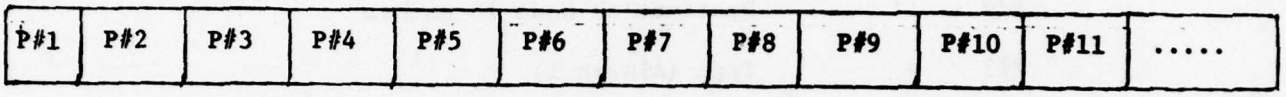
$LS\$ACK_{jtoi}$: denotes the logic switch that is set by ACK_{jtoi}.

$$LS\$ACK_{jtoi} = \begin{cases} 3 \times N + (i - 1) \times (N - 1) + j & \text{for } j<i \text{ and } 1<i \leq N \\ 3 \times N + (i - 1) \times (N - 1) + j - 1 & \text{for } j>i \text{ and } 1 \leq i < N \end{cases}$$

The total number of acknowledgement logic switches defined in the simulation is $N \times (N - 1)$. The total number of logic switches in the simulation is then $N \times (N + 2)$.

2.4.2 Parameter Set of Transactions

Except for the permit all the transactions have the following parameter set:



Information field — Control Field — Boolean field

Parameter Number — Contents

P#L : Source Node ID number

P#2 : Destination Node ID number

P#3 : Type 1. Permit
2. Update Begin
3. Update Complete
4. Update Acknowledgement

P#4 : Temporary Data

P#5 : A control Logic Switch ID number

P#6 : An Acknowledgement Logic Switch ID number

P#7 : Number of loops to be executed. For the loop counter.

P#8 : PER Switch ID number } Referred to by Boolean Variable 1

P#9 : MUP Switch ID number }

P#10 : ACK Switches } Referred to by Boolean Variable 2

P#11 : ID numbers }

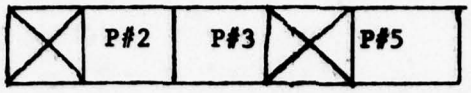
:

:

:

The number of parameters defined per transaction in a simulation is 8 + N.

The Permit's transaction parameter set has the following format:



Information field — Control field

Parameter Number — Contents

P#2 : Destination Node ID number

P#3 : Type (Always 1)

P#5 : Destination L.F.M.'s 'PER' Control Logic Switch ID number

2.4.3 Use of G.P.S.S. Variables and Boolean Variables

Variables in the G.P.S.S. simulation program are for the purpose of,

(1) Entering pertinent information into the Boolean Field of the transaction.

The Boolean field of the parameter set contains logic switch ID numbers. The Logic Switch ID#'s depend upon the source node's ID # and are obtained using a set of linear algebraic equations. (See 2.4.1). Variables are used to represent these linear algebraic equations.

(2) Entering information into the control field parameter set of the transaction.

(a) When a Control Logic Switch is to be set or reset, P#5 is assigned the Logic Switch ID# using the appropriate Variable. The Logic Switch Block in the G.P.S.S. program address that ; particular switch indirectly via P#5.

(b) When an update complete is dispatched, an Acknowledgement Logic Switch ID# is entered into the parameter # 6 of the message. The ID# of the first of these logic switches is

obtained by referencing a particular Variable after which the subsequent ID#'s are obtained by successively incrementing the initial ID#. On receipt of the returned update acknowledgement, the particular acknowledge logic switch which was previously specified in P#6 is set.

Boolean Variables are used to test the condition of two or more logic switches simultaneously. Two Boolean Variables are used in the Simulation.

(1) Boolean Variable 1 : MUP + $\overline{\text{PER}}$

This is tested to allow a request for processing only if it has a value '0'. This occurs when the permit comes in and sets the PER Switch of the L.F.M. and the L.F.M. is not in master updating state. Once a request is taken for processing, the MUP Switch is set. Next time the Boolean Variable 1 has a value '0' only when the permit comes in again. P#8 and P#9 contain the ID# of the Control Logic Switches PER and MUP respectively. The Boolean Variable addresses these Logic Switches indirectly through P#8 and P#9.

(2) Boolean Variable 2: LS$ACK#1 x LS$ACK#2 etc.

When an update acknowledgement is received, it sets a predetermined Acknowledgement Logic Switch. Only when all the Acknowledgements are received, this Boolean Variable returns a '1'. Once this occurs, the L.F.M. is allowed to exit the update session and send out the permit.

The specific use of variables is explained below:

Variable 1 : Constant

to specify the number of nodes in the simulation

Variable 2 : K3 + K3 * (P1 - K1)

to specify the MUP Control Logic Switch of a source node (see 2.4.1). Before setting the MUP logic switch at the beginning of an update session and resetting it at the end of an update session, this Variable is referenced and assigned to P#5. This variable is also used to enter the MUP Logic Switch ID# to P#9 of the Boolean field of the transaction parameter set.

Variable 3 : K3 * V1 + (V1 - K1) * P1

to specify the last Logic Switch ID# of the set of Acknowledgment Logic Switches of a source node. This variable is used to initialize P#6 on transferring out of the MUP state. P#6 is used to reference all the Acknowledgement Logic Switches. Resetting of the corresponding logic switch and decrementing of P#6 occurs alternatively.

Variable 4 : V1 - K1

to initialize the loop counter. P#7 is used as the loop counter.

Variable 5 : K2 + K3 * (P2 - K1)

to specify the PER Control Logic Switch of a receiving node (see 2.4.1). This variable is referenced in the receiving section of the program and assigned to P#5 prior to setting the PER logic switch.

Variable 6 : K1 + K3 * (P2 - K1)

to specify the SUP Control Logic Switch of a receiving node. This variable is referenced in the receiving

section of the program and assigned to P#5 prior to setting the SUP logic switch on receipt of an update begin. It is also assigned to P#5 prior to resetting the SUP logic switch on receipt of an update complete.

Variable 7 : K3 * V1 + K1 + (P1 - K1) * (V1 - K1)

to specify the first Logic Switch ID# of the set of Acknowledgement Logic Switches of a source node. This variable is used to enter Acknowledgement Logic Switch ID#'s into P#6 of an update complete before it was sent out. This variable is assigned to P#6 which in turn is successively incremented before a copy of the update begin is sent out.

Variable 8 : K2 + K3 * (P1 - K1)

to enter the PER Logic Switch ID# into P#8 of the Boolean Field of the transactions parameter set.

Variable 9 : K3 * V1 + (P1 - K1) * (V1 - K1) + P7

to specify the Acknowledgement Logic Switch ID#'s of a node. This variable in conjunction with variable 10 is used to enter Acknowledgement Logic Switch ID#'s into Boolean Field of the parameter set of a transaction (see 2.4.2).

Variable 10 : K8 + V1

to specify the total length of the parameter set. Starting from P#10 successive Acknowledgement Logic Switch ID#'s are entered into successive parameter locations. This is achieved as follows.

Variable 10 is assigned to P#4. The loop counter P#7 is initialized with the loop count variable 4. Variable 9 provides the ID# of the last logic switch of the Acknowledgement Logic Switch set. Variable 9 is assigned to the parameter location indirectly addressed by the contents of P#4. P#4 is then decremented and the transaction is looped around for another indirect assignment. The loop counter P#7 is also decremented. As a result, V9 specifies Acknowledgement Logic Switch ID#'s in descending order. When the loop counter P#7 becomes zero, the assignment of logic switch ID#'s to this section of the transaction's Boolean Field is complete.

2.4.4 Details of the Simulation Program (see Figure 2.4 and Appendix A)

The Sending Section:

Transactions (requests) are generated at individual generate blocks. The source ID# is assigned, and the transactions are transferred to a common section of the program. Even though the blocks are common, G.P.S.S. keeps track of transactions belonging to the same node by using the source ID# specified in P#1. Next, the Boolean Field is filled up with pertinent information as explained in the discussion involving the use of variables. The transactions are then queued up for processing. The transaction at the head of each queue is stopped unless the value of BV1 is zero (see 2.1.3.). Only when BV1 of a particular node has a value '0', a transaction from that node is permitted to proceed to commence updating. The transaction then departs the queue, and the MUP logic switch is set. After a .1 second delay, a copy is made, labelled as update begin, and

transferred to a section of the program (BTXT) which assigns individual destinations to the copy and sends it onto the link. To maintain generality of this section so that destination assignments can be made regardless which node the transaction belongs to, the following procedure is resorted to:

Step 1: The destination node ID# is specified as 1, i.e. P#2 is assigned a value of 1.

Step 2: The counter is initialized to the number of nodes in the network.

Step 3: A check is made to see if the destination specified is the same as the source (P#1 = P#2?). If so, step 4 is skipped.

Step 4: A copy is dispatched onto the link.

Step 5: The destination node ID# is incremented, i.e. P#2 is incremented.

Step 6: The counter is decremented. If it is not zero, then the program jumps to step 3.

Step 7: The transaction is terminated when all the update begins have been sent out.

After a fixed one second delay which represents the updating period, a copy of the transaction is made. It is labelled as update complete and transferred to a section of the program (ETXT). This section is very similar to the BTXT section except that Acknowledgement Switch ID#'s are entered into P#6 of the outgoing update complete. This is achieved as follows:

Step 1: The destination node ID# is specified as 1.

Step 2: Variable 7 is assigned to P#6. P#6 now contains the first Acknowledge Logic Switch ID# of the set of Acknowledgement Logic Switches.

Step 3: The counter is initialized to the number of nodes.

Step 4: A check is made to see if the destination specified is the same as the source. If it is so, step 5 and step 6 are skipped.

Step 5: A copy is dispatched onto the link.

Step 6: P#6 is incremented. The next Acknowledgement Logic Switch ID# is obtained in P#6.

Step 7: The destination node ID# is incremented, i.e. P#7 is incremented.

Step 8: The counter is decremented. If it is not zero, the program jumps to step 4.

Step 9: The transaction is terminated when all the update completes have been sent out.

Having sent out update completes, the resident transaction keeps testing the value of BV2 for a value of '1' at every simulation clock update (see Sec. 2.4.3.). BV2 becomes '1' when all update acknowledgements are received. When this happens, the MUP switch and the PER switch are reset. After this, all the Acknowledge Logic Switches are reset. This is done as follows:

Step 1: V3 is assigned to P#6 (Block 23)
P#6 now contains the node's last Acknowledgement Logic Switch number.

Step 2: Initialize Loop counter.

Step 3: Reset Logic Switch addressed by P#6.

Step 4: Decrement P#6.

Step 5: Decrement counter. Jump to step 3 is not zero.

Step 6: Terminate the transaction.

Receiving Section: When a transaction is received, it is identified in

the following order:

(1) Permit: P#3 = 1

A check is made to see if the local queue is empty. If the queue is empty, the ID# of the next node is entered into the destination field, P#2, and the PER logic switch ID# of the next node is entered into P#5. The permit is then sent out onto the link. Before a destination is specified, a check is made to see if the destination is N. In which case, the next destination specified should be 1. If the queue is not empty, the PER logic switch is set. The node retains the permit until the updating session is over. When the sending section resets the PER switch, the permit is allowed to leave. The next destination node ID# and its corresponding PER switch ID# are entered into the transaction field as before, and the permit is sent out.

At the start of the program, a single transaction is generated by a generate block. This transaction is labelled as a permit. After labelling the corresponding PER logic switch ID# to node 1 is done, this permit is dispatched to node #1 in the beginning of the simulation.

(2) Update Begin: (P#3 = 2)

If an incoming transaction is not a permit, it is checked to see if it is an update begin. If it is, then the SUP logic switch is set and the transaction is terminated.

(3) Update Complete: (P#3 = 3)

If an incoming transaction is identified as an update complete, the SUP logic switch is reset. The transaction is labelled as an update acknowledgement. The source and destination fields are switched around, and the transaction is sent out to the link.

(4) Update Acknowledgement: (P#3 = 4)

If the incoming transaction is none of the above, it must be

an update acknowledgement. The logic switch ID# in P#6 is set indirectly and the transaction is terminated. In the simulation, the communication is modeled by an advance block with a statistical delay.

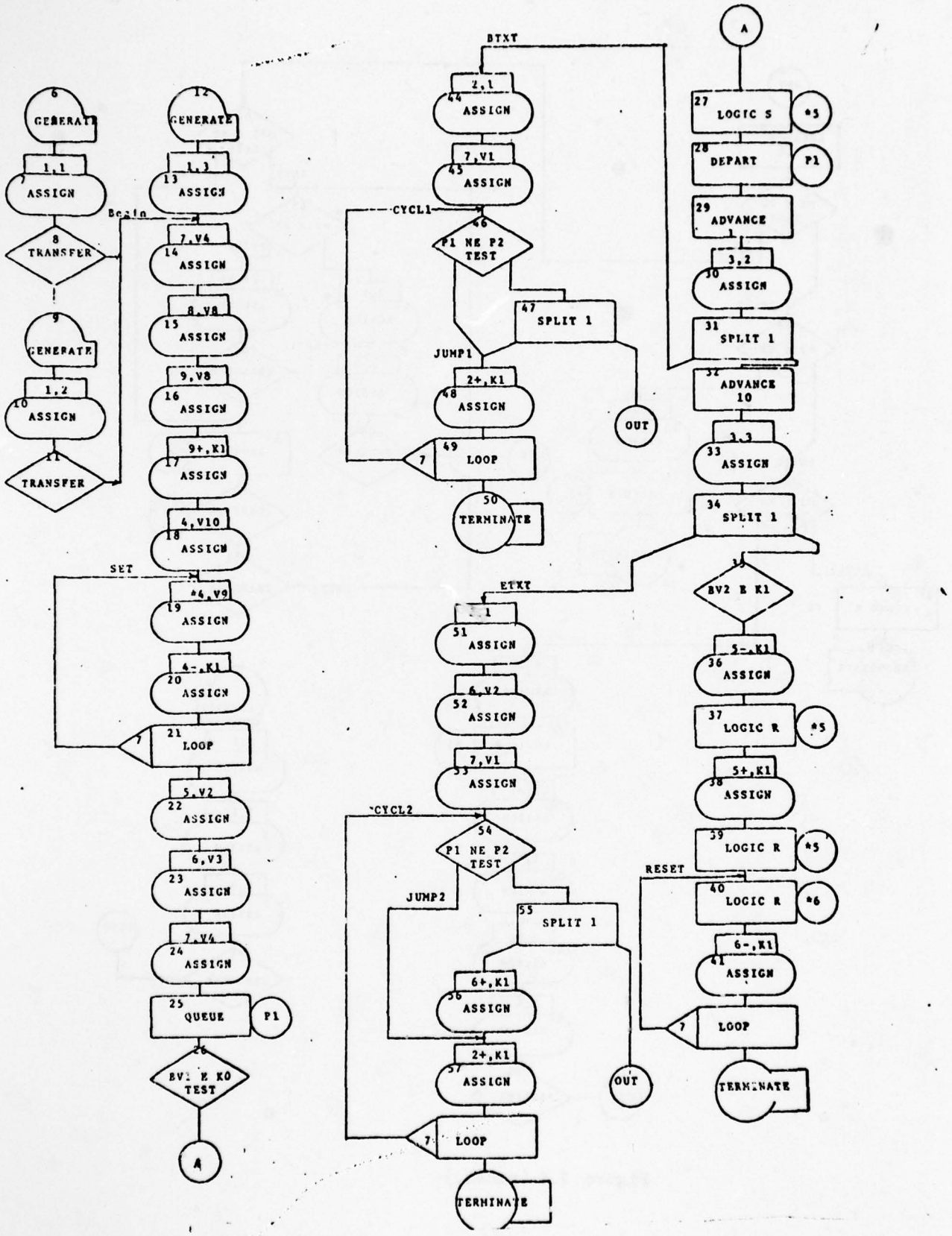


Figure 2.4 G.P.S.S. flowchart of the Hopping Permit Scheme

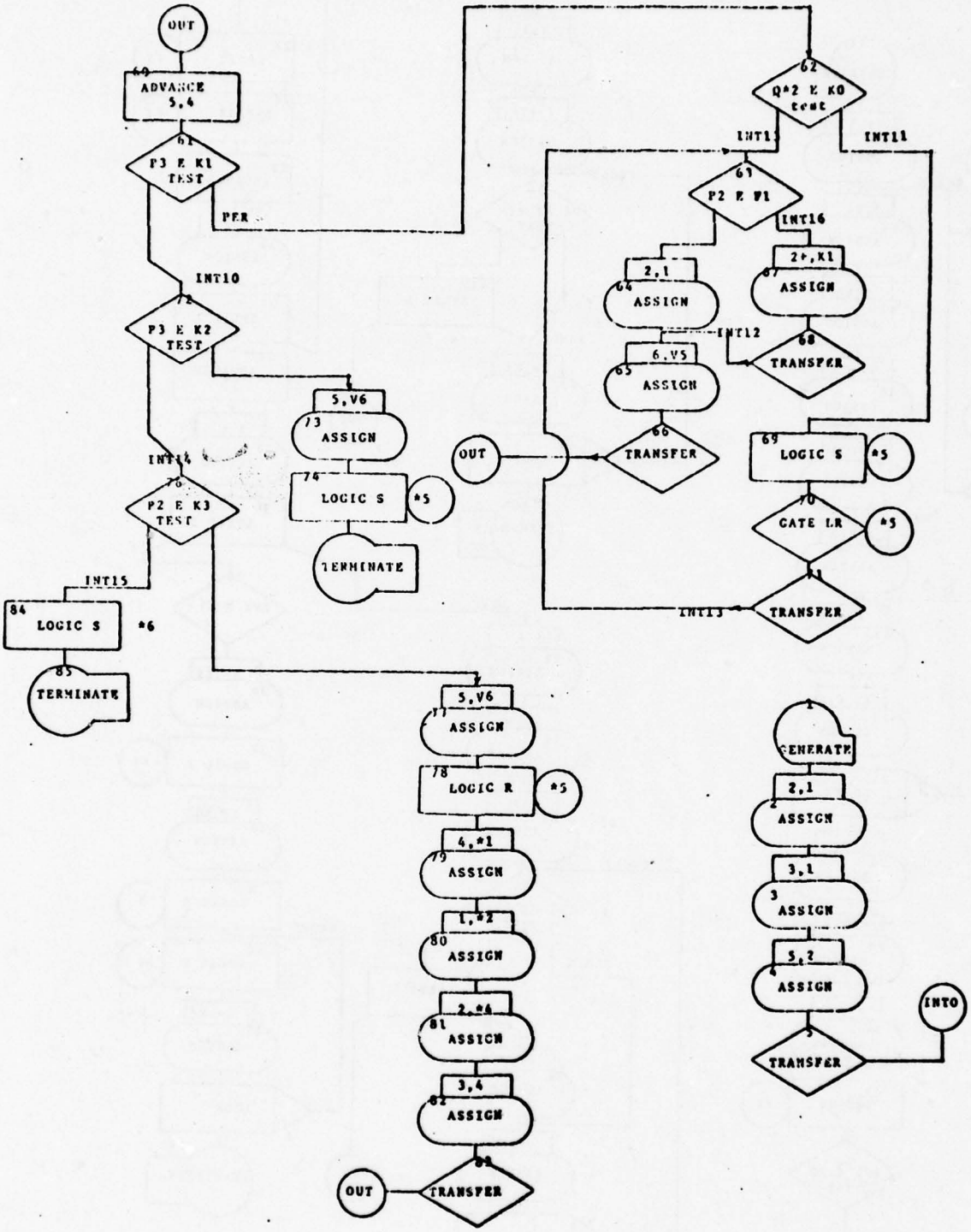


Figure 2.4 (contd.)

Chapter 3

NETWORK-WIDE SEMAPHORE SCHEME

In the Hopping Permit Scheme, a node must have possession of the pre-established permit before transferring into the updating phase. In contrast, a semaphore is established in the Network-Wide Semaphore Scheme by one of the nodes prior to its transition into the updating phase. The establishing of the semaphore is accomplished on the basis of request times after the synchronizing phase. In this chapter, the basic operation of this control scheme is presented. A detailed discussion of the use of G.P.S.S. entities and a description of the simulation program are also included.

3.1 Description of the Network-Wide Semaphore Scheme

In a network environment file requests might be generated at any node. It is necessary to control and grant these requests in an appropriate order. The situation is complicated by the fact that this global information about where a request is generated is not known to a local file manager. The distributed control scheme has to be able to achieve coordinating requests in an appropriate order without complete global information. In network-wide semaphore schemes, the local file manager engages in two phases of activities periodically. In the synchronizing phase, local file managers broadcast local requests to all the nodes. The protocal will work in such a way that only the local file manager with the earliest local request can obtain the file access right. In the following updating phase, all the other nodes will lock out file access and wait for updating information from the node with access right.

Essentially, the resolution to grant a file access request in the synchronizing -phase is by open bidding. Each local file manager has its

local request queue. It will broadcast the local request time to all other nodes. The local file manager with the earliest request will be completely acknowledged, and consequently obtain the file access right. R. H. Thomas and P. R. Johnson had proposed a time stamp approach to implement the RSEXEC capability of the ARPA network [10]. Though the details of implementation is different, their approach is very similar to the network-wide semaphore scheme studied here.

Requests for file access are queued up at each node on a first come first serve basis. In the synchronizing phase, the Local File Manager (L.F.M.) sends out the arrival time of the request at the head of the queue to all other participating nodes in messages. These messages are called 'requests'. A request $R_i(T_i,P_i)$ originating from L.F.M.$_i$ is said to be earlier than a request $R_j(T_j,P_j)$ originating from L.F.M.$_j$.

i.e. $R_i\ (T_i,P_i) < R_j\ (T_j,P_j)$ if

(i) $T_i < T_j$

or

(ii) $T_i = T_j$ and $P_i > P_j$

where:

T_i,T_j : are the times when requests were made at their respective nodes

P_i,P_j : are the pre-assigned source node priorities

There is a finite probability that two requests $R_i\ (T_i,P_i)$ and $R_j\ (T_j,P_j)$ might have the same generation times ($T_i = T_j$). Such a tie is broken by taking account of the arbitrarily assigned source priority (P_i,P_j).

During the synchronizing phase, L.F.M.'s broadcast the request at the head of the queue. Having sent out a request, if the L.F.M. receives a foreign request which is earlier than the local request sent, the foreign request is acknowledged by a request acknowledgement message. The local request is returned to the head of the queue for consideration at the next synchronizing phase. If the foreign request is later than the local request sent, it is ignored. If at a synchronizing phase an L.F.M. possesses no local request, the first incoming foreign request will be acknowledged. Having acknowledged a foreign request the L.F.M. goes into an intermediate phase wherein it is inhibited from sending out any local requests. In this phase the L.F.M. is waiting for updating information. The updating information is signaled by an update begin message which initiates the transition from the intermediate phase into the updating phase. Only the L.F.M. with the earliest request will be completely acknowledged and granted permission to proceed with the file access.

Updating information is proceeded by an update begin message and followed by an update complete message. The interval between receiving an update begin and an update complete is dedicated to the updating of the resident file by the L.F.M. During this interval the local users cannot access the file. On receipt of an update complete, the receiving L.F.M. acknowledges the receipt of the updating information by dispatching an updating acknowledgement. Then, it re-enters the synchronizing phase. The L.F.M. that originates the updating has to wait until it receives update acknowledgements from all the nodes before it can go into the next synchronizing phase.

3.2 State Transition Diagram

The state diagram of the local file manager (L.F.M.) is shown in figure 3.1 to accomodate every possible situation in the open bidding procedure. The following is a description of the operation of LFMs.

Possible states of the local file manager (L.F.M.) are:

(IDLE): The Idle State

In this state the local request queue is empty and the L.F.M. is not aware of any foreign request for file updating after the last updating phase.

(MREQ): The Master Request State

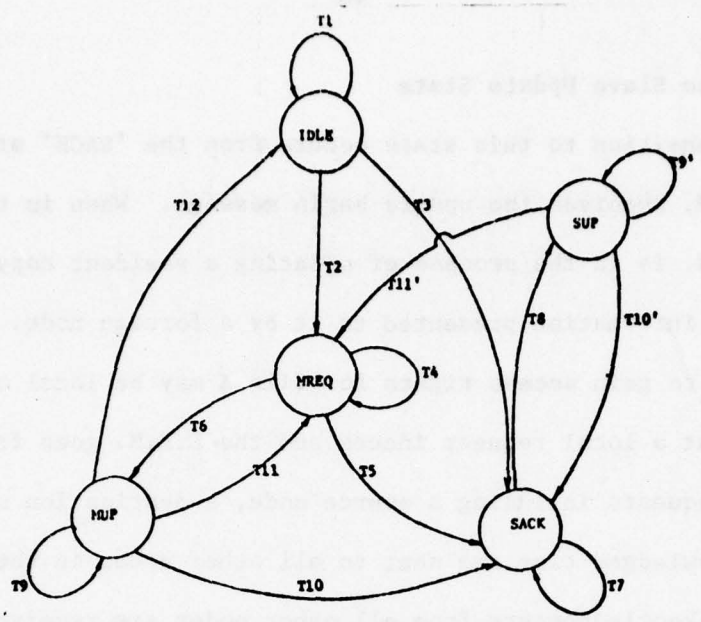
The L.F.M. would be in this state if it has sent out a request and is unaware of earlier requests (if any) in the network. In this state, the L.F.M. is waiting for request acknowledgements from the foreign nodes. If instead of receiving a request acknowledgement the L.F.M. receives an earlier request, it relinquishes its interest, requeues its pending old request and sends out a request acknowledgement to the source of the earlier new request.

(MUP): The Master Update State

If the L.F.M. receives request acknowledgements from all the other L.F.M.'s in the network, it proceeds to the updating phase wherein the local request is granted file access rights.

(SACK): The Slave Acknowledgement State

A transition to this state occurs when the L.F.M. acknowledges a foreign request. The L.F.M. is then ready to accept incoming updating information. When the L.F.M. is in this state, in the event of an earlier request arriving, the incoming request is duly acknowledged.



T1 : No requests, foreign or local to be serviced

T2 : The L.F.M. makes a request and is not aware of any foreign requests

T3 : A foreign request is received. There being no local request it is immediately acknowledged.

T4 : A foreign request is received with L.F.M. is awaiting request acknowledgements. The local request is earlier than the foreign request and no action is taken.

T5 : A foreign request is received which earlier than the local request and hence is acknowledged.

T6 : All request acknowledgements have been received and the L.F.M. proceeds to the updating phase of modifying the local file and sending updating information to all other participating nodes.

T7 : A foreign request is received. If it is earlier than the last acknowledged request, the incoming request is acknowledged, else ignored.

T8 : Updating information is received and the L.F.M. proceeds into the updating phase and updates the resident copy of the file with foreign updating information.

T9, T9'* : A request that arrives is saved if it is to be acknowledged on exiting either of these states.

T10, T10'* : The request to be acknowledged is a foreign request.

T11, T11'* : The request to be acknowledged is a local request.

T12, T12'* : No requests came in while the L.F.M. was in the updating phase.

* Transition out of the MUP state occurs when the L.F.M. receives update acknowledgements from all the other nodes. Transition out of the SUP state occurs when an update complete message is received

Figure 3.1 State transition diagram of the Network-Wide Semaphore Scheme.

(SUP): The Slave Update State

A transition to this state occurs from the 'SACK' state when the L.F.M. receives the update begin message. When in this state, the L.F.M. is in the process of updating a resident copy of the file with information presented to it by a foreign node.

A request to gain access rights for file X may be local or foreign. Assume that a local request incurs and the L.F.M. goes from IDLE to MREQ state. Requests including a source node, a destination node, time, and last acknowledged time are sent to all other nodes in the network. When request acknowledgements from all other nodes are received, L.F.M. has gained the access right and will go into MUPD state and start the updating session. When L.F.M. receives all updating acknowledgements, it will go back to either IDLE state or SACK state. A received foreign request generated earlier can move L.F.M. from MREQ state where L.F.M. is waiting for request acknowledgements to SACK state and restore the local request back to local queue. In SACK state, L.F.M. will respond by sending out request acknowledgements to any foreign request that has an earlier time than the last acknowledged time. In SACK state or SUPD state, L.F.M. can recognize the legitimate requests by consulting the last acknowledged time and keep only the earliest one. Legitimate request can be received by L.F.M. before or during updating sessions of a foreign node because of the stochastic network delay. A received update begin message will move L.F.M. to SUPD state.

In both the Master Update State and the Slave Update State, the L.F.M.'s may receive a request from nodes that have finished their update sessions early. There is a distinct likelihood that this can occur due to the randomness of communication delay. If the request comes in

when the L.F.M. is in any of these states, a comparison is made with the request times of the earliest of all requests (if any) already received. The source and time of the earlier of the two requests is saved to be acknowledged in the future when the L.F.M. exits from these states. If the local queue is not empty, a comparison is made between the request times of the request at the head of the queue and that of the incoming request. The earlier request time and source is saved. If the L.F.M. is in the Master Update State, the above situation can occur after it has sent out an update complete message, i.e., it has completed transmission of updating information. It could be waiting for update acknowledgements and may receive a request from an L.F.M. that has received the update complete message and sent out the update acknowledgement. If an L.F.M. is in the Slave Update State, the above situation may happen when another node has received its update complete message early and hence broadcasted its requests.

Transition out of a Master Update State and the Slave Update State can occur at the end of the update session into:

(1) The Idle State: No local request needs to be served and there were no foreign requests received during the updating phase.

(2) The Master Request State: The local request is the earliest of all requests that the L.F.M. is aware of.

(3) The Slave Acknowledgement State: A foreign request is the earliest request known to the L.F.M. at the end of the updating phase. The foreign request is acknowledged when the L.F.M. is transferred into the SACK state.

Figure 3.1 is the state transition diagram of an L.F.M. under the distributed control of the Network-wide Sempahore Scheme.

It may be noted that:

(1) The synchronizing phase comprised of the IDLE and the MREQ, and other SACK states.

(2) The updating phase comprises of the MUP and SUP states.

There are two weak points in this network-wide semaphore scheme. The first weakness is the requirement of synchronized precise clocks at every computer site. Keeping accurate time information at each site of a large network is difficult [20]. The second weakness is the network traffic overhead associated with this control scheme. Before all file managers go into critical states, a lot of ineffective requests are allowed to enter into the network. It is possible that a request from computer B, to access file x, goes out to link, while an earlier request to access the same file from computer A is already in transit to B. If this information were known to B, the request could be prevented from going into the link.

3.3 Simulation Program

The simulation program may be broadly partitioned into the sending section and the receiving section.

(1) The Sending Section:

In the sending section, transactions are generated and queued up for processing in their respective queues. In the synchronizing phase, a copy of the transaction is labelled as a request and dispatched to all the other nodes. The transaction proceeds for updating only if request acknowledgements are received from all the nodes. This becomes known to the sending section if all the acknowledgement switches in the receiving section

are set. However, if an earlier request is received, the local transaction is requeued at the head of the local queue.

If the transaction is allowed to proceed for updating, copies of it are labelled as update begins and is dispatched to the other nodes. The acknowledgement logic switches are then reset. After a one second delay copies of the local transaction are labelled as update complete and dispatched. On receipt of all update acknowledgements the transaction is allowed to proceed. It resets all the acknowledgement logic switches. The L.F.M. at this juncture leaves the update phase.

(2) The Receiving Section:

an incoming transaction is identified in the following order:

Update Begin: On receipt of this, the request acknowledgement logic switches will be reset, and the L.F.M. is transferred to the SUP state.

Request: An incoming request undergoes the most extensive processing. This is explained in the detailed program description. In short words, an earlier request is acknowledged, and a later request is ignored.

Request Acknowledgement: When a request acknowledgement comes in, it sets a predetermined logic switch. When all request acknowledgement logic switches are set, the local file manager is allowed to begin updating.

Update Complete: The L.F.M. transfers out of the 'SUP' state and begins the synchronizing phase.

Update

Acknowledgement: When an update acknowledgement comes in, it sets a predetermined logic switch. When all update acknowledgement logic switches are set, the L.F.M. transfers out of the updating phase.

3.3.1 Use of G.P.S.S. Logic Switches

Before a detailed description of the program is presented, the use of G.P.S.S. entities is discussed.

Logic switches are used in very much the same way as in the Hopping Permit Scheme. The Control Logic Switch ID#'s are specified as:

MREQ : $(i - 1) \times 4 + 1$

MUP : $(i - 1) \times 4 + 2$ $\quad i = 1,2.....N$

SACK : $(i - 1) \times 4 + 3$

SUP : $(i - 1) \times 4 + 4$

where: i is the ID# of the node

N is the total number of nodes in the simulation.

A total of 4 x N control logic switches are defined for control in the simulation.

Acknowledgement Logic Switch ID#'s are specified in much the same way as in the Hopping Permit Scheme. In this case, each node has four control logic switches, hence:

$$L\$ACK_{jtoi} = \begin{cases} 4 \times N + (i - 1) \times (N - 1) + j & \text{for } j < i \leq N \\ 4 \times N + (i - 1) \times (N - 1) + j - 1 & \text{for } j > i \text{ and } 1 \leq i < N \end{cases}$$

The number of acknowledgement logic switches defined in a simulation is

the same as for the Hopping Permit Scheme, which is N x (N - 1). Therefore, N x (N + 3) is the total number of Logic Switches defined in the simulation.

3.3.2 Use of G.P.S.S. Savevalues

Savevalues are used to store request times and source ID#'s. Four Savevalues per node are used and they are numbered the same way as the State control logic switches:

Savevalue # 1	:	(i - 1) x 4 + 1	
Savevalue # 2	:	(i - 1) x 4 + 2	i = 1,2.....N
Savevalue # 3	:	(i - 1) x 4 + 3	
Savevalue # 4	:	(i - 1) x 4 + 4	

The following information is stored in the Savevalues:

Savevalue # 1 : SLACK

The time of the request responsible for the last updating session is stored into this savevalue. In other words, this is the time of the last request that all the L.F.M.'s are acknowledged. SLACK is updated with every incoming update begin message. All request messages between update begin and end messages are checked for the condition $SLACK_j = P\$LACK_i$. Where, $P\$LACK_i$ of the request is assigned to $SLACK_i$ by the foreign $node_i$ before it was sent to $node_j$. This serves as a safeguard against invalid requests being considered. R.H. Thomas and P.R. Johnson also encounter similar problems like this in their work [10].

Savevalue # 2 : REQT

The arrival time of the request at the head of the local queue is stored into the Savevalue # 2.

Savevalue # 3 : OREQT

The earliest foreign request's time or the last acknowledged request's time is stored in this savevalue. If an incoming request finds that the receiving L.F.M.'s OREQT = SLACK, this would mean that either L.F.M. has no local request or the incoming request is the first request coming in since the commencement of the synchronizing phase.

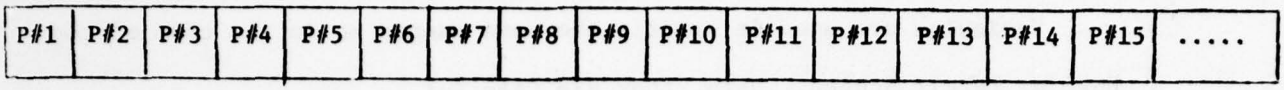
Savevalue # 4 : ASID

The source ID# corresponding to the request time stored in OREQT is stored in this savevalue.

In the event of foreign request arriving during an MUP or SUP session of an L.F.M.'s, the requests are retained until the transition out of these states occurs. In transferring out of these states, the earliest of all foreign request's times is stored in OREQT and its corresponding Source ID# is stored in ASID. If there is a local request, OREQT is compared with REQT to determine the earlier request of the two. Otherwise, the foreign request is acknowledged. If OREQT is less than REQT, a request acknowledgement is sent, or else the L.F.M. proceeds to send out its own request.

3.3.3 Parameter Set of a Transaction

All the transactions in the Network-Wide Semaphore Scheme have the following parameter set:



Information field | Control field | Boolean field

Parameter #		Contents
P#1	:	Request Time
P#2	:	Source Node ID#
P#3	:	Destination Node ID #
P#4	:	P$LACK (see 3.3.2)
P#5	:	Type : 1. Request 2. Request Acknowledge 3. Update Begin 4. Update Complete 5. Update Acknowledge
P#6	:	Temporary Data
P#7	:	Loop Counter, and Temporary Data
P#8	:	A Control Logic Switch ID#
P#9	:	An Acknowledgement Logic Switch ID#
P#10	:	MREQ Switch ID# — Contents referred to by
P#11	:	MUP Switch ID# — Boolean Variable 1
P#12	:	SACK Switch ID# — Contents of P#12 is also referred to
P#13	:	SUP Switch ID# — by Boolean Variable 3
P#14	:	ACK Switches ID#'s — Referred to by Boolean
P#15	:	" " " — Variable 2

.
.
.
.

A total of (12 + N) parameters need to be specified for a simulation with N nodes.

3.3.4 Use of G.P.S.S. Variables and Boolean Variables

Variables are used in exactly the same manner as in the case of the Hopping Permit Scheme. Here, they are used to:

(1) Enter pertinent information into the Boolean Field of the parameter set (see 2.4.3).

(2) Enter information into the Control Field of parameter set (see 2.2.3).

P#8 of a message is assigned a Control Logic Switch ID# using the appropriate variable prior to setting or resetting a particular Control Logic Switch.

P#9 of an update complete and request is assigned an Acknowledge Logic Switch ID# using the appropriate variable prior to dispatching (see Blocks 67 - 75 in figure 3.2)

The Boolean Variables in this simulation program perform the following functions:

(1) Boolean Variable 1 : MREQ + MUP + SUP + SACK.

A request is allowed for processing only if this Boolean Variable has a value of '0'. This occurs when the L.F.M. is in the IDLE State. In the simulation, a transition from the SUP or MUP State to the MREQ or SACK State is achieved by transmitting through the IDLE State. P#10 - P#13 of a transaction contains the ID#'s of the control logic switches which are referred to by the Boolean Variable indirectly addressed by these parameters.

(2) Boolean Variable 2 : LS$ACK1 x LS$ACK2 ..., etc. This Boolean Variable is used to check if all request acknowledgements and update acknowledgements are received when the L.F.M. is expecting them. This Boolean Variable returns a '1' if,

(a) All request acknowledgements are received. In this case, the L.F.M. moves itself from the MREQ to the MUP state.

(b) All update acknowledgements are received. In this case, the L.F.M. is allowed to exit the MUP state.

(3) Boolean Variable 3 : SUP + BV2

In the synchronizing phase when the L.F.M. sends out a request, it may either get acknowledgements from all the foreign L.F.M.'s, or it may receive an earlier foreign request. This Boolean Variable checks for either of these events when an L.F.M. is in the MREQ state. The local request which is made to wait is returned back to the head of the local queue for future consideration if an earlier request is received. Otherwise, the request is totally acknowledged, and the L.F.M. is allowed to proceed to the MUP state.

The specific use of variables is explained below. Variables 1 - 9 are referred to by transactions in the sending section. Variables 10 - 14 are referred to by transactions in the receiving section.

Variable 1 : Constant
to specify the number of nodes in the simulation.

Variable 2 : K1 + K4 * (P2 - K1)
to specify the MREQ Control Logic Switch ID# of a source node (see 3.3.1). Also used to specify Savevalue # 1 (see 3.2.2).

Variable 3 : K2 + K4 * (P2 - K1)
to specify the MUP Control Logic Switch ID# of a source node and Savevalue #2.

Variable 4 : K3 + K4 * (P2 - K1)
to specify the SACK Control Logic Switch ID# of a

node and Savevalue #3.

Variable 5 : K4 + K4 * (P2 - K1)

to specify the SUP Control Logic Switch ID# of a source node as well as Savevalue #4.

Variable 6 : V1 - K1

to initialize the loop counter. P#7 is used as the loop counter.

Variable 7 : K12 + V1

to specify the total length of the parameter set. Starting from P#14 successive Acknowledgement Logic Switch ID#'s are entered into successive parameter locations (see 3.3.2).

Variable 8 : K4 * V1 + (P2 - K1) * (V1 - K1) + P7

to specity Acknowledgement Logic Switch ID#'s of a source node. This variable in conjunction with V7, is used to enter Acknowledgement Logic ID#'s in the Boolean Field of the parameter set. This is achieved as follows. V7 is assigned to P#6. The loop counter is initialized using V6. With the loop counter initialized, V8 provides the ID# of the last logic switch from the Acknowledgement Logic Switches set. V8 is assigned indirectly to the parameter location specified by the contents of P#6. P#6 is then decremented and the transaction is looped around for another indirect assignment. The loop counter P#7 is decremented. As result, V8 specifies Acknowledgement Logic Switch ID#'s in descending order. When the loop counter becomes zero, the assignment of logic switch ID#'s

to this section of the transaction's Boolean Field is completed.

Variable 9 : K4 * V1 + (P2 - K1) * (V1 - K1) + K1

to sepcify the first logic switch ID# of the set of Acknowledgement Logic Switches of the source node. This variable is used to:

(1) Initialize P#9. P#9 is used to reset all the Acknowledgement Switches. It is done by resetting the corresponding logic switch and incrementing P#9 alternatively until all the Acknowledgement Logic Switches are reset. These are done at the beginning and end of an update session.

(2) Initialize P#9 which is successively incremented before a copy of the request or the update complete is sent out.

Variable 10 : K1 + K4 * (P3 - K1)

to specify the MREQ Control Logic Switch of a destination node. Also used to specify Savevalue #1.

Variable 11 : K4 + K4 * (P3 - K1)

to specify the SUP Control Logic Switch ID# of a destination node and Savevalue #4.

Variable 12 : K4 * V1 + (P3 - K1) * (V1 - K1) + K1

to specify the first logic switch ID# of the set of Acknowledgement Logic Switches of a destination node. When an update begin is received, P#9 is initialized with V12. P#9 is used to reset all the Acknowledgement Switches as explained earlier.

Variable 13 : K2 + K4 * (P3 - K1)

to specify the MUP Control Logic Switch ID# of a node and Savevalue #2.

Variable 14 : K3 + K4 * (P3 - K1)

to specify the SACK Control Logic Switch ID# of a node and Savevalue #3.

3.3.5 Details of the Simulation Program

The Sending Section: (see Figure 8 or Appendix B)

Transactions (requests) are generated at individual generating blocks. The source ID# is assigned, and the transactions are transferred to a common section of the program. The time the request was generated is entered into P#1. The Boolean Field is filled with pertinent information as explained in the discussion involving the use of variables. The transactions are then queued up for processing. The transaction at the head of each queue is stopped unless the value of BV1 in zero (see 3.3.4.). Only when BV1 of a particular node has a value of '0', a transaction from that node is permitted to proceed. When this happens, the MREQ logic switch is set, and P$LACK (P#4) is assigned to P$SLACK (see 3.3.2.). REQT (Savevalue #2) is assigned to the request time. The transaction is labelled as a request, and a copy is transferred to a dispatching section of the program. The dispatching section assigns individual destinations, enters Acknowledge Switch ID#'s into P#9 of the transactions, and sends out the required number of messages. The following procedure accomplishes the dispatching function:

Step 1 : The destination node ID# is specified as 1, i.e. P#3 is assigned a value of 1.

Step 2 : The counter is initialized by setting P#7 to the number of nodes in the simulation.

Step 3 : V9 is assigned to P#9. P#9 will now contain the first Acknowledge Logic Switch ID# of the node.

Step 4 : A check is made to see if the destination specified is the same as the source (P#2 = P#3?). If so, step 5 and step 6 are skipped.

Step 5 : A copy is dispatched onto the link.

Step 6 : P#4 is incremented. The next Acknowledgement Logic Switch ID# is obtained.

Step 7 : The destination node ID# is incremented, i.e. P#3 is incremented.

Step 8 : The counter is decremented. If not zero, the program jumps to step 4.

Step 9 : The transaction is terminated when all the messages have been sent out.

This very same set of blocks is used to dispatch update completes.

Once the requests are sent, the resident transaction tests BV3 for a one. (see 3.3.4.). BV3 gets a value one if either an earlier request is received by this node or all request acknowledgements are received. When the transaction proceeds, the value of SACK switch is tested. If it is a one, this indicates that an earlier request was received during the synchronizing phase. The local transaction is then returned to the head of the queue with priority 1. In the case when all acknowledgements are received, the transaction is allowed to proceed into the updating phase. The transaction departs from the queue and transfers the node from the MREQ state to the MUP state. All the Acknowledgement Logic Switches are reset.

Savevalue #1 and #3 are assigned to the request time. P$LACK is

also assigned to the request time. Savevalue #3 is assigned the request time for reasons discussed in section 3.3.2. After a .1 second time delay, the transaction is labelled as update begin, and a copy is made and transferred to the update begin dispatch section. This section sends out update begin messages to all the nodes in the same way requests and update ends are dispatched, except that in this case no logic switch ID#'s are entered into the parameter field.

After a fixed one second delay which represents the updating time, the transaction is labelled as update complete, and a copy is transferred to the update complete/request dispatch section. Now the local transaction being processed has to wait for update acknowledgements from all the nodes to come in. When all the update acknowledgements come in, BV2 gets a one value (see 3.2.4). The transaction then proceeds to remove the L.F.M. from the MUP state, and all the Acknowledgement Logic Switches are reset.

The Receiving Section: When a transaction is received it is identified in the following order:

(1) Update Begin: (P#5 = 3)

On receipt of an update begin, the SUP logic switch is set and the SACK logic switch is reset. The local SLACK is updated by assigning Savevalue #1 to P#1. All the other message types are checked for P$LACK = SLACK. If there is an inequality, the incoming transaction is ignored.

(2) Request: (P#5 = 1)

An incoming request is allowed to proceed only if the MUP logic switch has a zero value. Next, it is checked to see if the MREQ logic switch has a value of one. If MREQ logic switch has a value of one, the incoming request's time is compared with the local request's time.

If the incoming request is an earlier request, its request time and source ID# is saved, the SACK logic switch is set, and the MREQ logic switch is reset. The transaction is labelled as a request acknowledgement, and the source and destination fields are switched around. Then, the transaction is sent out to the link. If the MREQ logic switch is zero, either the SACK logic switch or the SUP logic switch may have a value of one or all the logic switches may have a value of zero. A check is made to see if OREQT = SLACK (see 3.2.3.). If they are equal, one of the two conditions may hold:

(a) All the logic switches have values of zero (IDLE State). The incoming request is the first one to come in since the last update phase.

(b) The SUP logic switch has a value of one. A foreign node has finished its update session early and the incoming request is the first request of the current synchronizing phase. The local node will commence the current synchronizing phase on receiving an update complete message.

For both the above cases OREQT (Savevalue #3) is assigned to the incoming request's time, and the source ID# is saved for the purpose of comparing requests that may come in during the current synchronizing phase. If the SUP logic Switch has a value of one, this request is held back. Otherwise, the SACK logic switch is set, and the transaction is labelled as a request acknowledgement. The source and destination fields are switched around, and the transaction is sent out to link.

If OREQT ≠ SLACK, this would mean that the node is aware of a request from the current synchronizing phase. A comparison is made between the earliest request that the node is aware of and the incoming request. If the incoming request is earlier, the incoming request's time is

assigned to OREQT, and the source node ID# is saved. If the SUP logic switch has a value of one, the request is held back. Otherwise, it means that the SACK logic switch has a value of one. The request is labelled as a request acknowledgement. The source and destination fields are switched around, and the message is sent out to link.

If the SUP logic has a value of one and there are requests that have been held up at the end of the updating phase, only the request whose request time corresponds to the one that has been saved is allowed to proceed. If a local request exists, a comparison of request times is made. If the foreign request is earlier, the MREQ logic switch is reset, the SACK logic switch is set, and the foreign request is acknowledged. If no local request exists at the end of the foreign update phase and there is a held over request, the SACK logic switch is set and the request (earliest of all requests the node is aware of) is acknowledged.

Request Acknowledgement: (P#5 = 2)

When a request acknowledgement comes in, it sets a predetermined Acknowledgement Logic Switch. When all the Acknowledgement Logic Switches are set, BV2 and BV3 will have a value of one.

Update Complete: (P#5 = 4)

When an update complete comes in, the SUP logic switch is reset. The message is labelled as update acknowledge, and the source and destination fields are interchanged. Then, the message is sent out to the link.

Update Acknowledge: (P#5 = 5)

When an update acknowledgement is received, it sets a predetermined Acknowledgement logic switch. When all these switches are set, BV2 will have a value of one.

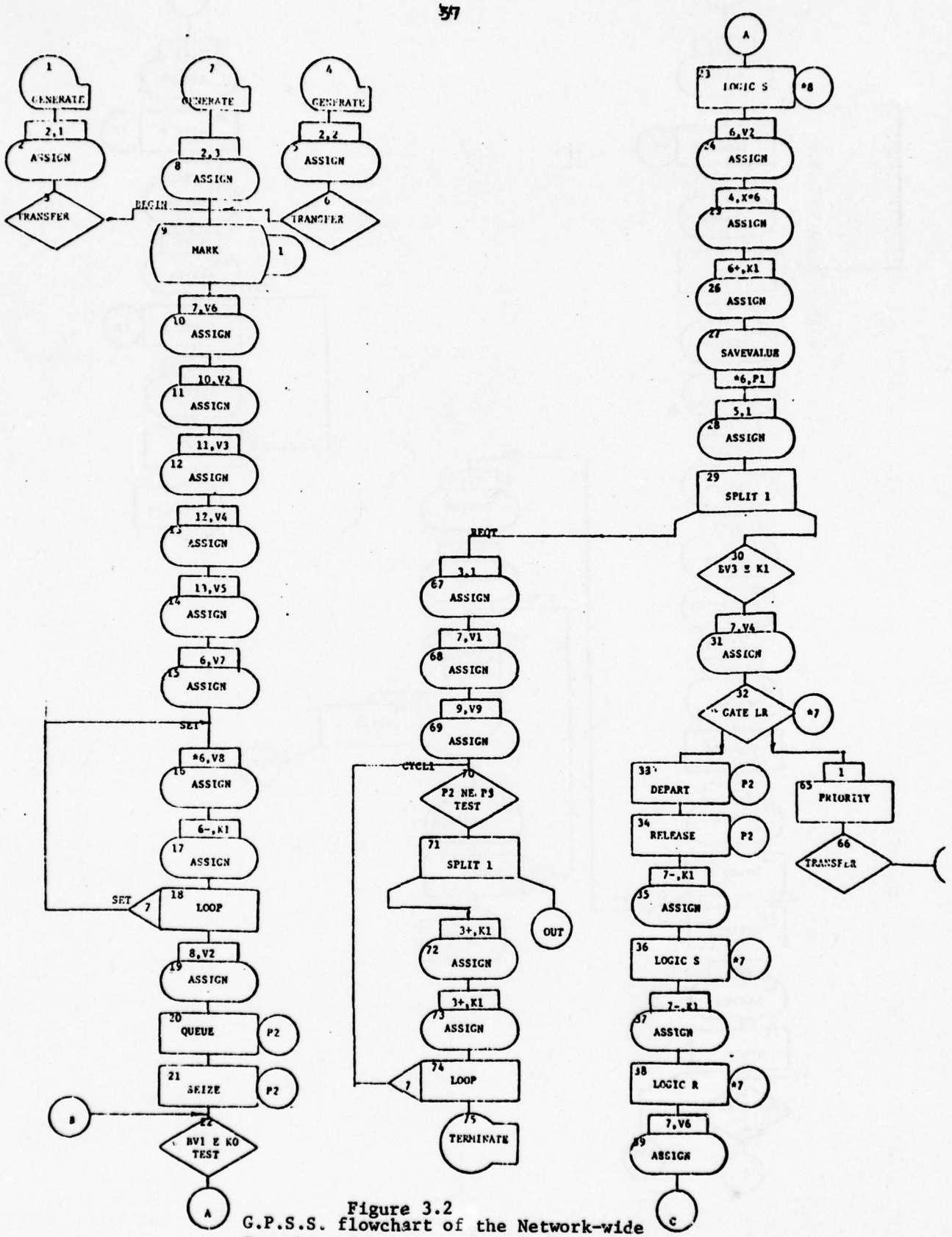


Figure 3.2
G.P.S.S. flowchart of the Network-wide Semaphore Scheme

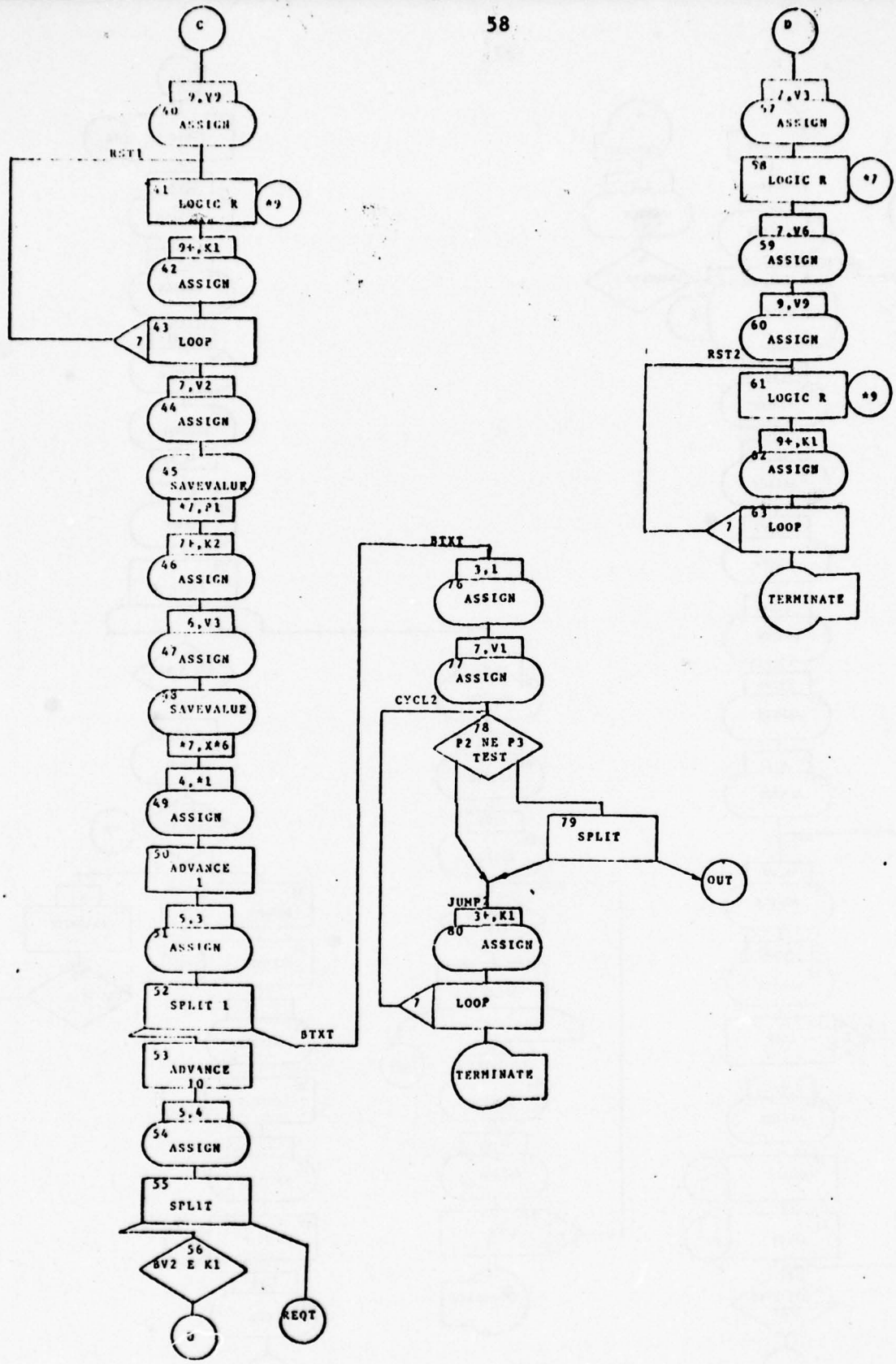


Figure 3.2 (contd.)

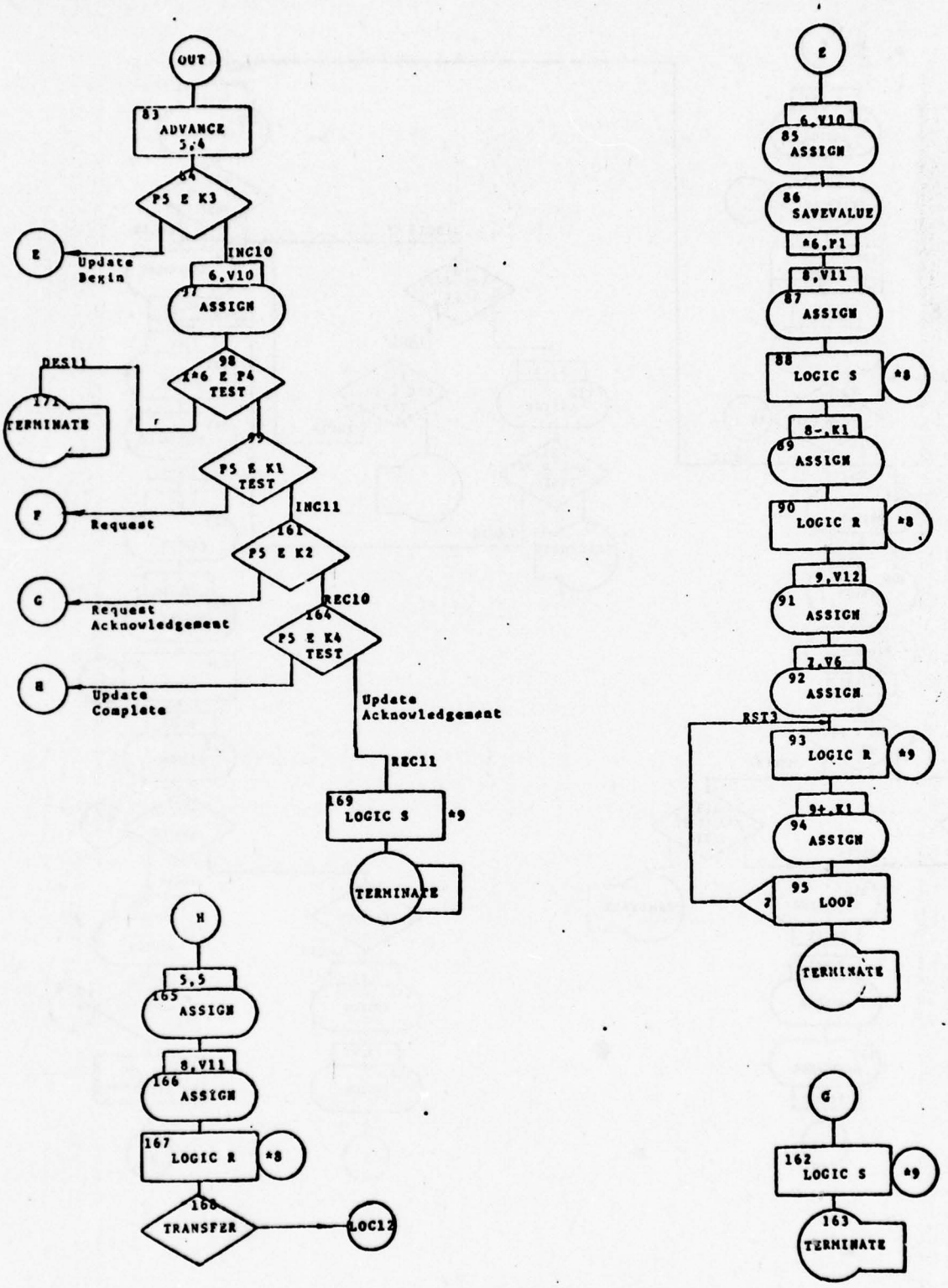


Figure 3.2 (contd.)

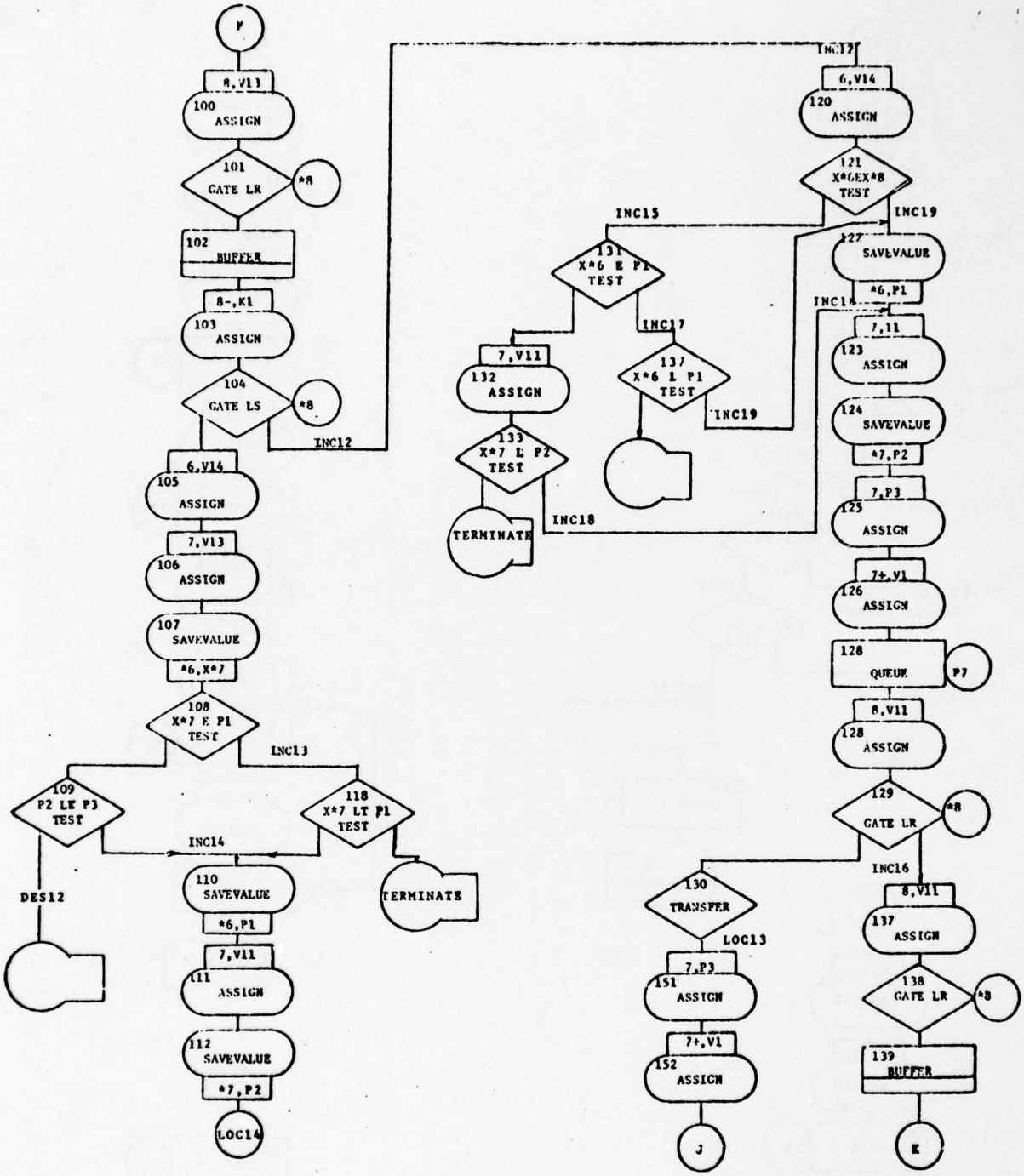


Figure 3.2 (contd.)

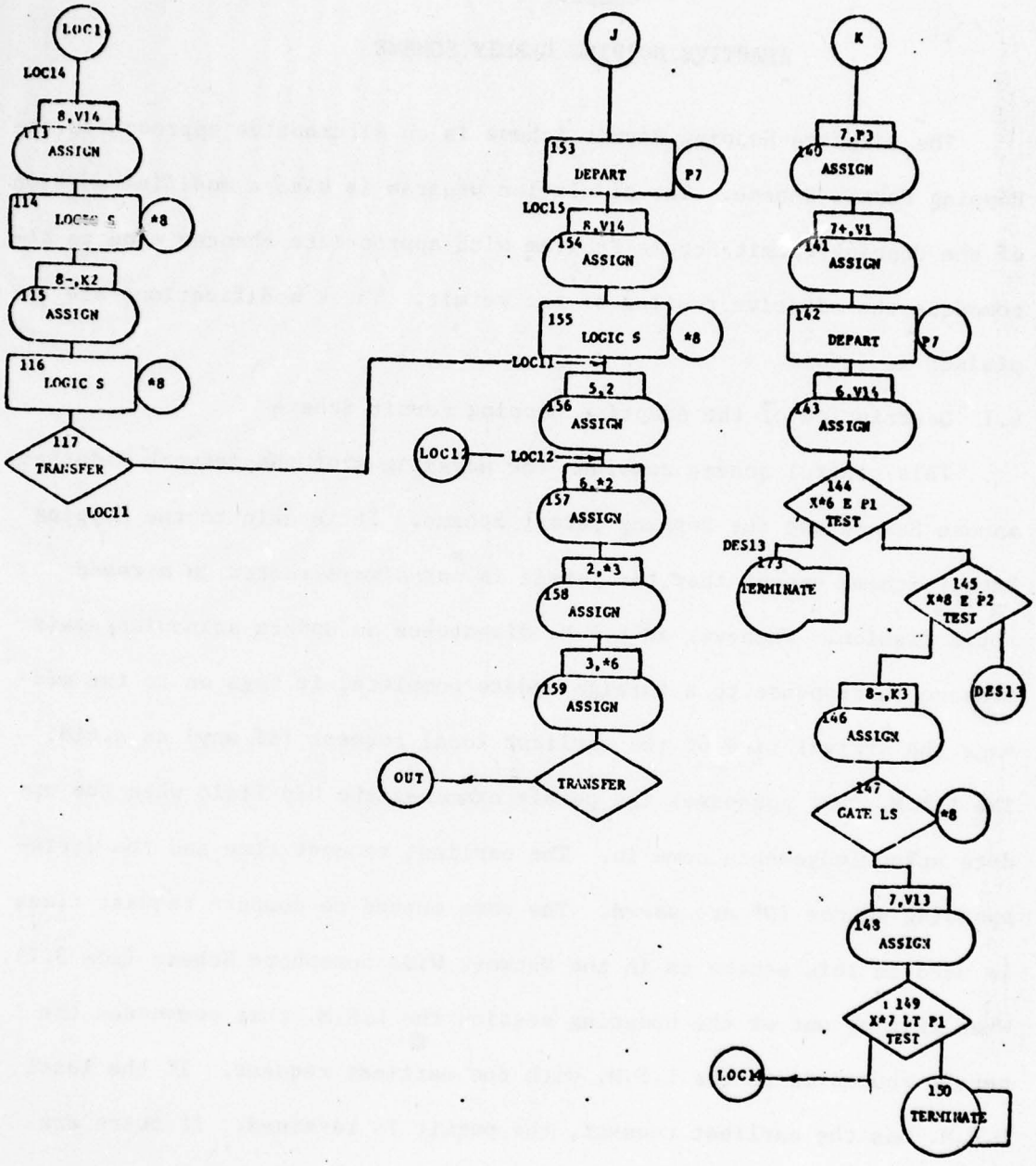


Figure 3.2 (contd.)

Chapter 4

ADAPTIVE HOPPING PERMIT SCHEME

The Adaptive Hopping Permit Scheme is an alternative approach to the Hopping Permit Scheme. The simulation program is also a modified version of the Hopping Permit Scheme Program with appropriate changes made to accomodate the adaptive routing of the permit. These modifications are explained in detail.

4.1 Description of the Adaptive Hopping Permit Scheme

This control scheme combines the advantages of the Network Wide Semaphore Scheme and the Hopping Permit Scheme. It is akin to the Hopping Permit Scheme except that the permit is not always routed in a round robin fashion. Whenever an L.F.M. dispatches an update acknowledgement message in response to a foreign update complete, it tags on to the message the arrival time of the earliest local request (if any) as a bid. The L.F.M. that possesses the permit examines the bid field when the update acknowledgements come in. The earliest request time and the corresponding source ID# are saved. The same method to compare request times is used in this scheme as in the Network Wide Semaphore Scheme (see 3.1). When exiting out of the updating session the L.F.M. that possesses the permit routes it to the L.F.M. with the earliest request. If the local L.F.M. has the earliest request, the permit is retained. If there are no requests in the network i.e. if all the foreign nodes respond with negative bids and the local queue is empty at the time when the last update acknowledgement comes in, the permit is passed down stream as in the Hopping Permit Scheme.

4.2 State Transition Diagram

The States in the transition diagram are identical to that of the

Hopping Permit Scheme, however, there is one more transition possible. An L.F.M. can continue to be in the MUP state if at the end of an update session it realizes that its own request is the earliest request. In such an event the permit is retained and the request at the head of the local quieu is serviced as shown in Figure 4.1.

4.3 Simulation Program

4.3.1 Use of G.P.S.S. Entities

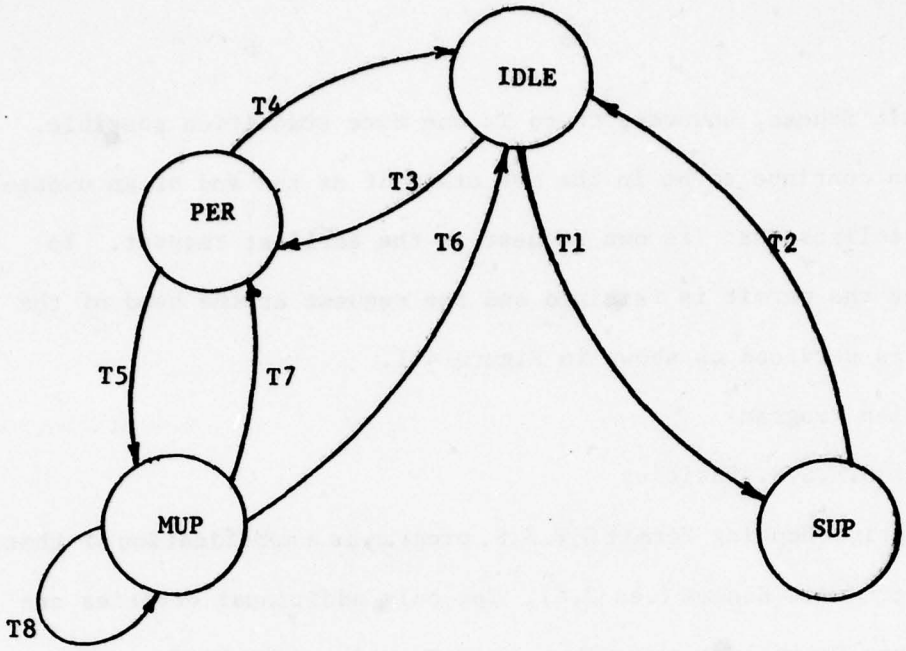
The Adaptive Hopping Permit G.P.S.S. program is a modification of that of the Hopping Permit Scheme (see 2.4). The only additional entities are the use of three Savevalues per node to store routing information. An extra parameter in a transaction is needed for the bid. Because of this additional parameter slight changes have to be made in the variable and Boolean variable expressions. The functions performed by these variables and Boolean variables are exactly the same as those in the Hopping Permit Scheme. Logic Switches are numbered exactly the same way as in the Hopping Permit Scheme (see 2.4.1). The same equations that are used to obtain the numbers of the Control Logic Switches are used to obtain Savevalue numbers for each computer. The Savevalues for each computer are numbered as follows:

Savevalue # 1: Contains the request times of the local request if any: $(i - 1) \times 3 + 1$

Savevalue # 2: Contains Source ID# of earliest request: $(i - 1) \times 3 + 2$

Savevalue # 3: Contains earliest request time: $(i - 1) \times 3 + 3$

When the updating L.F.M. sends out an updated complete message, the time of the request at the head of the local queue is entered into Savevalue #3 if any request exists. Otherwise, a future time is recorded and the source ID# is entered into Savevalue #2. As the update acknowledgements come in, the bids are compared with Savevalue #3. The earlier re-



T1 : Updating information is received and the L.F.M. proceeds into the foreign updating phase.

T2 : An update complete is received.

T3 : The permit is received.

T4 : The local queue is empty and the permit is passed on.

T5 : The local queue is non-empty and the L.F.M. proceeds into updating phase.

T6 : Update Acknowledgements from all participating nodes are received by the updating L.F.M.

T7 : No requests exist in the network at the end of the update session.

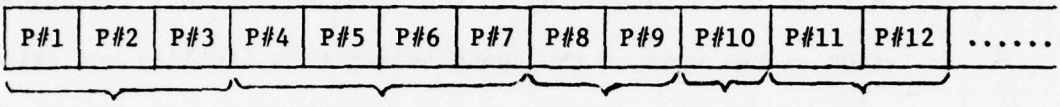
T8 : The local request is the earliest request in the network.

Figure 4.1. State transition diagram of the Adaptive Hoppin Permit Scheme.

quest and its corresponding source ID# of the two is recorded into these Savevalues.

4.3.2 Parameter Set of Transactions

The permit in the Adaptive Hopping Permit Scheme has the same parameter set as that in the Hopping Permit Scheme. All the other transactions have the following parameter set:



Information Field | Control Field | Boolean Field | Bid Field | Boolean Field

Parameter Number		Contents	
P#1	:	Source Node ID#	
P#2	:	Destination Node ID#	
P#3	:	Type (Same as for H.P.S., see 2.4.2)	
P#4	:	Temporary Data	
P#5	:	Control Logic Switch ID#	
P#6	:	Acknolwedgement Logic Switch ID#	
P#7	:	Loop Counter	
P#8	:	Per Switch ID#	Referred to by Boolean
P#9	:	MUP Switch ID#	Variable 1
P#10	:	Bid Field	
P#11	:	Ack. Switches	Referred to by Boolean
P#12	:	ID#'s	Variable 2

The number of parameters defined per transaction in this simulation is 9 + N.

4.3.3 Details of the Simulation Program (see Figure 4.2 or Appendix C)

The Sending Section

The following differences between the Hopping Permit Scheme program and the Adaptive Hopping Permit Scheme program are outlined below. Here, the program is basically the same as that of the Hopping Permit Scheme with a few modifications to include the adaptive routing feature.

(1) The request's generation time is entered in P#10.

(2) A facility is introduced in the sending section. The transaction at the head of the queue seizes this facility and releases it only when proceeding into the update phase. The arrival time request is stored in Savevalue #1.

(3) Prior to dispatching an update begin, the status of the above facility is checked. If this is '0' then it means that the queue is empty. In that case a future time (present simulation + 10 seconds) is stored in Savevalue #3. Otherwise, Savevalue #1 is assigned to Savevalue #3.

The Receiving Section

The only change made in the permit processing section is that the node ID# stored in Savevalue #2 is assigned as the next destination. This is done only if the node retains the permit. If the destination specified is the node that possesses the permit, then the permit is recycled within the node. Otherwise, it is dispatched.

When an update complete is received the local queue is checked to see whether it is empty or not. If it is not empty, then the contents of Savevalue #1 is assigned to P#10 as the bid. If the local queue is empty, a future time (present simulation time + 30 seconds) is assigned to P#10 as the bid.

When an update acknowledgement is received, the bid is compared with the earliest bid received thus far. If the incoming update acknowledgement has an earlier bid, Savevalue #3 is assigned the new bid, and Savevalue #2 is assigned the source node ID# of the bid. If all the nodes reply with no bid the permit is recycled within the node. If a request arrives between the time an update complete is sent out and all update acknowledgements are received, this request will be processed.

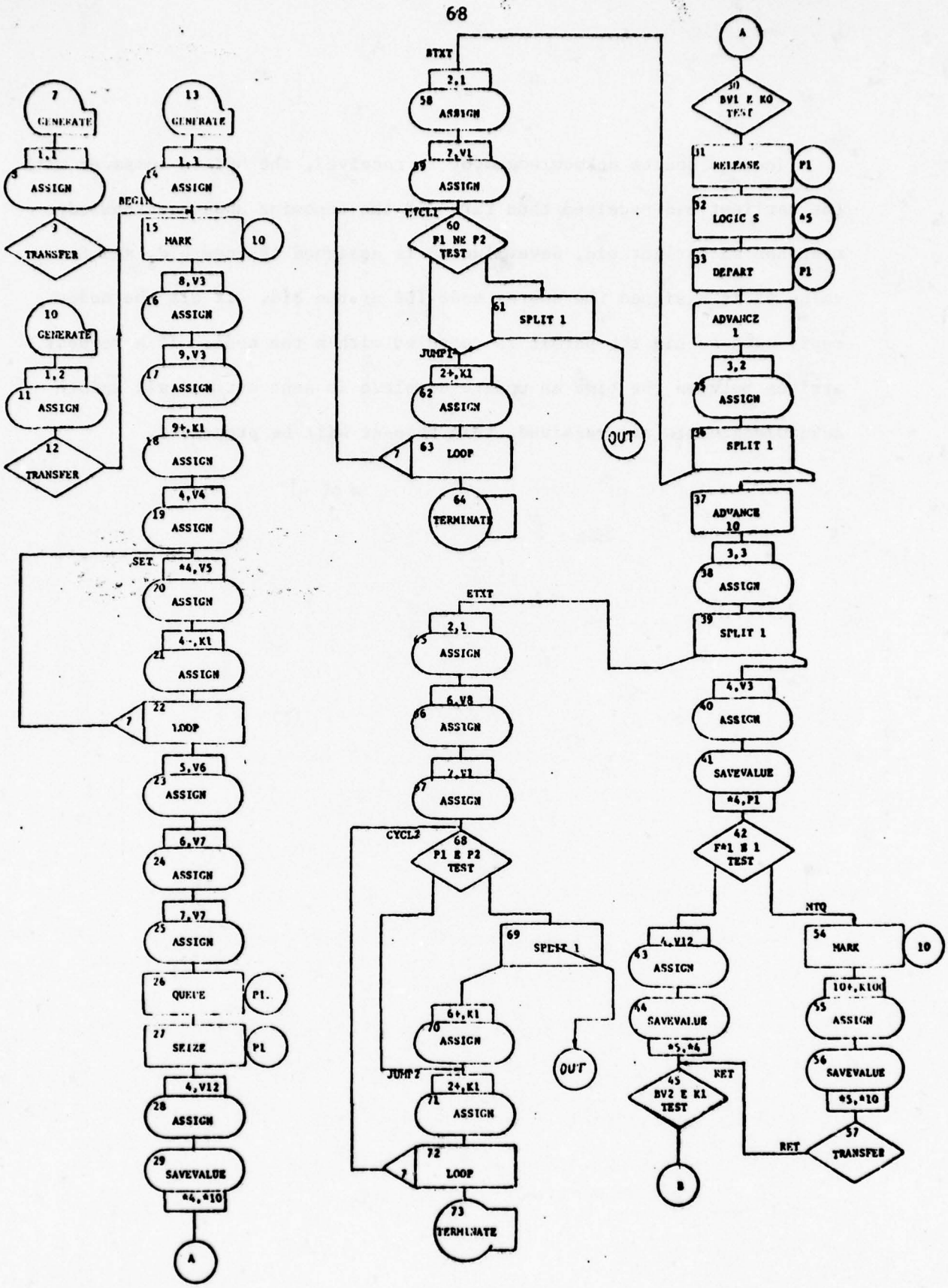


Figure 4.2. G.P.S.S. flow chart of the Adaptive Hopping Permit Scheme.

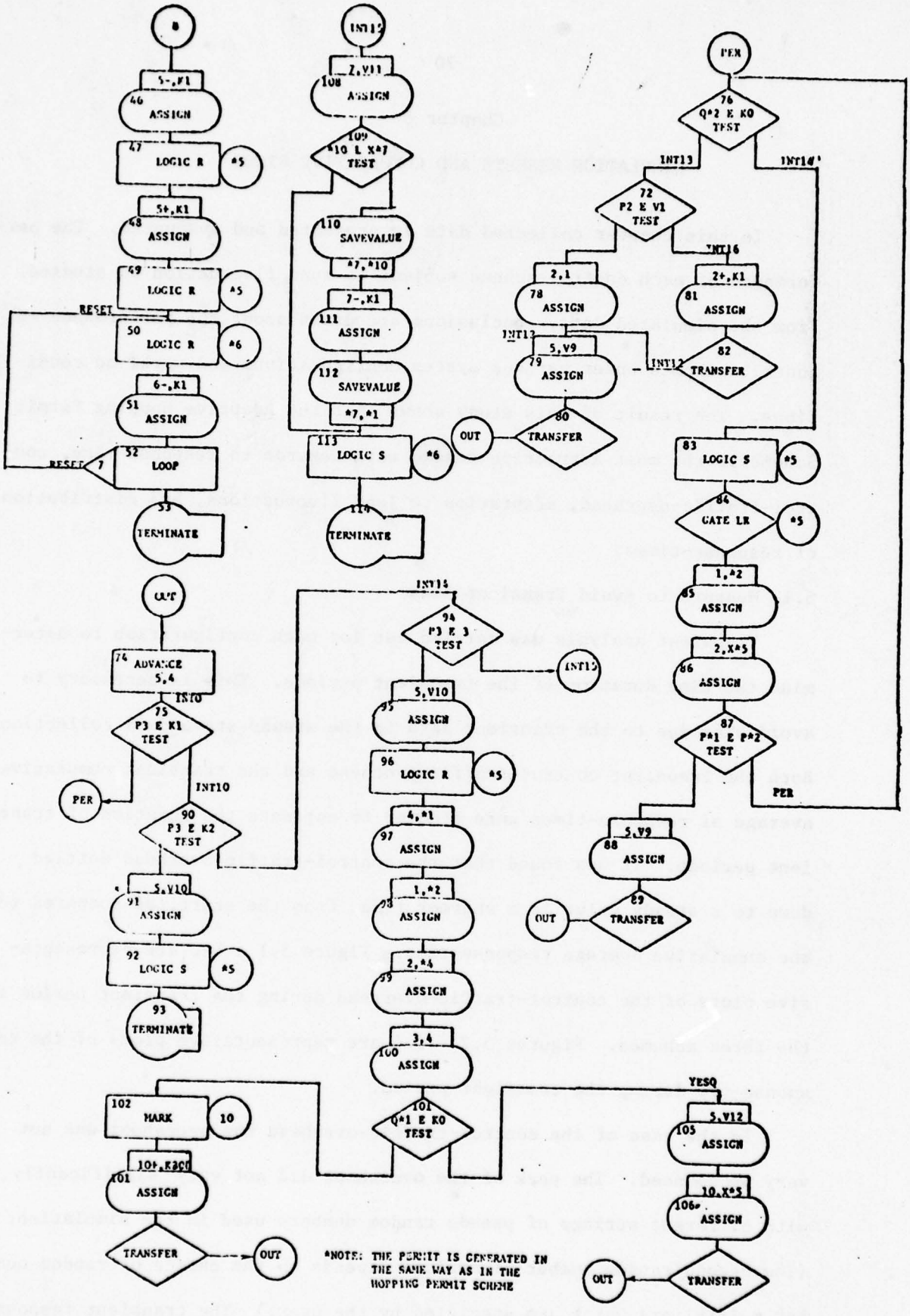


Figure 4.2 (contd.)

Chapter 5

SIMULATION RESULTS AND COMPARATIVE STUDY

In this chapter collected data is presented and evaluated. The performance of each control scheme subject to load fluctuation is studied. From the simulated data, conclusions are drawn about the performance of control schemes under various system configurations and workload conditions. The result of this study shows that the Adaptive Hopping Permit Scheme is the most attractive scheme with regards to response-time, control-traffic-overhead, adaptation to load fluctuations, and distribution of response-times.

5.1 Measure to Avoid Transient Bias

Transient analysis was carried out for each configuration to determine the time duration of the transient periods. This is necessary to avoid bias due to the transient data in the steady state data collection. Both the transient control-traffic-overhead and the transient cumulative average of response-times were studied to estimate the duration of transient periods. It was found that the control-traffic-overhead settled down to a steady value in a shorter time, from the start, as compared to the cumulative average response-time. Figure 5.1 a-b-c are representative plots of the control-traffic-overhead during the transient period for the three schemes. Figures 5.2 - 5.4 are representative plots of the response-tim during the transient period.

In the case of the control-traffic-overhead the overshoot was not very pronounced. The peak of the overshoot did not vary significantly with different strings of pseudo random numbers used in the simulation. (The Pseudo random number of strings depends on the choice of random number multipliers which are specified by the user.) The transient response

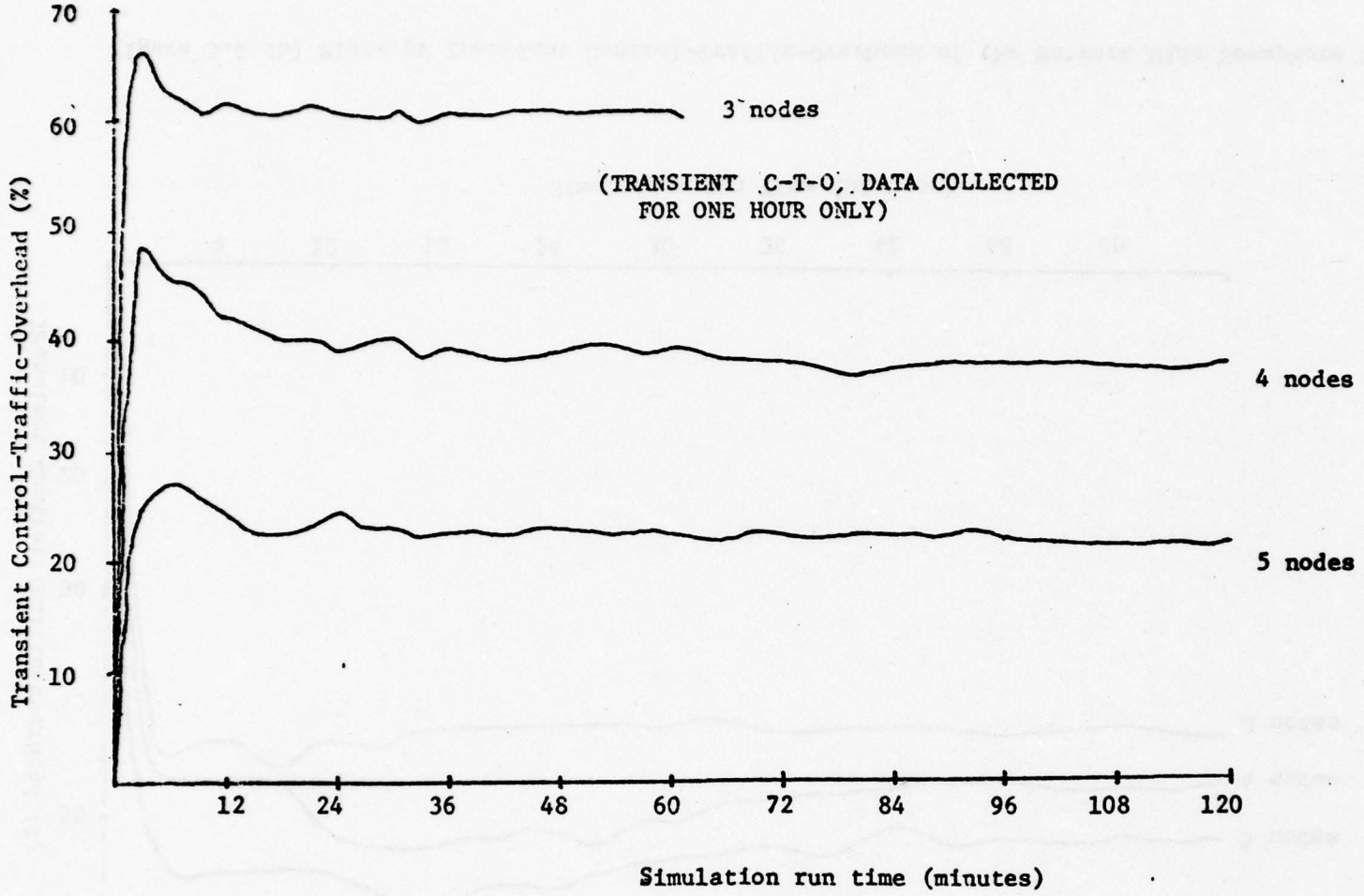


Figure 5.1 (a) Plots of Transient Control-Traffic Overhead of the Hopping Permit Scheme

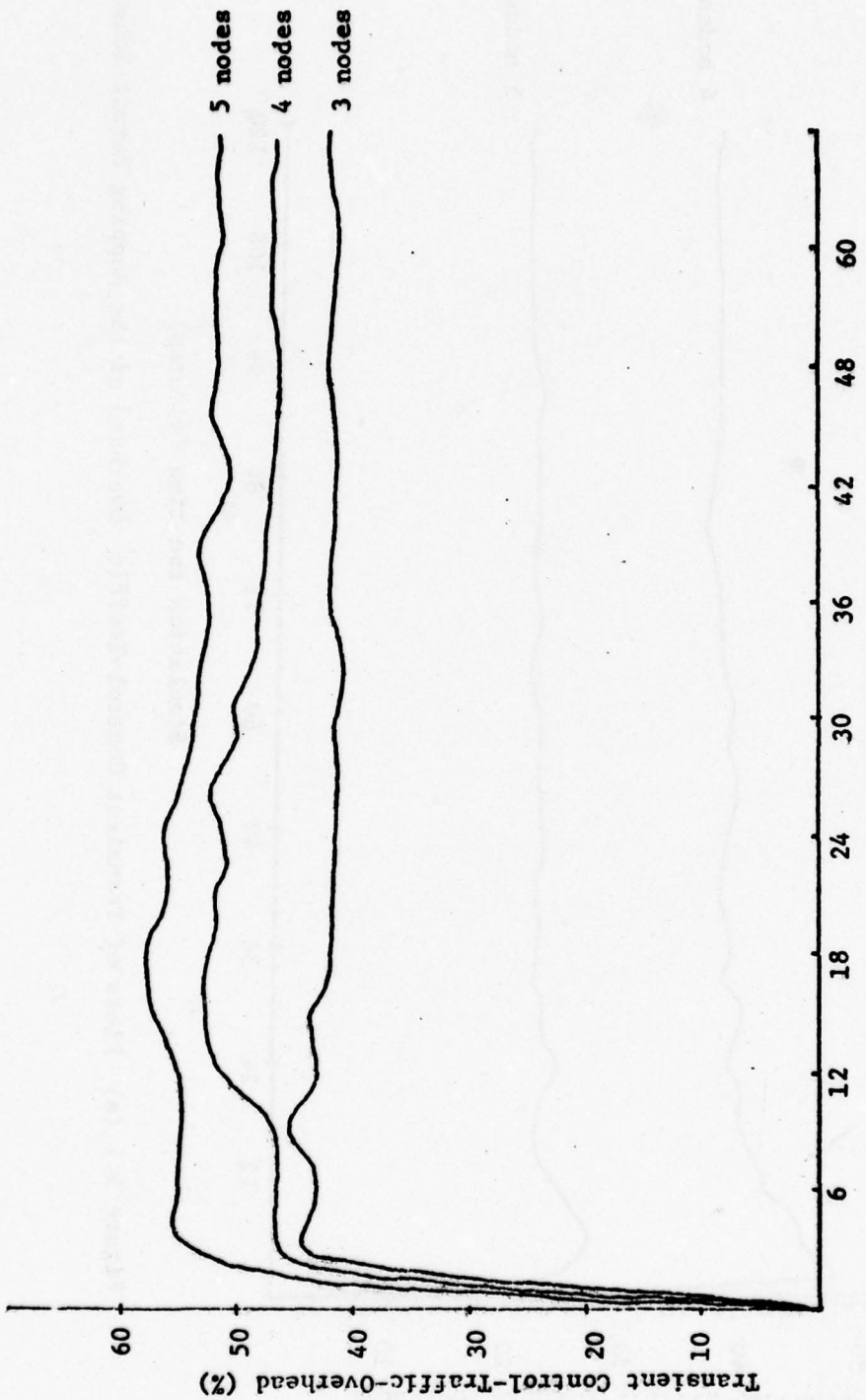


Figure 5.1 (b) Plots of Transient Control-Traffic-Overhead of the Network Wide Semaphore Scheme

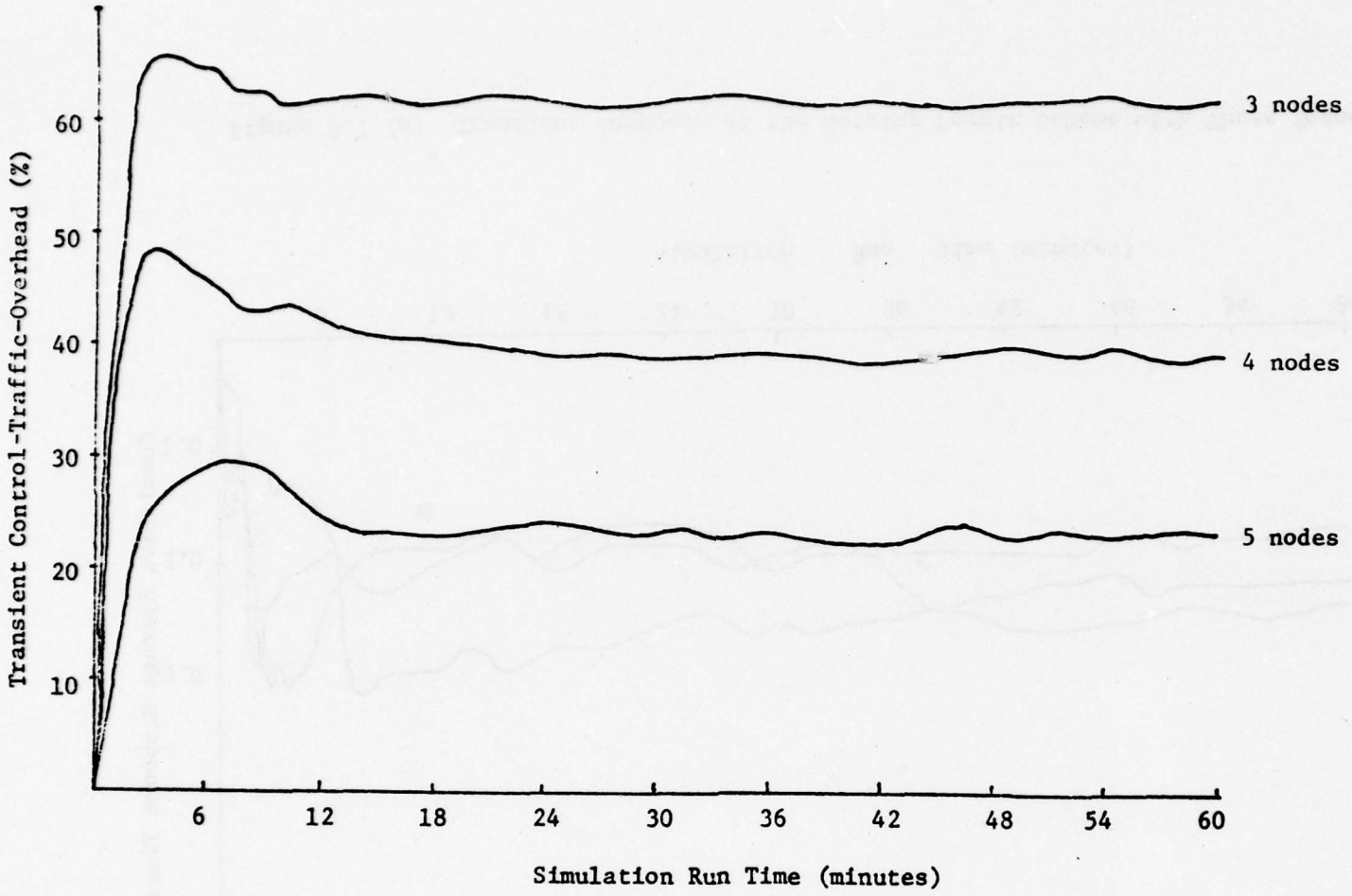


Figure 5.1(c) Plot of Transient Control-Traffic-Overhead of the Adaptive Hopping Permit Scheme

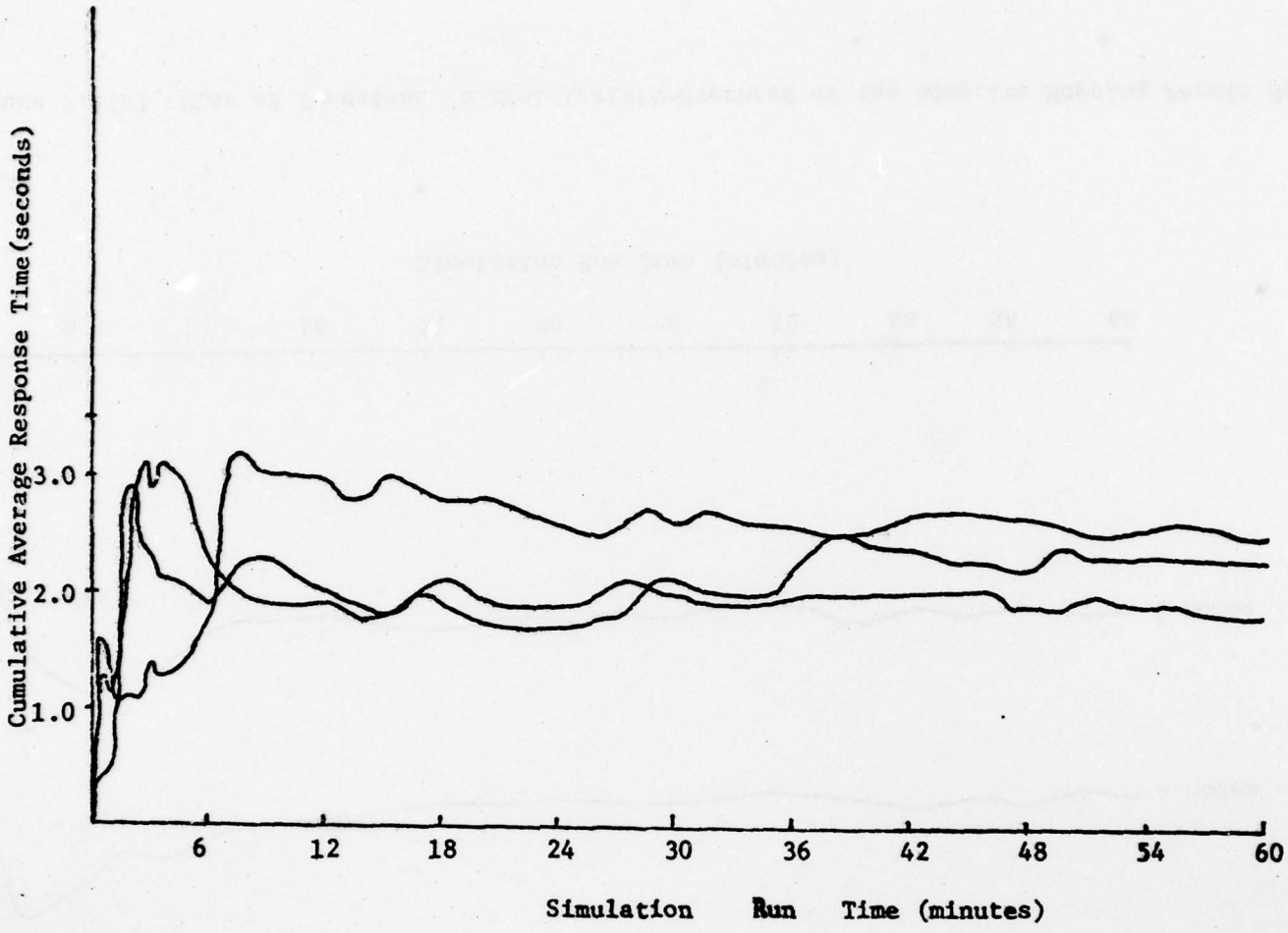


Figure 5.2 (a) Transient Response of the Hopping Permit Scheme with Three Nodes

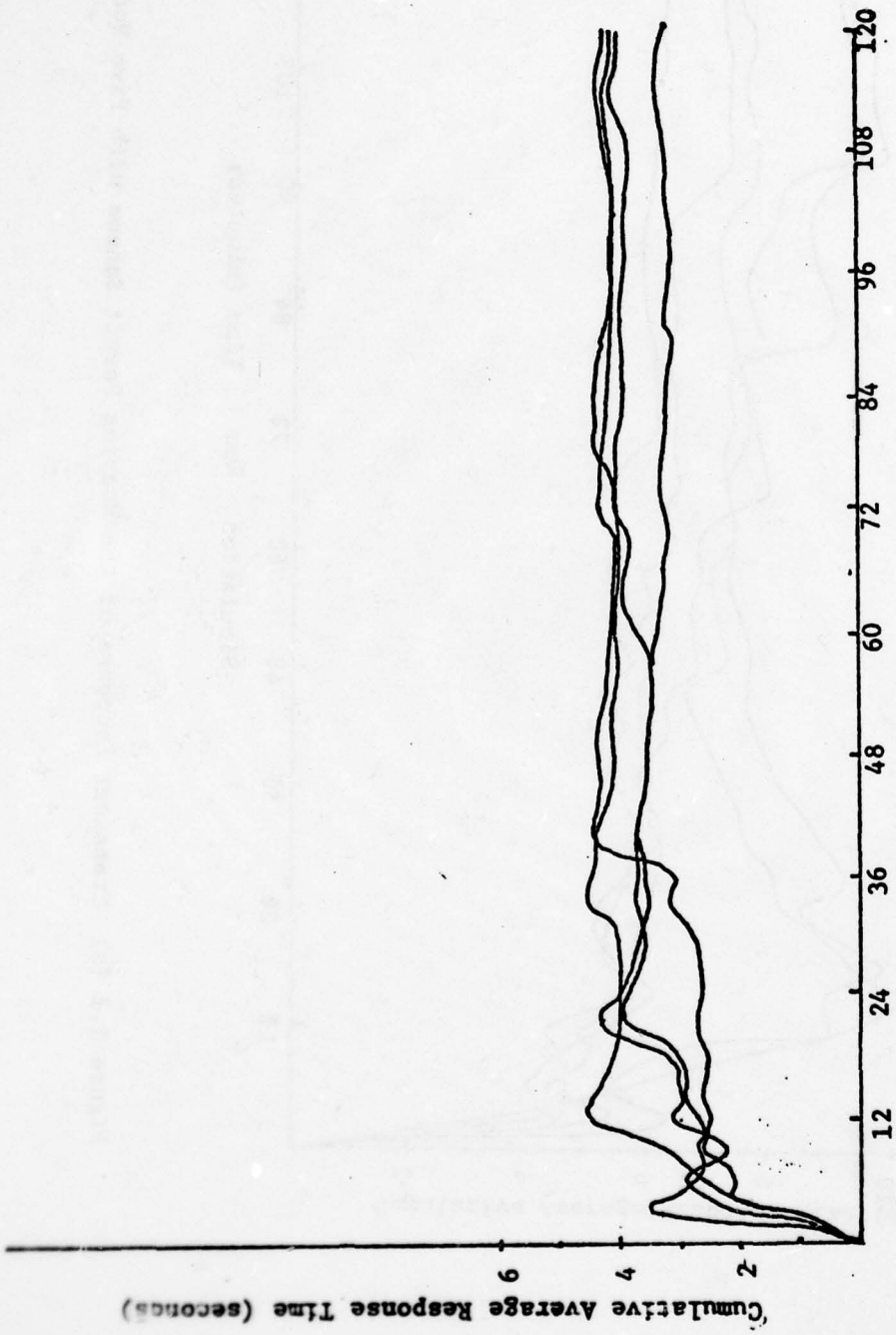


Figure 5.2 (b) Transient Response of the Hopping Permit Scheme with Four Nodes

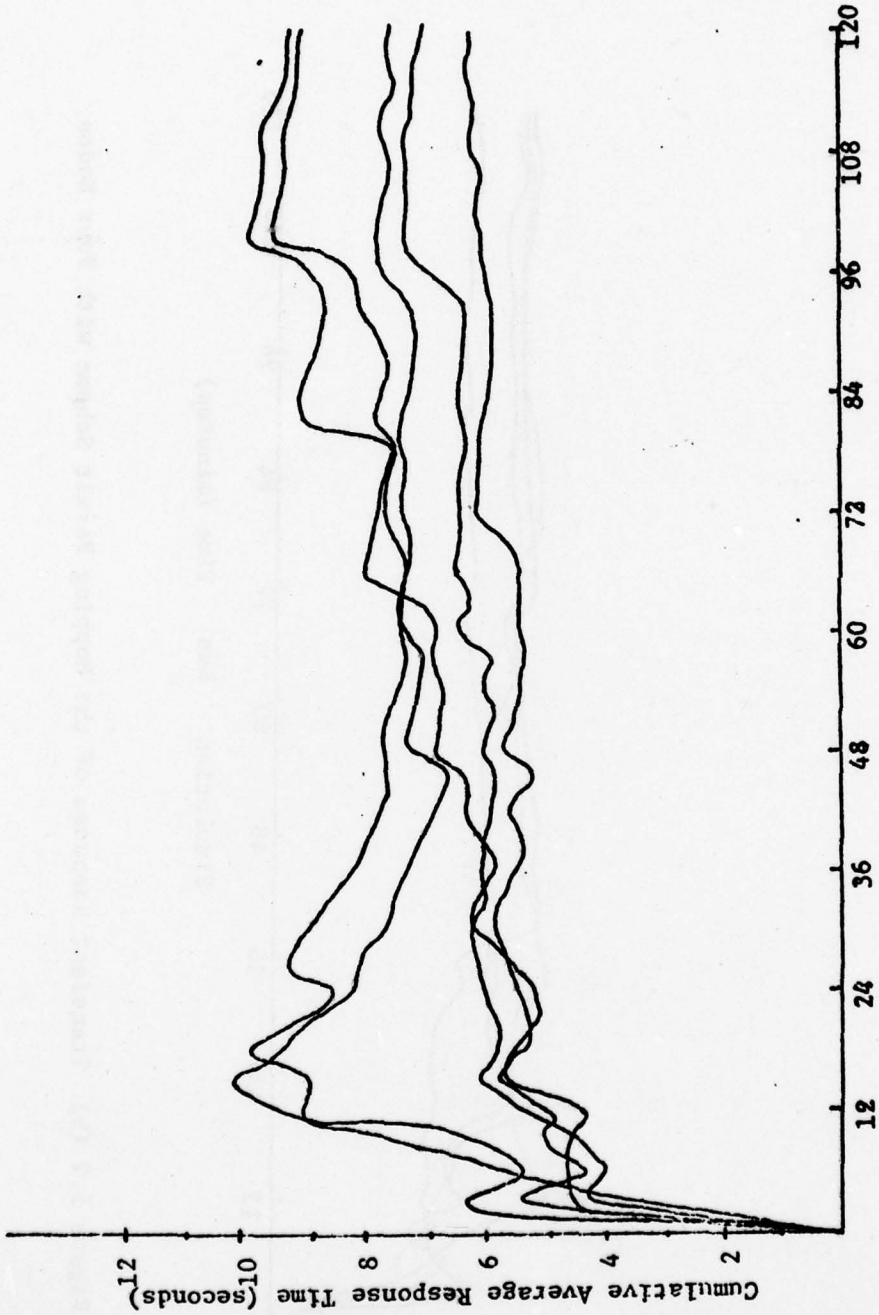


Figure 5.2 (c) Transient Response of the Hopping Permit Scheme with Five Nodes

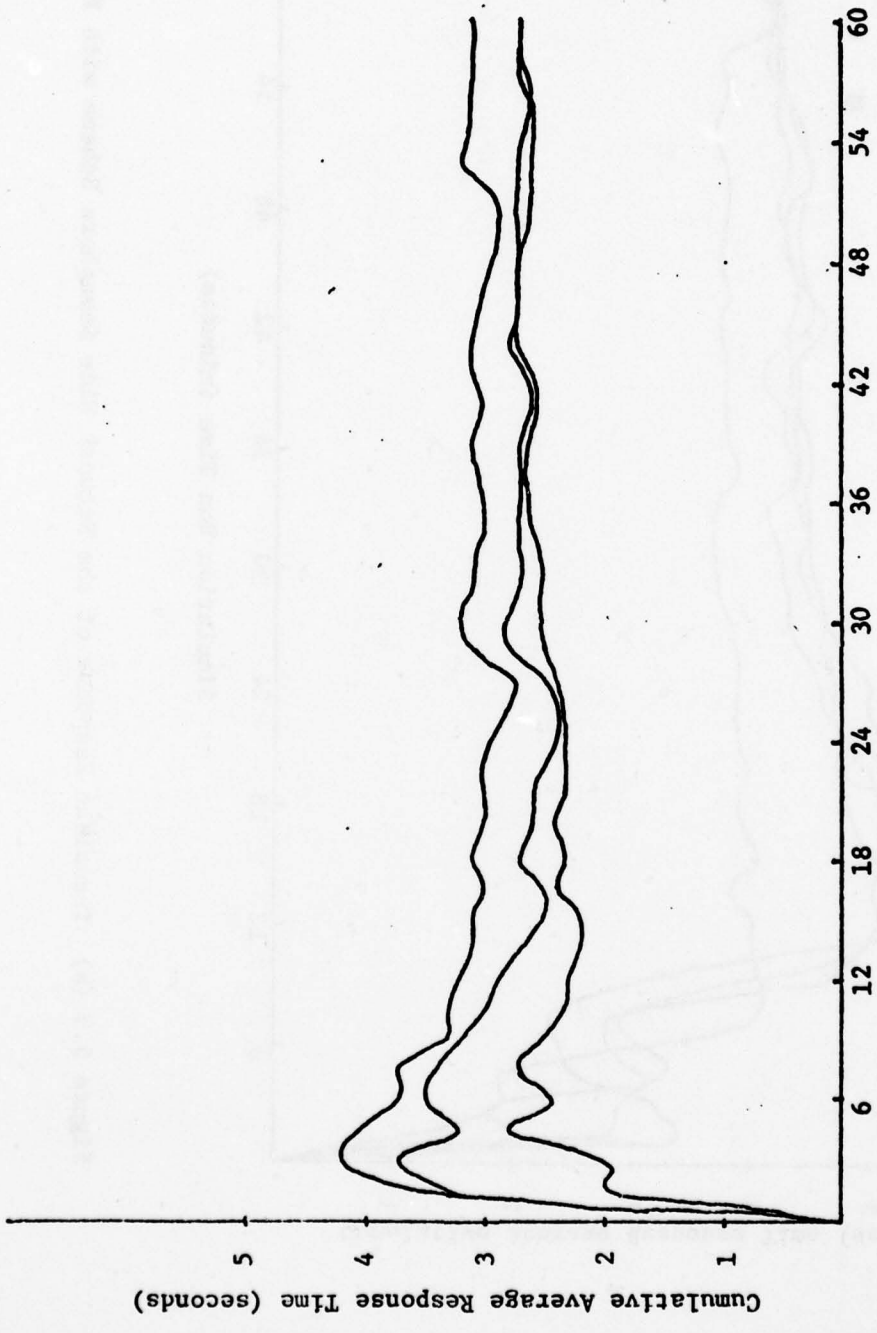


Figure 5.3 (a) Transient Response of the Network Wide Semaphore Scheme for Three Nodes

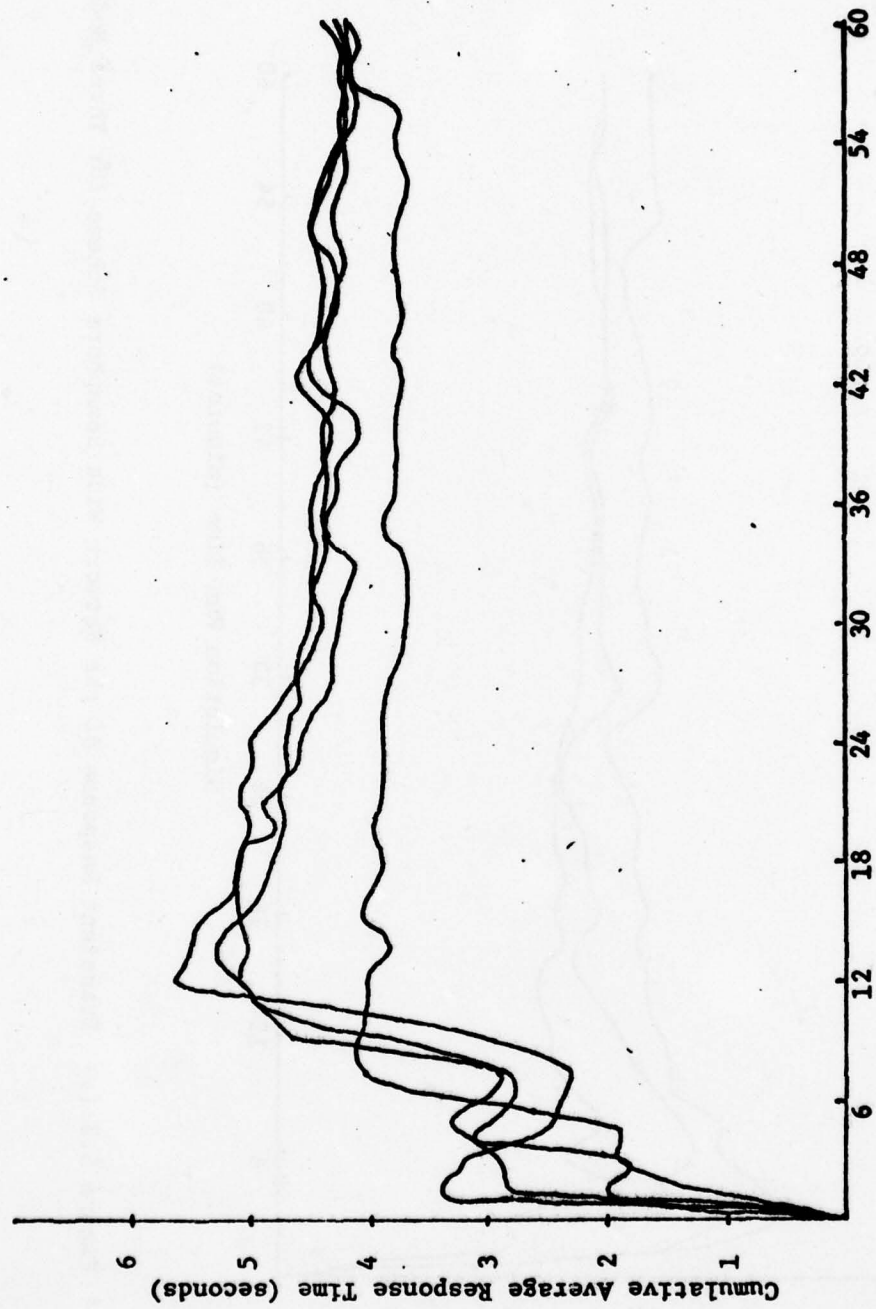


Figure 5.3 (b) Transient Response of the Network Wide Semaphore Scheme with Four Nodes

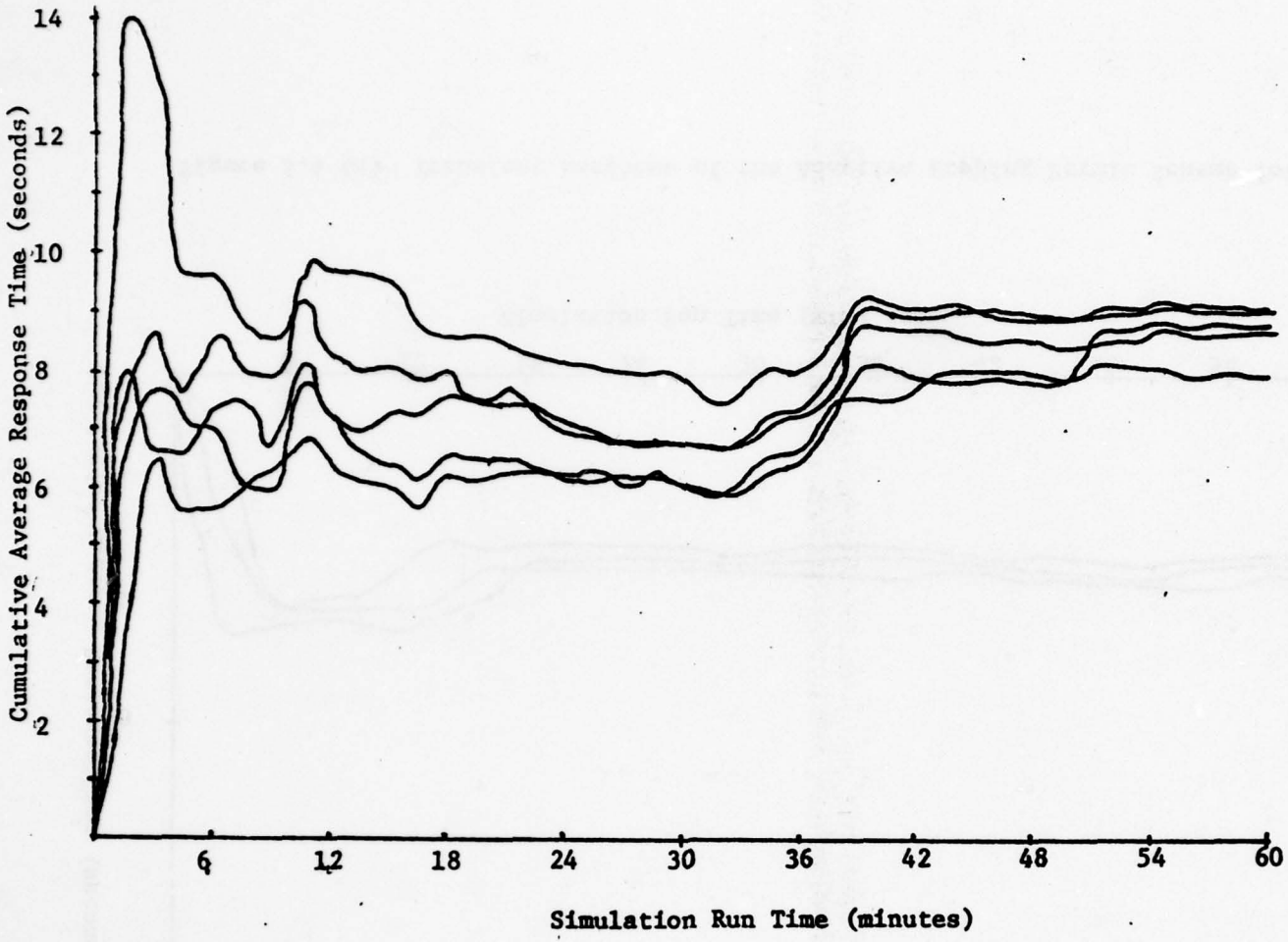


Figure 5.3 (c) Transient Response of the Network Wide Semaphore Scheme with Five Nodes

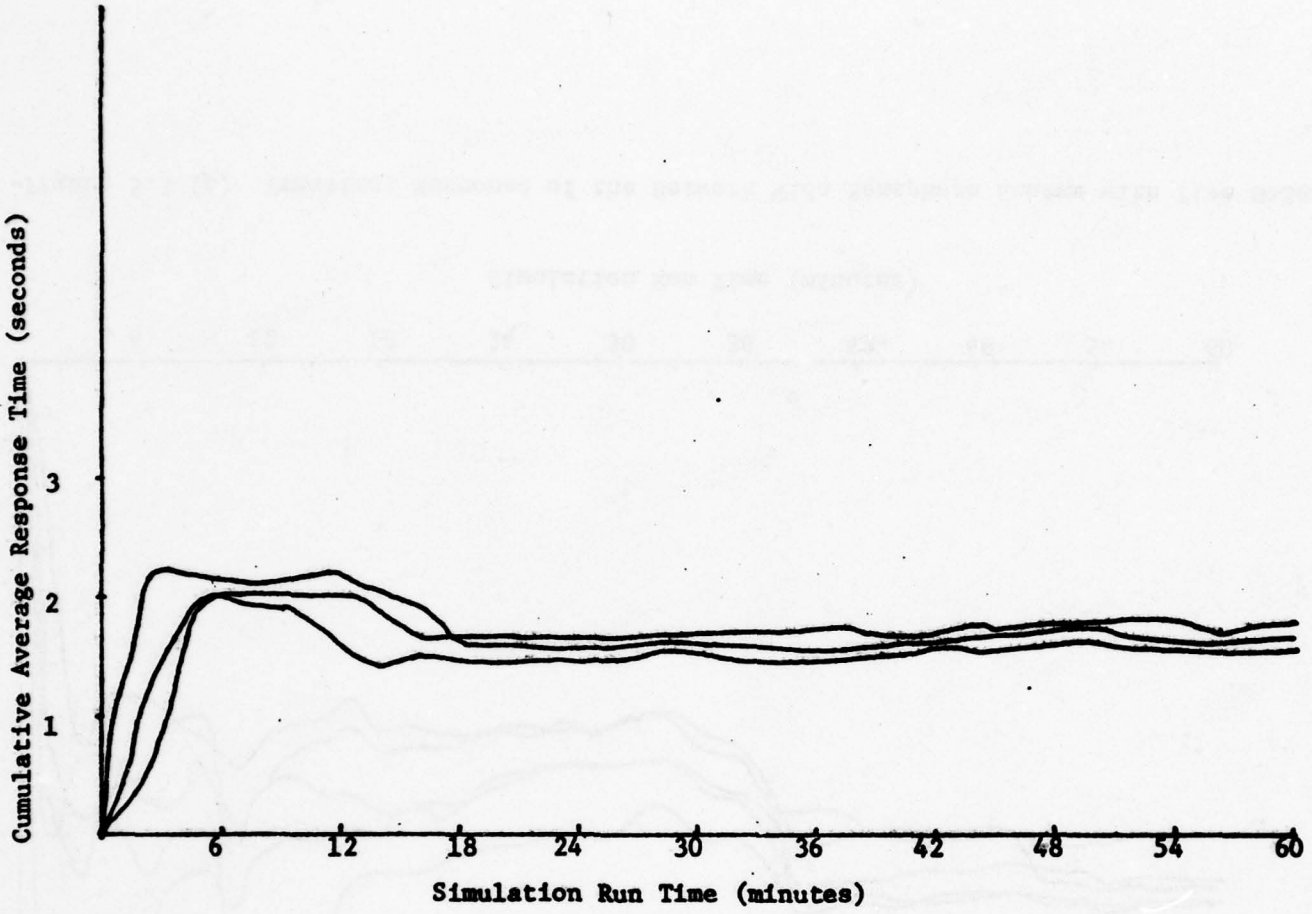


Figure 5.4 (a) Transient Response of the Adaptive Hopping Permit Scheme for Three Nodes

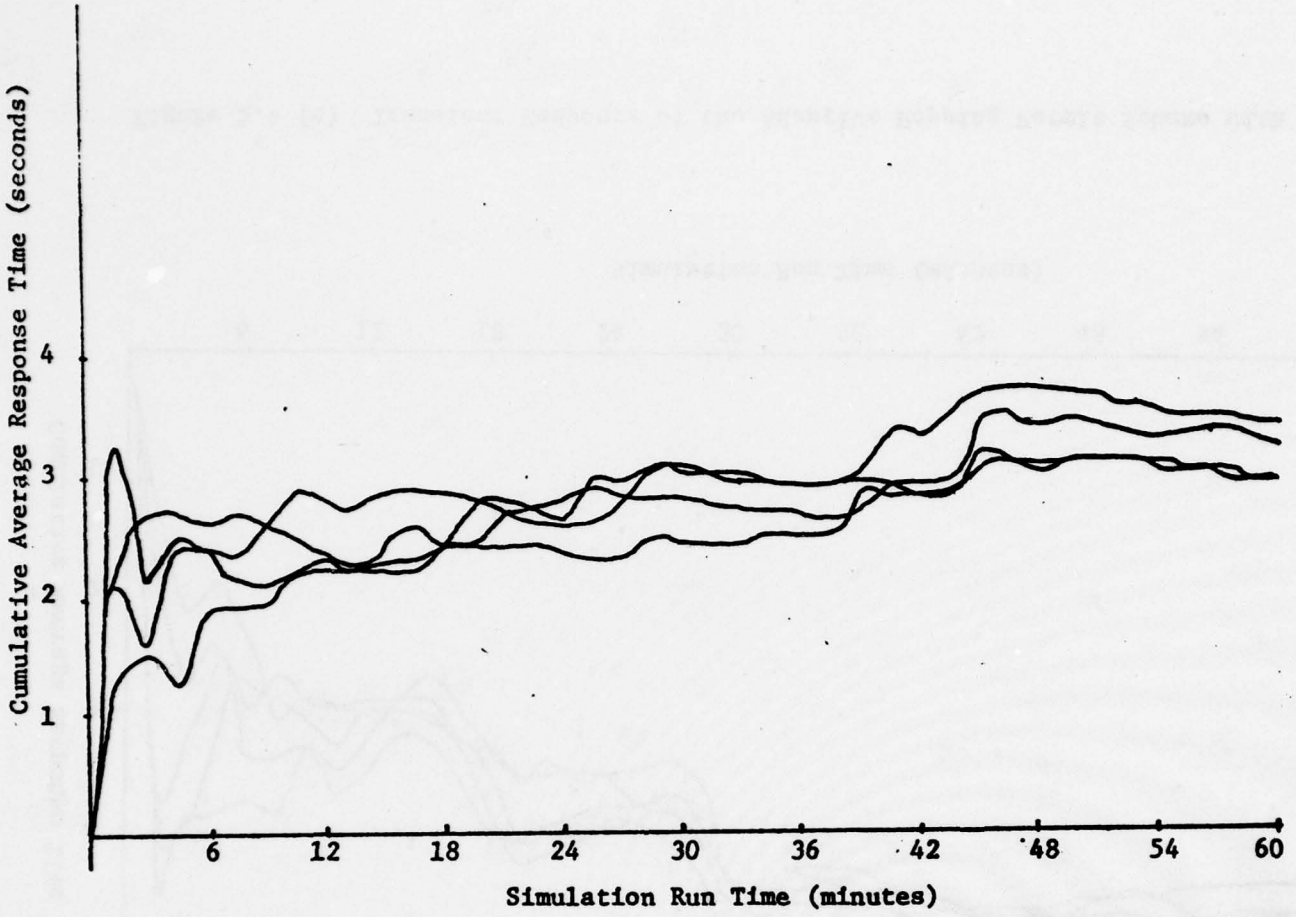


Figure 5.4 (b) Transient Response of the Adaptive Hopping Permit Scheme with Four Nodes

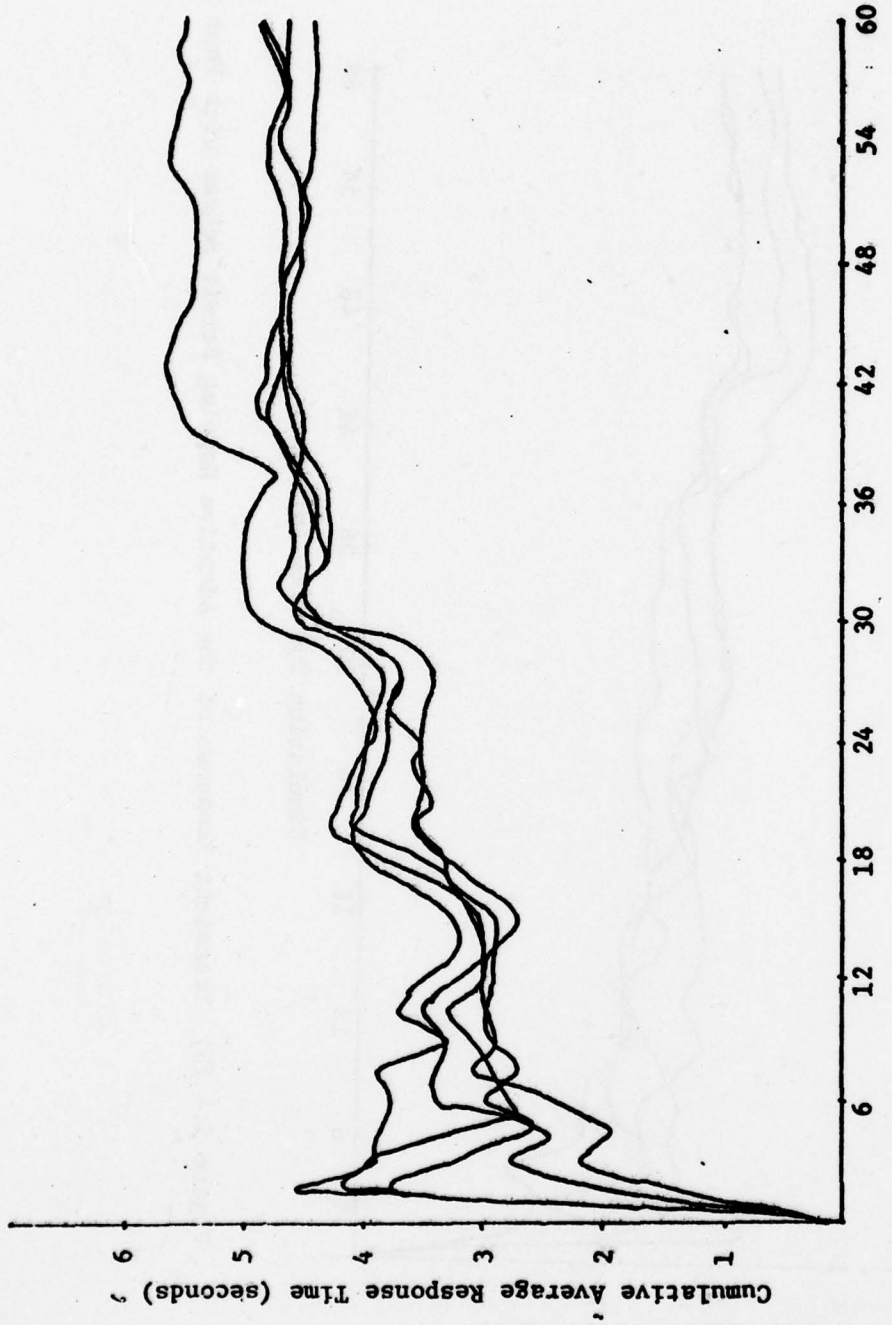


Figure 5.4 (c) Transient Response of the Adaptive Hopping Permit Scheme with Five Nodes

time however, was sensitive to the random number multiplier. The overshoots varied for different nodes in the same run. In some simulation runs the value at which the cumulative average response time settled was higher than the first peak. This effect becomes more pronounced as the number of nodes in the network increases. Obviously the simulation model with a large number of nodes is more sensitive to arrival patterns of the requests than the ones with the smaller number of nodes.

The transient duration of the average response time was always much longer than the transient duration of the control-traffic-overhead. It was noticed that the transient duration increased with the increase of nodes in the simulation model for each scheme. This may be ascribed to the increase in the utilization factor which will be discussed later. The transient period is then decided by an educated guess which varies from run to run. Refer to Table 5.2 for the estimated transient periods. In a simulation run, the model program is reset at the end of the transient period. Data collection will be started and continued for the next hour.

5.2 Collection of Steady State Data

Steady state data was obtained by both the method of replication and the method of batch means (21, 22, 23). However, due to cost considerations the number of replications were not large enough to do a statistical analysis for a confidence interval within which the true mean may be located with a finite probability. Since the aim of this study is to obtain a method for comparison, this is overlooked initially.

Steady state data was collected at intervals of 90 seconds for a period of one hour of system time for each simulation run. Figure 5.5 - 5.7 are representative plots of the steady state cumulative average response-times.

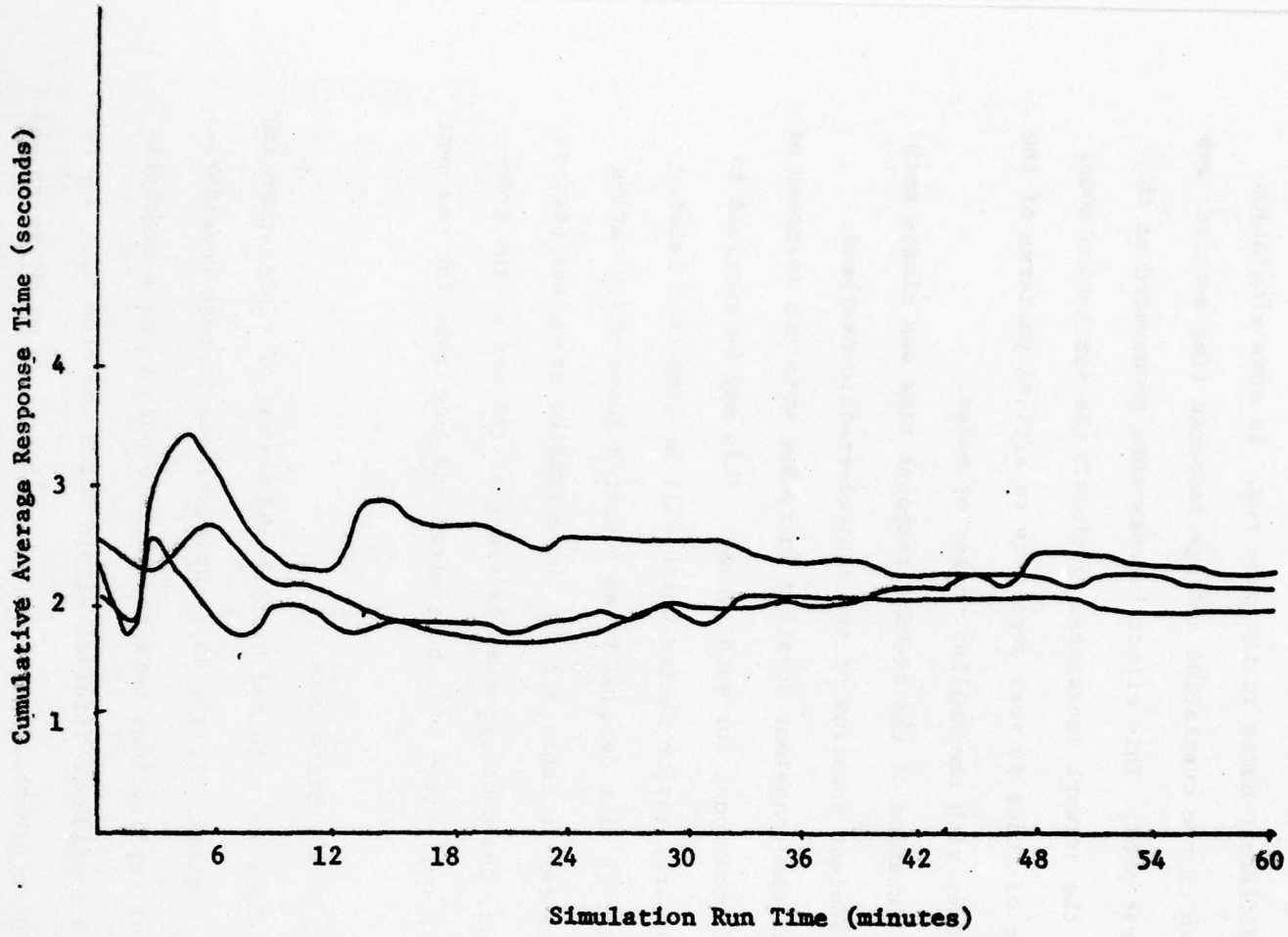


Figure 5.5 (a) Steady State Response of the Hopping Permit Scheme with Three Nodes

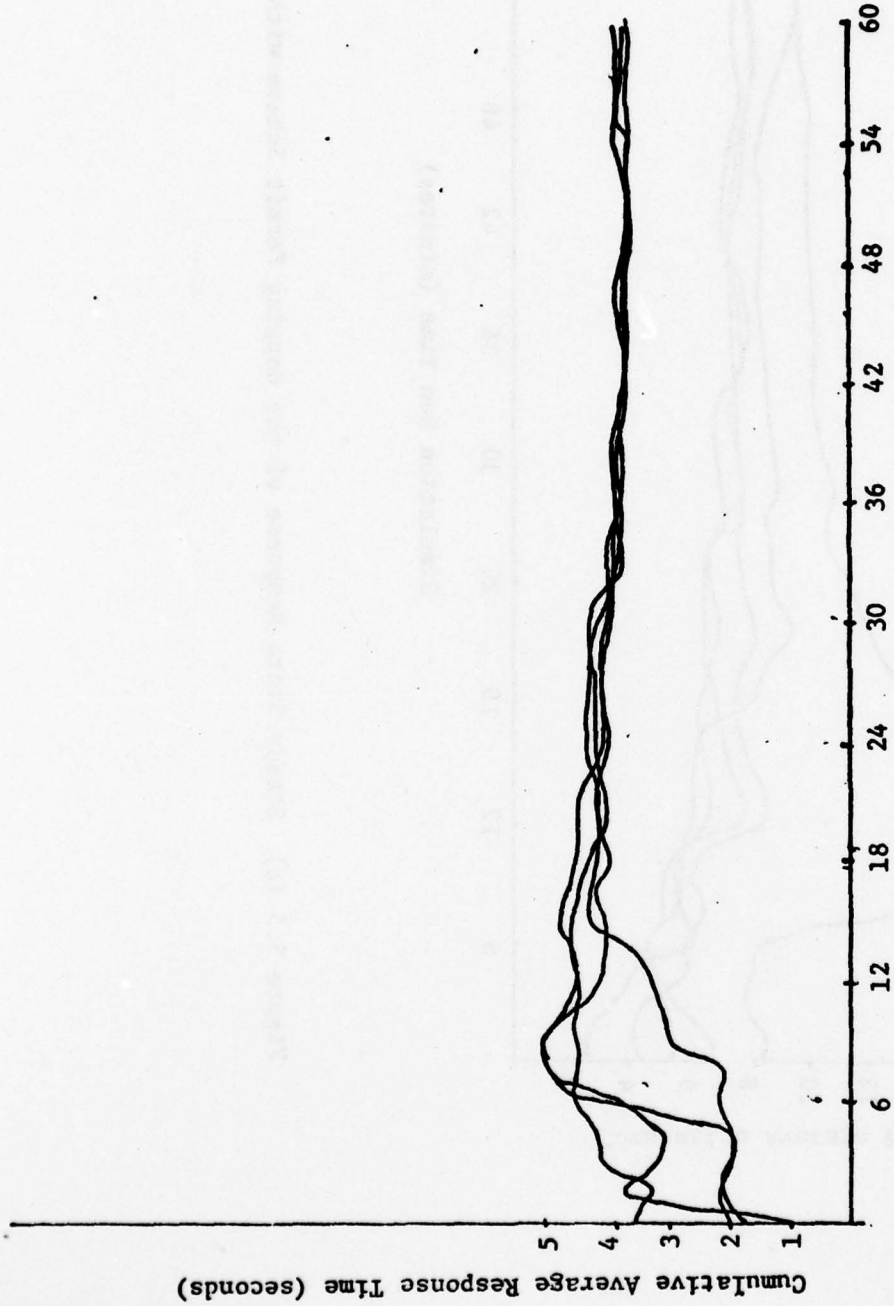


Figure 5.5 (b) Steady State Response of the Hopping Permit Scheme with Four Nodes

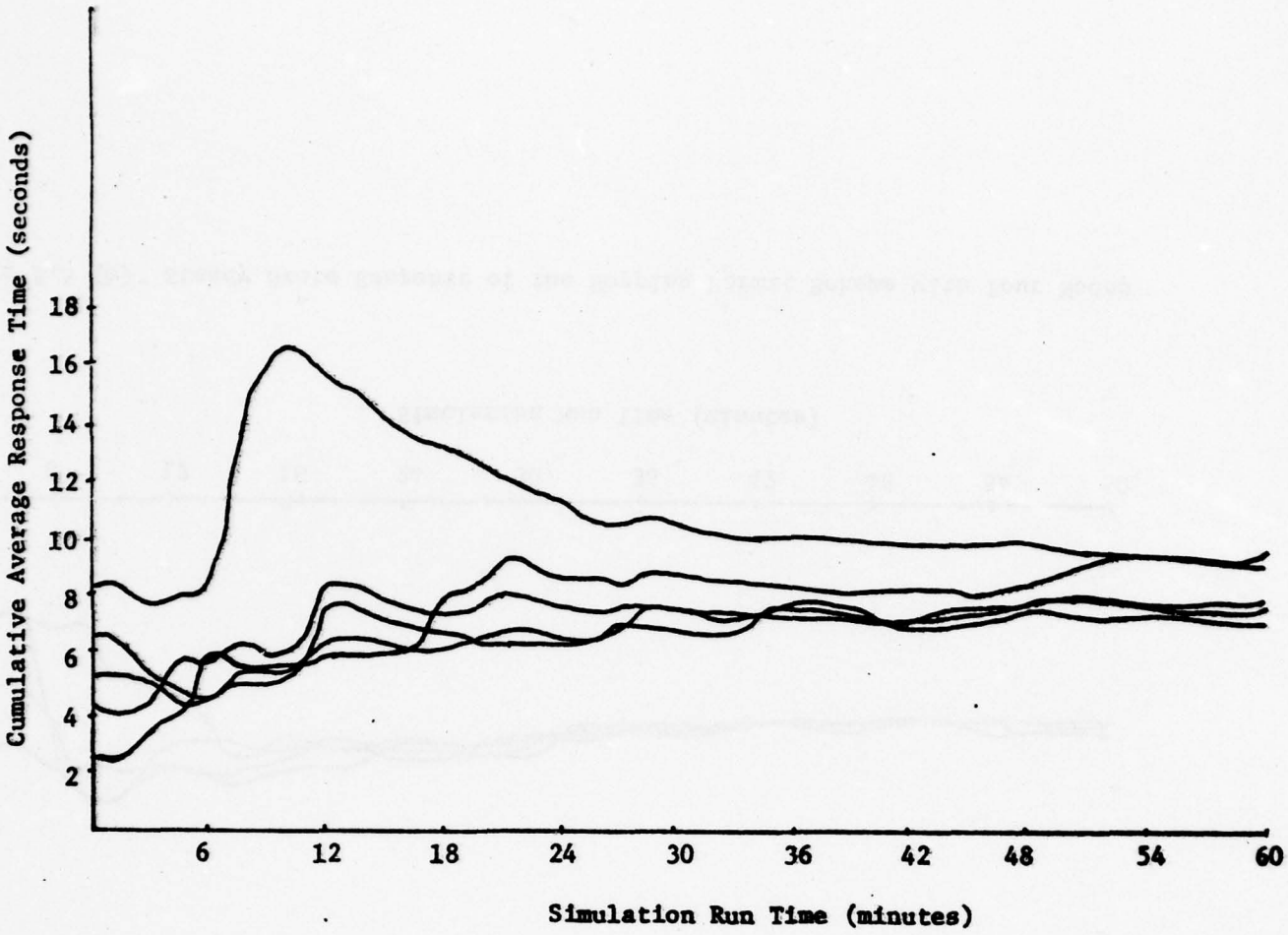


Figure 5.5 (c) Steady State Response of the Hopping Permit Scheme with Five Nodes

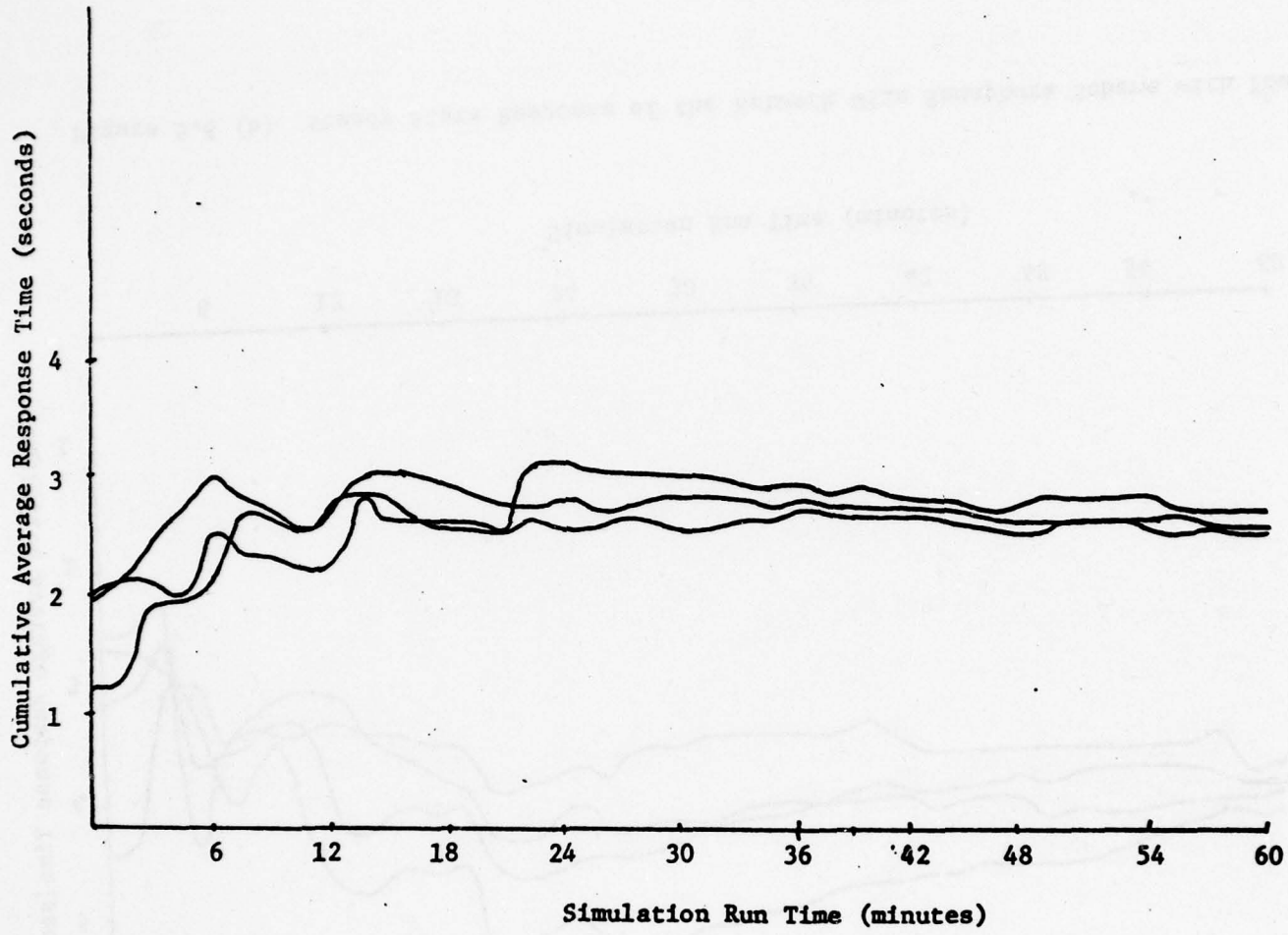


Figure 5.6 (a) Steady State Response of the Network Wide Semaphore Scheme with Three Nodes

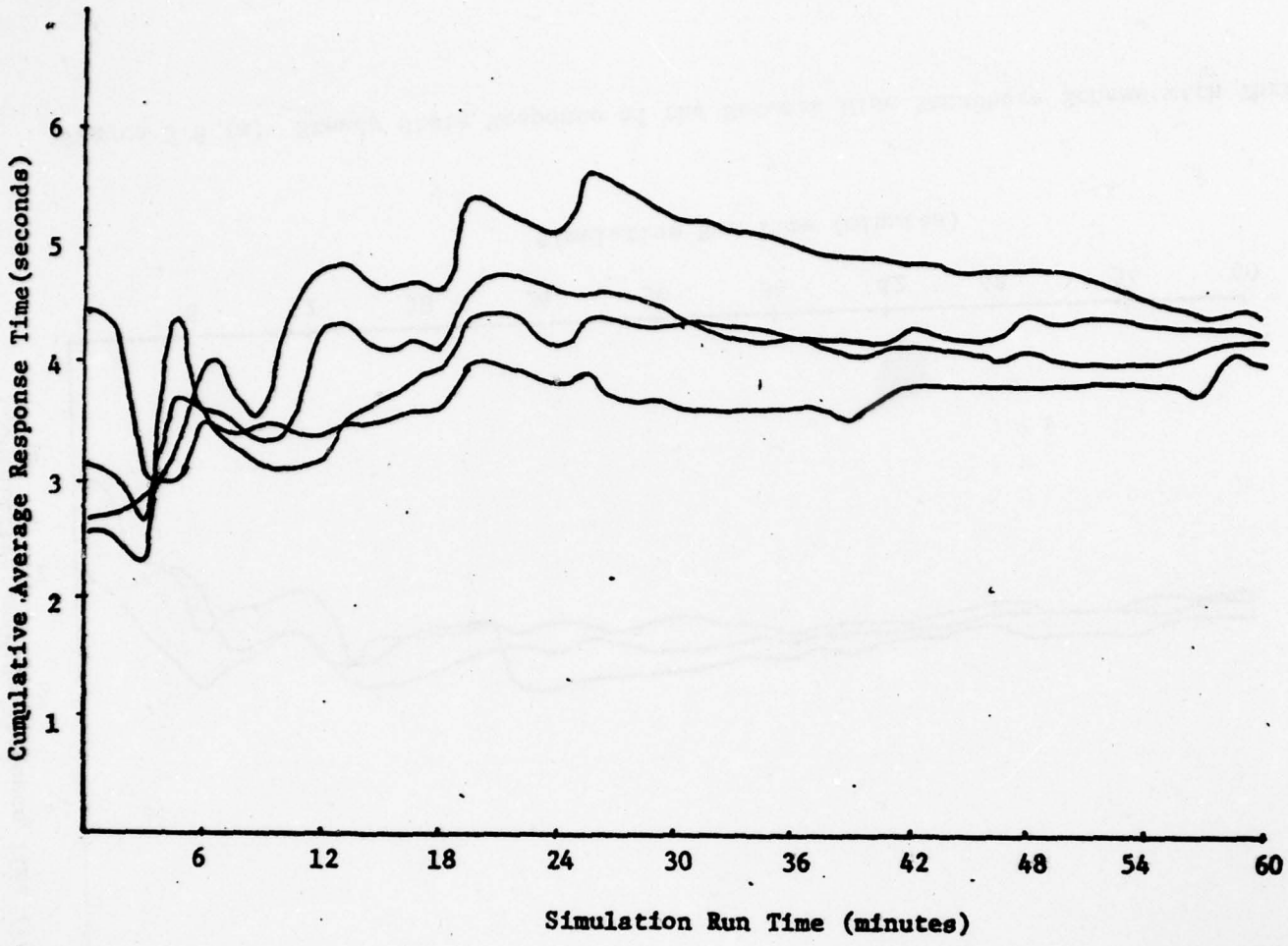
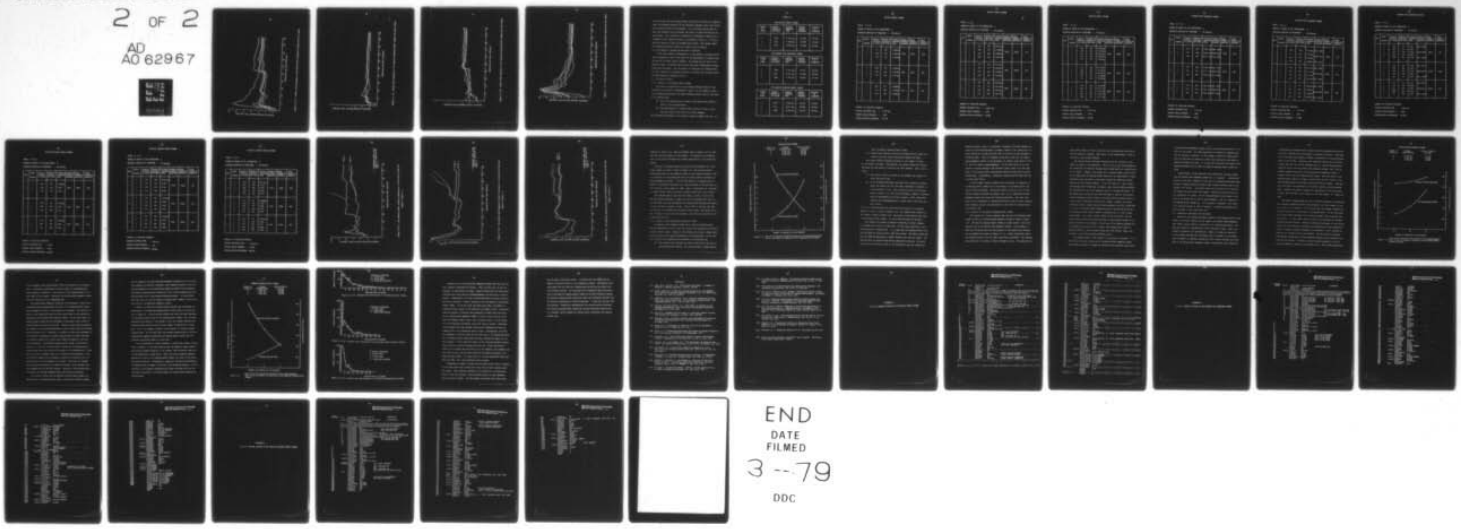


Figure 5.6 (b) Steady State Response of the Network Wide Semaphore Scheme with Four Nodes

AD-A062 967 SYRACUSE UNIV N Y F/G 9/2
DATA SYNCHRONIZATION SCHEMES FOR MULTIPLE COPIED DATA BASES.(U)
DEC 78 C LEE F30602-75-C-0121
UNCLASSIFIED RADC-TR-78-240 NL

2 OF 2

AD
A062967



END
DATE
FILMED
3-79
DDC

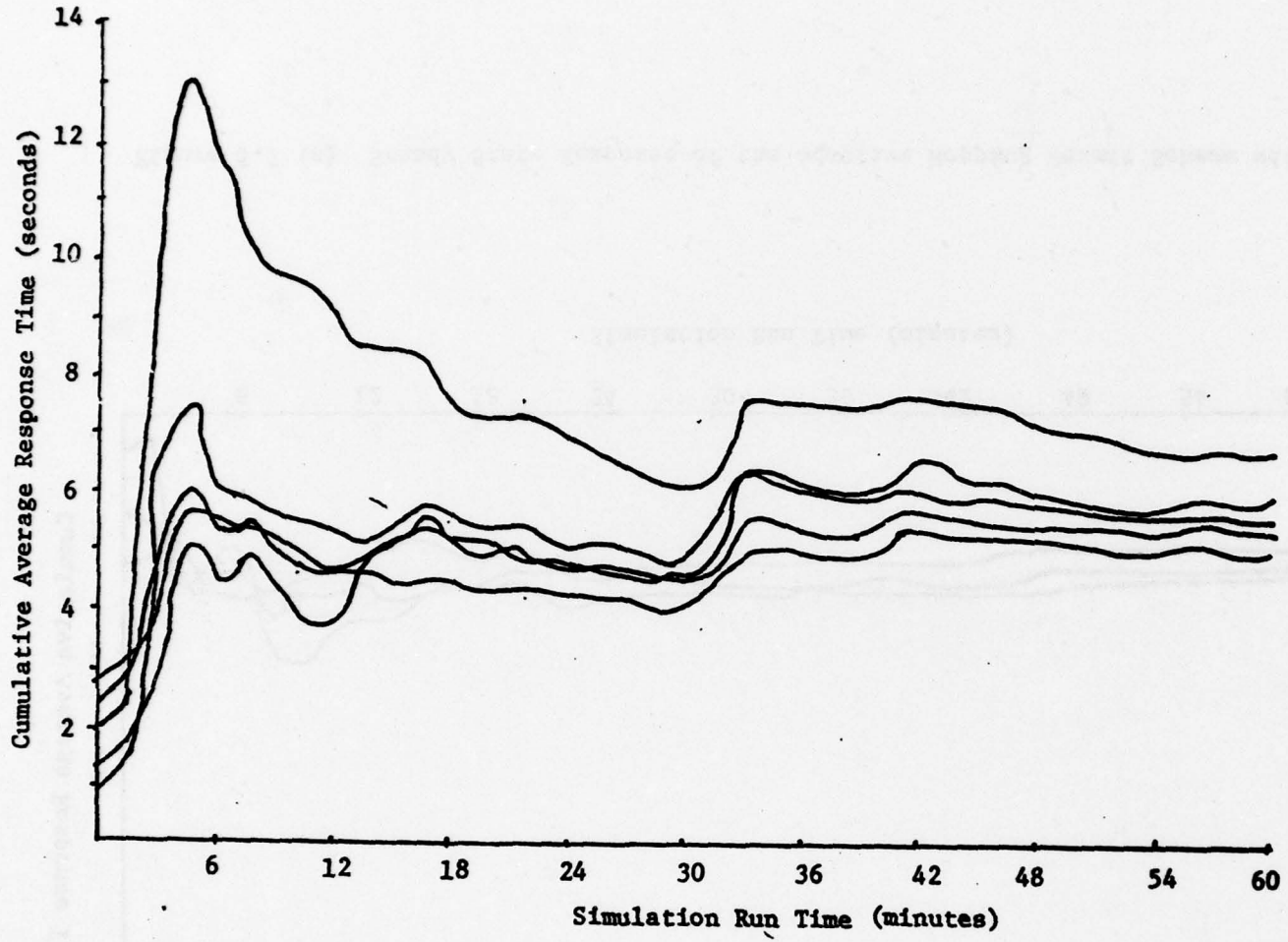


Figure 5.6 (c) Steady State Response of the Network Wide Semaphore Scheme with Five Nodes

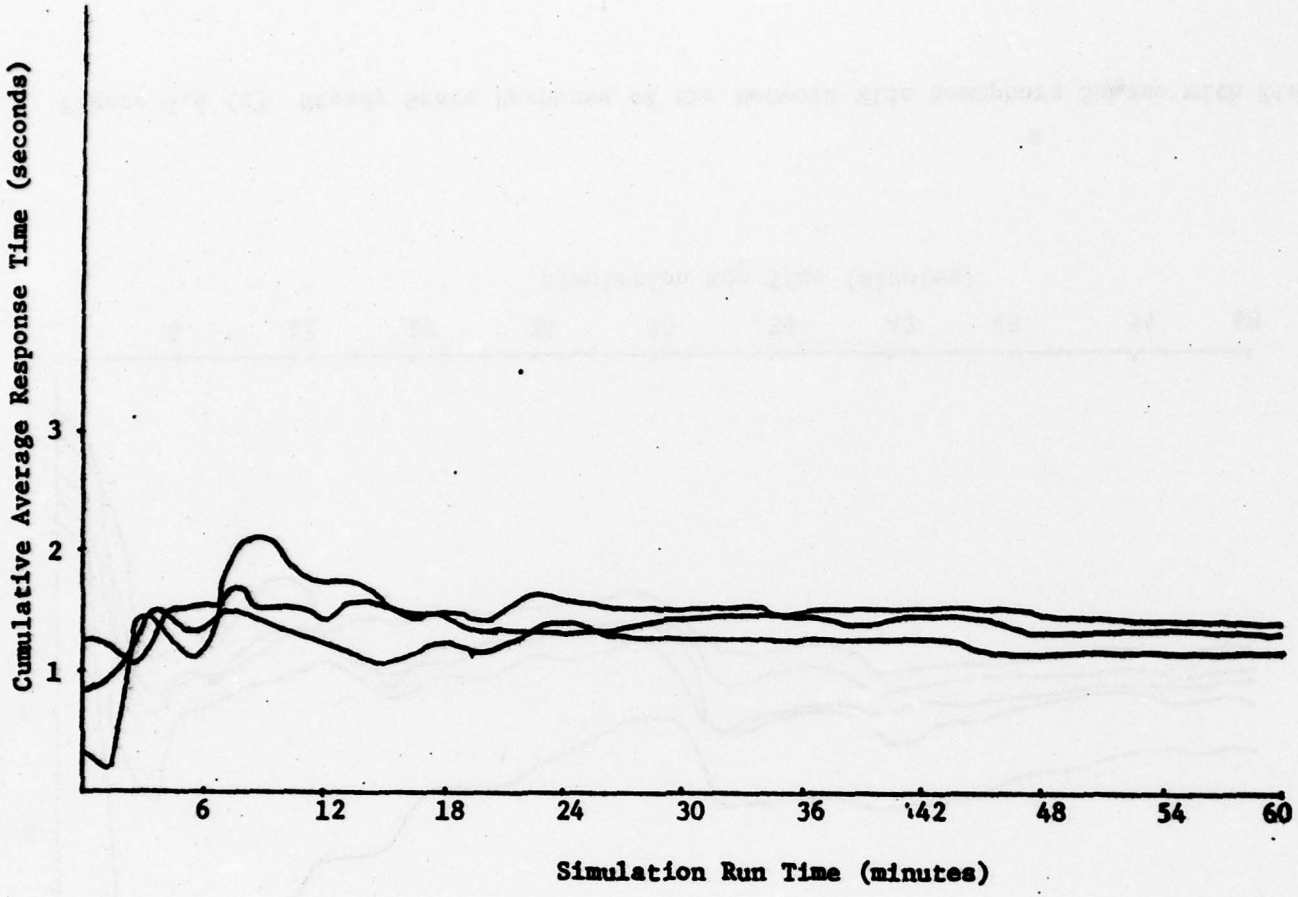


Figure 5.7 (a) Steady State Response of the Adaptive Hopping Permit Scheme with Three Nodes

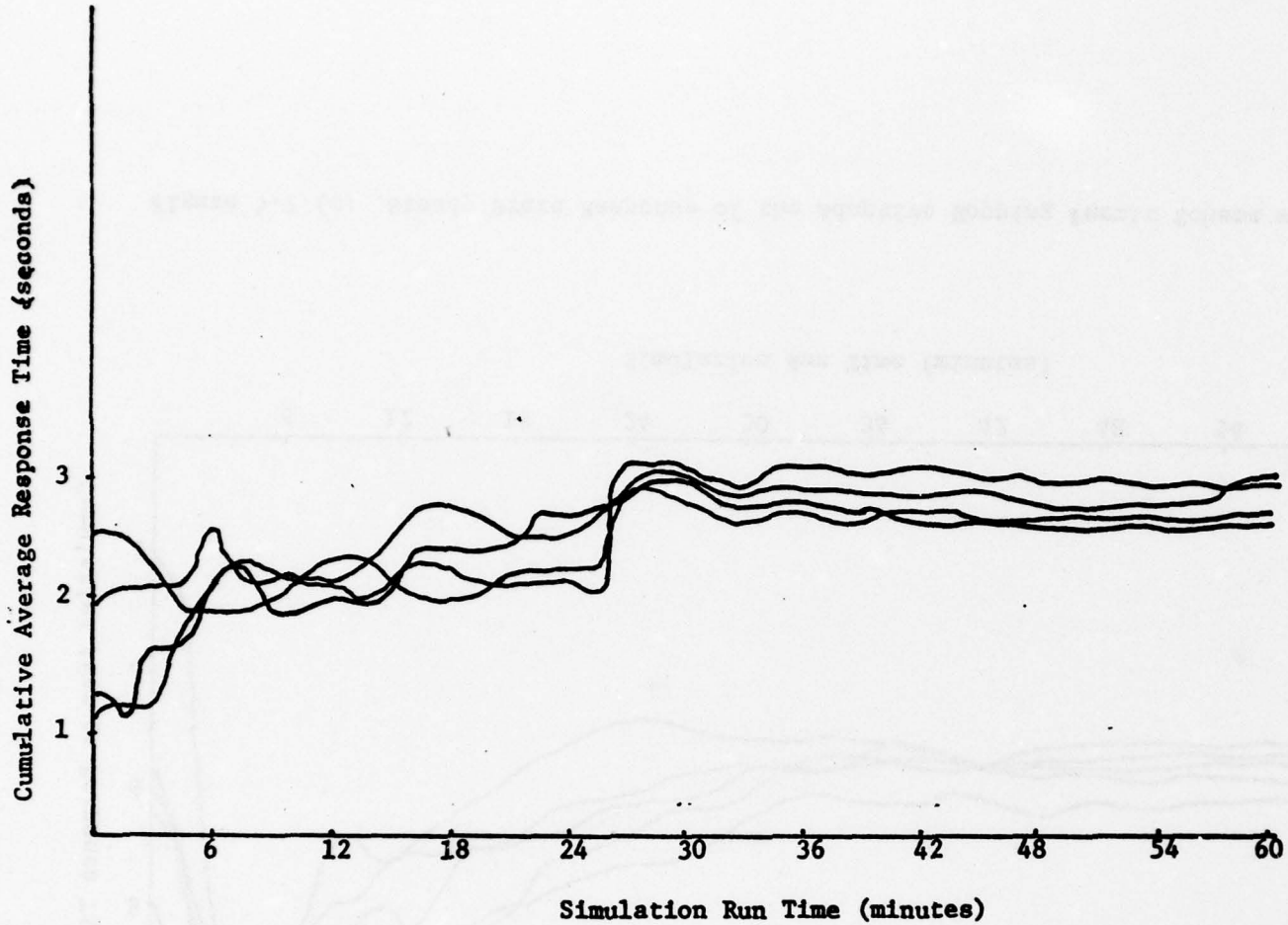


Figure 5.7 (b) Steady State Response of the Adaptive Hopping Permit Scheme with Four Nodes

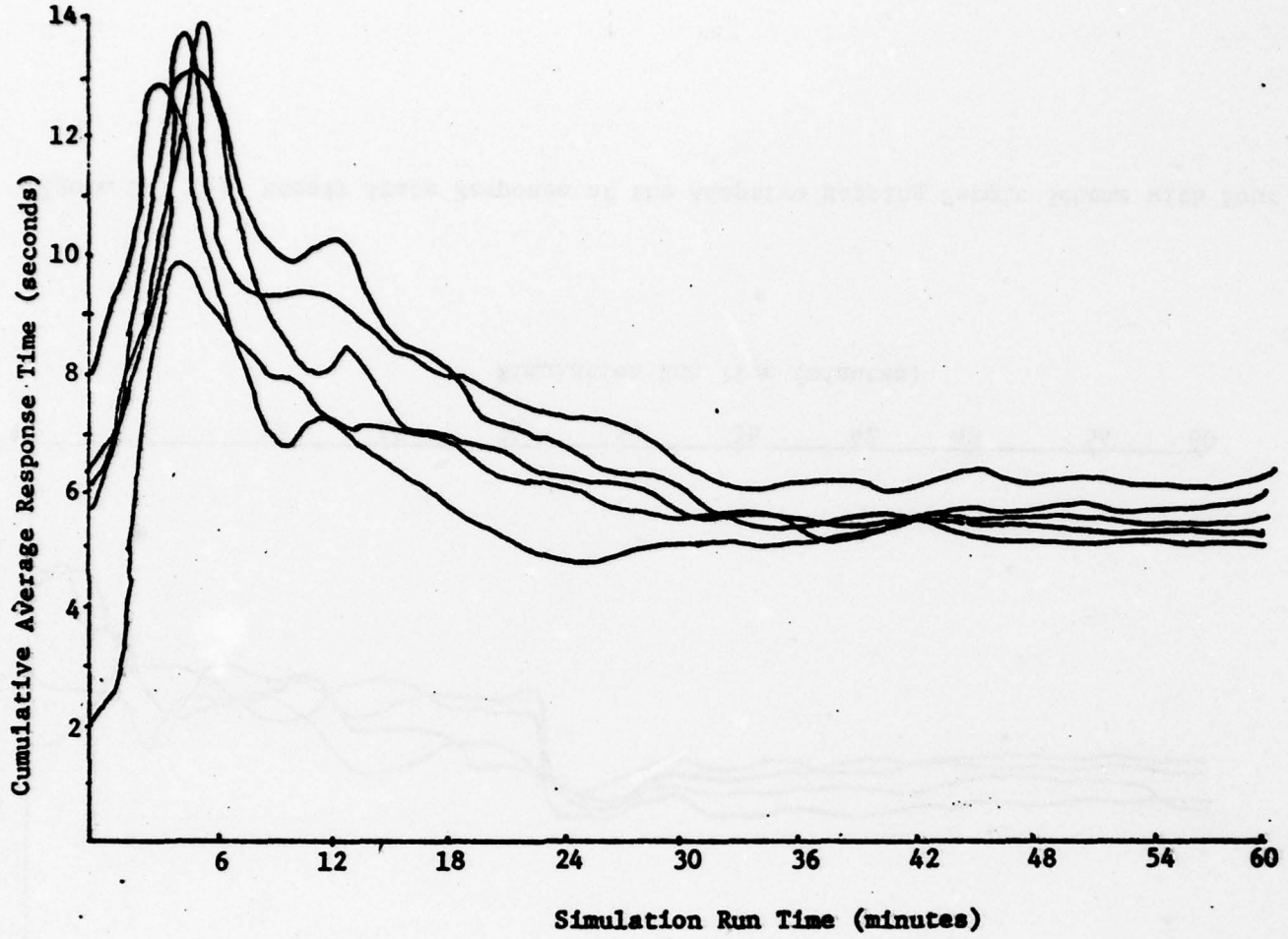


Figure 5.7 (c) Steady State Response of the Adaptive Hopping Permit Scheme with Five Nodes

It was noticed that even though steady state data collection was commenced after the estimated duration of the transient response, there would always be an initial period of fluctuations. It is felt that during this one hour time window of data collection the effect of these fluctuations can be overcome. Data collected in simulation is presented in Tables 5.2a - i. A summary of the simulated results is presented in Table 5.1. Of interests to an analyst is also the average queue length. The average number of requests waiting for service at a node is also presented.

5.3 Step Change in Average Arrival Rate of Requests

A five node network is subjected to a step change in average arrival rate of requests at one of the nodes for the measurement of response-time for each of the three control schemes. The average arrival rate of requests at node 1 is doubled half an hour later after commencement of steady state data collection. For the purpose of comparison the response-time of node 1 and node 3 is plotted both with and without the average arrival rate modification in figures 5.8 a-b-c.

5.4 Simulation Results

5.4.1 Results of the Hopping Permit Scheme

The inherent characteristics of the Hopping Permit Scheme is that the permit is routed in a deterministic manner. As the number of nodes in the simulation increases with equal request arrival rates at each node, it is observed that:

(1) Due to the deterministic routing of the permit this scheme results in long response-times.

(2) The deterioration of response-time is more of a fact in this case than those of the other two control schemes.

The response-time spread for five nodes is shown in Figure 5.12 (a). As

TABLE 5.1

THE HOPPING PERMIT SCHEME

NUMBER OF NODES	AVERAGE NUMBER IN THE QUEUE	AVERAGE RESPONSE TIME	CONTROL TRAFFIC OVERHEAD	DURATION OF TRANSIENT
3	.1047	2.107 sec.	59.17%	30 min.
4	.2114	4.0957sec.	35.96%	50 min.
5	.4713	9.055 sec.	18.55%	90 min.

THE NETWORK WIDE SEMAPHORE SCHEME

NUMBER OF NODES	AVERAGE NUMBER IN THE QUEUE	AVERAGE RESPONSE TIME	CONTROL TRAFFIC OVERHEAD	DURATION OF TRANSIENT
3	.1384	2.713 sec.	41.9 %	30 min.
4	.210	4.173 sec.	44.29%	60 min.
5	.3664	7.524 sec.	48.94%	75 min.

THE ADAPTIVE HOPPING PERMIT SCHEME

NUMBER OF NODES	AVERAGE NUMBER IN THE QUEUE	AVERAGE RESPONSE TIME	CONTROL TRAFFIC OVERHEAD	DURATION OF TRANSIENT
3	.0747	1.536 sec.	60.27%	20 min.
4	.143	2.82 sec.	36.85%	30 min.
5	.2538	5.109 sec.	20.57%	60 min.

HOPPING PERMIT SCHEME

TABLE # 5.2a

NUMBER OF NODES IN THE SIMULATION: 3

OBSERVED DURATION OF TRANSIENT : 30 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.11	190	2.088sec	4693	7873	530
	2	.092	162	2.062sec			
	3	.105	179	2.125sec			
2	1	.098	175	2.036sec	4692	7991	549
	2	.108	184	2.126sec			
	3	.123	190	2.33 sec			
3	1	.084	161	1.886sec	4712	7957	540
	2	.114	189	2.184sec			
	3	.109	190	2.083sec			

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 2.107 sec

AVERAGE QUEUE CONTENTS : .1047

CONTROL-TRAFFIC-OVERHEAD: 59.17%

HOPPING PERMIT SCHEME

TABLE # 5.2b

NUMBER OF NODES IN THE SIMULATION: 4

OBSERVED DURATION OF TRANSIENT : 50 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.151	156	3.502 sec	3782	10217	715
	2	.188	165	4.1206sec			
	3	.262	202	4.6846sec			
	4	.284	193	5.310 sec			
2	1	.211	193	3.95 sec	3661	10438	753
	2	.194	173	4.057sec			
	3	.235	195	4.345sec			
	4	.230	194	4.274sec			
3	1	.226	196	4.167sec	3665	10231	729
	2	.195	174	4.04 sec			
	3	.219	202	3.917sec			
	4	.142	158	3.24 sec			

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 4.0957 sec

AVERAGE QUEUE CONTENTS : .2114

CONTROL-TRAFFIC-OVERHEAD: 35.96%

HOPPING PERMIT SCHEME

TABLE # 5.2 c

NUMBER OF NODES IN THE SIMULATION: 5

OBSERVED DURATION OF TRANSIENT : 90 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.504	201	9.037sec	2528	13800	940
	2	.322	163	7.123sec			
	3	.514	193	9.60 sec			
	4	.423	203	7.556sec			
	5	.372	181	7.403sec			
2	1	.581	198	10.57sec	2462	13974	959
	2	.448	185	8.71sec			
	3	.568	194	10.54sec			
	4	.485	205	8.519sec			
	5	.434	182	8.585sec			
3	1	.475	192	8.911sec	2680	13560	907
	2	.680	214	11.44 sec			
	3	.429	157	9.84 sec			
	4	.348	163	7.69 sec			
	5	.486	181	9.67 sec			

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 9.055 sec

AVERAGE QUEUE CONTENTS : .4713

CONTROL-TRAFFIC-OVERHEAD: 18.55%

NETWORK WIDE SEMAPHORE SCHEME

TABLE # 5.2 d

NUMBER OF NODES IN THE SIMULATION: 3

OBSERVED DURATION OF TRANSIENT : 30 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.144	194	2.679 sec	REQ: 1404	6226	594
	2	.174	206	3.042 sec	RACK:1238		
	3	.154	195	2.95 sec	TOT: 2642		
2	1	.146	189	2.786 sec	REQ: 1214	5576	540
	2	.114	161	2.57 sec	RACK:1108		
	3	.135	190	2.564 sec	TOT: 2322		
3	1	.118	162	2.643 sec	REQ: 1172	5401	524
	2	.130	179	2.62 sec	RACK:1073		
	3	.126	183	2.493 sec	TOT: 2245		

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 2.713 sec

AVERAGE QUEUE CONTENTS : .1384

CONTROL-TRAFFIC-OVERHEAD: 41.9%

NETWORK WIDE SEMAPHORE SCHEME

TABLE # 5.2 e

NUMBER OF NODES IN THE SIMULATION: 4

OBSERVED DURATION OF TRANSIENT : 60 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.234	189	4.473sec	REQ: 2839	11810	730
	2	.2	169	4.276sec	RACK:2372		
	3	.223	191	4.214sec			
	4	.205	181	4.087sec	TOT: 5211		
2	1	.193	181	3.83 sec	REQ: 2856	11665	719
	2	.224	192	4.2 sec	RACK:2330		
	3	.211	184	4.14 sec			
	4	.213	164	4.69 sec	TOT: 5186		
	1	.19	174	3.933sec	REQ: 2851	11827	730
	2	.226	196	4.157sec	RACK:2389		
	3	.231	202	4.129sec			
	4	.173	158	3.962sec	TOT: 5240		

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 4.173 sec

AVERAGE QUEUE CONTENTS : .210

CONTROL-TRAFFIC-OVERHEAD: 44.29%

NETWORK WIDE SEMAPHORE SCHEME

TABLE # 5.2 f

NUMBER OF NODES IN THE SIMULATION: 5

OBSERVED DURATION OF TRANSIENT : 75 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.246	177	5.05	REQ: 5069	19013	837
	2	.284	171	5.991			
	3	.226	150	5.443	RACK:3869		
	4	.227	176	5.666			
	5	.308	164	6.768	TOT: 8938		
2	1	.335	185	6.536	REQ: 6107	21190	895
	2	.400	196	7.364			
	3	.335	162	7.443	RACK:4299		
	4	.376	178	7.615			
	5	.384	175	7.902	TOT: 10406		
	1	.421	174	8.713	REQ: 6765	22436	924
	2	.459	176	9.406			
	3	.451	169	9.619	RACK:4552		
	4	.588	224	9.450			
	5	.457	182	9.051	TOT: 11317		

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 7.524 sec

AVERAGE QUEUE CONTENTS : .3664

CONTROL-TRAFFIC-OVERHEAD: 48.94%

ADAPTIVE HOPPING PERMIT SCHEME

TABLE # 5.2 g

NUMBER OF NODES IN THE SIMULATION: 3

OBSERVED DURATION OF TRANSIENT : 20 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.067	167	1.459sec	4891	7908	504
	2	.056	164	1.249			
	3	.073	173	1.531			
2	1	.066	161	1.495sec	4759	7998	539
	2	.088	189	1.679			
	3	.079	189	1.517			
3	1	.078	187	1.599sec	4766	8012	575
	2	.086	200	1.649			
	3	.08	188	1.595			

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 1.536

AVERAGE QUEUE CONTENTS : .0747

CONTROL-TRAFFIC-OVERHEAD: 60.27%

ADAPTIVE HOPPING PERMIT SCHEME

TABLE # 5.2 h

NUMBER OF NODES IN THE SIMULATION: 4

OBSERVED DURATION OF TRANSIENT : 30 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.164	194	3.05 sec	3796	10265	729
	2	.145	167	3.142			
	3	.131	179	2.651			
	4	.145	190	2.754			
2	1	.147	194	2.737sec	3760	10271	735
	2	.148	189	2.823			
	3	.152	163	3.362			
	4	.135	189	2.573			
3	1	.147	196	2.714sec	3784	10237	730
	2	.136	174	2.823			
	3	.151	202	2.695			
	4	.115	158	2.636			

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 2.82 sec

AVERAGE QUEUE CONTENTS : .143

CONTROL-TRAFFIC-OVERHEAD: 36.85%

ADAPTIVE HOPPING PERMIT SCHEME

TABLE # 5.2 1

NUMBER OF NODES IN THE SIMULATION: 5

OBSERVED DURATION OF TRANSIENT : 60 minutes

SET	QUEUE NUMBER	AVERAGE CONTENTS OF QUEUE	NUMBER OF REQUESTS PER NODE	AVERAGE RESPONSE TIME	NUMBER OF OVERHEAD MESSAGES	TOTAL NUMBER OF MESSAGES	TOTAL NUMBER OF REQUESTS
1	1	.247	178	5.0 sec	2734	13597	905
	2	.320	200	5.774			
	3	.294	201	5.275			
	4	.253	166	5.504			
	5	.281	165	6.143			
2	1	.263	191	4.969sec	2710	13586	906
	2	.319	214	5.376			
	3	.222	157	5.098			
	4	.240	163	5.317			
	5	.292	181	5.827			
3	1	.223	173	4.641sec	2901	13373	873
	2	.177	158	4.054			
	3	.213	166	4.63			
	4	.238	192	4.468			
	5	.226	184	4.434			

SUMMARY OF SIMULATED RESULTS:

AVERAGE RESPONSE TIME : 5.109 sec

AVERAGE QUEUE CONTENTS : .2538

CONTROL-TRAFFIC-OVERHEAD: 20.57%

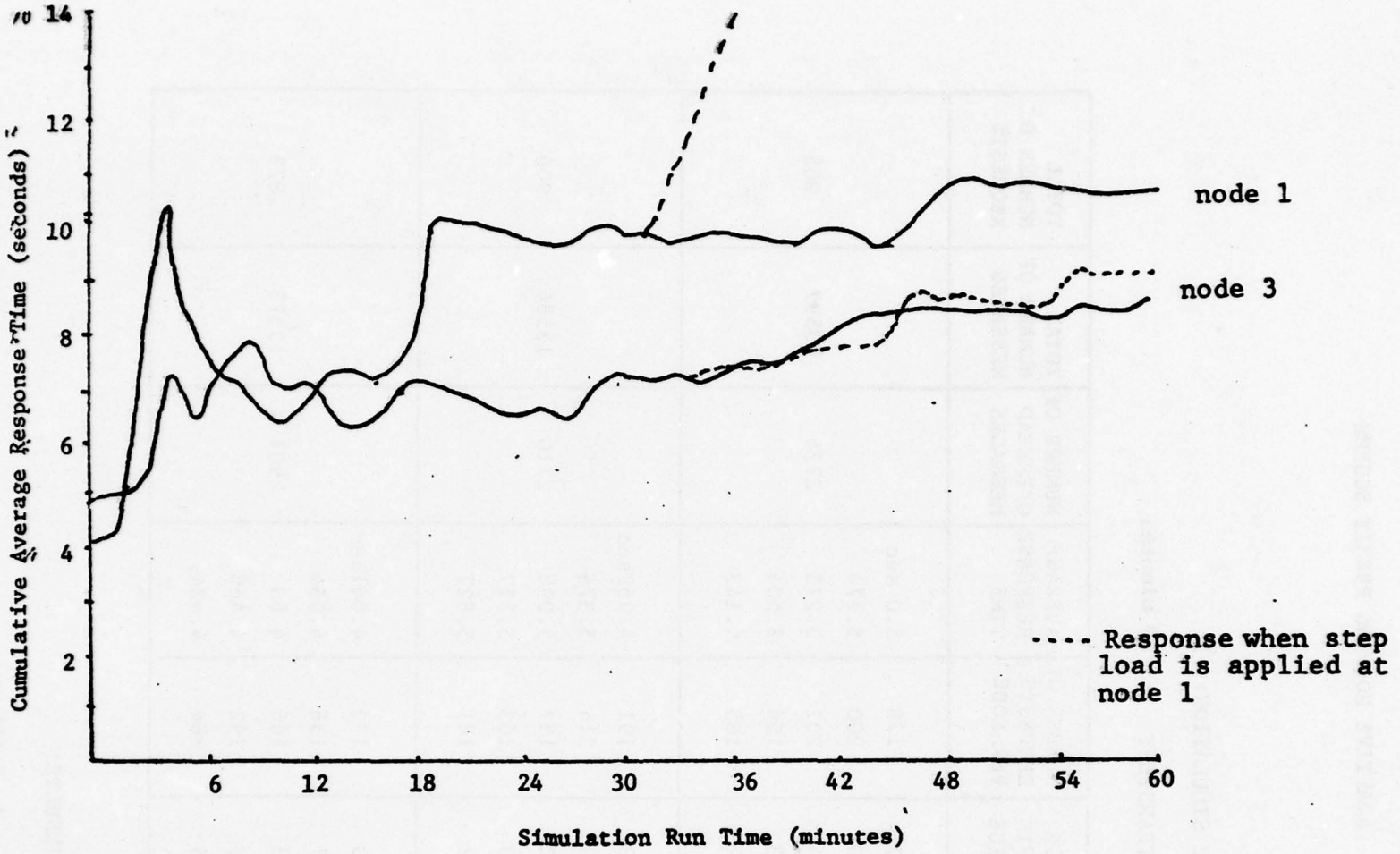


Figure 5.8 (a) The Hopping Permit Scheme's Response to a Step Load at Node One

Number of Nodes = 5

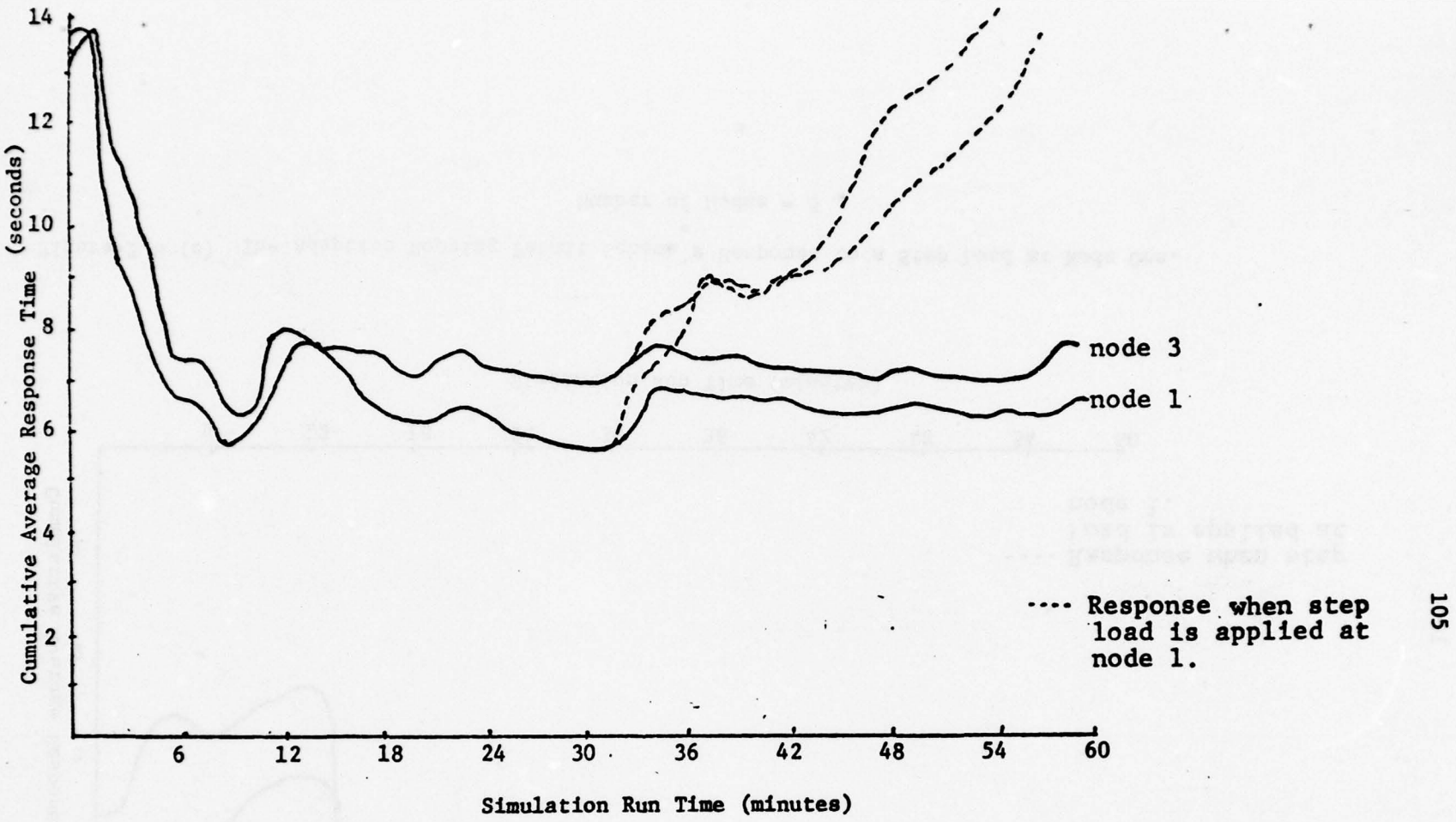


Figure 5.8 (b) The Network Wide Semaphore Scheme's Response to a Step Load at Node One.

Number of Nodes = 5

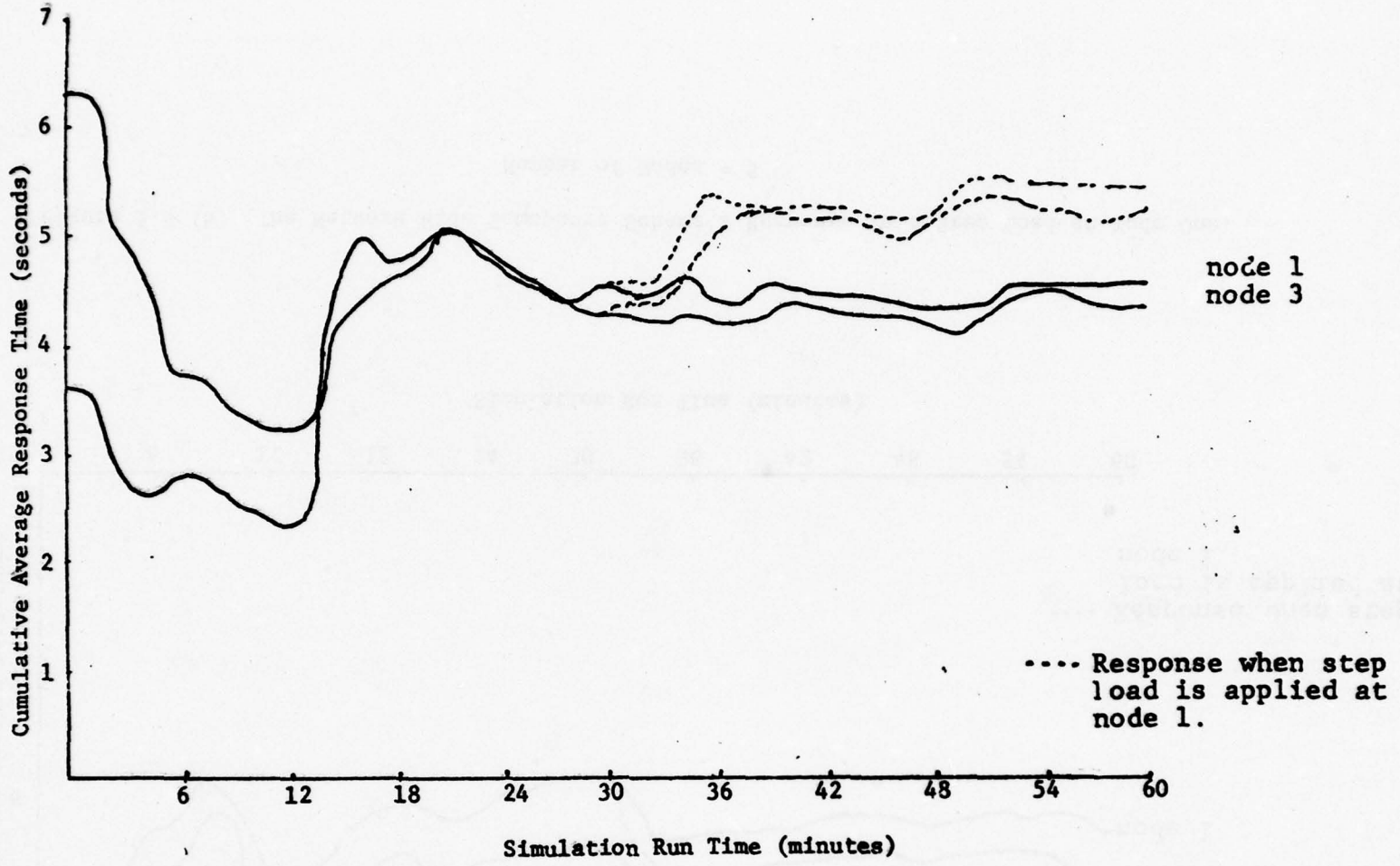


Figure 5.8 (c) The Adaptive Hopping Permit Scheme's Response to a Step Load at Node One.

Number of Nodes = 5

compared to those of the other two schemes shown in Figure 5.12 (b) both the mean and the variance are the largest. The spread of the response-time is very large and makes this scheme unattractive to the network designer.

The plot of Response-time and control-traffic-overhead for a different number of nodes is shown in Figure 5.9. The control-traffic-overhead decreases as the number of nodes in the network increases. This is so because for each additional node the update traffic increases by three messages per request. For the same duration of running time the permit is retained by the nodes for a greater period of time and sent out into the link a fewer number of times. What is experienced is a decrease in control traffic and an increase in update traffic and total traffic.

The response-time to the step load change at node 1 is very poor. The other nodes experience a slight increase in response-time. But, at the node where the step was applied the response-time deteriorates beyond recovery as shown in Figure 5.8 (a). This is due to the fact that only one request is processed when the permit comes around. The permit is not able to make the round trip fast enough to cope with the situation of increased arrival rate.

5.4.2 Results of the Network-Wide Semaphore Scheme

The Network Wide Semaphore Scheme has an improved response-time. This is explained by the fact that the requests are processed on the basis of their arrival times. Because of this feature the control scheme adapts itself to some kind of global minimization of response-time. As the number of nodes in the simulation is increased it is observed that:

(1) The response-time increases but better than that in the case of the Hopping Permit Scheme. The response-time is always greater

Hopping Permit Scheme

Number of Computers	Average Response-Time	Control-Traffic Overhead
3	2.107 sec.	59.17%
4	4.095 sec.	35.96%
5	9.055 sec.	18.55%

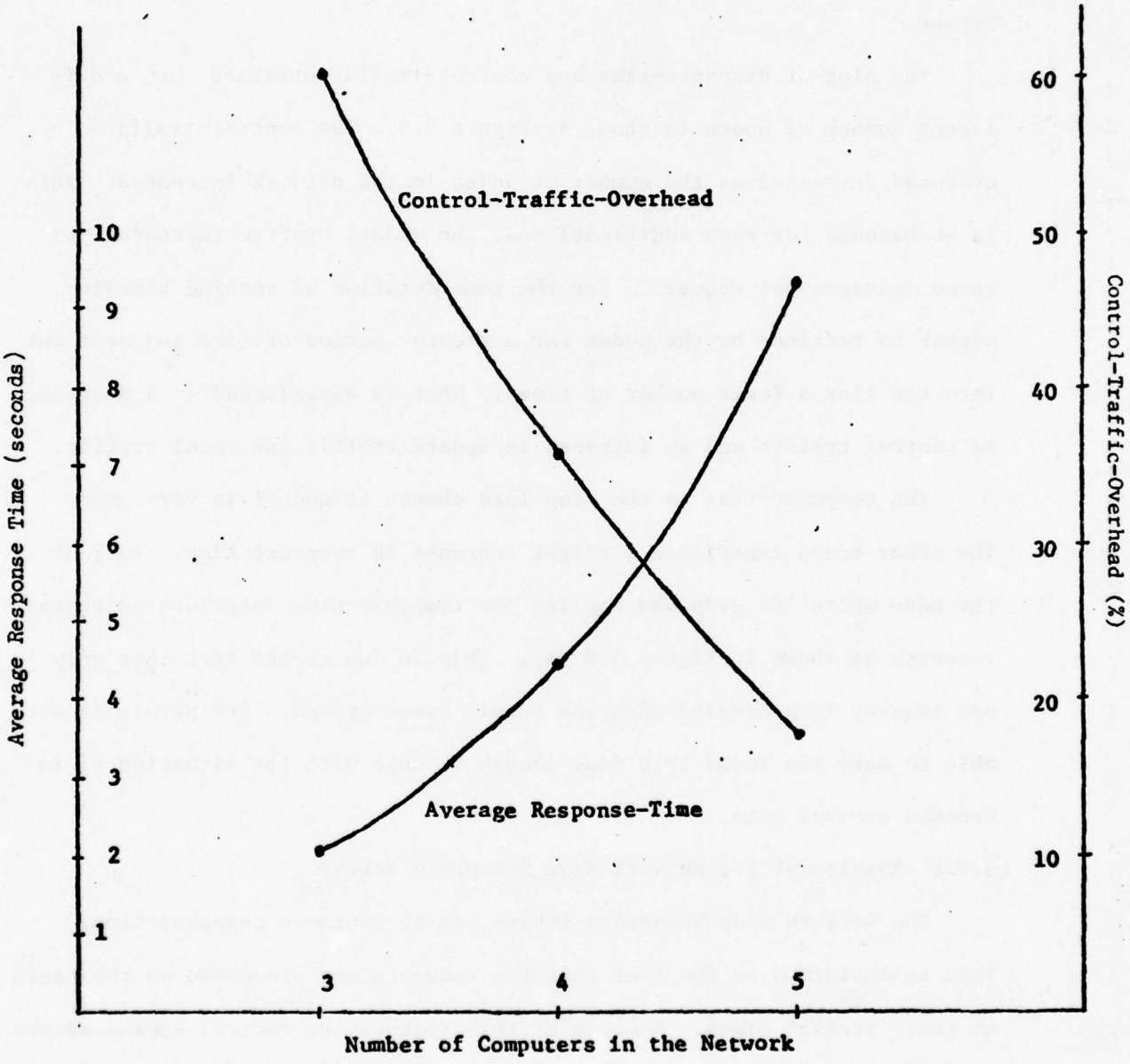


Figure 5.9 Plot of the Control-Traffic-Overhead and Average Response-Time versus the Number of Computers of the Hopping Permit Scheme.

than the Adaptive Hopping Permit Scheme.

(2) Steady State response is prone to fluctuations but these fluctuations are more evenly distributed amongst the nodes.

The control-traffic-overhead increases as the number of nodes in the network increases. Nevertheless, the update traffic per request increases the same amount as those in the other schemes. This is so because:

(1) The control traffic increases by two messages per request for every additional node.

(2) In the synchronizing phase each node with a queued up request sends the request out onto the links, regardless of whether it is the earliest or not. This accounts for part of the excess control traffic. An L.F.M. may acknowledge a number of requests before acknowledging the earliest request. Those ineffective requests and acknowledgements of traffic add to the total control traffic.

The plot of Response-Time and Control-Traffic-Overhead for different numbers of nodes is shown in Figure 5.10. The response-time spread for five nodes is shown in Figure 12b. The spread is much smaller than that of the Hopping Permit Scheme. It has an improved mean and variance.

Even though this scheme is capable of adapting to global changes, it responds very poorly to a step load. All the nodes experience a uniform increase in response-time. It is seen that the response-time in Figure 5.8b does not settle within a half hour period. The node at which the step was applied experiences a longer response time that the others. The deterioration of response-times may be explained as follows. As the arrival rate is increased at node 1, the probability that two consecutive

requests belong to node 1 is very high. Because of the open bidding procedure the synchronizing phase is slightly longer if two consecutive requests belong to the same node than that of the case if they belonged to different nodes. This is so because a node has to wait for all update acknowledgements before it can broadcast its request, after which it has to wait all request acknowledgements. On the other hand, if two consecutive requests belong to two different nodes, there is an over-lapping of the updating node's updating phase and the foreign node's synchronizing phase. Consequently, consecutive updates from the same node has a slowing down effect.

In this scheme a better response-time is obtained in comparison to the Hopping Permit Scheme, but at the expense of increased traffic and control-traffic-overhead. This excess traffic is due to a lot of ineffective requests entering the network, and a lot of ineffective acknowledgements being sent during the synchronizing phase. The other weak point is the requirement of synchronized precise clocks at every computer node; keeping accurate time information at each node of a large network is difficult (20).

5.4.3 Results of the Adaptive Hopping Permit Scheme

This scheme has a better response-time than both the Hopping Permit Scheme as well as the Network Wide Semaphore Scheme as shown in Figure 5.11. The Adaptive Hopping Permit Scheme can adapt itself to load fluctuations just as the Network Wide Semaphore Scheme. As the number of nodes are increased there are fluctuations in the steady state response of the response-time shown if Figure 5.7 (c). These fluctuations affect the response-times of all the other nodes fairly uniformly. The response-time spread for five nodes is shown in Figure 5.12 (c). The Adaptive Hop-

ping Permit Scheme is found to have the best response-time distribution with the smallest variance. This aspect of the scheme makes it very attractive to the network designer.

The Control-Traffic-Overhead decreases with the increase of the number of nodes in the simulation. This is true in both the Hopping Permit Scheme and the Adaptive Hopping Permit Scheme which is a modification of the former. However, this scheme has a slightly higher control-traffic-overhead than the Hopping Permit Scheme for the same number of nodes. Simulation runs were made of both these schemes for identical arrival patterns of file access requests. In all the cases (3, 4 and 5 nodes), the Hopping Permit Scheme had a slightly lower control-traffic-overhead. The difference is due to the adaptive nature of routing the permit. If the permit was utilized by each node everytime it came by, the control-traffic-overhead would be 16.66%, 11.11% and 8.33% for networks with 3, 4 and 5 nodes respectively for both the schemes. However, the actual simulation results showed control-traffic-overhead of 59.17%, 35.96% and 18.55% for the Hopping Permit Scheme and 60.27%, 36.85% and 20.57% for the Adaptive Hopping Permit Scheme of networkds with 3, 4 and 5 nodes respectively. The excess control traffic is due to the fact that the permit idles around in the link from node to node because lacking in requests occurred in the network. In the case of the Adaptive Hopping Permit Schemes, the service rate is faster, the average queue length is smaller (Table A), and the permit spends more time idling. Hence, the increase in the Control-Traffic-Overhead occurs.

The response-time of this scheme to a step load is favorable. It is far better than that of the case of the Network Wide Semaphore Scheme. This scheme can adapt to changes in request arrival patterns faster than

the Network Wide Semaphore Scheme, because the synchronizing phase is cut short by the permit. All nodes experience an increase in response-time, and the increase is uniform. In this scheme consecutive updates made from the same node can finish in a short time because the time spent in passing the permit is saved. It is seen here that where the Network Wide Semaphore Scheme fails is where the Adaptive Hopping Permit Scheme succeeds.

Superficially, it may seem that this scheme faces the same problem that the Network Wide Semaphore Scheme has to maintain synchronized clocks at each node. However, in the case of the Adaptive Routing Scheme instead of tagging absolute arrival times, relative waiting times of the requests are tagged. The permit would then be routed to the node with the request that has waited the longest. Therefore, precise clocks are not required at the nodes. One of the weaknesses of this scheme is the loss of the permit due to node or link breakdown. This is a weakness of the Hopping Permit Scheme too. It is possible to generate a new permit by an election process in case of failures. Le Lahn has proposed the above procedure in his recent work (14).

5.5 Comparative Discussion and Conclusion

Because no open bidding procedure happens in the Hopping Permit Scheme there is no control traffic such as request and request acknowledgement. Only permit routing traffic is involved in the synchronizing phase. Permits are passed in a deterministic route from node to node. There is no attempt to minimize the response-time. Figure 5.9 shows the variation of the average response-time increasing from 2.1 seconds to 9.0 seconds when the number of computers increases it has a faster increase rate than that of the network-wide semaphore scheme. Nonetheless, this scheme has

a very superior performance in the control-traffic-overhead which decreases from 59% to 18% when the total number of computers increases. That means only a very small percentage of traffic is involved in the synchronizing-phase of this scheme. Therefore, the simulation results have indicated clearly that from the traffic overhead point of view the hopping permit scheme is better than the network wide semaphore scheme. But, from the average response-time point of view network-wide semaphore scheme is slightly better than the hopping permit scheme. From the nature of the hopping permit scheme it is easy to notice that the only traffic in the synchronizing phase is the traffic associated with the permit. When network size grows, the percentage of the permit traffic certainly becomes small. Because the permit is passed in a deterministic route from node to node, each node has to wait longer for the next permit in a large network.

The average response-time and control-traffic-overhead of the Network Wide Semaphore Scheme have been shown in Figure 5.10 for a number of computer nodes in the networks. It is noticed that the average response-time increases nonlinearly with respect to increasing size. On the other hand, the control-traffic-overhead also increases slightly with increasing size of networks. In other words, when a networks size increases the average response-time deteriorates. During the synchronizing phase of this scheme, every local file manager is allowed to participate in an open bidding procedure. Because no global information is available, a lot of ineffective requests are allowed to enter the networks. Had the local file manager known global information, a later generated request can be prevented from entering the networks. Therefore, control-traffic-overhead of the network-wide semaphore scheme is not optimal at all. The second disadvantage

Network-Wide Semaphore Scheme

Number of Computers	Average Response-Time	Control -Traffic-Overhead
3	2.713 sec.	41.9 %
4	4.173 sec.	44.29%
5	7.524 sec.	48.94%

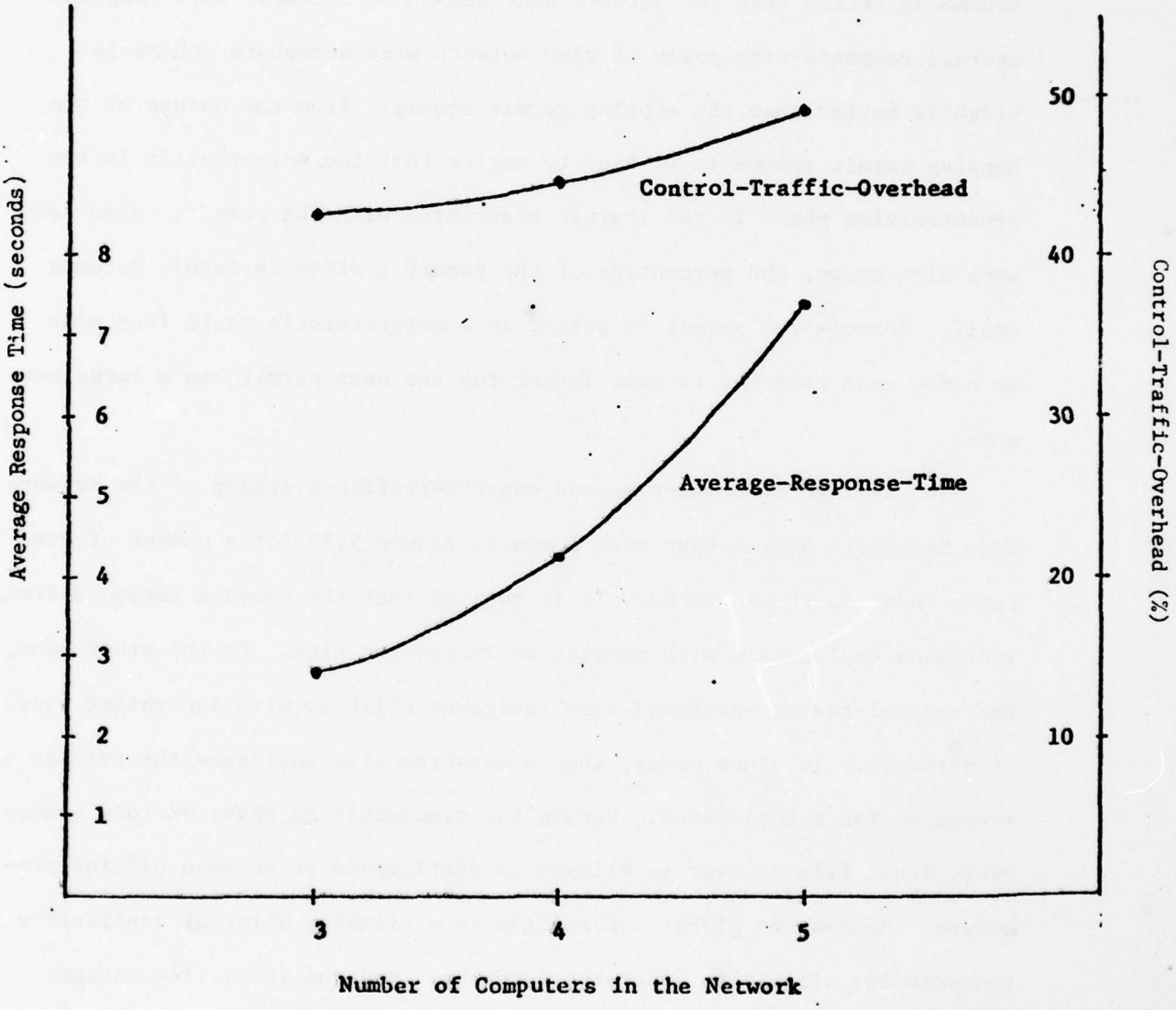


Figure 5.10 Plot of the Control-Traffic-Overhead and Average-Response-Time versus the Number of Computers of the Network-Wide Semaphore Scheme.

of this scheme is that synchronized clocks are required at the computer node. To maintain synchronized clocks at nodes of a geographically spread network is difficult. The success of this scheme depends on the time label of the request. Therefore, the network-wide semaphore scheme still has difficulty in its implementation.

The network-wide semaphore scheme has the advantage of slower deteriorating average response-time than that of the hopping permit scheme when the number of hosts in the networks is increased. The earliest request in the network will gain the file access right in the synchronizing-phase of the network-wide semaphore scheme. On the other hand, the hopping permit scheme has the advantage of decreasing control-traffic-overhead when the network size grows. Because no open bidding procedure is involved in the synchronizing-phase, control traffic does not increase with the network size. Therefore, to combine the above two advantages it is necessary to minimize the average response-time by always servicing the earliest request and to avoid open bidding procedures in the synchronizing-phase. The adaptive hopping permit scheme is conceived in this way. It will be in every respect the same as the hopping permit scheme except that the routing of the permit is not deterministic. Whenever a local file manager sends out an updating acknowledgement, an extra field is tagged in this message to indicate the earliest local request time to the file manager with a permit. The local file manager will use this information and transmit the permit to the foreign local file manager with the earliest request. Because of this routing policy for permit, the average response within the system is optimized.

It is easy to agree that the adaptive routing permit scheme is a special case of a hopping permit scheme. Because the earliest request

time is tagged in the updating acknowledgement message back to the local file manager with a permit, autonomous open bidding procedure in the synchronizing-phase of the network-wide semaphore scheme is not necessary. The adaptive hopping permit scheme has forced this autonomous open bidding procedure into a synchronized bidding procedure. In other words, from this point of veiw the adaptive hopping permit scheme is also a special case of a network-wide semaphore scheme.

Figure 5.11 shows the control-traffic-overhead and the average response-time of the adaptive hopping permit scheme versus the total number of computers. Control-traffic-overhead has shown the same advantage of decreasing trend as that of the hopping permit scheme. But, the most impressive performance of this scheme is that the average response-time increases almost linearly with the total number of computers at a small rate. For a five computer network, it can achieve 4.5 seconds average response-time. On the other hand, the average response-time of both the network-wide semaphore scheme and the hopping permit scheme show very nonlinear increasing trends at a fast rate.

In our simulation of control schemes, a steady state request arrival rate is assumed. It has been reported that the adaptive scheme usually has superior response behavior to the request load fluctuation than that of the deterministic scheme (8,9). When file access requests suddenly increase at a host in the hopping permit scheme, the route of the permit will not be affected. Consequently, congestion and severe deterioration of response-time can happen to the host with increasing requests. On the contrary, in the adaptive hopping permit scheme the permit will be directed more frequently to the heavy loaded host because more requests are waiting there.

Adaptive Hopping Permit Scheme

Number of Computers	Average Response-Time	Control-Traffic Overhead
3	1.536 sec.	60.27%
4	2.82 sec.	36.85%
5	5.109 sec.	20.57%

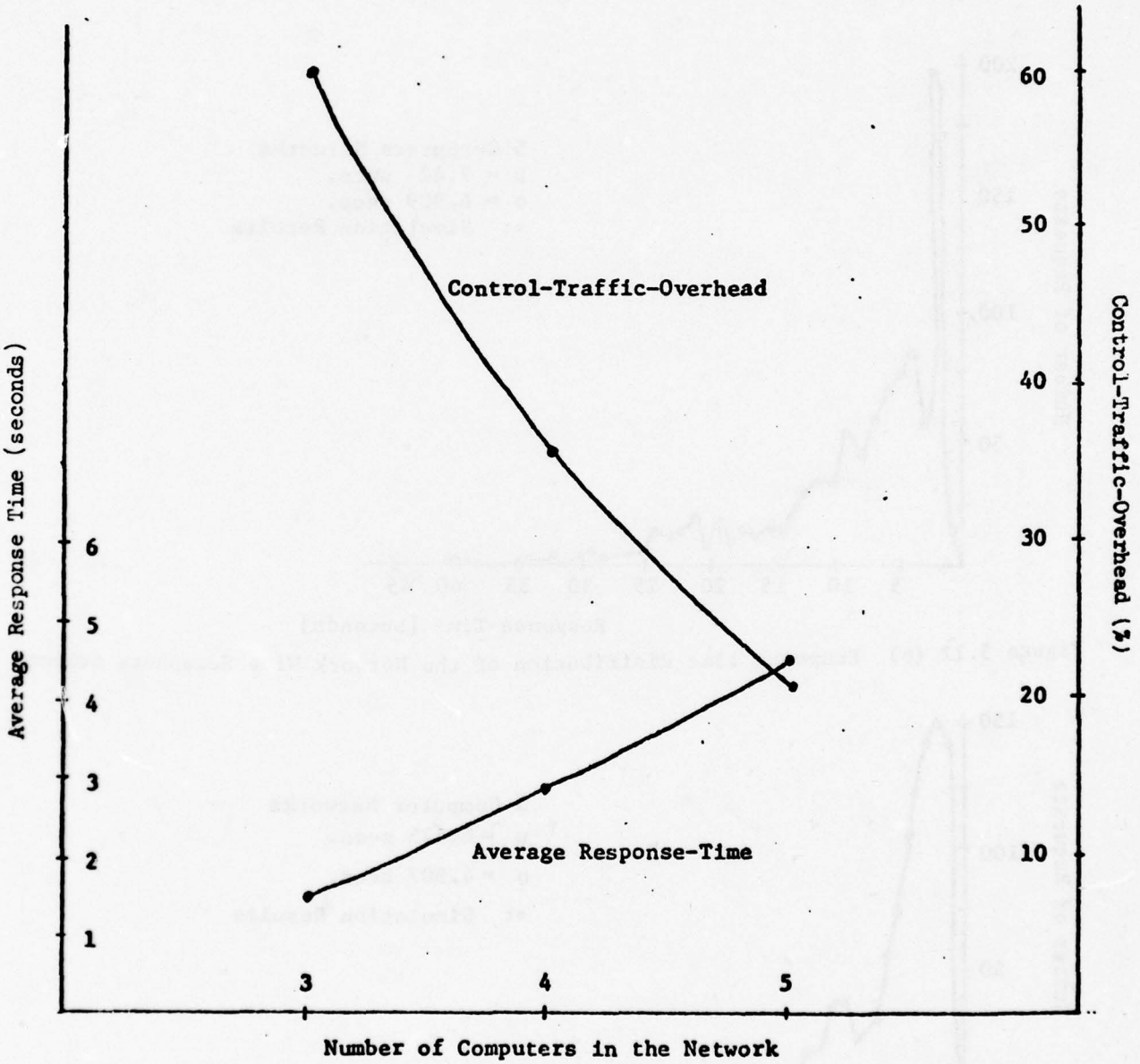


Figure 5.11 Plot of the Control-Traffic-Overhead and Average Response-Time versus the Number of Computers of the Adaptive Hopping Permit Scheme.

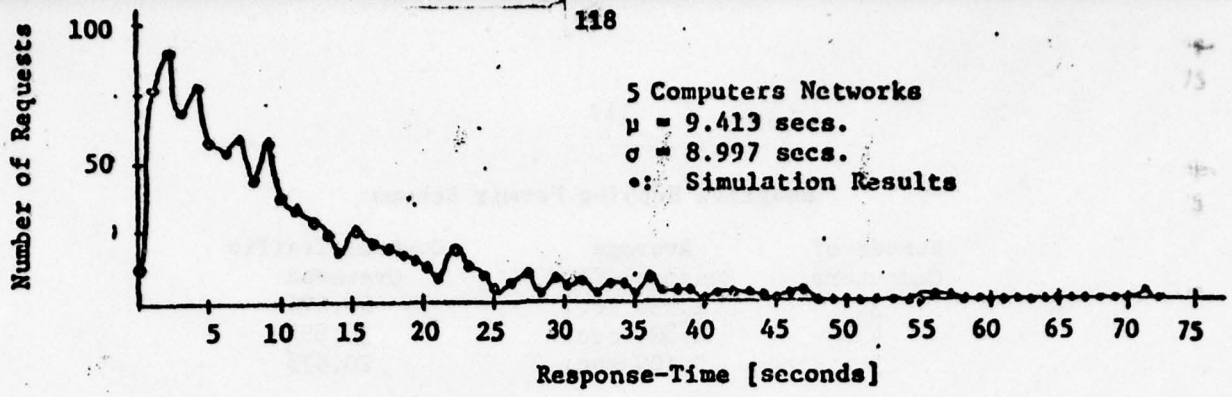


Figure 5.12 (a) Response time distribution of the Hopping Permit Scheme..

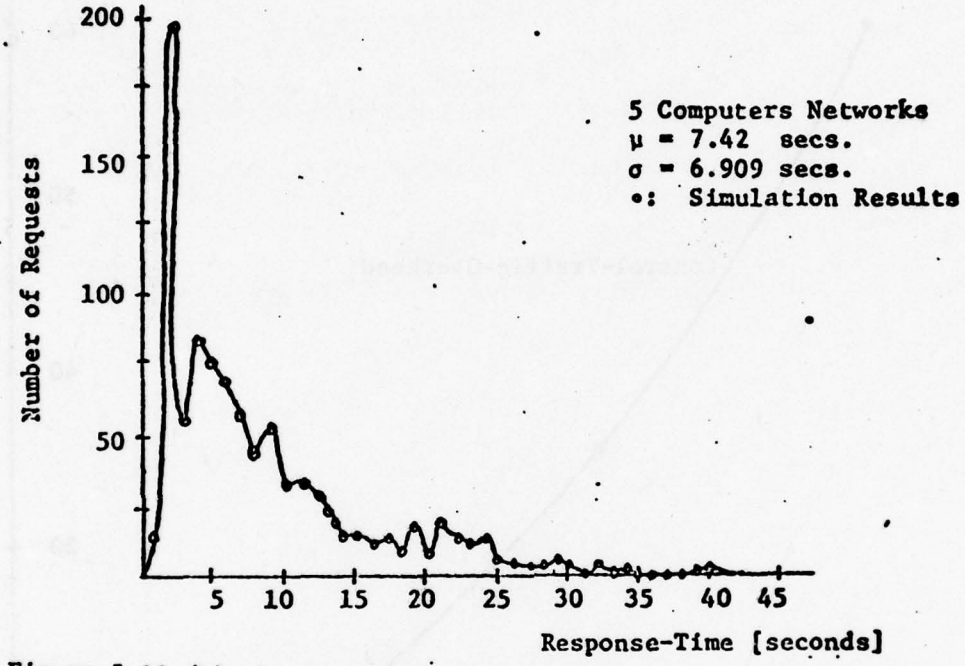


Figure 5.12 (b) Response time distribution of the Network Wide Semaphore Scheme.

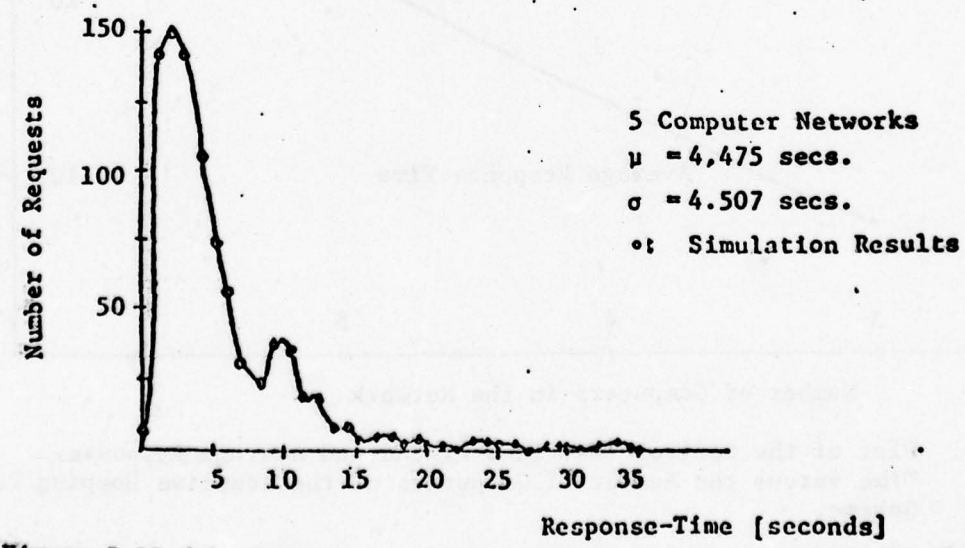


Figure 5.12 (c) Response time distribution of the Adaptive Hopping Permit Scheme.

Assume that in the network-wide semaphore scheme there was only one host computer requesting file access. After the host sent out all the requests, it experienced an outage. Assume further that all other hosts responded with their positive acknowledgements and went into a critical session. Consequently, the file is dead-locked and no further access of the file is possible. Similar situations can also happen in the hopping permit scheme. If the host with the permit goes down, the permit is trapped and other copies of the same file are deadly locked. Conceivably there are methods to decrease the probability of dead lock situtations for the network-wide semaphore scheme, a time out can be set up in the local file manager when it goes into a critical state. It will expect to receive updating information within the time-out period. Otherwise, it will assume that some abnormal situation has happened and that the original host in the requesting state is down. Consequently, the local file managers in critical state can be reset back to the requesting state. For the hopping permit scheme after any host released the permit to its next neighbor, it will check the status of the receiving neighbor periodically until the permit has again been sent out. After the period of time-out if no permit has been sent out by the neighbor, the neighbor computer will be down, and the host releasing the permit perviously, will create another permit. Of course there is a finite possibility that both hosts are down. But, the probability will be small.

Increasing the number of nodes with the same arrival rate of requests as the other nodes will increase the total rate at which requests enter the system. This indirectly amounts to an increase of the utilization factor of the total system. The utilization factor is also dependent upon the control scheme. For each scheme the network with three nodes

has the least utilization factor. A network with five nodes has the highest utilization factor in the simulation studies. The adaptive hopping permit has both the best response-time as well as the lowest control-traffic-overhead. For networks with a moderately high utilization factor the adaptive hopping permit scheme is the most attractive shceme. The adaptive hopping permit scheme has other very desirable features such as efficient adaptations to load fluctuations. It also has the best time distribution. Taking into consideration the various factors discussed the adaptive hopping permit scheme has the most desirable features and is a feasible Control Scheme for varied traffic conditions and computer network size.

References

[1] Pepe, M.E., and Fry, J.P., "Distributed Data Bases - A Summary of Research", Computer Networks 1 (1976), 130-138.

[2] Thomas, Robert H. "A Resource Sharing Executive for the ARPANET", Proc. AFIPS 1973 National Computer Conference, Vol. 42, AFIPS Press, Montvale, NJ, 1973, pp. 155-164.

[3] Kimbleton, S.R. and Schneider, G.M., "Computer Communications Networks Approaches, Objectives, and Performance Considerations", ACM Computing Surveys, Vol. 7, No. 3, Sept. 1975.

[4] Carlson, W.E. and Crocker, S.D., "The Impact of Networks on the Software Marketplace", Proc. Electronic and Aeorspace Conference, Washington, D.C., Oct. 1974.

[5] Chu, W.W., "Optimal File Allocation in a Multiple Computer System", IEEE Transaction Computers, c-18(10), 1969, 885-889.

[6] Whitney, V.K.M., "A Study of Optimal File Assignment and Communication Network Configuration" Ph.D. Dissertation, University of Michigan, 1970.

[7] Casey, R.G., "Allocation of Copies of a File in an Information Network", Proc. of AFIPS Vol. 40, 1972.

[8] Levin, K.D., "Organizing Distributed Data Bases in Computer Networks", Ph.D. Dissertation, University of Pennsylvania, 1974.

[9] Dijkstra, E.W., "Self-stabilizing Systems in Spite of Distributed Control", Communication of the ACM, Vol. 17, No. 11, Nov. 1974.

[10] Johnson, P.R., and Thomas, R.H., "The Maintenance of Duplicate Data Bases", ARPA Network Working Group, R.F.C. # 677, NIC #31507, Jan. 1975.

[11] Sutherland, W.R., "Distributed Computation Research at B.B.N., Vol. III B.B.N. Technical Report 2976, Dec. 1974, N.T.I.S. Cat. No. AD/A 003479.

[12] Dijkstra, E.W., "Cooperating Sequential Processes", in Programming Languages, F. Genuys Ed., Academic Press, NY, 1968, pp. 43-112.

[13] Rothnie J., Berstein P., and Goodman N., "Analysis of Serializability in SDD-1: A System for Distributed Data Bases (The Fully Redundant Case)", Proc. of COMPSAC 77, Nov. 8-11, 1977.

[14] Le Lahn G., "Distributed System - Towards a Formal Approach" Proc. of IFIPS, Information Processing '77. Aug. 81-2, 1977.

[15] P. Gilbert and W.J. Chandler, "Interference Between Communicating Parallel Processes" Communication of the ACM, Vol. 15, No. 6, June 1972.

[16] "Proceeding of the ACM Interprocess Communication Workshop", ACM Operating Systems Review, Vol. 9, No. 3, July 1975.

[17] C.H. Lee, R. Shastri, and R. Metzger, "Distributed Control Schemes for Multiple-Copies File Access in a Network Environment", proceeding of COMPSAC 77, Chicago, Nov. 8-11, 1977.

[18] C.H. Lee, "Adaptive Lockout Synchronization of Multi-Copied File Access in a Network". Proceedings of the Eleventh Hawaiian International Conference on System Sciences, Jan. 5-6, 1978.

[19] C.H. Lee and R. Shastri, "Modeling of Two Distributed Schemes for Data Synchronization in a Computer Network", proceeding of the Eleventh Annual Simulation Symposium, Tampa, Florida, March 15-17, 1978.

[20] R.H. Bittel et. al., "Clock Synchronization Through Discrete Control Corrections", IEEE Trans. on Communication, Vol. 22, No. 6, pp. 836-839, June 1974.

[21] Sargent, R.G., "Statistical Analysis of Simulation Output Data", Symposium on the Simulation of Computer Systems IV'" August 10-12, 1976, pp. 39-50.

[22] Schriber, T.J., "Simulation Using G.P.S.S." John Wiley and Son 1974.

[23] General Purpose Simulation System/360. User's Manual. IBM Corporation Manual Number GH20 - 0326-4.

APPENDIX A

G.P.S.S. PROGRAM LISTING OF THE HOPPING PERMIT SCHEME

```
BLOCK
NUMBER   *LOC    OPERATION   A,B,C,D,E,F,G                    COMMENTS
                 SIMULATE
                 RMULT       42315,75482
          1      FUNCTION    RN2,C24
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
          1      VARIABLE    K3
          2      VARIABLE    K3+K3*(P1-K1)          TO SPECIFY MUP SWITCHES
          3      VARIABLE    K3*V1+(V1-K1)*P1        LAST OF ACK SW OF EACH COMP
          4      VARIABLE    V1-K1                    FOR LOOP COUNTER
         *RECEIVING SECTION VARIABLE
          5      VARIABLE    K2+K3*(P2-K1)          TO ASSIGN PER SW
         *RECEIVING SECTION VARIABLE
          6      VARIABLE    K1+K3*(P2-K1)          TO ASSIGN SUP SW
          7      VARIABLE    K3*V1+K1+(P1-K1)*(V1-K1)  1 ST OF ACK SW
          8      VARIABLE    K2+K3*(P1-K1)            PER AND MUP SW
          9      VARIABLE    K3*V1+(P1-K1)*(V1-K1)+P7
          10     VARIABLE    K8+V1
          1      BVARIABLE   LS*9+LR*8
          2      BVARIABLE   LS*10*LS*11
          1      MATRIX      X,2,3
  1               GENERATE    ,,,1,,12
  2               ASSIGN      2,1
  3               ASSIGN      3,1
  4               ASSIGN      5,2
  5               TRANSFER    ,INTO
  6               GENERATE    200,FN1,,,,12
  7       COMPA   ASSIGN      1,1
  8               TRANSFER    ,BEGIN
  9               GENERATE    200,FN1,,,,12
 10       COMPB   ASSIGN      1,2
 11               TRANSFER    ,BEGIN
 12               GENERATE    200,FN1,,,,12
 13       COMPC   ASSIGN      1,3
 14       BEGIN   ASSIGN      7,V4              SET LOOP COUNTER
 15               ASSIGN      8,V8              SET P#8=PER SW
 16               ASSIGN      9,V8              SE
 17               ASSIGN      9+,K1             SET P#9=MUP SW
 18               ASSIGN      4,V10             SET LAST OF P# FOR ACK SW
 19       SET     ASSIGN      *4,V9
 20               ASSIGN      4-,K1
 21               LOOP        7,SET
 22               ASSIGN      5,V2
 23               ASSIGN      6,V3              SPECIFY ACK SWITCHES
 24               ASSIGN      7,V4              SET LOOP COUNTER
 25               QUEUE       P1
 26               TEST F      BV1,K0
 27               LOGIC S     *5
 28               DEPART      P1
 29               ADVANCE     1
 30               ASSIGN      3,2             P$TYPE =UPDATE BEGIN
 31               SPLIT       1,BTXT           SEND UPDATE BEGIN
 32               ADVANCE     10
 33               ASSIGN      3,3             P$TYPE=UPDATE COMPLETE
 34               SPLIT       1,ETXT           SEND UPDATE COMPLETE
```

```
35                  TEST E      RV2,K1
36                  ASSIGN      5-,K1
37                  LOGIC R     *5
38                  ASSIGN      5+,K1
39                  LOGIC R     *5
40          RESET   LOGIC R     *6
41                  ASSIGN      6-,K1
42                  LOOP        7,RESET
43                  TERMINATE
44          BTXT    ASSIGN      2,1
45                  ASSIGN      7,V1
46          CYCL1   TEST NE     P1,P2,JUMP1
47                  SPLIT       1,OUT
48          JUMP1   ASSIGN      2+,K1
49                  LOOP        7,CYCL1
50                  TERMINATE
51          ETXT    ASSIGN      2,1
52                  ASSIGN      6,V7
53                  ASSIGN      7,V1
54          CYCL2   TEST NE     P1,P2,JUMP2
55                  SPLIT       1,OUT
56                  ASSIGN      6+,K1
57          JUMP2   ASSIGN      2+,K1
58                  LOOP        7,CYCL2
59                  TERMINATE
60          OUT     ADVANCE     5,4
61          INTO    TEST E      P3,1,INT10  TEST INCOMING XACT FOR TYPE
62          PER     TEST E      Q*2,K0,INT11
63          INT13   TEST E      P2,V1,INT16
64                  ASSIGN      2,1
65          INT12   ASSIGN      5,V5
66                  TRANSFER    ,OUT
67          INT16   ASSIGN      2+,K1
68                  TRANSFER    ,INT12
69          INT11   LOGIC S     *5
70                  GATE LR     *5
71                  TRANSFER    ,INT13
72          INT10   TEST E      P3,2,INT14  0  TEST INCOMING XACT FOR TYPE
73                  ASSIGN      5,V6
74                  LOGIC S     *5
75                  TERMINATE
76          INT14   TEST E      P3,3,INT15  0  TEST INCOMING 7A&3 F69  ,76
77                  ASSIGN      5,V6
78                  LOGIC R     *5
79                  ASSIGN      4,*1
80                  ASSIGN      1,*2
81                  ASSIGN      2,*4
82                  ASSIGN      3,4
83                  TRANSFER    ,OUT
84          INT15   LOGIC S     *6
85                  TERMINATE
86                  GENERATE    900,,18000,,,1,F
87                  MARK        1
88          STAT    MSAVEVALUE  1,1,1,N$PER
89                  MSAVEVALUE  1,1,2,N$OUT
90                  MSAVEVALUE  1,1,3,P1
91                  MSAVEVALUE  1,2,1,QT1
92                  MSAVEVALUE  1,2,2,QT2
93                  MSAVEVALUE  1,2,3,QT3
94                  PRINT       1,1,MX,1
95                  TERMINATE
96          TIMER   GENERATE    1000
97                  TERMINATE   1
                    START       1,NP
                    RESET
                    START       2
                    RESET
                    START       2
```

APPENDIX B

G.P.S.S. PROGRAM LISTING OF THE NETWORK WIDE SEMAPHORE SCHEME

```
BLOCK
NUMBER   *LOC    OPERATION  A,B,C,D,E,F,G                    COMMENTS
                SIMULATE
                RMULT      23501,53206
          1     FUNCTION   RN2,C24
C,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
          *THIS IS A SIMULATION OF THREE COMPUTETS IN NWSS SCHEME
            1      VARIABLE   K3
            2      VARIABLE   K1+K4*(P2-K1)          TO SPECIFY MREQ SWS
            3      VARIABLE   K2+K4*(P2-K1)          TO SPECIFY  MUP SWS
            4      VARIABLE   K3+K4*(P2-K1)          TO SPECIFY SACK SWS
            5      VARIABLE   K4+K4*(P2-K1)          TO SPECIFY  SUP SWS
            6      VARIABLE   V1-K1
            7      VARIABLE   K12+V1
            8      VARIABLE   K4*V1+(P2-K1)*(V1-K1)+P7
            9      VARIABLE   K4*V1+(P2-K1)*(V1-K1)+K1
          * RECEVEING SECTION VARIABLES
           10      VARIABLE   K1+K4*(P3-K1)          TO ASSIGNX#1 AND MREQ SW
           11      VARIABLE   K4+K4*(P3-K1)          TO ASSIGNX#4 AND  SUP SW
           12      VARIABLE   K4*V1+(P3-K1)*(V1-K1)+K1
           13      VARIABLE   K2+K4*(P3-K1)          TO ASSIGNX#2 AND  MUP
           14      VARIABLE   K3+K4*(P3-K1)          TO ASSIGNX#3 AND  SACK
            1      BVARIABLE  LS*10+LS*11+LS*12+LS*13
            2      BVARIABLE  LS*14*LS*15
            3      BVARIABLE  LS*12+BV2
            1      MATRIX     X,2,4
                   INITIAL    X1-X12,0
1                  GENERATE   200,FN1,,,,15,F
2                  ASSIGN     2,2
3                  TRANSFER   ,BEGIN
4                  GENERATE   200,FN1,,,,15,F
5                  ASSIGN     2,1
6                  TRANSFER   ,BEGIN
7                  GENERATE   200,FN1,,,,15,F
8                  ASSIGN     2,3
9         BEGIN    MARK       1
10                 ASSIGN     7,V6          SET LOOP COUNTER
11                 ASSIGN     10,V2         MREQ TO BV P#10
12                 ASSIGN     11,V3         MUP TO BV P#11
13                 ASSIGN     12,V4        SACK TO BV P#12
14                 ASSIGN     13,V5         SUP TO BV P#13
15                 ASSIGN     6,V7
16        SET      ASSIGN     *6,V8
17                 ASSIGN     6-,K1
18                 LOOP       7,SET
19                 ASSIGN     8,V2          ASSIGN MREQ SW
20                 QUEUE      P2
21                 SEIZE      P2
22        BACKA    TEST E     BV1,K0
23                 LOGIC S    *8
24                 ASSIGN     6,V2
25                 ASSIGN     4,X*6
26                 ASSIGN     6+,1
27                 SAVEVALUE  *6,P1
28                 ASSIGN     5,1
```

```
29                SPLIT     1,REQT
30                TEST E    BV3,K1
31                ASSIGN    7,V4
32                GATE LR   *7,BACK1
33                DEPART    P2
34                RELEASE   P2
35                ASSIGN    7-,K1
36                LOGIC S   *7
37                ASSIGN    7-,K1
38                LOGIC R   *7
          *RESET ACK SW
39                ASSIGN    7,V6
40                ASSIGN    9,V9
41          RST1  LOGIC R   *9
42                ASSIGN    9+,K1
43                LOOP      7,RST1
44                ASSIGN    7,V2
45                SAVEVALUE *7,P1
46                ASSIGN    7+,2
47                ASSIGN    6,V3
48                SAVEVALUE *7,X*6
49                ASSIGN    4,*1
50                ADVANCE   1
51                ASSIGN    5,3
52                SPLIT     1,BTXT
53                ADVANCE   10
54                ASSIGN    5,4
55                SPLIT     1,REQT
56                TEST E    BV2,K1
57                ASSIGN    7,V3
58                LOGIC R   *7
          *   RESET ACK SW
59                ASSIGN    7,V6
60                ASSIGN    9,V9
61          RST2  LOGIC R   *9
62                ASSIGN    9+,K1
63                LOOP      7,RST2
64                TERMINATE
65          BACK1 PRICRITY  1
66                TRANSFER  ,BACKA
67          REQT  ASSIGN    3,1
68                ASSIGN    7,V1
69                ASSIGN    9,V9
70          CYCL1 TEST NE   P2,P3,JUMP1
71                SPLIT     1,OUT
72                ASSIGN    9+,K1
73          JUMP1 ASSIGN    3+,K1
74                LOOP      7,CYCL1
75                TERMINATE
76          BTXT  ASSIGN    3,1
77                ASSIGN    7,V1
78          CYCL2 TEST NE   P2,P3,JUMP2
79                SPLIT     1,OUT
80          JUMP2 ASSIGN    3+,K1
81                LOOP      7,CYCL2
82                TERMINATE
83          OUT   ADVANCE   5,4
```

```
84                TEST E      P5,3,INC10
          *XACT IS AN UPDATE BEGIN
85                ASSIGN      6,V10
86                SAVEVALUE   *6,P1
87                ASSIGN      8,V11
88                LOGIC S     *8
89                ASSIGN      8-,K1
90                LOGIC R     *8
          *  RESET ACK SW
91                ASSIGN      9,V12
92                ASSIGN      7,V6
93          RST3  LOGIC R     *9
94                ASSIGN      9+,K1
95                LOOP        7,RST3
96                TERMINATE
97          INC10 ASSIGN      6,V10
98                TEST E      X*6,P4,DES11
99                TEST E      P5,1,INC11
          *    XACT IS A REQUEST
100         CCNT1 ASSIGN      8,V13
101               GATE LR     *8
102               BUFFER
103               ASSIGN      8-,K1
104               GATE LS     *8,INC12
105               ASSIGN      6,V14
106               ASSIGN      7,V13
107               SAVEVALUE   *6,X*7
108               TEST E      X*7,P1,INC13
109               TEST L      P2,P3,DES12
110         INC14 SAVEVALUE   *6,P1
111               ASSIGN      7,V11          ASIIGN #4 TO P#7
112               SAVEVALUE   *7,P2          COPY ACKNOWLEDGED SOURCE
113         LOC14 ASSIGN      8,V14
114               LOGIC S     *8
115               ASSIGN      8-,K2
116               LOGIC R     *8
117               TRANSFER    ,LOC11
118         INC13 TEST L      X*7,P1,INC14
119               TERMINATE
120         INC12 ASSIGN      6,V14
121               TEST E      X*6,X*8,INC15
122         INC19 SAVEVALUE   *6,P1
123         INC18 ASSIGN      7,V11
124               SAVEVALUE   *7,P2
125               ASSIGN      7,P3
126               ASSIGN      7+,V1
127               QUELE       P7
128               ASSIGN      8,V11
129               GATE LR     *8,INC16
130               TRANSFER    ,LOC13
131         INC15 TEST E      X*6,P1,INC17
132               ASSIGN      7,V11
133               TEST L      X*7,P2,INC18
134               TERMINATE
135         INC17 TEST L      X*6,P1,INC19
136               TERMINATE
137         INC16 ASSIGN      8,V11
```

```
138                GATE LR     *8
139                BUFFER
140                ASSIGN      7,P3
141                ASSIGN      7+,V1
142                DEPART      P7
143                ASSIGN      6,V14
144                TEST E      X*6,P1,DES13
145                TEST E      X*8,P2,DES13
146                ASSIGN      8-,K3
147                GATE LS     *8,LOC15
148                ASSIGN      7,V13
149                TEST L      X*7,P1,LOC14
150                TERMINATE
151          LOC13 ASSIGN      7,P3
152                ASSIGN      7+,V1
153                DEPART      P7
154          LOC15 ASSIGN      8,V14
155                LOGIC S     *8
156          LOC11 ASSIGN      5,2
157          LOC12 ASSIGN      6,*2
158                ASSIGN      2,*3
159                ASSIGN      3,*6
160                TRANSFER    ,OUT
161          INC11 TEST E      P5,2,REC10
162          CONT2 LOGIC S     *9
163                TERMINATE
164          REC10 TEST E      P5,4,REC11
165                ASSIGN      5,5
166                ASSIGN      8,V11
167                LOGIC R     *8
168                TRANSFER    ,LOC12
169          REC11 LOGIC S     *9
170                TERMINATE
171          DES11 TERMINATE
172          DES12 TERMINATE
173          DES13 TERMINATE
174                GENERATE    900,,,,,1,F
175                MARK        1
176                MSAVEVALUE  1,1,1,N$CONT1
177                MSAVEVALUE  1,1,2,N$CONT2
178                MSAVEVALUE  1,1,3,N$OUT
179                MSAVEVALUE  1,1,4,P1
180                MSAVEVALUE  1,2,1,QT1
181                MSAVEVALUE  1,2,2,QT2
182                MSAVEVALUE  1,2,3,QT3
183                PRINT       1,1,MX,1
184                TERMINATE   1
                   START       40
                   CLEAR
                   START       40
                   END
```

APPENDIX C

G.P.S.S. PROGRAM LISTING OF THE ADAPTIVE HOPPING PERMIT SCHEME

```
BLOCK
NUMBER    *LOC    OPERATION   A,B,C,D,E,F,G                COMMENTS
                  SIMULATE
          * ADAPTIVE HOPPING PERMIT SCHEME WITH            COMPUTERS
                  RMULT       23501,53206
            1     FUNCTION    RN2,C24
          0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.
          .8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3
          .97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
            1     VARIABLE    K3
            2     VARIABLE    V1-K1                        FOR LOOP COUNTER
            3     VARIABLE    K2+K3*(P1-K1)                PER AND MUP SW
            4     VARIABLE    K9+V1
            5     VARIABLE    K3*V1+(P1-K1)*(V1-K1)+P7
            6     VARIABLE    K3+K3*(P1-K1)          TO SPECIFY MUP SWITCHES
            7     VARIABLE    K3*V1+(V1-K1)*P1     LAST OF ACK SW OF EACH COMP
            8     VARIABLE    K3*V1+K1+(P1-K1)*(V1-K1) 1 ST OF ACK SW
            9     VARIABLE    K2+K3*(P2-K1)                TO ASSIGN PER SW
            10    VARIABLE    K1+K3*(P2-K1)                TO ASSIGN SUP SW
            11    VARIABLE    K3+K3*(P2-K1)
            12    VARIABLE    K3+K3*(P1-K1)
            1     BVARIABLE   LS*9+LR*8
            2     BVARIABLE   LS*11*LS*12
1                 GENERATE    ,,,1,,12
2                 ASSIGN      2,1
3                 ASSIGN      3,1
4                 ASSIGN      5,2
5                 TRANSFER    ,INTO
6                 GENERATE    200,FN1,,,,14,F
7           COMPA ASSIGN      1,1
8                 TRANSFER    ,BEGIN
9                 GENERATE    200,FN1,,,,14,F
10          COMPB ASSIGN      1,2
11                TRANSFER    ,BEGIN
12                GENERATE    200,FN1,,,,14,F
13          COMPC ASSIGN      1,3
14          BEGIN ASSIGN      7,V2              SET LOOP COUNTER
15                MARK        10
16                ASSIGN      8,V3              SET P#8=PER SW
17                ASSIGN      9,V3              SE
18                ASSIGN      9+,K1             SET P#9=MUP SW
19                ASSIGN      4,V4              SET LAST OF P# FOR ACK SW
20          SET   ASSIGN      *4,V5
21                ASSIGN      4-,K1
22                LOOP        7,SET
23                ASSIGN      5,V6
24                ASSIGN      6,V7              SPECIFY ACK SWITCHES
25                ASSIGN      7,V2              SET LOOP COUNTER
26                QUEUE       P1
27                SEIZE       P1
28                ASSIGN      4,V12
29                SAVEVALUE   *4,*10
30                TEST E      BV1,K0
31                RELEASE     P1
32                LOGIC S     *5
33                DEPART      P1
34                ADVANCE     1
```

```
35                ASSIGN      3,2              P$TYPE =UPDATE BEGIN
36                SPLIT       1,BTXT            SEND UPDATE BEGIN
37                ADVANCE     10
38                ASSIGN      3,3              P$TYPE=UPDATE COMPLETE
39                SPLIT       1,ETXT            SEND UPDATE COMPLETE
40                ASSIGN      4,V3
41                SAVEVALUE   *4,P1
42                TEST E      F*1,1,MTQ
43                ASSIGN      4,V12
44                SAVEVALUE   *5,*4
45          RET   TEST E      BV2,K1
46                ASSIGN      5-,K1
47                LOGIC R     *5
48                ASSIGN      5+,K1
49                LOGIC R     *5
50          RESET LOGIC R     *6
51                ASSIGN      6-,K1
52                LOOP        7,RESET
53                TERMINATE
54          MTQ   MARK        10
55                ASSIGN      10+,K100
56                SAVEVALUE   *5,*10
57                TRANSFER    ,RET
58          BTXT  ASSIGN      2,1
59                ASSIGN      7,V1
60          CYCL1 TEST NE     P1,P2,JUMP1
61                SPLIT       1,OUT
62          JUMP1 ASSIGN      2+,K1
63                LOOP        7,CYCL1
64                TERMINATE
65          ETXT  ASSIGN      2,1
66                ASSIGN      6,V8
67                ASSIGN      7,V1
68          CYCL2 TEST NE     P1,P2,JUMP2
69                SPLIT       1,OUT
70                ASSIGN      6+,K1
71          JUMP2 ASSIGN      2+,K1
72                LOOP        7,CYCL2
73                TERMINATE
74          OUT   ADVANCE     5,4
75          INTO  TEST E      P3,1,INT10   TEST INCOMING XACT FOR TYPE
76          PER   TEST E      Q*2,K0,INT11
77          INT13 TEST E      P2,V1,INT16
78                ASSIGN      2,1
79          INT12 ASSIGN      5,V9
80                TRANSFER    ,OUT
81          INT16 ASSIGN      2+,K1
82                TRANSFER    ,INT12
83          INT11 LOGIC S     *5
84                GATE LR     *5
85                ASSIGN      1,*2               ASSIGN P#1=P#2
86                ASSIGN      2,X*5              ASSIGN NEXT DESTINATION
87                TEST NE     *1,*2,PER          TEST  IF NEXT DESTINATION IS SELF
88                ASSIGN      5,V9
89                TRANSFER    ,OUT
90          INT10 TEST E      P3,2,INT14    0  TEST INCOMING XACT FOR TYPE
91                ASSIGN      5,V10
```

```
92                LOGIC S     *5
93                TERMINATE
94          INT14 TEST E      P3,3,INT15    0  TEST INCOMING 7A83 F69   ,7E
95                ASSIGN      5,V10
96                LOGIC R     *5
97                ASSIGN      4,*1
98                ASSIGN      1,*2
99                ASSIGN      2,*4
100               ASSIGN      3,4
101               TEST E      0*1,K0,YESQ
102               MARK        10
103               ASSIGN      10+,K300
104               TRANSFER    ,OUT
105         YESQ  ASSIGN      5,V6
106               ASSIGN      10,X*5
107               TRANSFER    ,OUT
108         INT15 ASSIGN      7,V11
109               TEST L      *10,X*7,IGNOR
110               SAVEVALUE   *7,*10
111               ASSIGN      7-,K1
112               SAVEVALUE   *7,*1          COPY SOURCE
113         IGNOR LOGIC S     *6
114               TERMINATE
115               GENERATE    18000
116               TERMINATE   1
                  START       1
                  RESET
                  START       2
                  END
```