

A GRAPHICAL SIMULATION TOOL FOR  
LOGICAL TOKEN-BASED DISTRIBUTED  
MUTUAL EXCLUSION ALGORITHMS

By

LATA R.N. SINGH

Master of Science (Physics)

School of Sciences, Gujarat University,

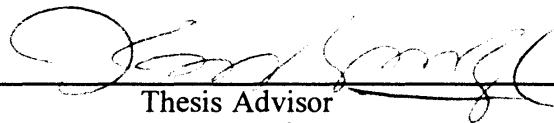
Gujarat, INDIA

1988

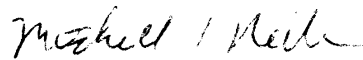
Submitted to the faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirement for  
the Degree of  
MASTER OF SCIENCE  
December 1994

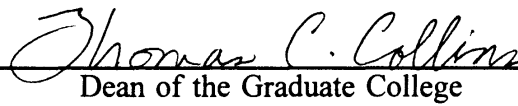
A GRAPHICAL SIMULATION TOOL FOR  
LOGICAL TOKEN-BASED DISTRIBUTED  
MUTUAL EXCLUSION ALGORITHMS

Thesis Approved:

  
\_\_\_\_\_  
Thesis Advisor

  
\_\_\_\_\_  
H. Lu

  
\_\_\_\_\_  
Michael Neith

  
\_\_\_\_\_  
Dean of the Graduate College

## ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my major advisor, Dr. K.M. George, for his guidance, inspiration, and friendship. Many thanks also go to Dr. Mitch L. Neilsen without whose constant guidance and valuable suggestion would not have led me to the successful completion of this work. My sincere thanks to Dr. Huizhu Lu for serving on my graduate committee.

Special thanks go to Dr. Dan Storm, who supported me during my masters course and gave opportunity to apply my skills in the area of Agriculture Engineering.

My deepest gratitude is due to those nearest my heart. Many thanks to my supporting and loving husband, who inspired me and helped me in the completion of this thesis. Finally, I would like to express my gratitude to my parents who provided everything I wanted in my life till now and to my brother and sisters who constantly encouraged me to achieved my goals.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION .....	1
II. OVERVIEW OF MUTUAL EXCLUSION IN DISTRIBUTED SYSTEMS .....	4
2.1 Definitions .....	4
2.1.1 Token .....	4
2.1.2 Time Stamping .....	5
2.2 Principles for distributed mutual exclusion .....	6
2.2.1 Permission based algorithms .....	6
2.2.2 Token Based Algorithms .....	6
III. LITERATURE SURVEY .....	9
3.1 Survey of Permission Based Algorithms .....	9
3.1.1 Lamport's Mutual Exclusion Algorithm .....	9
3.1.2 Ricart and Agarwala's Algorithm .....	10
3.1.3 Carvalho and Roucairol's Algorithm .....	10
3.1.4 Maekawa's Algorithm in Decentralized System .....	11
3.2 Survey of Token Based Algorithms .....	12
3.2.1 Algorithms with no Logical Structure .....	12
3.2.1.1 Suzuki and Kasami's Algorithm .....	13
3.2.1.2 Heuristically-aided Algorithm .....	13
3.2.1.3 Algorithm Using Dynamic Information Structure .....	14
3.2.2 Algorithms with Logical Structure .....	15
3.2.2.1 Token Based Algorithm using Logical Ring .....	15
3.2.2.2 Tree Based Algorithm by Raymond .....	15
3.2.2.3 A Distributed Algorithm by Naimi & Trehel .....	18
3.2.2.4 A Distributed Algorithm by Raynal .....	22
3.2.2.5 A Generalized Structure for Tree Based Algorithms .....	23
3.2.2.6 Fair Mutual Exclusion on a Graph of Process .....	29
3.2.2.7 Dag Based Algorithm for Distributed Mutual Exclusion .....	30
3.2.2.8 A Distributed Algorithm on a Ring of Processes .....	33

Chapter	Page
IV. DESIGN AND IMPLEMENTATION ISSUES . . . . .	35
4.1 Implementation Platform and Environment . . . . .	35
4.1.1 Sequent Symmetry S/81 . . . . .	36
4.1.2 X11 Window system . . . . .	36
4.1.3 Motif . . . . .	37
4.2 Design of the Tool . . . . .	37
4.3 Implementation Details . . . . .	40
4.4 Overview of Graphical User Interface . . . . .	42
V. SIMULATION RESULTS . . . . .	50
5.1 Sample Simulation . . . . .	50
5.2 Discussion of Results . . . . .	54
5.2.1 A New Classification of Logical Token-Based Mutual Exclusion Algorithms . . . . .	54
5.2.2 Statistical Analysis . . . . .	57
5.2.2.1 Neilsen's Algorithm . . . . .	65
5.2.2.2 Raymond's Algorithm . . . . .	65
5.2.2.3 Naimi's Algorithm . . . . .	66
5.2.2.4 Raynal's Algorithm . . . . .	66
VI. CONCLUSIONS . . . . .	69
6.1 Summary . . . . .	69
6.2 Conclusions . . . . .	69
6.3 Future Work . . . . .	70
REFERENCES . . . . .	72
APPENDICES . . . . .	74
APPENDIX A - GLOSSARY AND TRADEMARK INFORMATION . . . . .	75
APPENDIX B - USER GUIDE FOR "SIMME" . . . . .	78
APPENDIX C - SYSTEM ADMINISTRATOR GUIDE FOR "SIMME" . . . . .	82
APPENDIX D - LISTING OF PROGRAM FILES . . . . .	84

## LIST OF TABLES

Table	Page
I. Simulation Configuration . . . . .	58
II. Simulation Results obtained for Heavy Load . . . . .	59
III. Simulation Results obtained for Light Load . . . . .	60
IV. Simulation Results obtained for 50% Load . . . . .	60

## LIST OF FIGURES

Figure	Page
1. Taxonomy of Distributed Mutual Exclusion Algorithms . . . . .	8
2. Example of Logical Tree Based Algorithm . . . . .	19
3. Initial situation of an Example of Tree Based Algorithm . . . . .	25
4. Final Configuration of an Example of Tree Based Algorithm . . . . .	27
5. Example of Dag Based Algorithm . . . . .	31
6. Developing a Site Neighbors File for a Tree Based Network . . . . .	41
7. Linear representation of a Star Network . . . . .	42
8a. Initial Screen . . . . .	43
8b. Data Menu with "Node Information" selected . . . . .	44
8c. Data Menu with "Site Request Rate" selected . . . . .	44
8d. Data Menu with "Site Behavior" selected . . . . .	45
8e. Data Menu with "Site Neighbor File" selected . . . . .	45
8f. A Snapshot of "Visualization" Mode . . . . .	46
8g. A Snapshot of "Execute" Mode . . . . .	47
8h. Statistics Summary Screen . . . . .	48
8i. A Graph plotted through Statistics Menu . . . . .	48
8j. Help Menu Screen . . . . .	49

Figure	Page
8k. Exit Menu Screen . . . . .	49
9a. Summary of Statistics obtained from "Visualization" mode . . . . .	51
9b. Summary of Statistics obtained from "Execute" mode . . . . .	51
10a. Avg. Synchronization Delay Time Graph for "Visualization" mode . . . . .	52
10b. Avg. Message Complexity Graph for "Visualization" mode . . . . .	52
10c. Avg. Synchronization Delay Time Graph for "Execute" mode . . . . .	53
10d. Avg. Message Complexity Graph for "Execute" mode . . . . .	53
11. A Classification of Logical Token-Based Algorithms in terms of Statitic and Dynamic Structure . . . . .	55
12. A Classification of Logical Token-Based Algorithm in terms of Transit and Proxy mode . . . . .	56
13. A Two-Dimensional Classification of Logical Token-Based Algorithms using adaptive reversal techinques . . . . .	57
14a. Site Neighbor File (linear6.nbr) . . . . .	58
14b. Behavior Data File (raynal.bev) . . . . .	58
15a. Avg. Synchronization Delay Time Graph for Neilsen's Algorithm . . . . .	61
15b. Avg. Message Complexity Graph for Neilsen's Algorithm . . . . .	61
15c. Avg. Synchronization Delay Time Graph for Raymond's Algorithm . . . . .	62
15d. Avg. Message Complexity Graph for Raymond's Algorithm . . . . .	62
15e. Avg. Synchronization Delay Time Graph for Naimi's Algorithm . . . . .	63
15f. Avg. Message Complexity Graph for Naimi's Algorithm . . . . .	63
15g. Avg. Synchronization Delay Time Graph for Raynal's Algorithm . . . . .	64
15h. Avg. Message Complexity Graph for Raynal's Algorithm . . . . .	64



Figure	Page
16. List showing Algorithms and its corresponding Files . . . . .	81

## CHAPTER I

### INTRODUCTION

In multiprocessing systems, there are many processes working together. They often share resources, such as memory, files or printers. A situation may arise where many processes are using a shared resource and correct operation of the system depends on when processes access that shared resource. This is called a *race condition*. In order to avoid race conditions, *mutual exclusion* is required, that is, when one process is accessing the resource, other processes are excluded from accessing the same resource. The part of program where the shared resource is accessed is called a *critical section* "cs". To have a good solution to mutual exclusion, the following conditions should be satisfied always [TAN 91]:

- One and only one process can be inside its critical section at any given time.
- No assumptions should be made regarding the speed or number of processors.
- No process running outside its critical section may block other processes.
- Each process should be allowed to enter its critical section within a finite time.

Multiprocessing systems can be either centralized or distributed. A *Centralized system* is composed of several terminals sharing single resources like CPU, Memory and other peripherals. Thus, in centralized systems, controlling algorithms are based on the existence of shared memory and all processes have access to this common memory. Mutual exclusion in the centralized system can be solved by either "lock" variables or by

interprocess communication primitives such as semaphores, event counters, and monitors. Various algorithms have been proposed, like Decker's algorithm, Dijkstra's algorithm, Hyman's algorithm, and Peterson's algorithm [RAY 86]. All these algorithms use a shared variable that acts as a lock on a resource whenever a process is accessing it and when the process has finished accessing the resource the lock is removed.

*A distributed system* is one that runs on a collection of machines, not having shared memory or a global clock, but still appears to function like a single machine. In general, a distributed system will have the following properties [TAN 91]:

- There is no shared memory, and therefore all the information is scattered on various machines connected by a computer network.
- Processes make decisions based on locally available information.
- Failure of one process should not effect the system.
- There is no global clock to synchronize the event.
- Processes communicate with each other through message passing only.

The mechanism of sharing resources in a distributed system is different from a centralized system. Mutual exclusion in a distributed system can be achieved by receiving permission from all sites explicitly or implicitly, for instance by receiving the single token in the system. Definition of terms associated with mutual exclusion in distributed system is explained in Chapter II.

Algorithms to achieve mutual exclusion in the distributed systems can be classified into two classes: Permission based and Token based. The token based algorithms can be further classified into two classes: Algorithms with no logical structure imposed on the

system and Algorithms with logical structure imposed on the system. These algorithms are reviewed in the literature survey of Chapter III.

The purpose of this research is to develop a package to simulate the logical token-based distributed mutual exclusion algorithm using an adaptation of reversal techniques. These reversal techniques are discussed in chapter V. The tool, thus can be useful for the following purposes.

- The graphical tool can be used to understand the working of the mutual exclusion algorithm, with a view to improve them. The visualization capabilities can help the user, to identify any hidden problems that make the algorithm unsuitable for his particular network.
- The tool can be used as an aid to educational purposes.
- The performance study of algorithms, in terms of Message Complexity and Synchronization Delay can be done under different load conditions.

The design and implementation details of the simulation tool are discussed in Chapter IV. The performance study of the algorithms implemented, and the statistical results obtained, are discussed in Chapter V.

Finally, the thesis concludes with summary, conclusion and a brief discussion of future work in Chapter VI.

## CHAPTER II

### OVERVIEW OF MUTUAL EXCLUSION IN DISTRIBUTED SYSTEMS

The concept of mutual exclusion exists in both centralized and distributed systems. In distributed systems several uncoordinated users from different sites may access a shared resource concurrently. Therefore, all these concurrent requests should be serialized using some algorithm so that each request is satisfied within a finite time. The problem of mutual exclusion in distributed systems is more complex because a distributed system consists of geographically dispersed information sites:  $s_1, s_2, s_3, \dots, s_n$ , which are connected by computer network and they communicate only via message passing. There is no common clock and the sites do not share memory. Further the messages can be delayed or lost and the nodes as well as the channels connecting the nodes can fail. In addition there is no centralized coordinator to coordinate all the activities.

#### 2.1 Definitions

##### 2.1.1 Token

*Token* is a privilege or priority that circulates the logical structure in a distributed system. The site possessing the token can enter critical section while other sites have to wait till they receive the token.

### 2.1.2 Time Stamping

Since in distributed systems there is no common physical clock, there should be some mechanism to set order, on a set of messages. This mechanism was proposed by Lamport [LAM 78] and is called *Time Stamping* or principle of a Logical clock.

The mechanism works as follows:

Each site possesses a logical clock  $h_i$  which is set to zero initially. Every time a message is to be issued by process  $p_i$ , it stamps the message by  $(m, h_i, i)$  where "m" is the message, " $h_i$ " is the clock value, and "i" is the process id.

The clock is managed as follows:

- When process  $p_i$  issues a message  $(m, h_i, i)$ , the  $h_i$  value is incremented by 1 and then issued.
- When  $p_i$  receives a message  $(m, h_j, j)$  it sets  $h_i$  to the value  $(\max(h_i, h_j) + 1)$ , i.e., it sets its own clock with maximum of local clock value and the clock value of the requesting process, so that any problem of drift among various clocks of communicating processes is avoided.

Thus each event or message is time stamped and the order of events is maintained. To resolve any conflict among the events with same time stamped value, the following rules are observed.

1.  $h_i < h_j$  implies  $m_i$  precedes  $m_j$  in the logical clock.
2. if  $h_i = h_j$  and  $i < j$  then  $m_i$  precedes  $m_j$  else  $m_j$  precedes  $m_i$ .

And, this way a total ordering of events can be maintained in the system.

## 2.2 Principles for distributed mutual exclusion

Various algorithms have been designed for achieving mutual exclusion. These algorithms are based on any of the two principles, permission based [RAY 91] and token based [RAY 91].

### 2.2.1 Permission based algorithms

Consider geographically dispersed information sites  $s_1, s_2, \dots, s_n$ , of a distributed system. In permission based algorithms, a site wanting to enter a critical section, first gets permission from all other sites, i.e., it sends request messages to all sites or set of sites depending on the algorithm adopted and waits for their permission. The other sites give permission to the requesting site if they are not using critical section. Otherwise they delay giving permission until they finish using the critical section. The requesting site on receiving permission from all the sites enters the critical section.

Refinements have been done to reduce the number of the messages required to send per critical section request. Timestamping is done on each event to maintain the order of events.

### 2.2.2 Token Based Algorithms

Systems based on the token concept are simple. Token-based algorithms work as follows. A process holding token can enter critical section while other processes just wait till they get the token. Always only one token is present in the system. This gives mutual exclusion among processes. The movement of token can be either perpetual mobile or

token-requesting method.

In perpetual mobile method, the processes in the system are arranged in a logical ring structure and the token keeps rotating in the ring. Process holding a token, if it so wants, can access critical section and after using it, passes the token to its neighbor. But, if a process does not want to enter critical section, it simply passes the token to its neighbor. Thus, the token keeps revolving in the ring.

In token-requesting method, a node makes request to all other nodes and the token holding node after using the critical section passes the token to the requesting node. The token-based method can be further classified into two categories.

- Token-based method with Logical structure imposed on the physical network.
- Token-based method with no Logical structure imposed on the physical network.

In logical structured token-based method, a logical structure like a ring, a tree, or a dag is imposed on the physical network.

Refinements have been done to decrease the number of messages per critical section such as logically structuring the requesting processes as a tree or making a heuristic guess to locate the token holder or using parallel flooding technique. A taxonomy of distributed mutual exclusion algorithms is shown in the figure 1.



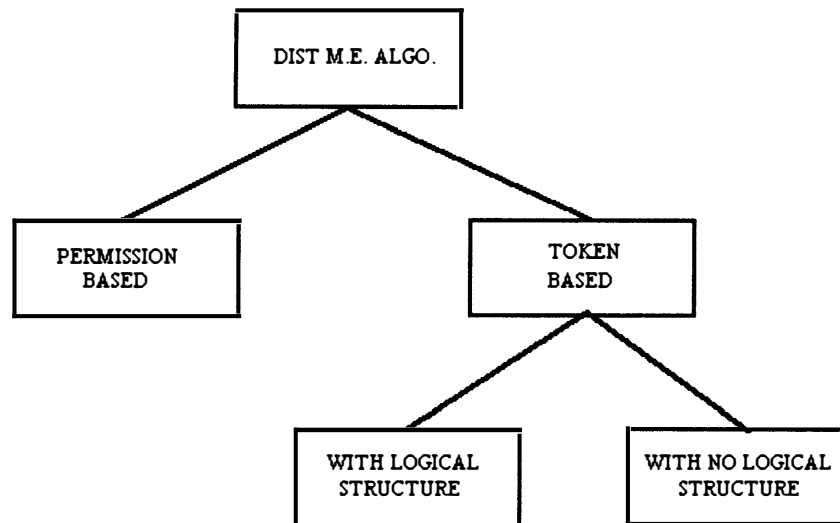


Figure 1: Taxonomy of Distributed Mutual Exclusion Algorithms

## CHAPTER III

### LITERATURE SURVEY

As shown by the taxonomy (Figure 1), distributed mutual exclusion algorithms can be classified into:

- Permission based algorithms
- Token based algorithms

Algorithms related to each of these categories are discussed in this chapter.

#### 3.1 Survey of Permission Based Algorithms

In permission-based algorithms, each site before entering critical section, takes permission from all other sites or set of sites depending on the algorithm. Some permission-based algorithms are as follows:

##### 3.1.1 Lamport's Mutual Exclusion Algorithm [RAY 86]

Lamport, in order to implement mutual exclusion in a distributed system, has adapted the centralized algorithm for the same. In the algorithm for the centralized system, a queue is maintained for request and release of messages. Requests are enqueued in the order of their arrival and the serving is done in the same order.

Lamport's algorithm maintains order of the message by using timestamping of each event. A site wanting to enter critical section broadcasts the message to all sites. All

sites acknowledge the request. The site after using the critical section again broadcasts the release of critical section message to all sites. Each site has a single message queue with messages totally ordered. Site having the oldest timestamped value enters the critical section next. To enforce mutual exclusion,  $3(n-1)$  messages are required, where "n" is number of sites in the distributed system.

### 3.1.2 Ricart and Agarwala's Algorithm [RIC 81]

Ricart and Agarwala's Algorithm has further reduced the number of messages required per critical section. The site wanting to enter critical section sends request messages to all other sites. Other sites on receiving the request, reply giving permission to enter critical section. The reply can be sent immediately or can be delayed until the site completes the processing of critical section. Timestamping technique is used to maintain the order of events. Number of messages required per critical section is  $2(n-1)$ .

### 3.1.3 Carvalho and Roucairol's Algorithm [CAR 83]

This algorithm is a modified form of Ricart and Agarwala's Algorithm. Here, the number of messages per critical section request is further reduced. In this algorithm, a site, if using critical section can keep using it until another site sends a request for critical section. A single queue server is maintained and the ordering of messages is maintained using timestamping of the events. The number of messages here varies between 0 and  $2(n-1)$ .

### 3.1.4 Maekawa's Algorithm in Decentralized System [MAE 85]

Maekawa has divided all sites into groups or sets. This further reduces the number of messages per critical section to  $\sqrt{n}$ . A site in a system has to get permission only from its group members and if the permission is granted then that site can go ahead to use critical section. Otherwise the site has to wait till it gets permission from all other members of the group.

Groups so formed have the following properties:

- For any combination of  $i$  and  $j$ ,  $1 \leq i, j \leq n$ .

$$s_i \cap s_j \neq \emptyset$$

- The set  $s_i$ ,  $1 \leq i \leq n$  always contains  $i$
- The size of  $s_i$ ,  $|s_i|$ , is  $k$  for any  $i$

$$|s_1| = |s_2| = \dots = |s_n| = k$$

- Any  $j$ ,  $1 \leq j \leq n$ , is contained in  $D$   $s_i$ 's,  $1 \leq i \leq n$

Where constant  $k$  denotes the number of members in a group, and  $D$  denotes the number of groups in which a site is a member.

The group can be formed in many ways. one way is;

$$N = (D - 1) K + 1$$

where  $k = \sqrt{n}$ .

There can be simultaneous requests from various sites, which can result in deadlock. This is solved in the following way:

- A site makes REQUEST to members in its group.

- Each member gives permission and locks itself.
- If a site gets permission from all its members, it executes critical section and then sends a RELEASE message to unlock them.
- If a member is already locked by another site, it sends an INQUIRE message to other members of the group.
- On receiving an Inquire message, the priority is checked by sequence number and the corresponding action is taken to RELINQUISH the lock.

Thus the number of messages required per critical section varies from  $3\sqrt{n}$  to  $5\sqrt{n}$ , i.e.,  $c\sqrt{n}$ , Where  $c$  is a constant ranging from 3 to 5.

### 3.2 Survey of Token Based Algorithms

Algorithms using token requesting method to achieve mutual exclusion in distributed system are discussed in this section. Token-based algorithms can be further classified as those,

- With logical structure
- With no logical structure

#### 3.2.1 Algorithms with no Logical Structure

In this category, the algorithms do not impose any logical structure on the physical network. However, a site can enter the critical section only after possessing a token. Related algorithms are discussed in this subsection.

3.2.1.1 Suzuki and Kasami's Algorithm [RAY 86]: Message traffic is further reduced in this algorithm to  $n$  messages per critical section request, where  $n$  is the number of sites in the system. Each node requesting to use critical section, sends a message to all other nodes. Token contains the timestamp value of all processes visited by it. Site holding the token, and using critical section, checks the request array and token array to find the first site whose request timestamp is greater than its timestamp in token array and sends the token to it.  $n$  messages are required:  $(n-1)$  messages to send request to all sites and 1 message to receive a token.

3.2.1.2 Heuristically-aided Algorithm [SIN 92]: Heuristically-aided algorithm is an improvement on the Suzuki-Kasami's algorithm. All above-mentioned algorithms are deterministic in nature. This means, each site does not maintain the state information of all other sites but, when a site wants to make a request for critical section, it sends the request message to all the other sites and the token is assigned to a site depending on the lowest timestamp value of the received request messages from different requesting sites.

In Heuristic method, each site maintains the state information of all sites. Thus a site wanting to make request knows the set of sites that may probably be holding the token and hence the request is sent only to those sites. This decreases the number of messages sent per critical section. The set of sites are those sites that are either holding the token or requesting the token. Heuristic guessing can be achieved by the following data structure. Each site has 2 state vectors. One state vector stores the state of sites and other stores highest sequence number of each site. The state of sites can be any of the four.

R = Requesting the token, N= Not requesting the token,

H = Holding the token, but not executing critical section,

E = Executing critical section.

Token also maintains state vector and sequence number of all sites. Whenever the request or token is received, the state vector is updated. Each set should have at least one sites in the state vector that is requesting. Number of messages vary from 0 to n, where n is number of sites in the system.

3.2.1.3 Algorithm Using Dynamic Information Structure [SIN 89]: This algorithm is based on having dynamic state information of all the sites. The state information is updated continuously as requests are made. The sites are arranged in a specific order say  $s_1, s_2, \dots, s_n$  such that each site has to request only to all the sites in the right of it and to no sites in the left of it. Only after getting the permission, a site can enter the critical section. Thus each site maintains a *request set* (i.e., sites from which it has to obtain permission before entering critical section) and an *inform set* (sites to which it has to inform after its completion of critical section).

An interesting feature of the algorithm is that its information structure adapts itself to the environments of heterogeneous traffic of critical section requests and therefore to statistical fluctuations in traffic of critical section requests to optimize the performance. A site that is always requesting will be on the right-hand side and hence reduces the number of messages. While the sites that are less busy will be clustered on the left. This helps in increasing performance of the system in terms of number of messages sent per critical section. Messages vary from 0 to  $2(n-1)$ .

### 3.2.2 Algorithms with Logical Structure

Algorithms that impose a logical structure on the physical network are discussed in this subsection. The logical structure may, however, be static or dynamic in nature. If the logical structure imposed on physical network does not change with the movement of request message or token, then the structure is *static*. But, if the logical structure keeps changing dynamically with the movement of token, then the structure is *dynamic*.

3.2.2.1 Token Based Algorithm using Logical Ring [RAY 86]: In this algorithm, all the nodes of the distributed system are arranged in a logical ring. The token, a special privilege to enter critical section keeps rotating in the logical ring. A node that possesses a token and wants to enter critical section, can enter the critical section, otherwise it waits for the turn. After a node exits critical section, it passes the token to its neighbor. The token moves in one direction and so there is no starvation. Messages required in the best case is 1 and in the worst case is infinity.

3.2.2.2 Tree Based Algorithm by Raymond [RAY 89]: Here the request of any site (X) is only sent to its neighboring node (Y) which in turn forwards it, to its neighbor, in the direction of a token holder. Thus the number of messages per critical section depends on the topology that is a tree. The node does not have to know the whole tree but only knows its neighbor. Each node has a variable HOLDER that shows the direction of the token holder node.

Thus when a node sends a request to neighbor Y, neighbor Y will start functioning on X's behalf and forwards the message.



Hence 3 kinds of messages are sent REQUEST, PRIVILEGE, and INITIALIZE.

A REQUEST message is sent when a site wants to access critical section, to the neighboring site in the direction of the token holder node. A PRIVILEGE message is sent by the token holder node to its neighboring node that is in the direction of the requesting node. INITIALIZE message is sent at the start of the whole process, when one site is arbitrarily given the privilege and that site sends a message to its neighbors about its privilege. The neighboring site initializes its HOLDER variable and passes it further to its neighbor.

ALGORITHM :

Procedure initialization()

```
begin
    if (holding the token) then
        begin
            holder = self;
            using = true;
        end
    else
        begin
            holder = name of neighbor in whose subtree the token is present;
            using = false;
        end
    send initialize(i) message to all neighboring nodes;
end.
```

Procedure send\_req()

```
begin
    if (holder  $\neq$  self and request_q  $\neq$  empty and not asked) then
        begin
            send request to holder;
            asked = true;
        end;
end.
```

procedure recv\_req()

```
begin
```

```

        enqueue request to request_q;
        if (holder  $\diamond$  self)
            send request to holder;
    end.

procedure execute_cs()
begin
    if (holder  $\diamond$  self)
    begin
        send request to holder;
        wait until a privilege message is received;
    end
    using = true;
    CRITICAL SECTION;
    using = false;
    if (request_q  $\diamond$  empty)
    begin
        assign_privilege();
    end;
end.

procedure assign_privilege()
begin
    if (holder = self and not using and request_q  $\diamond$  empty) then
    begin
        holder = dequeue(request_q);
        asked = false;
        if (holder = self) then
        begin
            using = true;
            (initiate entry into critical section)
        end
        else
            send privilege to holder;
        end
    end
end

```

Various other topologies like line, radiating star, ring etc., can be applied using the same algorithm. This algorithm can be further optimized by using piggy back strategy and greedy strategy.

3.2.2.3 A Distributed Algorithm by Naimi & Trehel [TRE 87]: The algorithm uses logical rooted tree structure where each node that is requesting the critical section becomes the root of the logical tree. Thus, the structure of the tree keeps changing dynamically. Each requesting node sends the request to only one site and that site sends permission to the requesting site. This single node to which the request is sent is the last node that was holding the token. Thus, it has two data structures.

- Logical rooted tree: The rooted tree is maintained by updating the tree such that the new requesting site, if it is not the root, is transformed to be the root and sites that are between the root and the requesting node will have the new root as "last".
- Waiting queue : It holds the order in which the privilege is given.

Thus, there are cases when many sites are requesting simultaneously and hence several rooted trees are formed and when all the transit messages arrive they form a single rooted tree.

In this algorithm, there is no need to maintain the logical clock. Number of messages per critical section is order of  $\log |n|$ . Example 1, illustrates this algorithm and Figures 2(a), 2(b), 2(c), 2(d), 2(e) respectively show the dynamic change in the logical structure.

Example 1: [TRE 87]

- \* Initial state of the distributed system. Site 1 has privilege.

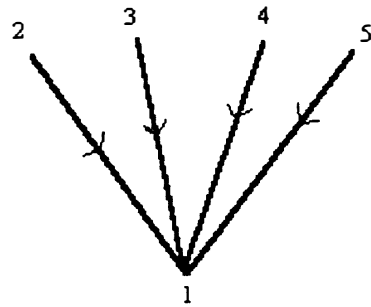


Figure 2 (a):

- \* Site 2 invokes the critical section. It sends a request to site 1. Site 2 becomes the root. Site 2 has privilege and enters the critical section.

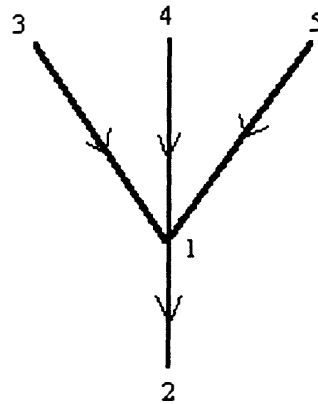


Figure 2 (b):

- \* Site 3 invokes the critical section. It sends a request to site 1 which transmits to site 2. Site 2 is in the critical section : site 3 waits

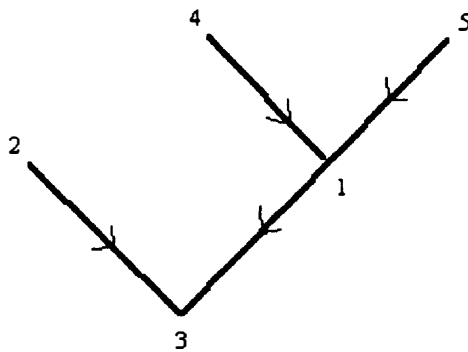


Figure 2(c):

- \* Site 4 requests the critical section. It sends a request to site 1 which transmits to site 3. Site 2 is in the critical section; sites 3 and 4 wait.

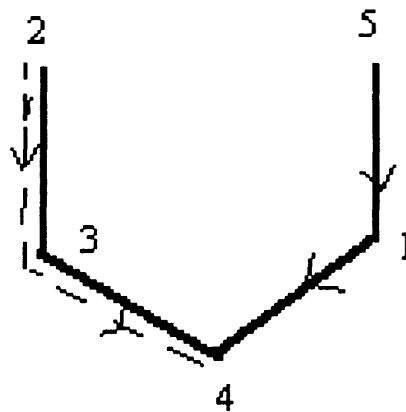


Figure 2(d):

- \* Site 2 releases the critical section. It gives privilege to site 3. Site 2 requests critical section again. It sends a request to site 3 which transmits to site 4. Site 3 is in the critical section. Sites 4 and 2 wait.

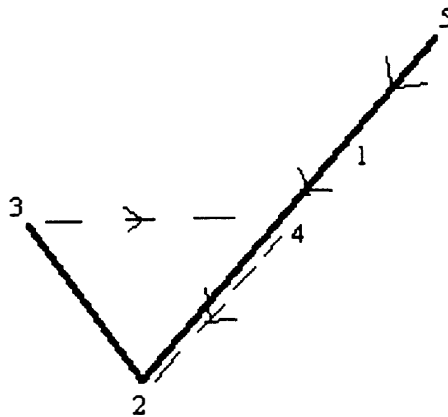


Figure 2(e):

Specification of the algorithm are given below:

All the sites execute the same algorithm. Each site has local variables namely;

Privilege, requesting\_c\_s: Boolean

Where, Privilege is true if site  $i$  controls the critical section and requesting\_c\_s is true if site  $i$  has invoked the critical section and remains true until it releases the critical section.

- A site  $i$  may enter the critical section if "privilege" = true and "requesting\_c\_s" = true.
- A site  $i$  gives privilege to another site if "privilege" = true and "requesting\_c\_s" = false.
- A site  $i$  may transmit a request to another site if "last"  $\diamond$  nil.

Algorithm:

procedure initialization()

begin

    last = 1;

    % initialization of last is the same for every site %

    next = nil ; requesting\_c\_s = false;

    if last = me then

        privilege = true;last = nil;

    else

```

        privilege = false;
    endif;
end % initialization %

procedure request_c_s()
begin
    requesting_c_s = true;
    if last <> nil then
        send(req,me) to last
        last = nil;
    endif
end %request_c_s %

procedure release_c_s()
begin
    requesting_c_s = false;
    if last <> nil then
        send(ok,me) to next,
        next = nil;
        privilege = false;
    endif
end %release_c_s %

procedure receiving_message(message,k)
case
    message = REQ --> case
        last=nil--> if requesting_c_s then
                    next =k;
                else
                    privilege = false;
                    send(ok,me) to next
                    next = nil;
                endif
        last <> nil --> send(req,k) to last;
    endcase;
    last = k;
    message= OK --> privilege = true;
endcase

```

3.2.2.4 A Distributed Algorithm by Raynal [RAY 86]: Raynal has proposed a token-based algorithm which can work for various network topologies like tree, line, ring,

and complete network. The number of messages per critical section depends on the topology of the particular network.

In the arbitrary network, a node requesting for token sends its request to its neighboring node. The REQUEST message consists of the following structure. *req-id* and *req-info*. *req-id* consists of *req-origin* and *req-time* (*req-time* is the logical time as specified by lamport). *req-info* contains a sender and already seen nodes. When the neighbor receives the request, if it has the token, checks the request-array for pending request and sends the token to the oldest of them through the already-seen nodes. If the neighbor does not have the token, it forwards it to its neighbor and adds its node number to the already seen node list. The node also updates the request array to delete any old request received from the same node earlier or adds it in as a new request.

This algorithm can be used on any arbitrary network. Each node maintains local information only. This algorithm uses distributed communication techniques to achieve mutual exclusion.

3.2.2.5 A Generalized Structure for Tree Based Algorithms [RAYUR]: This algorithm imposes a dynamic rooted tree structure to connect nodes logically. Each node contains local information regarding the state, behavior and its relative position in the system. The variables used in the algorithm are as follows :

1. Token-here<sub>i</sub>: True if node i has the token.
2. Asked<sub>i</sub>: True if node i is currently waiting for the token or executing critical section.
3. Parent<sub>i</sub>: If node i is not possessing the token and wants to get it, it sends a request(i) to parent<sub>i</sub>.



4. Lender<sub>i</sub>: It indicates the node to which node *i* will have to give back the token once it exits critical section.

5. Behavior(*i*): It can be either of the following:

*Transit*: When node *i* receives a request from node *j*, it just forwards it to its parent<sub>i</sub>. It sets parent<sub>i</sub> = *j*.

*Proxy*: When node *i* receives a request from node *j*, it takes the request on its own account. It now considers node *j* as its mandator and requests the token for itself from its parent<sub>i</sub>. When node *i* receives the token from some node *k* then it passes it on to node *j*. If the token has been lent to node *i* then node *i* sets its parent<sub>i</sub> = *k* else it sets parent<sub>i</sub> = NIL and node *i* will become the lender.

6. Mandator(*i*): This is used in proxy behavior of nodes as explained above.

If more than one request is queued on a node then they are handled in a fair way by using policies like FIFO. This happens when more than one node have parent<sub>i</sub> = same node *j*. Then node *j* can receive multiple requests.

Example [RAYUR] :

The set of variables defined above constitute the *information structure* of each site in a tree structured token based algorithm. The following example illustrates the algorithm. The Figure 3 below describes the initial situation. Node 8 wishes to enter the critical section, and the token is with node number 1; only nodes belonging to the oriented path (defined by the successive variables *parent*) linking node 8 to node 1 are shown in the Figure 3. Nodes 3, 5, and 6 are assumed to be permanently *proxy* (they are circles in the figures), whereas nodes 1, 2, 4, 7, 8, are assumed to be permanently *transit*.



Figure 3: Initial situation of an Example of Tree Based Algorithm

Node 8 wishes to enter the critical section and not *token\_here*<sub>8</sub>:

0 send *request*(8) to parent<sub>8</sub> = 7; asked<sub>8</sub> = true; mandator<sub>8</sub> = 8

Node 7 receives *request*(8) and behavior<sub>7</sub> = transit and not *token\_here*<sub>7</sub>:

send *request*(8) to parent<sub>7</sub> = 6; parent<sub>7</sub> = 8

Node 6 receives *request*(8) and behavior<sub>6</sub> = proxy and not *token\_here*<sub>6</sub>:

% 6 takes the request on its own account % send *request*(6) to parent<sub>6</sub> = 5; asked<sub>6</sub> = true;

mandator<sub>6</sub> = 8

Node 5 receives *request*(6) and behavior<sub>5</sub> = proxy and not *token\_here*<sub>5</sub>:

% 5 takes the request on its own account % send *request*(5) to parent<sub>5</sub> = 4; asked<sub>5</sub> = true;

mandator<sub>5</sub> = 6;

Node 4 receives *request*(5) and behavior<sub>4</sub> = transit and not *token\_here*<sub>4</sub>:

send *request*(5) to parent<sub>4</sub> = 3; parent<sub>4</sub> = 5

Node 3 receives *request*(5) and behavior<sub>3</sub> = proxy and not *token\_here*<sub>3</sub>:

% 3 takes the request on its own account % send *request*(3) to parent<sub>3</sub> = 2; asked<sub>3</sub> = true;

mandator<sub>3</sub> = 5;

Node 2 receives *request*(3) and behavior<sub>2</sub> = transit and not *token\_here*<sub>2</sub>:

send *request*(3) to parent<sub>2</sub> = 1; parent<sub>2</sub> = 3;

Node 1 receives request(3) and behavior<sub>1</sub> = transit and token\_here<sub>1</sub> and asked<sub>1</sub>:

% 1 gives up the token to 3 since its behavior is transit % send token(nil) to 3;

parent<sub>1</sub> = 3; token\_here = false;

Node 3 receives token(nil) and mandator<sub>3</sub> = 5:

% 3 becomes the lender

% parent<sub>3</sub> = nil ; send token(3) to mandator<sub>3</sub> = 5; mandator<sub>3</sub> = nil;

Node 5 receives token(3) and mandator<sub>5</sub> = 6:

parent<sub>5</sub> = 3 % 3 is the sender of the token %

% complete the mandate for node 6 % send token(3) to mandator<sub>5</sub> = 6; mandator<sub>5</sub> = nil;

asked<sub>5</sub> = false;

Node 6 receives token(3) and mandator<sub>6</sub> = 8:

parent<sub>6</sub> = 5 %5 is the sender of the token %

% complete the mandate for node 8 % send token(3) to mandator<sub>6</sub> = 8; mandator<sub>6</sub> = nil;

asked<sub>6</sub> = false;

Node 8 receives token(3) and mandator<sub>8</sub> = 8:

parent<sub>8</sub> = 6 %6 is the sender of the token %

lender<sub>8</sub> = 3 % the token will be returned to node 3 % token\_here<sub>8</sub> = true

CRITICAL SECTION

send token(nil) to lender<sub>8</sub> = 3; token\_here<sub>8</sub> = false; asked<sub>8</sub> = false

Node 3 receives token(nil) and mandator<sub>3</sub> = nil:

token\_here<sub>3</sub> = true; asked<sub>3</sub> = false;

At the end the rooted tree is shown below in Figure 4:

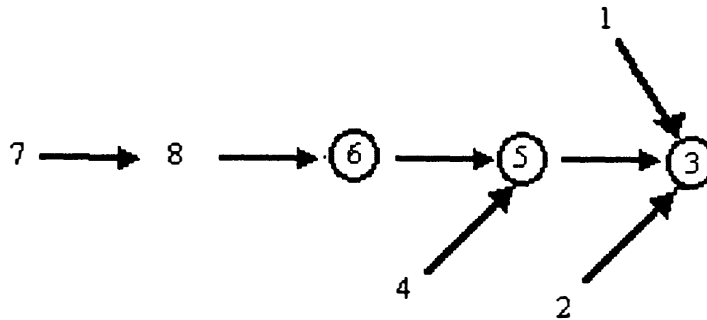


Figure 4: Final Configuration of an Example of Tree Based Algorithm

algorithm:

```

procedure call_to_enter_cs()
begin
  wait(not askedi );
  askedi := true;
  if not token_herei then mandatori := i;
                        send request(i) to fatheri;
                        wait(token_herei); % receipt of token sets lenderi %
  endif
end % enter_cs %

procedure call_to_exit_cs
begin
  if lenderi <> i then send token(nil) to lenderi; token_herei = false; endif;
  askedi = false;
end % exit_cs %

procedure receipt_request(j)
begin
  wait(not askedi);
  case of behaviori = proxy
  begin % i becomes proxy of j %
    askedi = true;
    if token_herei
    then % i temporarily lends the token %
      send token(i) to j; token_herei = false;
    else % i requires the token %

```

```

        mandatori = j;
        send request(i) to fatheri;
    endif
end
behaviori = transit
begin
if token_herei
    then % give up the token %
        lenderi = nil;
        send token(nil) to j; token_herei = false;
    else % forward the request %
        send request(j) to fatheri;
    endif
fatheri = j;
end
endcase
end % request %

procedure receipt_token
begin
    token_herei = true;
    case of mandatori = nil
        begin % case :return of the token after temporary lending %
            askedi = false
        end
    mandatori = i
    begin % claim of i will be satisfied %
        % update the position variables %
    if j = nil then % i will not have to give back the token (no lender) %
        lenderi = i; fatheri = nil
    else % i will have to give back the token %
        % update the path towards lender %
        lenderi = j; fatheri = k;
    endif;
    mandatori = nil
end
mandatori <math>\diamond</math> nil
begin % case : i honors the request of its mandator, %
    % meanwhile, its behavior may have changed %
    askedi = false;
    case of behaviori = proxy
        begin
            if j = nil then % i becomes the lender and temporary lends the token %
                lenderi = i; fatheri = nil
            end
        end
    end
end

```

```

        send token(i) to mandatori;
        askedi = true
    else % j is the lender of the token %
        fatheri = k;
        send token(j) to mandatori;
    endif
end
behaviori = transit
begin
    if j = nil then % the token must not be returned %
        lenderi = nil ; fatheri = mandatori;
        send token(nil) to mandatori;
    else % j is the lender of the token %
        fatheri =k;
        send token(j) to mandatori;
    endif
end
endcase
mandatori = nil ; token_herei = false
end
endcase
end % Token %

```

3.2.2.6 Fair Mutual Exclusion on a Graph of Process [VAN 87]: Van de Snepscheut suggested the following algorithm:

Each communication channel is made to point to the processes in the tree having the privilege. A process that wants the privilege, sends a request along the directed path of a communication channel toward the processes having the privilege. The token travels along the same path but in opposite direction. The privilege on traveling the channel reverses the direction of the communication channel. Once the privilege reaches a process all communication channel now points toward that process. If more than one request is queuing on a process then the privilege is sent to one process and a request is also sent so that the privilege comes back and all pending requests are satisfied. This algorithm is

extended to a graph as follows. Depth first search technique is used to construct a palm tree in the undirected graph whose root is the node with the privilege. All channels are directed to the root. The other modification is that all the outgoing channels from the node receiving privilege are inverted not just the channel on which the privilege travels.

3.2.2.7 Dag Based Algorithm for Distributed Mutual Exclusion [NEI 89]: The algorithm is as follows :

Each node maintains three variables.

1. Last: A logical dag structure is imposed by the last variable on the nodes. When a node receives a request message, it passes it to the neighboring node that is pointed by its "last" variable.
2. Next: This variable indicates the node that will be granted mutual exclusion after the current node. If it is less than zero then the current node will hold the privilege.
3. Holding: When a node is holding the privilege but is not in its critical section then holding is set to true.

A *sink node* is the last node in the implicit waiting queue also it is the last node in the path along which a request travels. When a non-sink node receives a request message, it passes it on to its neighboring node indicated by variable "Last". It then sets its variable "last" to point to the node from which it received the request.

When a sink node(i) receives a request message, it sets its variable "Next" to point to node(j) initiating the request. Now node(i) becomes a non-sink node and node(j) becomes the new sink node.

The above algorithm is illustrated by the following example [NEI 89]. Consider

a system consisting of 6 nodes, as shown in figure 5. Assume that Node 5 holds the token initially. Let the directed edges indicate the direction in which the LAST variables are pointing. The initial configuration is shown in Figure 5a. Suppose node 5 wants to enter the critical section. Since node 5 holds the token, it can enter immediately. Now, suppose node 3 wants to enter the critical section. It sends a REQUEST message to node 4 and sets its LAST variable to 0 to become a new sink (ref. figure 5b). Node 4 receives the request and sets its LAST variable to point to node 5, on behalf of node 3 (ref. to figure 5c). Node 5 receives the REQUEST message. Since node 5 is a sink node, it sets its NEXT variable to point to node 3 and sets its LAST variable to point to node 4 to become on-sink. When node 5 leaves the critical section, it sends a PRIVILEGE message to the node indicated by its NEXT variable, i.e., node 3. Finally, node 3 receives the PRIVILEGE message and enters the critical section (ref. figure 5d).



Figure 5 (a):



Figure 5 (b):



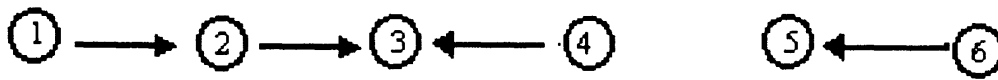


Figure 5 (c):

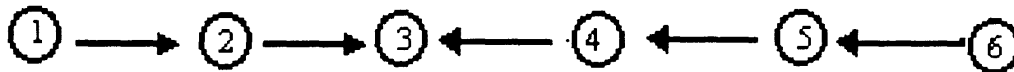


Figure 5 (d):

algorithm :

There are two procedures at each node: P1 and P2. Procedure P1 is executed when node  $i$  requests for entry into the critical section, and procedure P2 is executed when it receives a request from some other node.

procedure P1; (\* Enter critical section \*)

begin

  if (not HOLDING ) then

  begin

    send REQUEST(I,I) to LAST;

    LAST := 0;

    wait until a PRIVILEGE message is received

  end;

  HOLDING:= false;

    CRITICAL SECTION

  if (NEXT != 0) then

  begin

    send PRIVILEGE message to NEXT

    NEXT:=0;

  end;

  else HOLDING := true;

end;

PROCEDURE P2 (\* Handle REQUEST(X,Y) msg \*)

begin

  if (LAST=0) then

```

begin
    send PRIVILEGE message to Y;
    HOLDING := false;
end;
else send REQUEST(I<Y) to LAST;
LAST := X;
end;

PROCEDURE INIT; (* Initialize nodes *)
begin
    if (holding the token ) then
        begin
            HOLDING := true;
            LAST:= 0;
            NEXT:=0;
            send INITIALIZE(I) message to all neighboring nodes;
        end;
    else
        begin
            wait for INITIALIZE(J) message to arrive from node j;
            HOLDING :=false;
            LAST:=J;
            NEXT:=0;
            send INITIALIZE(I) message to all neighboring nodes, except node j;
        end;
    end;
end;

```

3.2.2.8 A Distributed Algorithm on a Ring of Processes [MAR 85]: The set of processes that want to use critical section is called *masters*. When a process wants to enter critical section, it communicates with another process called the *server*. Therefore, for N masters there are N servers. Each master communicates only with its server and servers communicate with each other.

A.J. Martin has suggested three solutions when the communication network among the servers is a ring. They are

1. Perpetuum Mobile: The privilege circulates continuously. When a process wants to

enter critical section, it waits for its turn till its server gets privilege. If a node does not want to use critical section then it passes on the privilege to a neighboring node.

2. Reflecting Privilege: In this solution the request moves in one direction and the privilege in the opposite direction.

3. Drifting Privilege: In this solution both the request and privilege move in the same direction.

Martin concludes that the second solution is most elegant and efficient among the three solutions. In the second solution the completion of pending request is used to transmit the privilege reducing the communication traffic to its minimum.

This thesis is concerned with the development of a graphical simulation tool for distributed mutual exclusion algorithms. The design and implementation issues associated to the tool are examined in the next chapter.

## CHAPTER IV

### DESIGN AND IMPLEMENTATION ISSUES

#### 4.1 Implementation Platform and Environment

The graphical tool for simulation has been implemented on the Symmetry S/81 system running the X Window system under the DYNIX/Ptx operating system. DYNIX/Ptx is a UNIX system port that is compatible with AT&T System V 3.2 [SEQ 90]. The X Window system or X allows programmers to develop portable graphical user interface (GUI). X allows programs to display windows containing text and graphics on any hardware that supports the X protocol. Thus, X-based application can work on heterogeneous environment consisting of mainframes, workstations and personal computers. The X Window system has Client-Server architecture. A Server is a process responsible for all input and output devices and a client is the application.

The X-interface, provides GUI capability to an application program. It is written either in Xlib or Toolkit. Xlib is a set of standard C library functions and works as a low level interface to X. Toolkits is a set of higher level subroutine libraries and is used to implement a set of user interface features. The simulation tool is developed using the Motif widget set, Xlib functions, and C programming language. Graphs are plotted using BLT package.

#### 4.1.1 Sequent Symmetry S/81

Sequent Symmetry is a main frame-class multiprocessing system. It can run both Dynix V 3.0 and Dynix/Ptx. It is implemented on the Unix platform and is a true multiprocessor system, having multiple CPUs and a single common memory. Sequent symmetry systems have the following characteristics[SEQ 90].

- True multiprocessor, having multiple CPUs.
- Single common memory shared by all processors.
- All processors, I/O controller and memory modules are plugged into a single high speed bus.
- All processors are tightly coupled.
- All processors are dynamically load balanced.
- All processors are symmetric.
- Hardware support for mutual exclusion, is support exclusive access to shared data structure.

#### 4.1.2 X11 Window system

X11 Window system developed by MIT, provided GUI capability to programs using it. X window system requires bitmapped graphic display terminal, so that the graphic image can be constructed using pixel. The window system uses client-server architecture to communicate with the X client, which is X window based application and the X server, a software to deal with I/O of the bitmapped graphic-display terminal, in the X protocol. From the X client point of view, X protocol is a collection of function

libraries. The Xlib is a library that implements the X protocol for the C programming language. These functions in Xlib are preprogrammed to create, move, and destroy windows or draw lines etc. A level above Xlib is X Toolkit or Xt, where X window applications can be written from a higher level. Widgets are preprogrammed graphics objects that can be used by GUI programs. Some popular widget sets on X window system are Motif by OSF, Open Look by AT & T, and Athena by MIT. These widget sets provide menus, buttons, dialogboxes etc., to be used in a GUI application.

#### 4.1.3 Motif

Motif is the widget set developed by OSF, which provides GUI on any system that supports X window system. Motif is a widely available application programming interface and provides distinctive three-dimensional appearance. It requires an X window system including Xlib and Xt intrinsics. Number of widgets can be defined and are instantiated into objects in the application program like Pushbutton, Text, MessageBox, and Scrolled window.

### 4.2 Design of the Tool

In the distributed system, sites communicate with each other only through message passing, since there is no shared memory. Therefore, for a site to request the service of resource or enter critical section, it has to receive permission from all other sites in the system. In the simulation of the algorithm in this thesis, the following assumptions are made:

- Messages are not lost or altered in the system, while they are sent from one site to another.
- The messages are delivered in the order they are sent.
- The communication medium is reliable and there is no node failure.
- The internode communication time is always the same between two sites during the entire simulation period and is set by the user.
- The critical section time for each site is the same and this constant is set by the user before the starting of the simulation.

Various parameters are given by the user like the topology of the network, number of sites in the system, internode communication time, critical section execution time by each site, request rate of each site as a probability also termed as traffic load, and period of the simulation.

Each of the implemented algorithms is described in Chapter III. Initially, the token holder node is selected at random. A global clock is maintained throughout the simulation period, which keeps track of the generation of request, forwarding of the request and critical section execution. At each clock tick, requests are generated by sites depending on the respective traffic load. These requests are sent in the direction of a site holding token by the method dictated by the chosen algorithm. The request for access to critical section by nodes is generated by Monte Carlo methods that is described below.

*Monte Carlo Method:* The Monte Carlo method used in the simulation is as follows :

1. The user sets the probability of request generation by each site. Each site is assumed to be independent.

Let  $f_1, \dots, f_n$  be the probabilities of generating request for critical section by site<sub>1</sub>, site<sub>2</sub>, . . . site<sub>n</sub>.

2. A random number (E) is generated.

If,  $0 < E \leq f_1$  then site 1 generates a request and so on.

3. Once, a site makes a request, it cannot make another request until the earlier request made by it has been satisfied.

The traffic load for each site is probabilistic in nature. Simulation can be done for different load factors, namely, Heavy Load, Light Load and as set by user.

*Heavy Load* is a situation when all sites have high probability of making request always. A site is never idle. Once the request made by a site is satisfied, it makes the request again for critical section. Thus, the probability of making request is 100%. Here, when a site makes a request, it always has to wait for the token because some other site is using the critical section. Maximum number of messages are flowing in the system during the simulation period.

*Light Load* is a situation when all sites have low probability of making request always. The sites in the system are idle most of the time. At any given instant of time, maximum of one site is making a request and hence the number of messages flowing in the system are minimum during the simulation period. There are various methods of simulating a light load. The method adopted in this thesis is as follows. When a site is executing critical section, no other site can make a request. When a site finishes executing critical section then one site is selected at random to make a request.

The traffic load of each site in terms of the probability can be set by the user to



test the particular case.

After the request generation is started, the working of the mutual exclusion algorithm can be visualized on the screen. Movement of requests and token can be dynamically visualized on the screen. However, the visualization can be done only for limited number of nodes to give clear graphics on a screen of limited size. Also, as described earlier, simulation is a dynamic process, large number of nodes would make the screen cluttered and difficult for a user to understand. Simulation for large number of nodes can be done without visualizing the sites using the 'Execute' option.

### 4.3 Implementation Details

For the system of  $n$  sites, the sites are always numbered from 0 to  $n-1$ . A global clock is maintained to keep track of internode communication time, simulation time and critical section execution time. A global array "global\_array" is also maintained for each algorithm. It contains five fields. These five fields in the structure are used to store 'code number', 'from node #', 'to node #', 'source node #' and 'time left'. Where, 'code number' represents '1' for request, '2' for token, and '3' for critical section; 'from node #' represents the node # from which the message is sent; 'to node #' represents the node # to which the message is sent; 'source node #' represents the node from which the request originated; 'time left' represents the time remaining for completion of the operation.

At each clock tick of the simulation period, Monte Carlo simulation method, discussed in section 4.2, is used to select the sites from the set of sites that can make

request then. All the information is stored in the "global\_array" described above. Processing of new request message, forwarding message, sending request, execution of the critical section and, processing of the new request messages are registered in this global array. The field indicating 'time left' in the global array is an important parameter. After each time unit, the 'time left' field is decrement by one unit and checked to see, if any of the above operations can be completed and appropriate action is then executed. Parallel handling of the request messages from various sites to their corresponding neighbor, is efficiently handled by the "global\_array".

The topology of the network is given through an external file indicating the neighbor of each sites in the system. For example, a tree based topology on a system of six sites, can be stored in a file, by knowing the neighbors of each site in that system. Figure 6 illustrates the development of site neighbor file, "nbhr\_file".

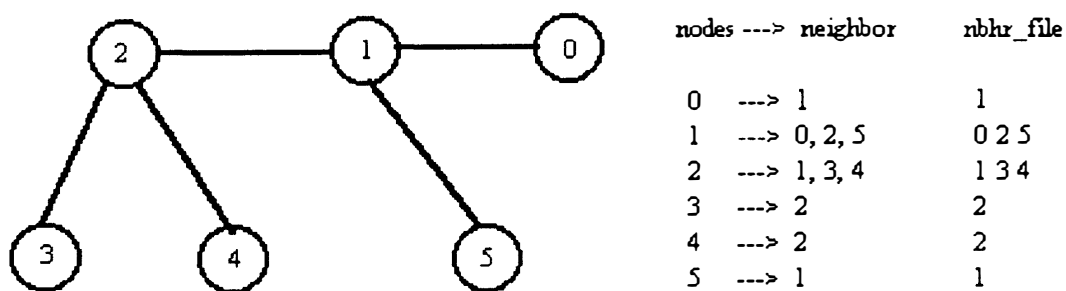


Figure 6: Developing a Site neighbors File for a Tree Based Network

There are two options in which the simulation can be executed, with or without visualization. Visualization is limited to a system consisting of maximum 8 nodes. But,

without visualization, in the 'Execute' option, the upper limit on the number of sites is imposed only by the memory capability of the computer on which the simulation is run. In 'visualization' option, the topology of the system is always represented in a linear form. This is done to maintain the symmetry in the visualization. The linear representation of a star-based network is shown in Figure 7.

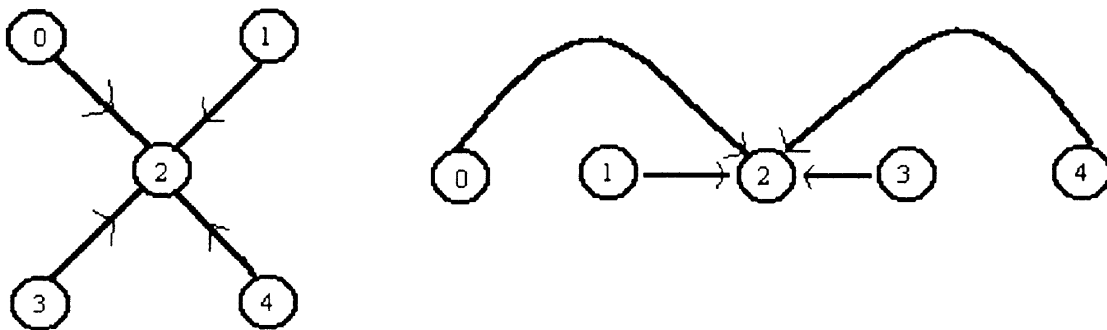


Figure 7: Linear representation of a Star Network

#### 4.4 Overview of Graphical User Interface

The data needed to run, the simulation is obtained from the user through the Graphical User Interface. It consists of a menubar having 'Algorithm', 'Data', 'Run', 'Statistics', 'Help' and 'Exit' menus. A snapshot of the initial screen of the tool is shown in figure 8a. An 'Algorithm' menu allows the user to choose the distributed mutual exclusion algorithm for simulation. The algorithms available for simulation are by Nielsen [NEI 89], Raymond [RAY 89], Naimi [TRE 87] and Raynal [RAYUR]. The required data for the simulation can be entered through the 'Data' menu. It consists of the following submenus: Node Information, Request Rate, Site Behavior and Site Neighbor Files.

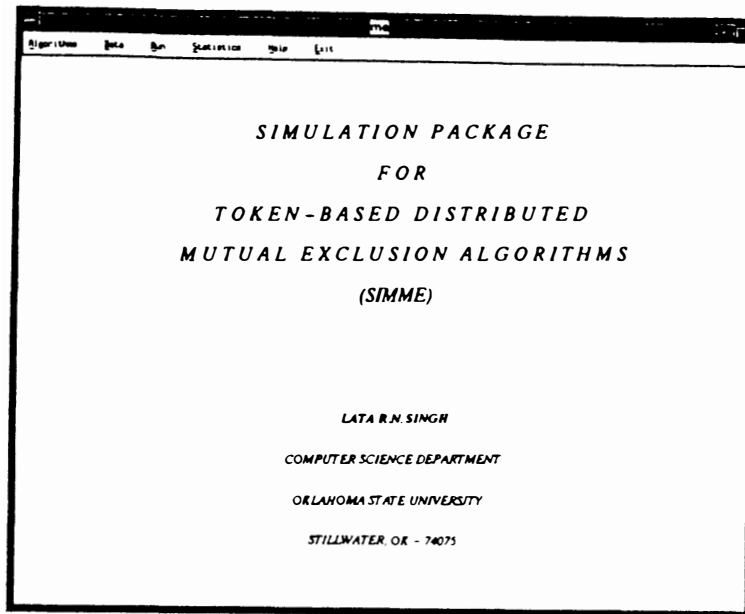


Figure 8a: Initial Screen

Number of nodes in the system, internode communication time, critical section execution time, and simulation period can be entered through the 'Node Information' submenu. The load of the site (Heavy load, Light load, or, as set by user) can be entered through the 'Request Rate' submenu. The behavior of the site (Transit or Proxy) is entered through the 'Site Behavior' submenu. The file providing, the topology of the network is chosen from 'Site Neighbor File' submenu. The snapshot of the Data menu is shown in Figures 8b, 8c, 8d, 8e.

The tool has an option of visualizing the simulation and executing without visualization. These can be done through the 'Run' menu. A Visualization screen consists of 3 subscreens namely display screen, description screen, and status screen. The visualization starts.

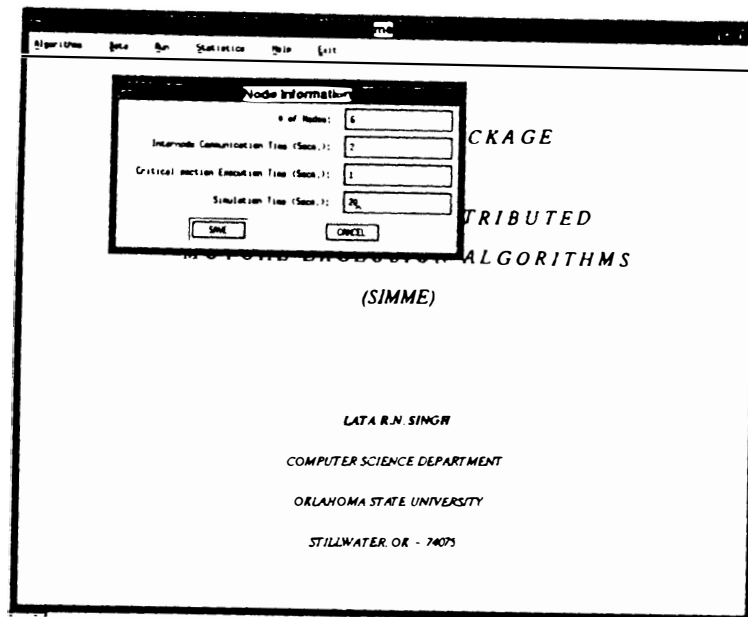


Figure 8b: Data Menu with "Node Information" selected

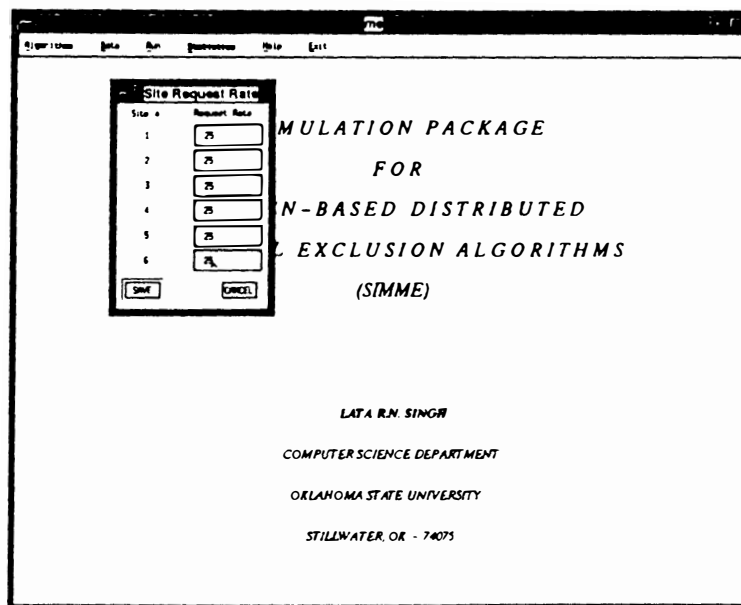


Figure 8c: Data Menu with "Site Request Rate" selected

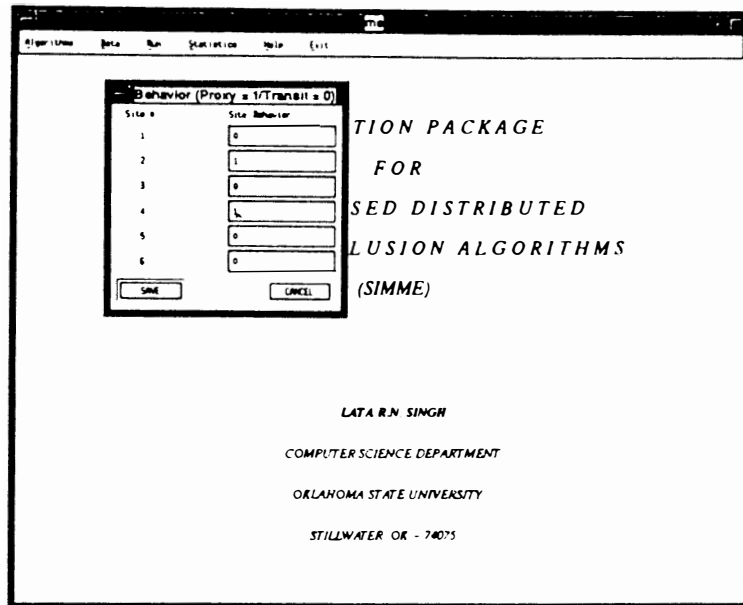


Figure 8d: Data Menu with "Site Behavior" Selected

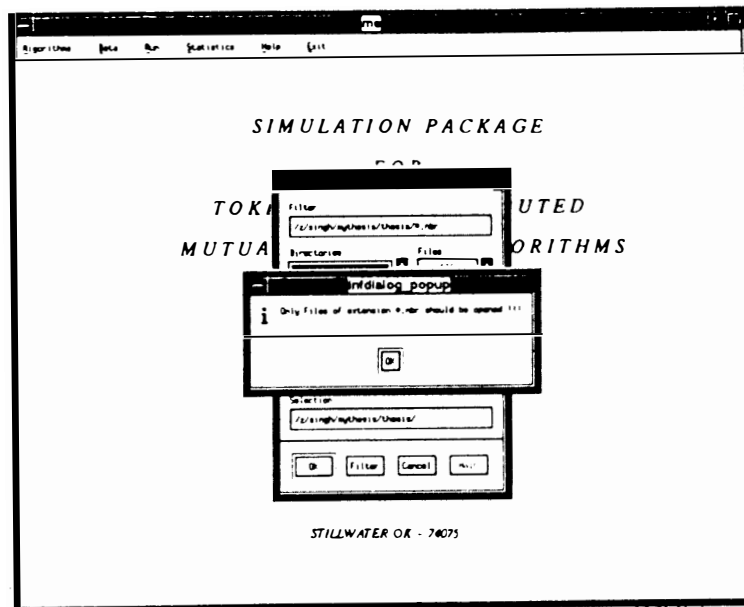


Figure 8e: Data Menu with "Site Neighbor File" selected

with displaying initial configuration in the display screen. The simulation proceeds with dynamically displaying the movement of token, the movement of request and the execution of the critical section. The description of the visualization is displayed in the description screen. The description highlights the following.

- Generation of new request at each clock tick.
- Sending/Forwarding the request to the neighboring site.
- Sending/Forwarding the token to the requesting site.
- Execution of critical section.

The information contained locally at each site in the system at every clock tick is displayed in the status screen. The snapshot of visualization is shown in Figure 8f.

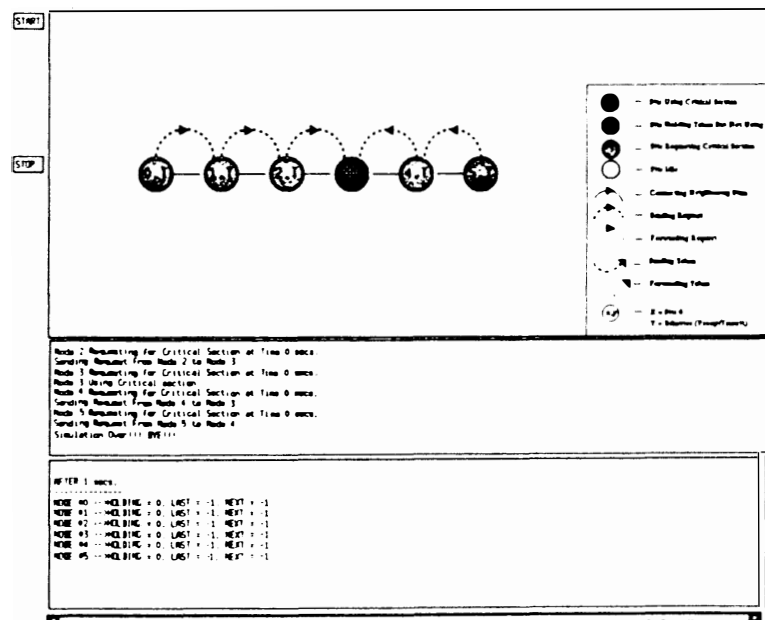


Figure 8f: A Snapshot of "Visualization" mode

Execution mode contains the description screen only. The snapshot of execution mode is shown in Figure 8g.

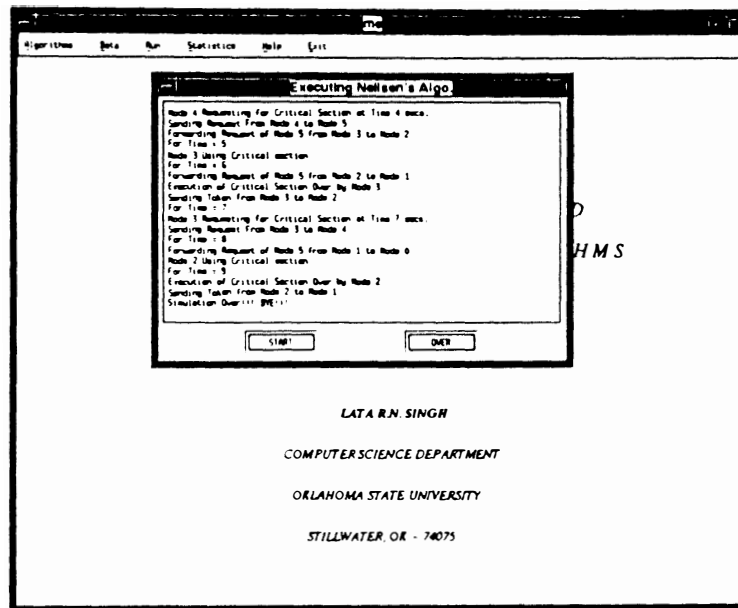


Figure 8g: A Snapshot of "Execute" mode

Statistics of the simulation can be obtained by choosing 'Statistics' menu. Statistics is displayed as a summary report in which the summary of the simulation at different load setting can be viewed at a time. The graphs are plotted by interfacing through BLT package. Snapshots for the Statistics menu are shown in figure 8h, 8i.

Help menu gives a brief description of all the algorithms implemented and also gives the key to use the package. Exit menu is used to exit the tool. The snapshot of the Help menu is shown in figure 8j and Exit menu in 8k.



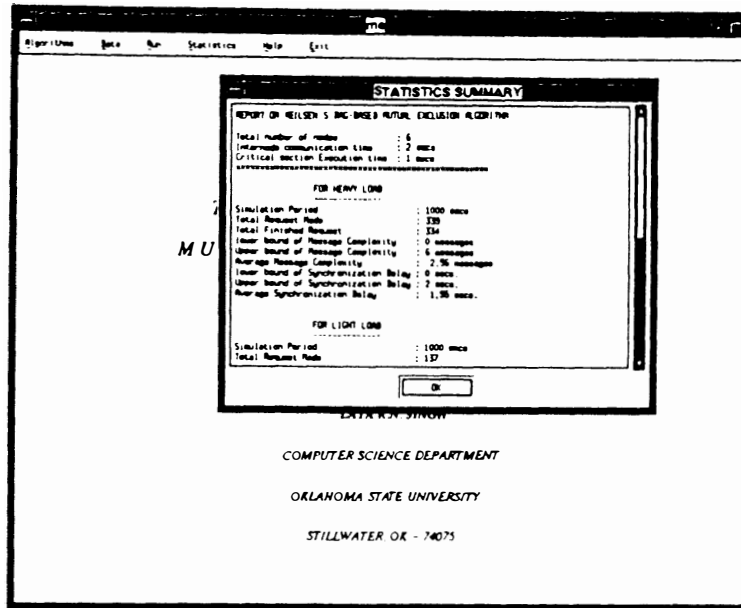


Figure 8h: Statistics Summary Screen

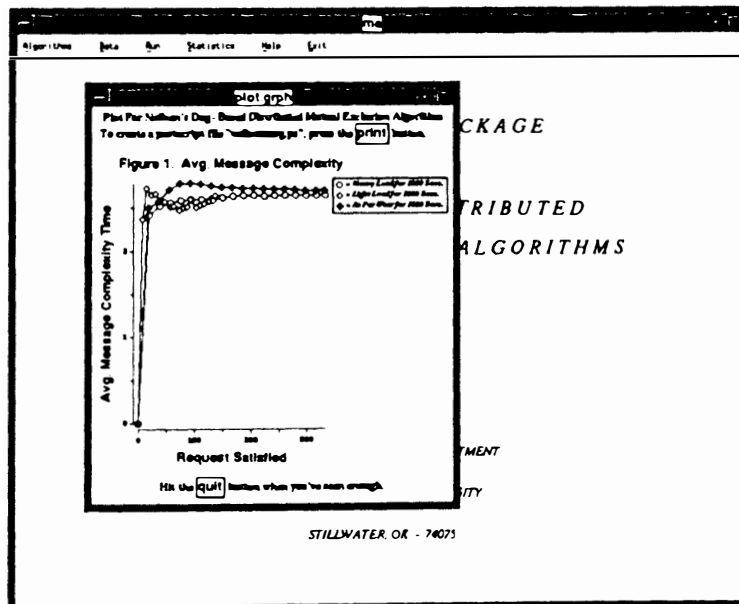


Figure 8i: A Graph plotted through Statistics Menu

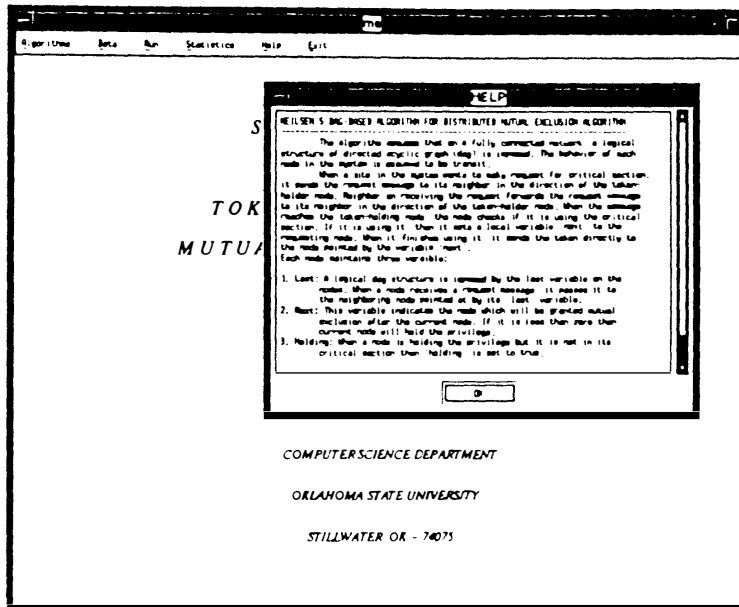


Figure 8j: Help Menu Screen

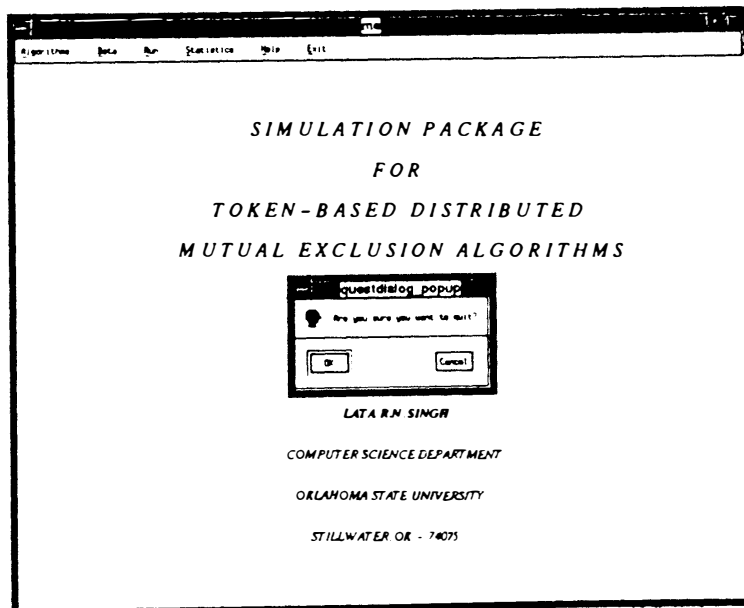


Figure 8k: Exit Menu Screen

## CHAPTER V

### SIMULATION RESULTS

#### 5.1 Sample Simulation

Sample simulations were performed using the following configuration:

Algorithm	: Neilsen's Dag-based Distributed Algorithm[NEI89]
# of Nodes	: 6 (for Visualization option) : 100 (for Execute option)
Internode Communication Time	: 7 Secs.
Critical Section Execution Time	: 5 Secs.
Simulation Period	: 100 Secs.
Requesting Probability	: Heavy Load (100%)
Sites Behavior	: Transit mode.
Topology	: Linear

The summary of statistics obtained in the visualization mode is shown in Figure 9a and that obtained in the execution mode is shown in Figure 9b. The respective graphs generated by the tool are shown in Figures 10a, 10b, 10c, and 10d.

---

 REPORT ON NEILSEN'S DAG-BASED MUTUAL EXCLUSION ALGORITHM
 

---

Total number of nodes : 6  
 Internode communication time : 7 secs  
 Critical section Execution time : 3 secs

---

 FOR HEAVY LOAD
 

---

Simulation Period : 100 secs  
 Total Request Made : 17  
 Total Finished Request : 11  
 lower bound of Message Complexity : 0 messages  
 Upper bound of Message Complexity : 5 messages  
 Average Message Complexity : 1.32 messages  
 lower bound of Synchronization Delay : 0 secs.  
 Upper bound of Synchronization Delay : 7 secs.  
 Average Synchronization Delay : 3.97 secs.

---

 FOR LIGHT LOAD
 

---

\*\*\*\* NO INFORMATION AVAILABLE \*\*\*  
 LOAD AS SET BY USER  
 \*\*\*\* NO INFORMATION AVAILABLE \*\*\*

---

Figure 9a: Summary of Statistics obtained from "Visualization" mode

---

 REPORT ON NEILSEN'S DAG-BASED MUTUAL EXCLUSION ALGORITHM
 

---

Total number of nodes : 100  
 Internode communication time : 7 secs  
 Critical section Execution time : 3 secs

---

 FOR HEAVY LOAD
 

---

Simulation Period : 100 secs  
 Total Request Made : 111  
 Total Finished Request : 11  
 lower bound of Message Complexity : 0 messages  
 Upper bound of Message Complexity : 2 messages  
 Average Message Complexity : 1.08 messages  
 lower bound of Synchronization Delay : 0 secs.  
 Upper bound of Synchronization Delay : 7 secs.  
 Average Synchronization Delay : 3.97 secs.

---

 FOR LIGHT LOAD
 

---

\*\*\*\* NO INFORMATION AVAILABLE \*\*\*  
 LOAD AS SET BY USER  
 \*\*\*\* NO INFORMATION AVAILABLE \*\*\*

---

Figure 9b: Summary of Statistics obtained from "Execute" mode

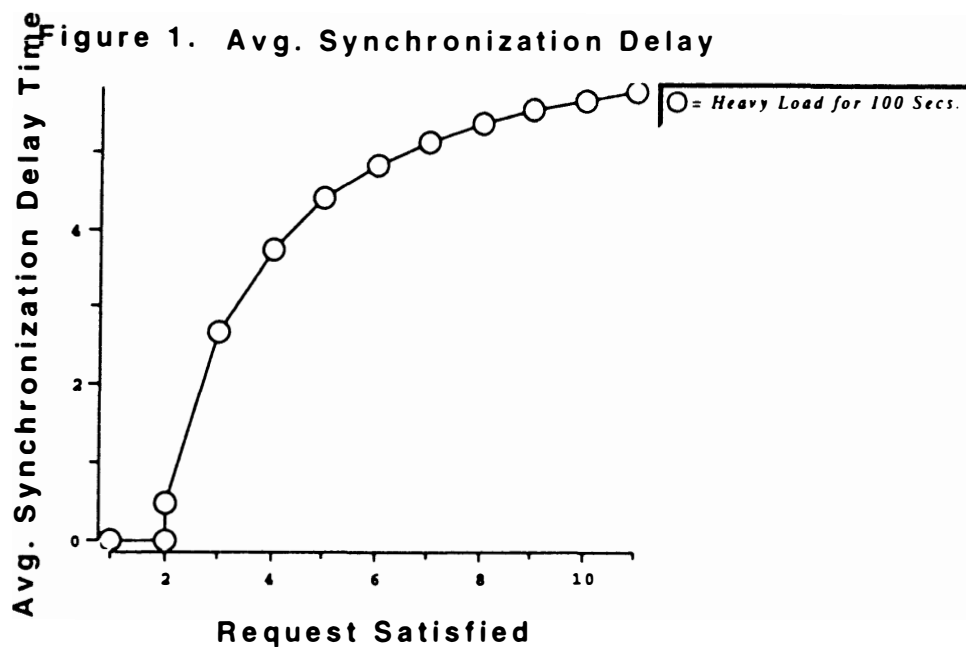


Figure 10a: Avg. Synchronization Delay Time Graph for "Visualization" mode

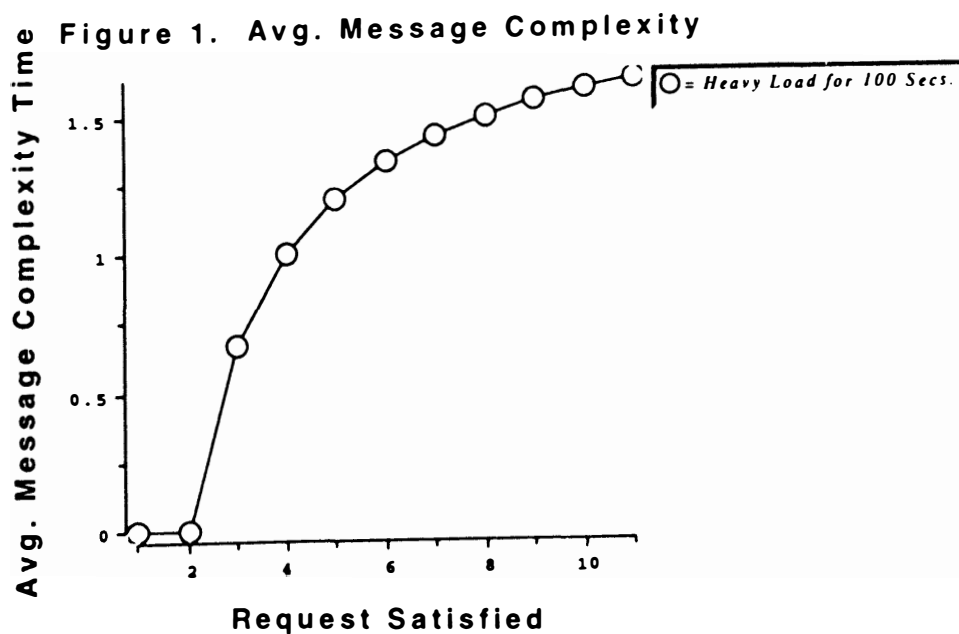


Figure 10b: Avg. Message Complexity Graph for "Visualization" mode

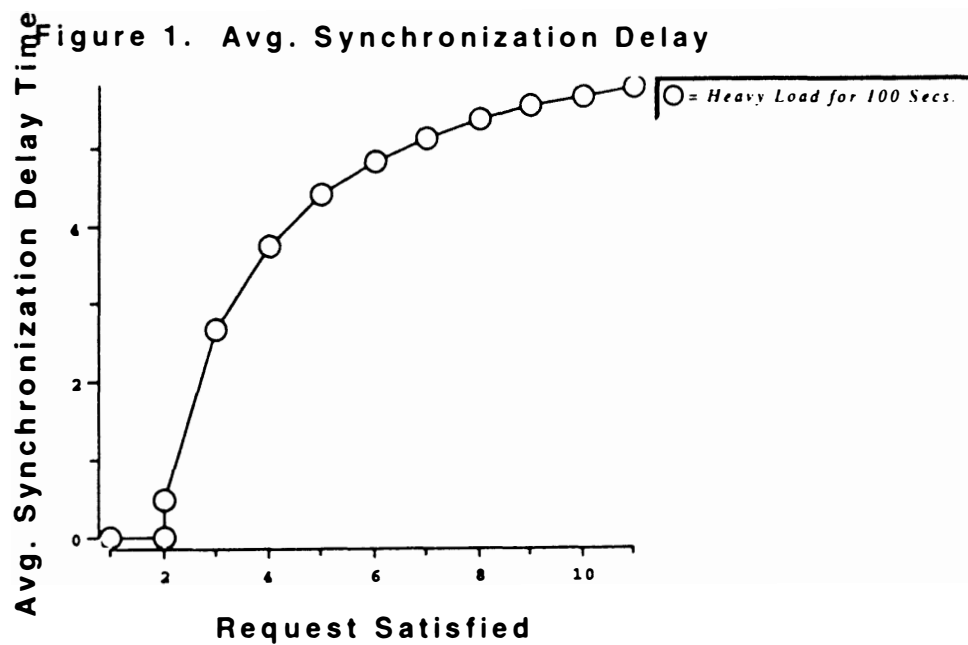


Figure 10c: Avg. Synchronization Delay Time Graph for "Execute" mode

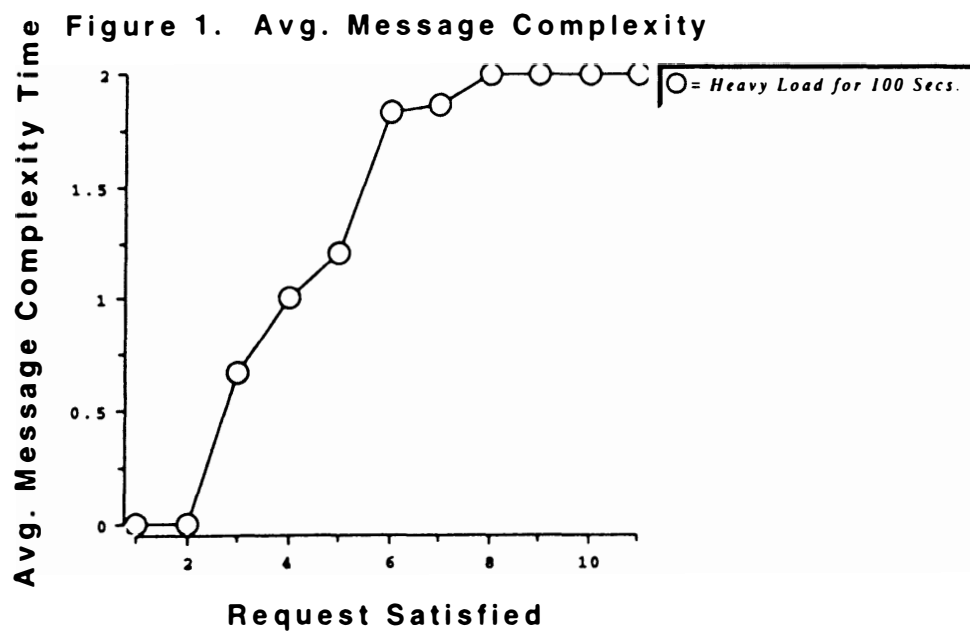


Figure 10d: Avg. Message Complexity Graph for "Execute" mode

## 5.2 Discussion of Results

After studying the algorithms and simulating each of them, the following results were obtained.

### 5.2.1 A New Classification of Logical Token-Based Mutual Exclusion Algorithms

Based on simulation, a new insight to classifying the mutual exclusion algorithms was observed. As stated earlier, distributed mutual exclusion algorithms are classified into two classes, namely: Permission-based algorithms and token-based algorithms. In token-based algorithms, there are two subclasses. They are token based algorithms with logical structure imposed on a physical network and with no logical structure imposed on a physical network. A taxonomy of distributed mutual exclusion algorithms is given in Chapter 2.

In this thesis the emphasis is on logical token based algorithms. A detailed study of these algorithms shows that the logical structure can be either static or dynamic in nature. *Static structure* is the logical structure imposed on the physical network that does not change with the movement of the request messages or the token message during the entire period of mutual exclusion by various sites in the system. However, the edges joining a site to its neighboring sites which direct the flow of messages in terms of requests or token passing, may or may not change their direction. Algorithm proposed by Neilsen and Mizuno[NEI 89] uses the technique of reversing the edges between the neighboring sites in the static structure. While the algorithm proposed by A.J. Martin [MAR 85] is with no *edge reversal* (direction between a node and its neighbor can

reverse), in which the direction of flow of request and token remain fixed. *Dynamic structure* is the logical structure imposed on the physical network that keeps changing depending on the movement of the token during the entire process of simulation. This would lead to the *reversal of the path* (direction of the path between the requesting node and the token holding node reverses) or reversal of path and edge. The algorithm proposed by Naimi and Trehel [NAI 87] uses path reversal technique in dynamic structure and the algorithm proposed by Raynal [RAYUR] is the combination of the path and the edge reversal. A Classification of logical token based algorithms in terms of static and dynamic structure is shown in Figure 11.

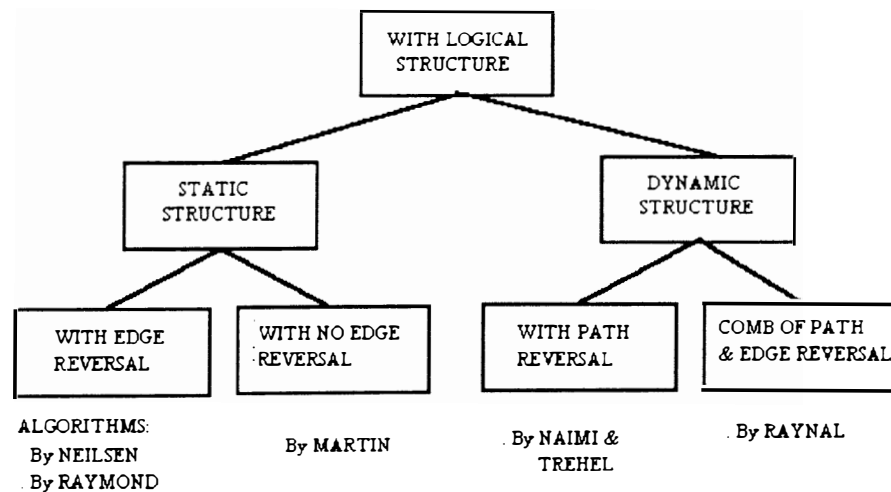


Figure 11: A Classification of Logical Token-Based Algorithms in terms of Static and Dynamic Structure

The above classification is in terms of logical structure of the system. A careful



study of the algorithms at each node suggests that algorithms can also be classified in terms of proxy and transit sites. Raymond's tree-based algorithm [RAY 89] treats each site as proxy while Neilsen in Dag-based algorithm [NEI 89] and Naimi [TRE 87] in logical rooted tree algorithm, treats each site as transit. Raynal [RAYUR] however, in the Generalized tree-based algorithm uses a combination of transit and proxy sites in the algorithm. A classification of logical token based algorithm in terms of transit and proxy sites is shown in Figure 12.

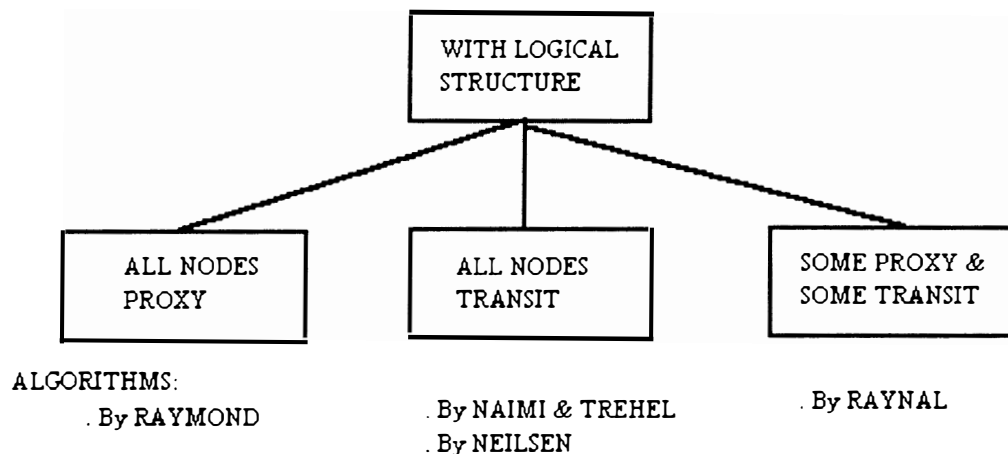


Figure 12: A Classification of Logical Token-Based Algorithm in terms of Transit and Proxy Nodes

By overlapping the above classifications for the logical token based algorithm, a two dimensional classification can be presented. Figure 13 shows the relationship between static and dynamic structure of the system verses proxy and transit behavior of sites for the logical token based mutual exclusion algorithms using some kind of reversal

technique.

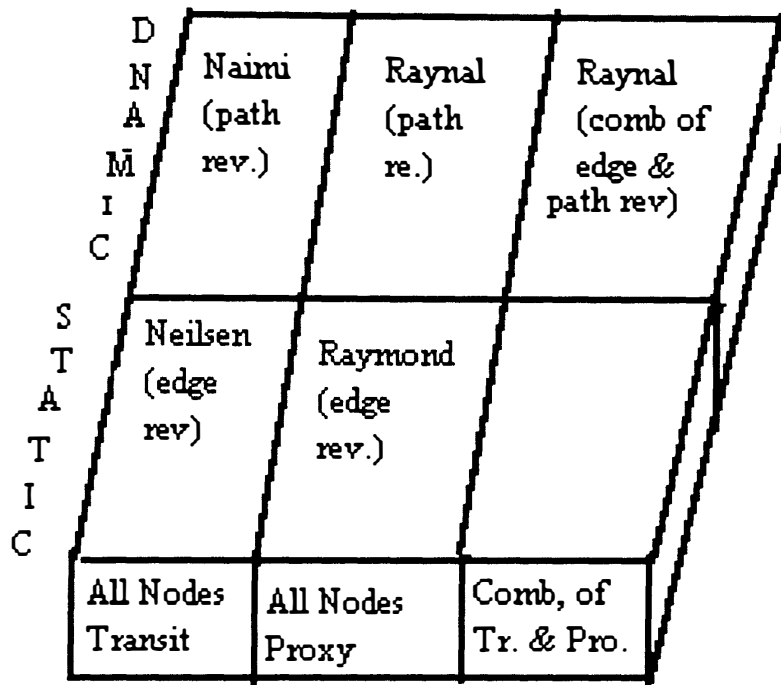


Figure 13: A Two-Dimensional Classification of Logical Token Based Algorithm using adaptive reversal techniques

### 5.2.2 Statistical Analysis

The simulation was done for logical token based algorithms using an adaptation of reversal technique, shown in Figure 13, for same number of sites, internode communication time, critical section execution time, traffic load, simulation period, and logical structure. The configuration used is shown in Table I. Linear topology of the system was set through a site neighbor file (linear6.nbr) shown in Figure 14a.

With the above configuration, the simulation was run for Heavy load, Light Load

and 50% load. The behavior of nodes for Raynal's algorithm [RAYUR] was set through a file (raynal.bev) shown in Figure14b.

# of Nodes	Comm. Time(secs)	CS Exec Time(secs)	Sim Period (secs)	Topology
6	2	1	1000	Linear

Table 1: Simulation Configuration

1	0
0 2	1
1 3	0
2 4	1
3 5	0
4	0

Figure 14a Site Neighbor File  
(linear6.nbr)

Figure 14b: Behavior Data File  
(raynal.bev)

Various statistical parameters were collected at each clock tick. These parameters were noted to calculate Message Complexity and Synchronization Delay for each

algorithm.

*Message Complexity* is defined as the average number of messages required by a node to enter the critical section [NEI 89]. Message complexity, however, depends on the topology of the system. *Synchronization Delay* is defined as the time gap, when a node, say node I, leaves the critical section and before another node, say node J, can enter the critical section. This measures the efficiency of the algorithm in passing the permission to the requesting node.

The results obtained by the simulation of all the algorithms with the above configuration are tabulated in Tables II, III, and IV. The graphical results are shown in Figures 15a, 15b, 15c, 15d, 15e, 15f, 15g, 15h.

Algo rithm	Mesg. Complexity in Msgs.			Synch. Delay Time in Secs.		
	Lower Bound	Upper Bound	Average	Lower Bound	Upper Bound	Average
Neilsen	0	6	2.56	0	2	1.95
Raymond	0	10	3.13	0	10	3.15
Naimi	0	6	3.01	0	2	1.95
Raynal	0	4	2.32	0	4	3.55

Table II: Simulation Results obtained for Heavy Load

Algo rithm	Mesg. Complexity in Msgs.			Synch. Delay Time in Secs.		
	Lower Bound	Upper Bound	Average	Lower Bound	Upper Bound	Average
Neilsen	0	6	2.56	0	12	5.12
Raymond	0	10	4.40	0	20	8.80
Naimi	0	6	2.23	0	10	4.47
Raynal	0	5	3.25	0	11	7.54

Table III: Simulation Results obtained for Light Load

Algo rithm	Mesg. Complexity in Msgs.			Synch. Delay Time in Secs.		
	Lower Bound	Upper Bound	Average	Lower Bound	Upper Bound	Average
Neilsen	0	5	2.69	2	4	2.03
Raymond	0	8	3.15	0	8	3.17
Naimi	0	6	2.91	0	2	1.95
Raynal	0	4	2.33	0	6	3.60

Table IV: Simulation Results obtained for 50% Load

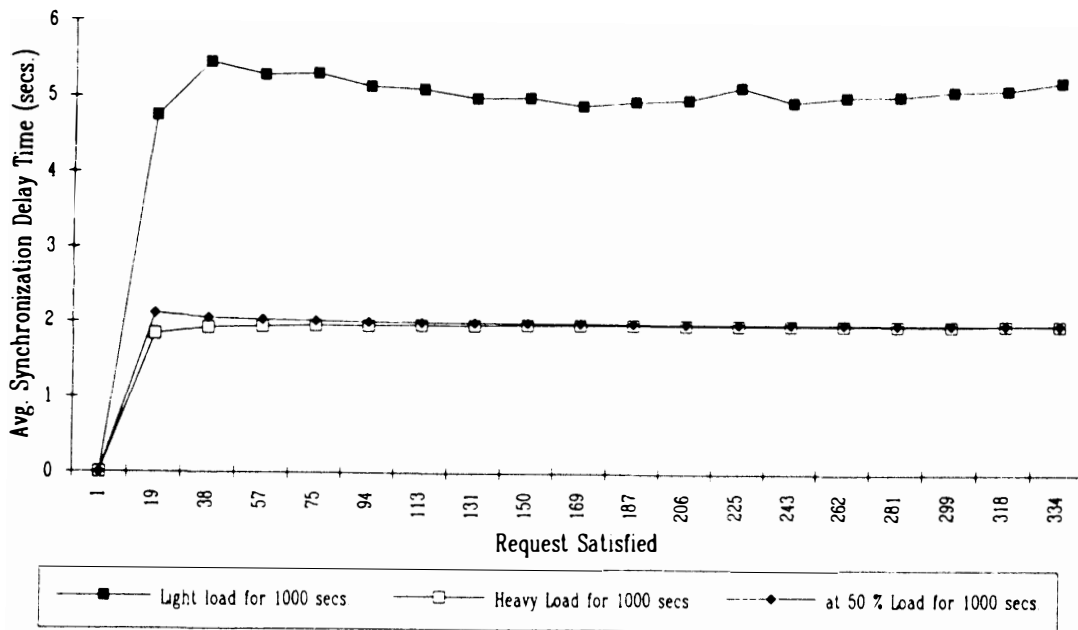


Figure 15a: Avg. Synchronization Delay Time Graph for Neilsen’s Algorithm

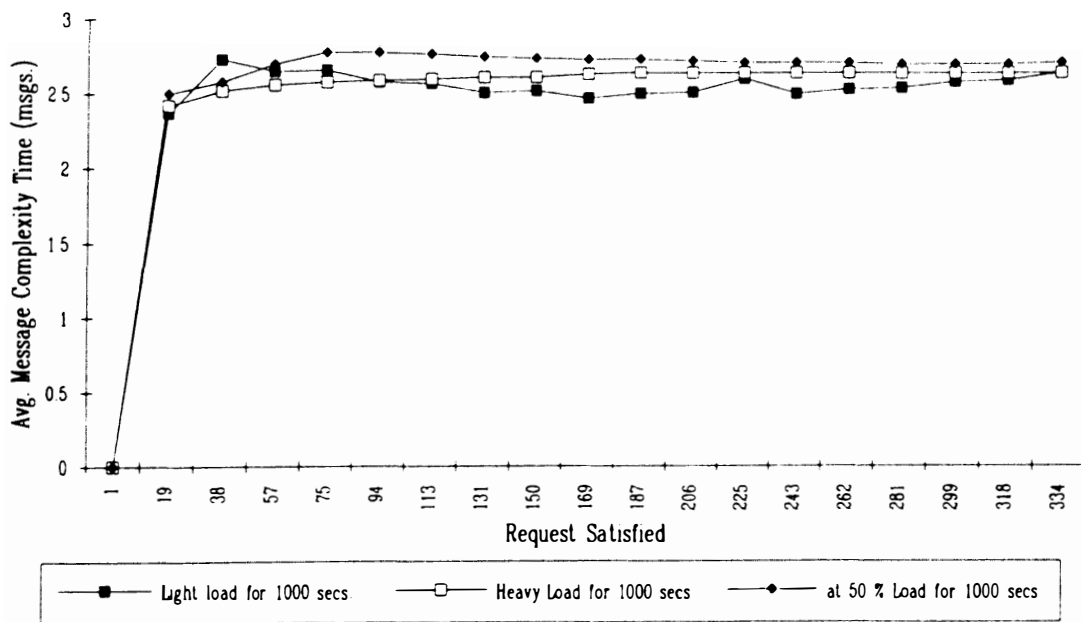


Figure 15b: Avg. MessageComplexity Graph for Neilsen’s Algorithm

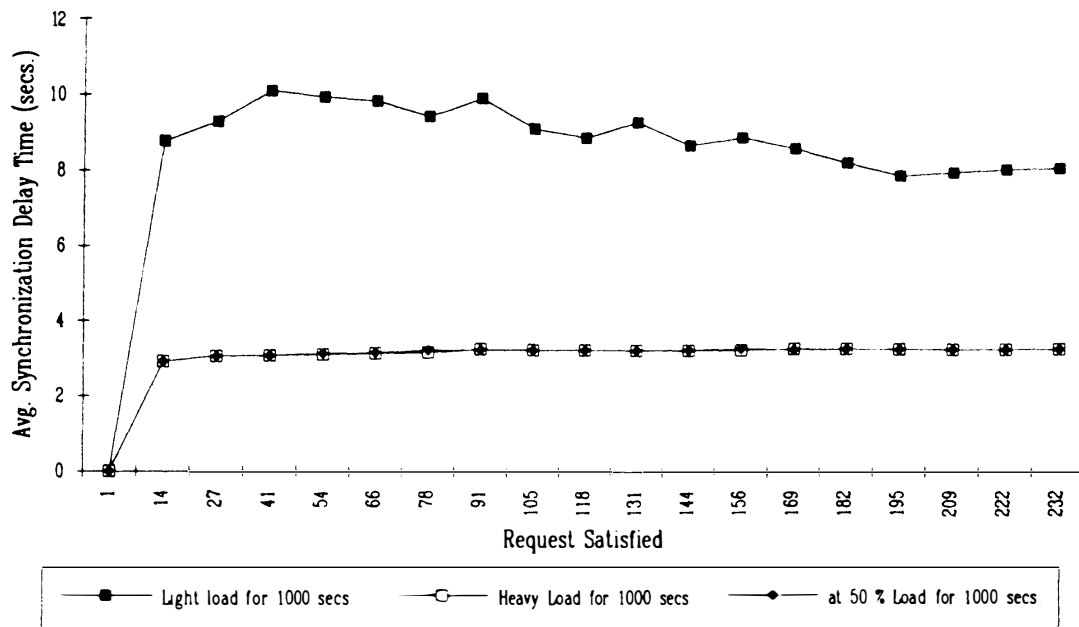


Figure 15c: Avg. Synchronization Delay Time Graph for Raymond's Algorithm

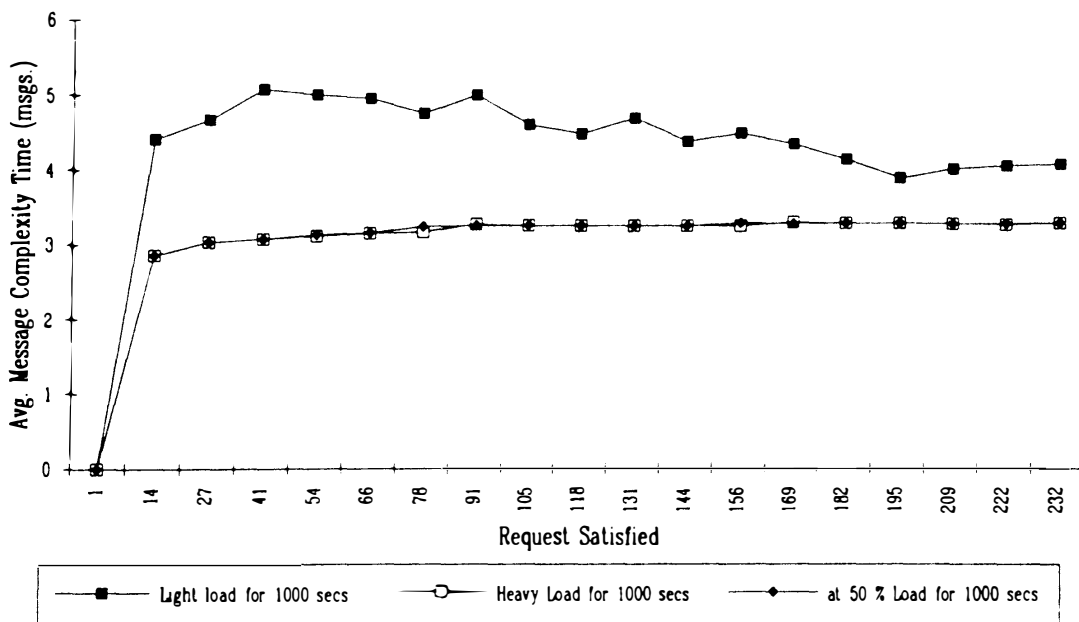


Figure 15d: Avg. Message Complexity Graph for Raymond's Algorithm

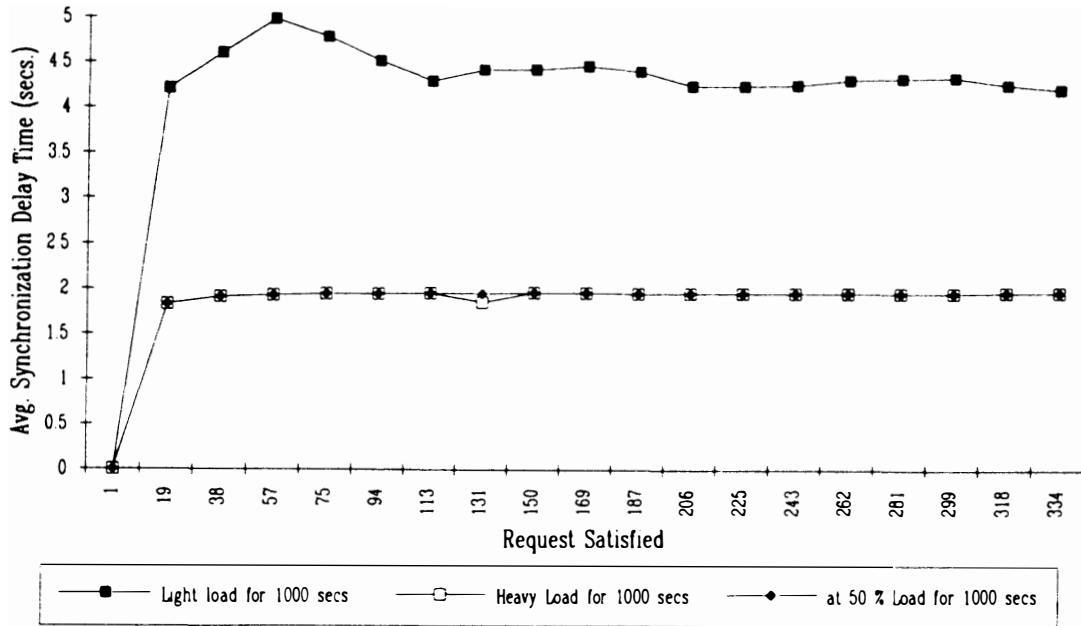


Figure 15e: Avg. Synchronization Delay Time Graph for Naimi's Algorithm

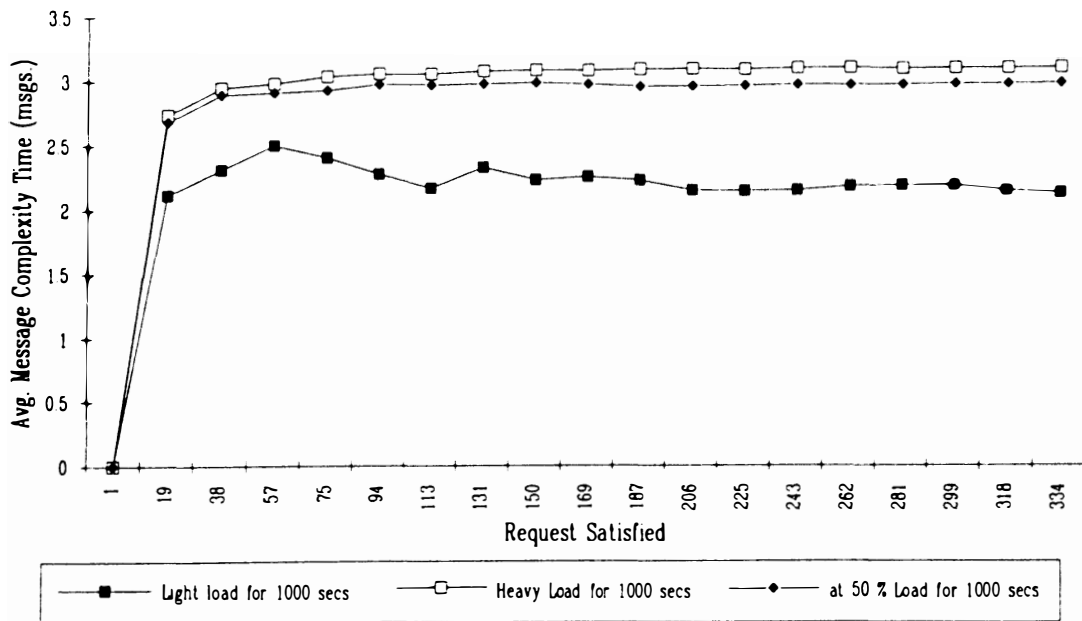


Figure 15f: Avg. Message Complexity Graph for Naimi's Algorithm



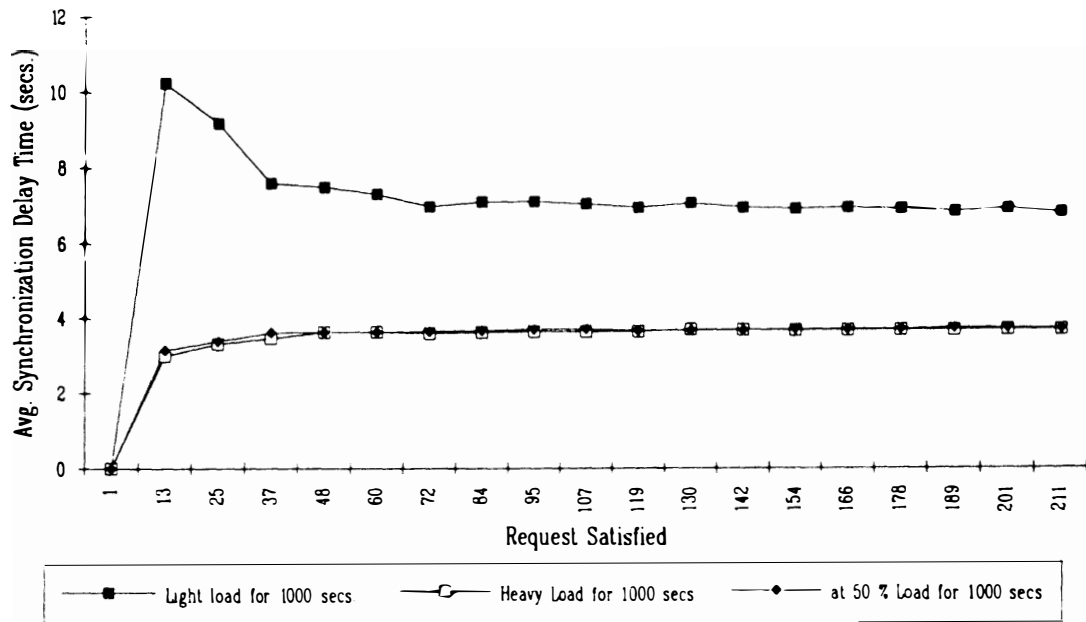


Figure 15g: Avg. Synchronization Delay Time Graph for Raynal's Algorithm

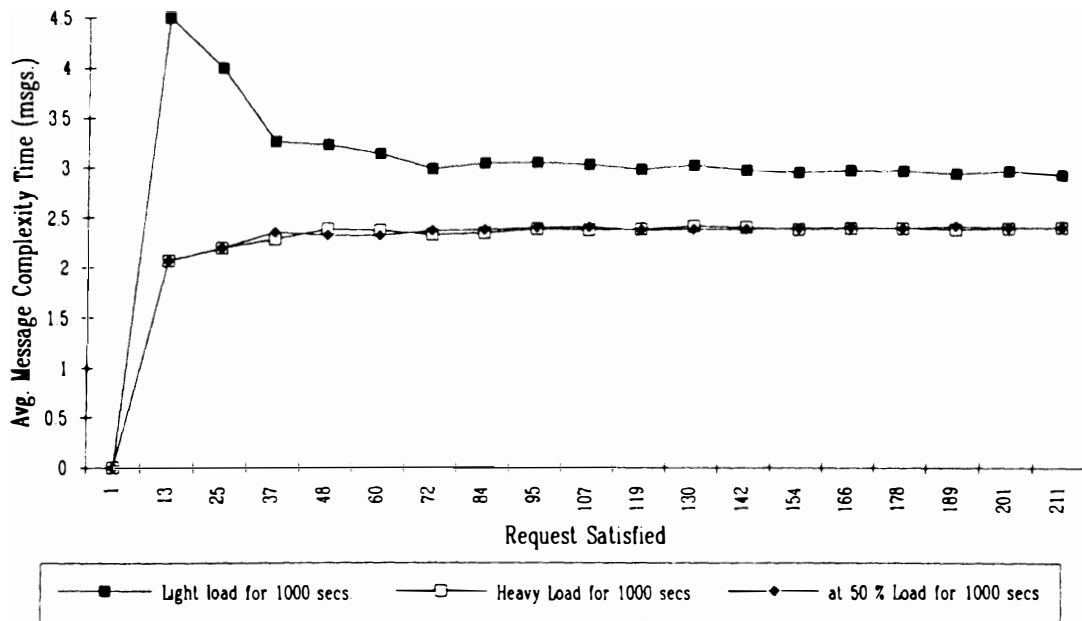


Figure 15h: Avg. Message Complexity Graph for Raynal's Algorithm

The analysis of the results obtained are given below.

5.2.2.1 Neilsen's Algorithm [NEI 89]: The maximum message complexity observed was 6 messages. This agrees with the theoretical result of  $D+1$  messages, where,  $D$  is the diameter of the topology or the length of the longest path[NEI 89]. This happens when the request is made by a node at one end of the diameter and the token is present at a node at the other end of the diameter and hence a request requires  $D$  messages to reach the token holding node and one message for sending the token. The minimum message complexity is zero message which occurs when the token holding node makes the request and so, minimum synchronization delay is zero.

The upper bound of synchronization delay varies with the loading factor. For a heavy loaded system, the delay observed was 2 secs(1 message). In the heavy load, all the time a request message is in the waiting queue of the token-holder node and hence, as soon as the token-holder node finishes the execution of the critical section, it passes the token to the requesting node directly which requires only 1 message. In the light node, the situation is different. Maximum of one request message exists in the system at all time and hence to get the token, a request has to travel all the way to the token-holder node and hence, the worst case of synchronization delay was observed as 12 secs (6 messages or  $D+1$  messages).

5.2.2.2 Raymond's Algorithm: The worst message complexity obtained for both heavy load and light load was 10 messages( $2*5d$  or  $2*D$  messages) where,  $D$  is the diameter of the system. This case occurs due to  $D$  messages required to send the request message to

the token-holder node at the other end of the system and again  $D$  messages to send token message from the token-holder node to the requesting node. Hence, the worst synchronization delay observed in light load was 20 secs (10 message =  $2 * D$  message). In the heavy load, a request is always queued in the token-holding node and atmost  $D$  messages are required to send token from the token holding node to the requesting node, so the worst synchronization delay is  $D$  messages. The observed delay in heavy load was 10 secs (5 message =  $D$  messages).

5.2.2.3 Naimi's Algorithm: This algorithm imposes a dynamic logical rooted tree structure. Each node sends request message to the node indicated by its local variable "last" and so on. The worst case is when a request has to pass through all the nodes before it reaches the token holder node and hence the upper message complexity is  $6$  messages ( $N$  messages) where,  $N$  is number of nodes in the system. The worst synchronization delay for the heavy load observed was 2 secs(1 message) and for the light load was 10 secs ( $O(\log N)$ ).

5.2.2.4 Raynal's Algorithm: Raynal's algorithm is a generalized dynamic tree-based algorithm where nodes can be either proxy or transit in behavior. The upper bound of message complexity observed was 5 messages. The upper bound of synchronization delay for light load was 11 secs (5 messages) and for heavy load was 4 secs (2 messages). The performance parameters depend on the behavior of the nodes in the system.

All the observed results agree with the theoretical results [NEI 89]. It can be observed that the heavy loaded system has less synchronization delay time than the light

loaded system. In the heavy loaded system, there is a pending request at all times in the system to be satisfied while in light loading, there is no pending request at any time in the system. Thus, it can be concluded that all algorithms perform better in the heavy loaded system.

It can also be observed that message complexity is independent of the loading factor. The minimum number of messages required to get a token remains invariant with the load as the token keeps moving in the system. Message complexity, however, depends on the topology of the system. The straight line logical structure requires the maximum number of messages to get a token while the radiating star logical structure requires the least messages per critical section request [NEI 89].

The graphs and the tables can be used to compare the performance of the algorithms. It is observed that the algorithm having transit behavior for all sites, proposed by Neilsen and Mizuno [NEI 89], gives less average message complexity and delay time than the algorithm, proposed by Raymond [RAY 89], having proxy behavior for all sites. Both of these algorithms have static structure. Naimi's dynamic logical tree-based algorithm (all sites have transit behavior) also gives good performance. Raynal's dynamic tree-based algorithm (sites can have either proxy or transit behavior) has variable performance depending on its node behavior. Thus a transit node performs better than proxy node but with transit node the system requires fully connected network, whereas system with proxy node need not have fully connected network. Dynamic structure can perform better than static structure when the loading factor will be different for all nodes. Dynamic structure would form a minimal logical structure around heavy loaded node and

hence would improve the performance.

## CHAPTER VI

### CONCLUSIONS

#### 6.1 Summary

The objective of this research was to develop a GUI tool to visualize logical token-based mutual exclusion algorithms. A simulation package was developed for several mutual exclusion algorithms. A new two-dimensional classification of logical token-based mutual exclusion algorithms showing the relationship between static and dynamic properties of sites versus proxy and transit behavior of sites in the system was developed. The classification was motivated by observing the simulation. The performance analysis of each of the algorithms in terms of message complexity and synchronization delay was done. All the implemented algorithms were studied under Heavy traffic load and Light traffic load.

The application package also had visualization capability. The working of the algorithms were visualized and the movement of the token and request messages could be seen dynamically.

#### 6.2 Conclusions

After studying the algorithms to achieve mutual exclusion in the distributed system, it can be concluded that each algorithm is based on the objective of decreasing

the message complexity and synchronization delay in the system. Each algorithm has optimized the information storage overhead at each node. In Raymond's algorithm [RAY 89], each node knows only the neighbor in the tree, while in Naimi and Trehel's algorithm [NAI 87], each node knows the "last node", the node which requested the last in the system so the request is sent only to this "last node". Neilsen [NEI 89] and Raynal's [RAYUR] algorithms also impose very little storage overhead on each node.

Topology of the network plays an important factor on its performance. Neilsen claimed that worst topology is straight line, where messages may have to be sent to the longest path or the diameter of the system. Raymond, on the other hand, claimed that the radiating star topology with one node in the center and all other in the leaf level, as the best topology. Each algorithm performed differently on the same topology adopted.

On simulation of algorithms under different traffic load, it was observed that heavy traffic load gives better performance in synchronization delay than the light traffic load. Thus, it has the obvious advantage of minimizing the number of messages when the network is complete.

### 6.3 Future Work

In developing the simulation package, assumptions were made that the messages are not lost or altered in the system, messages are delivered in the order they are sent, there is no site failure and the internode communication time and critical section execution time for all sites are same. However, the package can be made more realistic by removing all these assumptions.

A generalized algorithm can be developed by merging good features of all these algorithms in to a single algorithm. Similarly, the visualization can be improved further.



## REFERENCES

- [CAR 83] O.S.F. Carvalho and G. Roucairol. On mutual exclusion in computer networks. *Communications of the ACM*, 26(2):146-147, 1983.
- [DAN 91] Dan Heller. Motif programming manual for OSF/Motif Version 1.1, vol 6. *O'Reilly & Associates Inc.*
- [LAM 78] L.Lamport. Time,clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558-565,1978.
- [MAE 85] M.Maekawa. A  $\sqrt{n}$  algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145-159,1985.
- [MAR 85] A.J.Martin. Distributed mutual exclusion on a ring processes. *Science of Computer Programming*,5:265-276,1985.
- [NAI 87] M.Naimi and M.Trehel. How to detect a failure and regenerate the token in the  $\log(n)$  distributed algorithm for mutual exclusion. *Lecture notes in Computer Science*, 312:155-166,1987.
- [NEI 89] M.L. Neilsen and M.Mizuno. A dag-based algorithm for distributed mutual exclusion. *In IEEE* ,pp 354-360,1991.
- [RAY 89] K.Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61-77,1989.
- [RAY 86] M.Raynal. *Algorithms for Mutual Exclusion Press*, 1986.
- [RAY 91] M.Raynal. A simple taxonomy for distributed exclusion algorithms. *ACM Op. Systems Review*, Vol. 25,2 (1991), pp 47-50.
- [RAYUR] M.Raynal. A very general information structure of tree based distributed mutual exclusion algorithms. This paper is under review.
- [RIC 81] G.Ricart and A.K.Agarwala. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9-17,1981.
- [RIC 83] G.Ricart and A.K.Agarwala. Authors response to 'On mutual exclusion in computer networks' by Carvalho and Roucairol. *Communication of ACM* ,

26(2):147-148,1983.

- [SEQ 90] Symmetry Multiprocessor Architecture Overview. *Sequent Computer Systems, Inc.*
- [SIN 89] M. Singhal . A dynamic information-structure mutual exclusion algorithm for distributed systems. *In IEEE 9th International Conference on distributed Computing Systems*, pp 70-78,1989.
- [SIN 92] M Singhal. A heuristically-aided algorithm for mutual exclusion algorithm in distributed systems. *Transactions on Computers*, 38(5),1992.
- [STY 91] OSF/MOTIF Style Guide. *Prentice Hall.*
- [SUZ 85] I.Suzuki and T. Kasami. A distributed mutual exclusion algorithms. *ACM Transactions on Computer systems*, 3(4):344-349,1985.
- [TAN 91] A. Tanenbaum. Modern operating systems. *Prentice Hall.*
- [TRE 87] M.Trehel and M. Naimi . A distributed system for mutual exclusion based on data structure and fault tolerance. *In Proc. IEEE 6th Int. Conf. on Computers and Communication*. pp 35-39, 1987.
- [VAN 87] J.L.A. an de Snepscheut. Fair Mutual exclusion on a graph of processes. *Distributed Computing Vol 2*:113-115,1987.
- [XLi 91] Xlib Reference Manual, vol. 2. *O'Rielly & Associates Inc.*

## APPENDICES

## APPENDIX A

### GLOSSARY AND TRADEMARK INFORMATION

**Centralized System:** It is composed of the number of terminals sharing single resource like CPU, Memory, and other peripherals.

**Client-Server Model:** In client-server model, a process known as server is responsible for providing some facilities to other processes known as clients.

**Critical Section:** The part of program where the shared resource is accessed is called a critical section "cs".

**CPU:** Central processing unit.

**DAG:** Directed acyclic graph is a simple digraph which does not have any cycles.

**Deadlock:** A situation when processors get blocked forever and no more work can be done after that.

**Distributed System:** A distributed system is one that runs on a collection of machines, not having shared memory or a global clock, but still appears to function like a single machine.

**Dynamic Structure:** The logical structure imposed on the physical network that keeps changing depending on the movement of the token during the entire process of simulation.

**Event Counters:** Event counter is a special kind of variables to obtain mutual exclusion. The operations defined on event counter "E" are:  
    **Read(E):** Returns the current value of E.  
    **Advance(E):** Automatically increments E by 1.  
    **Await(E,V):** Wait until E has a value of V or more.

**Graph:** A graph G consists of a nonempty set V called the set of nodes (points, vertices) of the graph, a set E which is the set of edges of the graph, and a mapping from the set of edges E to the set of pairs of elements of V.

**Graphical User Interface (GUI):** A visual representation of a computer program functions that can be manipulated by nonprogrammatic means.

**Heavy Load:** A situation when all sites in the system have high probability of making request at all the time.

**Light Load:** A situation when all the sites in the system have low probability of making request at all the time.

**Message Complexity:** Average number of messages required by the node to enter the critical section.

**Line Topology:** All sites in the network are connected logically in form of a line.

**Monitor:** A monitor is collection of procedure, variable, and data structures that are all grouped together in a special kind of module. It has an important property that helps for achieving mutual exclusion: only one process can be active at a time in a monitor.

**Multiprocessing System:** A single system where more than one CPUS work together.

**Mutual Exclusion:** This means when one process is accessing the resource, other processes are excluded from accessing the same resource.

**Node:** Refer Site.

**Physical Network:** The actual network connection between different CPUS in the distributed system.

**Protocol:** A mutual agreement between a client and server to accomplish certain actions.

**Proxy Behavior:** When a node  $i$  receives a request from a node  $j$ , it takes the request on its own account. It now considers node  $j$  as its mandator and request the token for itself from its parent( $i$ ).

**Race Condition:** A situation where many processes are using a shared resource and correct operation of the system depends on when processess access that shared resource.

**Radiating Star Topology:** All sites in the network are connected logically in the form of a radiating star.

**Semaphor:** An abstract data type or object with data types as an integer or binary, a queue, and a number of operations such as P, V, initialization. P(s) is used to block a other process, if one is already in critical section. V(s) is to wakeup a process which is waiting for critical section.

**Site:** Each CPU or maching which is connected by a high speed network in a distributed

system.

**Starvation State:** The state of a process when it cannot access a shared resource for indefinite period of time in a multiprocessing system.

**Static Structure:** The logical structure imposed on the physical network that remains the same throughout the process of mutual exclusion of various sites in the system.

**Synchronization Delay:** Time gap between when a node *i* leaves the critical section and before another node *j* can enter the critical section.

**Token:** A privilege or priority that circulates around the logical structure in a distributed system.

**Transit Behavior:** When a node *i* receives a request from node *j* in a system, it just forwards it to its parent(*i*). This behavior is called transit behavior.

**Widgets:** A user interface component such as menu, scrollbar, or dialog box.

**X Window System:** It allows programmers to develop portable GUI.

## TRADEMARK INFORMATION

OSF/Motif is a registered trademark of the Open Software Foundation.

The X Window System is a registered trademark of the Massachusetts Institute of Technology.

UNIX is a registered trademark of AT&T.

DEC is a registered trademark of Digital Equipment Corporation.

DYNIX, DYNIX/ptx, Sequent, and Symmetry are registered trademarks of the Sequent Computer System, Inc.

## APPENDIX B

### USER GUIDE FOR "SIMME"

#### A. INTRODUCTION

The graphical simulation tool "SIMME" is developed to visualize logical token-based distributed mutual exclusion algorithms. The working of the algorithms can be visualized and the movement of the token and request messages could be seen dynamically. This tool has been implemented on Sequent Symmetry S/81 system running X window system using Motif widget set. Users may interact with the simulation package by selecting, clicking or dragging with mouse the graphic element or by typing in the data through the keyboard in the graphic element. Some of the graphic elements used in this package are MainWindow, Menubar, Menu, FileSelectionBox, Drawing Area Widget, Pushbutton, Text Widget, Scrollbars. They are explained briefly as follows:

MainWindow is used to organize the contents of a primary window. A main window frame is the client area and can include Menubar, Scrollbar, command area and message area.

Menu is used to organize a collection of buttons, labels, and separator in a horizontal, vertical or 2-dimensional layout within a separate menu window.

Menubar organizes a collection of CascadeButtons in a horizontal layout at the top of a MainWindow.

FileSelectionBox is used to select a file to be read or written from the list of files in a directory.

Drawing Area Widget provides a blank canvas for interactive drawing using basic Xlib drawing primitives.

Pushbutton is a button with a label on it which can be clicked using a mouse to perform the associated action.

Scrollbars are used to scroll the visible area of a component.

Text Widget provides a full-featured text editing capabilities to the user to enter

text.

## B. DESCRIPTION

Simulation tool consists of a horizontal menubar at the top of the MainWindow. Menubar consists of "Algorithms", "Data", "Run", "Statistics", "Help", "Exit" menupads. The menus can be pulled down by clicking with the mouse on the corresponding menupads. The order of entering data is important to run the simulation. The order to be followed should be the order in the menubar; i.e. first "Algorithm" menu; second "Data" menu; third "Run" menu; fourth "Statistics" and so on.

### 1. Algorithm Menu

This menu is used to select the algorithm to simulate. It consists of four options namely, "Nielsen", "Raymond", "Naimi", "Raynal". An algorithm is selected by clicking on one of these.

### 2. Data Menu

After the algorithm is selected, the next step is to provide require data. Data menu is used for this purpose. Data menu consists of four cascadebuttons namely, "Node Information", "Requesting Prob.", "Behavior Info", and "Site Neighbor File". Initially only "Node Information" button is active and the rest of the buttons are inactive. This is done to make user to enter data in this part first before they proceed furthur. When "Node Information" button is clicked, a "Node Information" dialog box comes up. Through this dialog box, number of nodes in the system, internode communication time (time to send message from one node to another node), critical section execution time, and simulation time (time one wants to run the simulation) can be entered. This information is fed in through the keyboard in the text widget. To go from one text field to another text field, use mouse to click on that text field. The dialog box has "Save" and "Cancel" pushbuttons. The information entered in the dialog box is saved by pushing the "Save" button and the information is ignored by pressing the "Cancel" key.

After the data is entered in the "Node Information" dialog box, all the other cascadebuttons of data menu are activated. When the "Requesting Prob." cacadebutton is clicked, another pulldown menu comes up to request the user to set the request rate of each sites in the system. The request rate can be set as heavy load by clicking on "Heavy Load (100%)" button or can be set as light load by clicking on "Light Load" pushbutton.



When "As set by user" button is clicked, one more pulldown menu comes up requesting for "From Screen" or "From External File". User can key in the individual request rate probabilities from range 0 to 100 for each node in the system by clicking "From Screen" button and through "Site Request Rate" dialog box. "From Screen" dialogbox contains two pushbottom; "Save" and "Cancel", when the number of sites in the system is less than 20. When the number of sites in the system is more than 20, two more pushbuttons will be visible. They are "pgup" and "pgdn". "pgup" will do page up function and "pgdn" will do page down function. When "save" is clicked, the data of this dialogbox is saved in the memory for the simulation and a dialogbox comes up to request for the filename to save so that next time this file can be retrieved directly instead of keying the data from the keyboard for this part of data. This also contains "Save" and "Cancel" button. A file containing request rate of each site can be retrieved by chosing "From External File" button. This button pops up a FileSelectionBox with an information box displaying the message regarding the file to be selected. After "OK" button is pressed, user can select a file from current directory or any directory by setting the filter. The file containing the required information is selected by double clicking on the filename or by selecting the filename and then clicking on "OK" pushbutton.

"Behavior Info" cascadebutton is clicked to set the behavior for each node. This menu is similar to "As set by user" button. Behavior is set as proxy by keying in "1" for that site and as transit behavior key in "0".

Similarly the topolgy of the network can be set through the external file. This can be done by clicking "Site Neighbor File" and selecting the desired file from the FileSelectionBox.

### 3. Run Menu

After all the data is entered, a user is ready to start the simulation. Run menu consists of two pushbuttons namely, "Visualization" and "Execute". When "Visualization" is clicked, the simulation can be visualized graphically. A window containing a "Start" and "Stop" button, a drawing area widget and two text widgets come up. The simulation starts by clicking the "Start" button and the "Stop" button is pressed to interrupt the simulation or close this window after the simulation is complete. The simulation is visualized graphically in the drawing area while in the text widget below drawing area displays the information about each action taken during the process of simulation. The content of data structure for each node is displayed in the last textwidget after every clock tick.

User can run the simulation without visulizing it by pressing "Execute" button of run menu. A dialog box consisting of the text widget and two pushbuttons namely "Start" and "Over" comes up. The simulation is started as described above by pressing "Start"

button. Once the "Start" button is clicked both the buttons of the dialogbox are deactivated and the progress of the simulation is displayed on the text widget by printing proper messages. After the simulation is over the buttons are activated again and the window can be closed by pressing "Over" button.

#### 4. Statistics Menu

This menu consists of "Summary" and "Graph" cascadebuttons. Summary of the simulation can be displayed by clicking on "Summary" button. When this is done, another pulldown menu comes up requesting the algorithm name for which the summary is to be displayed i.e. "Neilsen Algo.", "Raymond Algo.", "Naimi Algo.", and "Raynal Algo.". The statistical summary for any of these algorithms can be displayed by clicking on these buttons. When one of these algorithms is selected, "Statistics Summary" dialogbox comes up on which the summary is displayed. This dialogbox can be closed by pressing on "OK" button. Hard copy of summary can be obtained by printing files "neilsen.sum", "raymond.sum", "naimi.sum", "raynal.sum" files from the shell prompt.

Graphs can be displayed similarly i.e. press "Graph" button and then the algorithm name button and the graph window will come up. To print the graph, click on "Print" button on the window. This generates a postscript file and this can be later printed on a postscript printer. The window can be dismissed by pressing "Quit" button.

#### 5. Help Menu

Help menu gives help to use this tool. Help menu consists of 3 pushbuttons. By clicking on the related button the help information can be displayed on the dialogbox.

#### 6. Exit menu

To exit the tool, Exit menu can be clicked and then a question dialog box comes on the front screen to verify the same. By clicking "OK", button the tool can be quit otherwise the exit message is ignored.

## APPENDIX C

### SYSTEM ADMINISTRATOR GUIDE FOR "SIMME"

The simulation tool 'SIMME' is developed using Motif widget set release IV on Sequent Symmetry S/81 running X window system. 'Makefile' is provided to compile and generate executable code 'SIMME'. The tool can be used to simulate distributed mutual exclusion algorithms in two different modes namely, Visualization mode and Execute mode.

The procedures to simulate each algorithm and mode of operation are stored in a program file and their respective declaration are stored in a header file. The list of filenames, their respective algorithms, and the mode of execution are shown in Figure 16.

Algorithm	Mode of Execution	Program file	Header file
Neilsen	Execute	neilsen.c	neilsen.h
Neilsen	Visualization	neilvisu.c	neilvisu.h
Naimi	Execute	naimi.c	naimi.h
Naimi	Visualization	naivisu.c	naivisu.h
Raymond	Execute	raymond.c	raymond.h
Raymond	Visualization	rayvisu.c	rayvisu.h
Raynal	Execute	raynal.c	raynal.h
Raynal	Visualization	raynalvisu.c	raynalvisu.h
-----Main Program -----		tdmea.c	tdmea.h

Figure 16: List showing Algorithms and its corresponding files

In the Visualization mode, the simulation can be done for a system with a maximum of eight nodes. This constraint is due to limited size of the display screen. The data structure used in this mode is an array of size 10. Hence, with any increase in the maximum number of nodes in the system, the array size should also be increased accordingly. In the Execute mode, the dynamic movement of the token and the request message is not displayed on the screen and hence the simulation can be done for unlimited number of nodes in the system. The memory is allocated dynamically depending on the number of nodes in the system as entered by the user. However, the limitation on the

number of nodes is set by the memory of that computer.

In accepting the probability request rate and the behavior of each node from the screen, a dialogbox containing a table of maximum 10 entries is displayed on the screen. This can be increased by increasing `'max_table'` variable in `'tdmea.h'` header file. Correspondingly the table structure which is an array of 20 elements, should also be modified.

To accept filename, a string of 20 characters are used. For example, string `latestfile` in `'globalme.h'` header file and string `'okfname'` in `'tdmea.h'` header file use 20 characters and 15 characters respectively. These sizes can also be increased if needed.

The variable `'setwpr_pos'` is used to hold the last cursor position in the text dialogbox which is popped up in the Execute mode and Visualization mode to print the simulation process information on the screen. This variable is of the type integer and hence the maximum cursor position is restricted to range of the integer variable. After this it is reset to zero.

An output file is created to store the simulation results. While running the simulation, after every clock tick, various statistical parameters are noted. They are stored in the output file. The nomenclature of the output filename is as follows: first four character of the algorithm name + load factor of the simulation + extension of the file (`'out'`). For example, the filename of Raymond's algorithm running for light load will be `'raymlight.out'`.

These files are processed to create the summary file. To display statistical summary for a selected algorithm, files with extension `'out'` for that algorithm is opened and the final statistical result is calculated (Refer `PrtSumm()` function of `tdmea.c` program). The summary is stored in a file named as follows: algorithm name + file extension (`'sum'`). For example, summary file for raymond's algorithm will be `'raymond.sum'`.

For the help menu, the description of each option is stored in a separate file. The files describing the algorithms are `'naimi.hlp'`, `'raymond.hlp'`, `'raynal.hlp'`, and `'neilsen.hlp'`. The help file regarding simulation package is `'aboutsimsim.hlp'` and the help file about the tool description is `'misc.hlp'`.

A list of program files and header files is shown in Appendix D. If any of the program file is modified then "SIMME" should be recompiled. If any new program file is added in the application, Makefile should be modified accordingly.

## APPENDIX D

### LISTING OF PROGRAM FILES

The design and implementation issues of simulation package 'SIMME' is discussed in Chapter IV. The list of program files and header files are shown in this section.

#### HEADER FILES:

me.h  
globalme.h  
tdmea.h  
drawfunc.h  
neilsen.h  
neilvisu.h  
naimi.h  
naivisu.h  
raymond.h  
rayvisu.h  
raynal.h  
raynalvisu.h

#### PROGRAM FILES:

tdmea.c  
execfunc.c  
drawfunc.c  
neilsen.c  
neilvisu.c  
naimi.c  
naivisu.c  
raymond.c  
rayvisu.c  
raynal.c  
raynalvisu.c

The source code for "SIMME" can be obtained from the following address:

Dr. K.M. GEORGE (Prof.),

Computer Science Department,  
Math Science Building,  
Oklahoma State University,  
Stillwater, OK-74078 .

## VITA

Singh Lata R. N.

Candidate for the degree of

Master of Science

Thesis: A GRAPHICAL SIMULATION TOOL FOR LOGICAL TOKEN-BASED  
DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Ahmedabad, INDIA, on July 04, 1966, daughter of  
R.N. Singh and Suvidya.

Education: Graduated from St. Xavier's College, Ahmedabad, INDIA in May  
1986; received Master of Science degree in Physics from School of  
Science, Gujarat University, Ahmedabad, INDIA in June 1988. Completed  
the requirements for the Master of Science degree in Computer Science at  
the Computer Science Department at Oklahoma State University in  
December 1994.

Experience: Worked as Lecturer and Programmer in St. Xavier's College,  
Ahmedabad, INDIA; employed by Oklahoma State University, Biosystems  
and Agriculture Engineering Department as Graduate Research Assistant  
from January 1993 to May 1993.