Chapter 2

Performance Comparison of Token- and Tree based Algorithms

2.1 Introduction

In this chapter we present a performance comparison of token and tree based mutual exclusion algorithms. The effect of varying the connectivity of the network topology on various static and dynamic tree based mutual exclusion algorithms is studied. These algorithms have generally been evaluated according to the number of messages exchanged among the sites per CS execution and the average time delay in granting the CS. In practice, network topologies have significant impact on the design and performance of mutual exclusion algorithms. When a message has passed by (N-1) intermediate nodes before it arrives at the destination, the message transmission complexity should be N instead of one because the message has been stored and forwarded N times. Therefore, the message complexity should be measured by counting the number of communication hops in the network per CS execution instead of number of messages exchanged among sites per CS execution. In dynamic logical structure based algorithms, the network is assumed to be fully connected and the performance has been measured with this assumption [Cha96, Joh95]. If the underlying network is not fully connected, then the above mentioned performance measure for message traffic does not give the complete picture. The actual performance of these algorithms depend upon the
underlying physical network, to be more specific, on the degree of the connectivity of the underlying network. If the connectivity is very high, then the performance is very good, but as connectivity is reduced, the performance also deteriorates. In this paper we investigate the effect of the degree of connectivity of the physical network on the performance of the following token- and tree based algorithms: Raymond's algorithm [Ray89b], Helary et al's algorithm [HMR94], Chang et al's algorithm [CSL90a] and Naimi et al's algorithm [NTA96]. For this purpose we have chosen Chordal Rings as the network topology. In Chordal Rings [AL81] the connectivity can be varied by uniformly increasing or decreasing the degree of the nodes. Raymond's algorithm assumes a static logical tree superimposed on the underlying network and the root holds the token. As token is passed, the tree remains unchanged and only the direction of edges change. Here, we use the minimum spanning tree of the underlying physical topology as the static logical structure, with root holding the token. Naimi et al, Helary et al and Chang et al assume a dynamic rooted tree imposed on the underlying physical network.

In this chapter first we define the System model, Performance model and the Network topologies used to compare the performance of the algorithms which is followed by a brief description of these algorithms and a comparative performance analysis.

2.2 System Preliminaries

2.2.1 System Model

A distributed system consists of N sites, uniquely numbered from 1 to N. Each site contains a process that makes a request to mutually exclusively access the CS. This request is communicated to other processes. Message propagation delay is finite but
unpredictable. The communication network is assumed to be reliable (i.e., messages are neither lost nor duplicated and are transmitted error free) and sites do not crash. There is one CS in the system, and any process currently in the CS will exit in finite time. Each site has a single process running, therefore in this paper we use the terms site, node and process interchangeably. Moreover a site cannot issue another request until the current request is granted and the process itself exits the CS.

2.2.2 Performance Model

The operation of mutual exclusion algorithm is very complex and is quite difficult to analyze mathematically. Analytic performance study of mutual exclusion algorithms is intractable due to the rapid growth of the cardinality of the state space with the number of the sites in the system. Therefore we have carried out the performance analysis using the simulation techniques. The performance model used in this paper is similar to the one used in [Sin89]. We assume that the requests for CS execution arrive at a site according to the Poisson distribution with parameter $x$. Message propagation delay between any two sites is taken as 10 time units and the time taken by a site to execute the CS is taken as 1 time unit. Simulation experiments were carried out for a homogeneous system of 16 sites and 64 sites for various values of the traffic of CS requests ($x$). In this performance comparison the following two performance measures are considered:

- **Message traffic**: The average number of messages exchanged among the sites per CS execution. Message complexity is measured by counting the number of communication hops in the network for a given topology.
• **Time Delay**: The average time delay in granting the CS, which is the period of time between the instant a site invokes mutual exclusion and the instant when the site enters the CS.

These performance measures are probabilistic in nature, therefore we collect the values of these variables for 5000 CS executions. The performance of the algorithms is studied by simulating it using MAISIE simulation language [BL94]. Maisie is a C-based discrete event simulation language. It adopts the process interaction approach to discrete event simulation. An object in the physical system is represented by a logical process. Interactions among the physical processes are modeled by time stamped message exchanges among the corresponding logical processes. All sites of the distributed system are simulated as entities which have their local memory and communicate with each other via buffered message passing. Every entity is associated with a unique message buffer. Asynchronous send and receive primitives are provided by MAISIE to respectively deposit and remove messages from the message buffer of an entity.

### 2.2.3 Network Topology

The performance analysis is carried out on ring, fig. 2.1, and chordal ring topologies. Chordal rings are graphs in which the basic topology is that of a ring and chordal rings are obtained by increasing the degree of all nodes by adding extra links. In general, the more links added, the higher the node degree and the shorter the network diameter. Chordal rings of degree 3,4 are shown in fig. 2.2 & fig. 2.3 respectively. Another topology we use for the performance comparison is called barrel shifter, fig 2.4. It is
obtained from the ring by adding extra links from each node to those nodes having a
distance equal to a integer power of 2. This implies that node i is connected to node j if 
\[ j - i \mid = 2^r \] for some \( r = 0,1,2,\ldots,n-1 \) and the network size is \( N = 2^n \). Such a barrel
shifter has a node degree of \( d=2n-1 \) and a diameter \( D=n/2 \). The connectivity in the barrel
shifter is increased over that of any chordal ring of lower node degree. The degree and
diameter of these topologies for network sizes of 16 & 64 nodes are given below in
Table 1.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Degree 16 Nodes</th>
<th>Diameter 16 Nodes</th>
<th>Degree 64 Nodes</th>
<th>Diameter 64 Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>Chordal ring I</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>Chordal ring II</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Barrel Shifter</td>
<td>7</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table 1**

We study the effect of increasing the connectivity of the network from that of a ring
(which is one of the lowest connected network topologies) to a barrel shifter (which has
very low diameter and is almost comparable to fully connected networks) on the
performance of the token-tree based mutual exclusion algorithms.
2.3 Description of Algorithms

2.3.1 Raymond's Algorithm

This algorithm is based on static logical structure based approach. A tree structure is imposed on the sites and the root holds the token. The tree may either be a minimum spanning tree of the actual network topology, or merely a logical structure imposed on
the network. Requests must be sequentially propagated through the paths between the
requesting sites and the site holding the token, and so does the token. The direction of
edges is dynamically updated such that it always leads to the site holding the token.
Every site communicates only with its neighbors and holds information only about its
neighbors. Each site \( x \) maintains a pointer \( \text{holder}_x \), to the neighbor which is the root of
the subtree where the token is located. In addition each site keeps a FIFO queue of
pending requests. The queue records the identifiers of its requesting neighbors or of the
site itself:

![Diagram](attachment:image.png)

Fig 2.5

When a site \( x \) that does not hold the token receives a request for the token or generates a
request locally for itself, it puts the request into the queue. If the queue was previously
empty, it forwards the request to \( \text{holder}_x \). A sequence of request messages is sent
between the requesting site and the site holding the token, along the path constructed by
the holders, until a request message arrives at the site holding the token. Then the token is
sent along the same path in the reverse direction. As the token passes by, the direction of
dges traveled by the token is reversed such that every path always leads to the site
holding the token. When a site receives the token, it removes the entry at the head of the queue. If the entry is for the site itself, it accesses the CS. Otherwise, the token is forwarded to the neighbor which was at the head of the queue. If the queue is still not empty, a request message is sent to the neighbor. One property of the Raymond's algorithm is that the number of messages required for synchronization decreases as the load on the network increases i.e. as frequency of requests to enter CS increases. This is because if a request reaches a site which is busy serving a previous request, it is stored in the queue and no further messages are sent. The chance of this happening increases as the number of waiting sites increases. Fig 2.5 shows a scenario when site A holds the token. Edges from all other sites are directed towards the subtree containing site A. Now if site H wants to access the CS, it sends the request to its neighbor F which stores the request in its queue and sends a request on site H's behalf to site E, similarly site E sends the request to site A. On receiving the request, site A sends the token to site E. On receiving the token, site E removes the pending request from its FIFO queue and sends the token to site F. Site F sends the token to site H, which then accesses the CS. As token moves from site A to H, all the edges on its path are reversed so that now they point to the subtree which contains site H as shown in fig. 2.6.
2.3.2 Naimi et al's Algorithm

This algorithm is based on dynamic logical structure based approach, the network is logically fully connected and a dynamic logical tree is maintained such that the root is always the site which will hold the token in the near future i.e. the root is the last site to get the token among the current requesting sites when no message is in transit. A request is sequentially forwarded in the logical tree, but the token is directly sent to the next requesting site to execute the CS. Each site maintains a pointer "father", which indicates the site to which request for CS access should be forwarded, and the pointer "next" which indicates the site to which the token should be forwarded after a site leaves its CS. If next is nil, token is held with the site itself. When a site wants to enter the CS but does not hold the token, it sends a request message to its father and sets father variable to nil, thereby becoming the new root of the tree. When a site receives a request message, if it has the token, it sends the token to the requester, otherwise it forwards the request to its father. From father to father, a request is transmitted to the root (father_root = Nil). If the root is in requesting state then it sets its "next" variable to requester, so that when the root receives the token, it accesses the CS, and on exit from the CS, sends the token to next. These next variables at each site form a queue, the head is the process processing the token. The tail is the last process which has requested the CS. A path is so organized, that when a process requests entering the CS, the request message is transmitted to the tail. This shown in fig 2.7.
2.3.3 Helary et al’s Algorithm

Helary et al have proposed a general scheme for token and tree based algorithms which covers all algorithms based on the static and dynamic logical structure approaches. Their information structure contains a dynamic rooted tree structure logically connecting the sites involved in the system and a behavior attribute (Transit or Proxy) dynamically assigned to each site. A variable Father\(_i\) indicates, according to site \(i\)'s knowledge the site through which the token can be reached. All Father variables are set in such a way, they define a rooted tree structure over the sites, with the token located at the root. When a site \(i\) wants to access the CS, it sends a request message to Father\(_i\). On receiving a request message from site \(i\), site \(j\) behaves depending upon its behavior attribute. If site \(j\) has transit behavior, and does not hold the token, it forwards the request to Father\(_j\). If site \(j\) owns the token, it sends the token to site \(i\), in both cases site \(j\) sets Father\(_j\) = \(i\).

If a Proxy site \(j\) receives a request from node \(i\), it considers \(i\) its mandator and requests the token (to Father\(_i\)) for itself, thus becoming an asking (busy) site. When site \(j\) receives the token, it sends the token to site \(i\) and considers its mandate for site \(i\) as completed. If a site \(i\) with Proxy behavior is the token owner, it will send a token (\(i\)) message to site \(j\), implying that the token must be returned back to site \(i\). The variable Father\(_i\) is not
changed at a site with Proxy behavior. On receipt of token(j) from site k by site i, the site i behaves as follows:

- If mandator_i = nil, it implies that the token is returned to site i after a loan.
- If mandator_i = i, it implies that the request of site i would be satisfied. If j is nil, then site i retains the token, otherwise after accessing the CS it returns the token to site j.
- If mandator_i != i or nil, it implies that site i had requested the token for some other node x. If currently site i has Proxy behavior and j=nil, it sends the token (i) message to node x, if j!= nil, site i sends token (j) message to site x and updates Father_i = k.
  
  If site i has Transit behavior and j = nil, it sends token(nil) to site x and updates Father_j = x. If j != nil, site i sends token (j) to x and updates Father_i = x.

In Helary et al’s general algorithm, when every site has Transit behavior, the resulting algorithm is a variant of Naimi-Trehel’s algorithm [NTA96]. On the other hand, when the behavior of every site is Transit when it has the token and Proxy otherwise, resulting algorithm is a variant of Raymond’s algorithm. When every site has Proxy behavior, the resulting algorithm is a Centralized algorithm, in which the direction of paths constructed by father variables will not be changed and the token will always be sent back to the root when the site exits the CS. The performance of Helary et al’s algorithm really depends on the given topology and every site’s behavior. In this paper we assume all sites to have Transit behavior.

2.3.4 Chang et al’s Algorithm

The algorithm proposed by Chang, Singhal & Liu makes aggressive use of path compression to achieve good performance [CSL90a]. In the algorithm, the network is
assumed to be fully connected and a logical tree is imposed on it with all directed edges pointing to the root. Initially root is the site that holds the token. Later, root is the last site to get the token among the current requesting sites when no message is in transmission, it holds the token if no site is requesting. Every site i uses a variable holder_i to record the site possibly holding the token. A requesting site i sends a request (i) message to holder_i, and then sets holder_i to itself. When a site j receives a request(i) message, it forwards the message to holder_j, which may be holding the token, if (1) it is not requesting and it does not hold the token. (2) It is requesting and its queue is not empty. Otherwise site j adds the request from site i to the queue. Then site j sets holder_j = i, as site i will hold the token in near future. In finite time the request (i) message is forwarded to the root and then site i becomes the new root. The topology of the network is always maintained as a logical tree by the holder relationship when no message is in transmission. Each site maintains another pointer "next" which records the site id which is next in line for the token. When a site sends a request for itself, it sets its next to nil. If a site that holds or is waiting for the token receives a request, and its next pointer is nil, it sets next to the id of the site that sends the request. Among the sites that hold the token or are waiting, the next variable forms a queue of the blocked sites. If a site is waiting and its next pointer is nil, the site is at the end of the waiting queue. When token holder releases the token, it sends the token to next if next is not nil, otherwise the token holder keeps the token. Fig. 2.8 shows structure of the logical tree as the algorithm runs. The solid arrows represent the holder pointers and the dashed arrows represent the next pointer. The sites that are not requesting the token lie on a path that leads either to the token holder or to a site that is in the waiting list. the next pointer form a list of blocked
sites whose head is the token holder. Moreover, the holder pointers in the list of blocked sites point to another blocked site that is closer to the end of list.

Fig. 28

2.4 Performance Comparison

2.4.1 Time Delay

The average time delay per CS on varying the arrival rate of CS requests, for network size of 16 & 64 sites is shown in fig. 2.9 & fig. 2.10 respectively. All the algorithms exhibit similar delay curve. The delay in granting the CS at high loads is high. As the load decreases, the delay in receiving the token also decreases and finally becomes stable. The delay is high at high loads because number of pending requests increases and therefore “next” chain becomes very long. As the load decreases, the length of “next” chain also decreases and hence delay also decreases. Finally at light loads, when there is atmost one pending request at a given time in the system, the delay is the time required to
search the site having the token and time taken by the token to reach the site. At this point the delay curve stabilizes. We find that at lower connectivities Raymond’s algorithm has a lower delay than all other algorithms. This is because it is based on static logical structure approach and uses the minimum spanning tree of the underlying topology as the logical structure. This way it makes the best use of network topology. The minimum spanning tree gives it the benefit of most optimal logical structure that can be imposed upon a given topology. Dynamic logical structure based algorithms assume a fully connected network, therefore when the actual physical topology is of lower degree of connectivity, they do not exhibit good performance. As the connectivity is increased, there is a significant improvement in the response time of Naimi et al, Chang et al, and Helary et al’s algorithms. At higher degree of connectivity they prove to be better than Raymond’s algorithm. This is because with increased connectivity more number of edges of logical tree would exist in the physical topology and hence information dissemination would be faster. The response time of Naimi et al’s algorithm and Chang et al’s algorithm is exactly similar whereas Helary et al’s algorithm exhibits a lower response time at very high loads. When the network size is increased from 16 sites to 64 sites, the average response time increases because the diameter of the network increases. We see that the performance deterioration of dynamic logical structure based algorithm is higher as the network size increases as compared to static logical structure based algorithms.

2.4.2 Message Traffic

The average number of message hops per CS on varying the arrival rate of CS requests for network sizes of 16 and 64 sites are shown in fig. 2.11 and fig. 2.12 respectively.
Raymond's algorithm has the lowest message traffic. This is because it makes the best use of underlying topology. At high loads the message traffic of Raymond's algorithm is further less because as load increases, it becomes more likely that a request will reach a site that has pending requests and does not send the request message further. As load decreases, probability of finding a requesting site becomes less and therefore message traffic increases. Among dynamic logical structure based algorithms, Naimi et al's algorithm has the highest message traffic, followed by Chang et al's algorithm. Helary et al's algorithm has the lowest message traffic. At high loads, message traffic increases on decreasing the load, this is because at high loads more sites are in waiting chain and therefore number of messages needed to reach the waiting chain are less. Chang et al's algorithm has lower message traffic than Naimi et al's algorithm because in Chang et al's algorithm when a site A sends a token message to site B, it includes in the token message the information of the site that would last hold the token. So now when site B generates a new request, it would send directly to new root instead of following the complete chain. Therefore, the height of the logical tree is reduced. Dynamic logical structure based algorithms have similar message traffic at light loads. This implies that when the system has few pending requests, the searching of site holding the token require same number of message hops in all three algorithms. Dynamic logical structure based algorithms have a higher message traffic because the logical tree does not correspond to the physical network and hence one message in theoretical may mean several message hops in real time. As the connectivity increases, the probability of a logical message hop being the actual physical hop increases and therefore message traffic reduces on increase of connectivity. The message traffic in Raymond's algorithm also decreases as
connectivity increases, because as connectivity increases, the diameter of the network decreases and with it the height of the minimum spanning tree decreases, which results in lower message traffic.

2.5 Conclusion

We conclude that the performance of the logical structure based algorithms depends upon the degree of connectivity of the underlying topology. When degree of connectivity is low, static logical structure based algorithms like Raymond’s algorithm perform better because they are able to make best use of given topology, that too when the logical structure corresponds to the physical topology i.e. here the minimum spanning tree of the physical topology is considered. As the connectivity increases, dynamic logical structure based algorithms like Naimi et al’s algorithm, Chang et al’s algorithm and Helary et al’s algorithm perform better than static logical structure based algorithms. At high loads among the dynamic logical structure based algorithms, Helary et al’s algorithm has the best performance followed by Chang et al and then Naimi et al’s algorithm. Moreover at high loads and even at high connectivities Raymond’s algorithm has the lowest message traffic.
Fig. 2. 9 (a) Avg. time delay on a ring of 16 sites

Fig. 2. 9 (b) Avg. time delay on chordal ring I of 16 sites
Fig. 2.9 (c) Avg. time delay on chordal ring II of 16 sites

Fig. 2.9 (d) Avg. time delay on barrel shifter of 16 sites
Fig. 2. 10 (a) Avg. time delay on ring of 64 sites

Fig. 2. 10 (b) Avg. time delay on chordal ring I of 64 sites
Fig. 2. 10 (c) Avg. time delay on chordal ring II of 64 sites

Fig. 2. 10 (d) Avg. time delay on barrel shifter of 64 sites
Fig. 2. 11 (a) Avg. message traffic on ring of 16 sites

Fig. 2. 11 (b) Avg. message traffic on chordal ring 1 of 16 sites
Fig. 2. 11 (c) Avg. message traffic on chordal ring II of 16 sites

Fig. 2. 11 (d) Avg. message traffic on barrel shifter of 16 sites
Fig. 2. 12 (a) Avg. message traffic on ring of 64 sites

Fig. 2. 12 (b) Avg. message traffic on chordal ring I of 64 sites
Fig. 2. 12 (c) Avg. message traffic on chordal ring II of 64 sites

Fig. 2. 12 (d) Avg. message traffic on barrel shifter of 64 sites