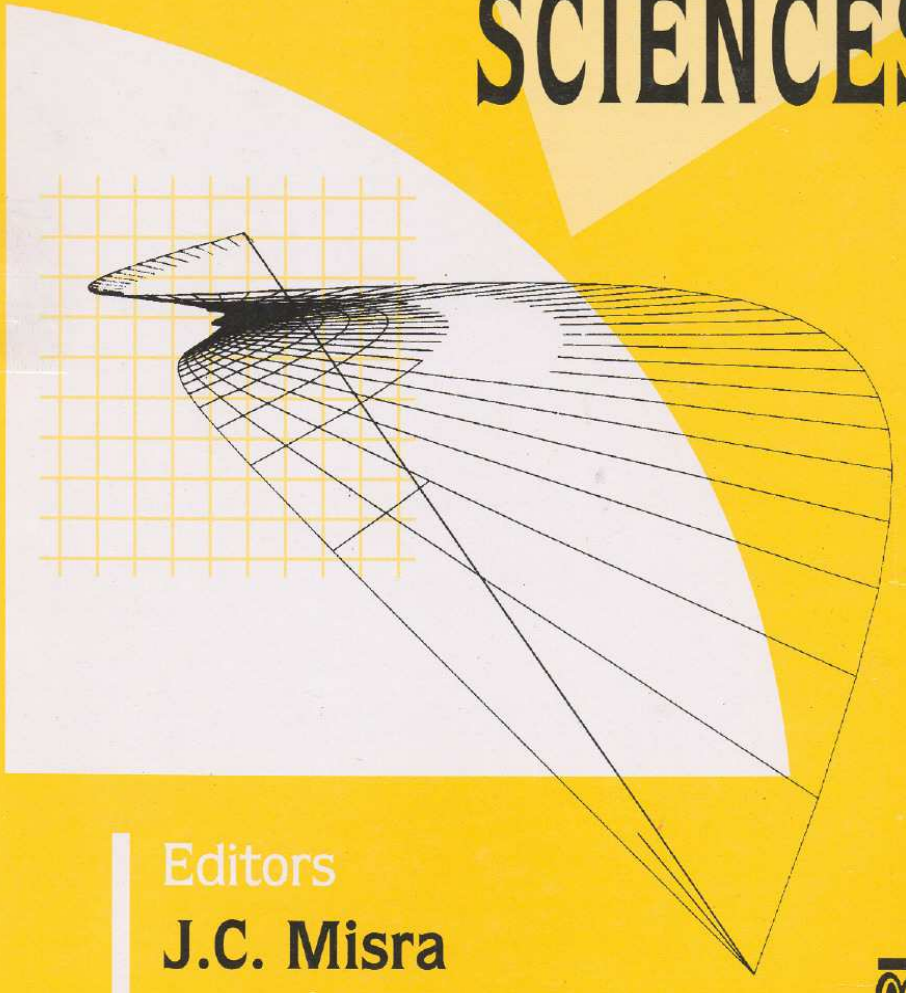
Recent Trends in MATHEMATICAL SCIENCES



Editors
J.C. Misra
S.B. Sinha

Narosa

EDITORS
J.C. Misra
Professor & Head, Department of Mathematics
Indian Institute of Technology, Kharagpur, India
S.B. Sinha
Professor, Department of Mathematics
Indian Institute of Technology, Kharagpur, India

NAROSA PUBLISHING HOUSE

22 Daryaganj, Prakash Deep, Delhi Medical Association Road, New Delhi 110 002
35–36 Greams Road, Thousand Lights, Chennai 600 006
306 Shiv Centre, D.B.C. Sector 17, K.U. Bazar P.O., Navi Mumbai 400 705
2F–2G Shivam Chambers, 53 Syed Amir Ali Avenue, Calcutta 700 019

ISBN 81-7319-405-X

Published by N.K. Mehra for Narosa Publishing House, 22 Daryaganj, Prakash Deep, Delhi Medical Association Road, New Delhi 110 002 and printed at Rajkamal Electric Press, Delhi 110 033 (India).

in: J.C.Misra and S.B.Sinha (Eds.), Recent Trends in Mathematical Sciences, Narosa Publishing, New Delhi, Dec. 2000, 53-63.

7. Shared Resource Allocation Using Token Based Control Strategy in Ring Extension Topologies[1]

R.Rajendra Prasath[2] and P. Thangavel[3]

Department of Computer Science,
University of Madras, Chennai - 600 005, India

Abstract

In this paper, we consider the problem of controlling the allocation of a shared resource among the processors in ring extension topologies. We propose a request message based token control strategy for the allocation of a shared resource among the processors in a logically structured ring extension topologies like touching rings and interconnected rings. This strategy assures that the request messages would be served with in a maximum delay. It is quaranteed that the protocols allow only a bounded number of messages per request and this complexity is unbounded for other classical solutions.

Keywords: Distributed computing, interconnection networks.

1. INTRODUCTION

Consider the problem of controlling the allocation of a shared resource among a set of n processors that are arranged in a logically structured ring extension topologies like touching rings and interconnected rings. We propose a solution by means of a control token. A processor which needs the shared resource must first get a free token and then change it into a busy token. After the utilization of the shared resource, the busy token is destroyed and a new free token is again circulated in the network for shared resource allocation.

This type of circular token based communication model is used in various combinatorial problems such as election problem, mutual exclusion problem[1-6]. All these problems use the same procedure: an entity holding the token will pass it along the circular communication model as soon as it no longer needs it [4]. But the unnecessary movements of the token amount to an unbounded number of message exchanges even for a finite set of requests. To avoid the unnecessary movements, we adopt the request message based token control strategy introduced by Feuerstein *et al*[7] inwhich token is circulated only if it is informed of a request message generated for the shared resource.

[1]This work is partially supported by Council of Scientific and Industrial Research (CSIR), Government of India.

[2]Email:rrajprasath@yahoo.com;

[3]Email:thangavelp@yahoo.com

Although a vast amount of literature[2,8] exists on the token based control strategy, but little is known about the use of circular token based control mechanism. In the context of distributed computing, the research has been mainly focused on the detection of token loss and on self stabilizing aspects[3,9]. It is assumed that the processors do not include any information to the request messages originated for the shared resource. Suppose if we include any information along the request messages then one can easily find out the shortest path[10,11] onwhich routing from source to destination is made and can be repeated for every processor by adding a queue of pending request messages.

Feuerstein *et al*[7] have proposed two protocols Q and D based on the request message based token control strategy for controlling the allocation of a shared resource in unidirectional fault-free ring network in which both the token as well as the request messages move in a single direction. The algorithm Q requires n messages per request and service traffic $\frac{3n^2}{2}$, where as algorithm D requires $2n$ messages per request and service traffic $3n-3$.

In this paper, we propose two protocols for shared resource allocation problem in touching rings and two interconnected ring networks. In the next section, we present preliminaries. Section 3 presents a protocol for touching ring networks. In section 4, we present an algorithm for two interconnected rings. Section 5 concludes the paper.

2. PRELIMINARIES

Consider a touching ring model in which n processors are logically structured in two subrings that touch at a common processor through which routing of the requests as well as token from one subring to another subring takes place either in an unidirectional or in a bidirectional way and an interconnected ring network model in which n processors, assumed to be even, are arranged in two subrings with $\frac{n}{2}$ processors each. In the touching ring network, processor next to the central touching processor is indexed as 1 and running over the remaining processors in clockwise direction; crossing the central touching processor; then again running over the processors in the next subring in clockwise direction and finally ending up the indexing with the processor before the central processor. The role of the central processor is significant as it has a special mechanism to switch the token as well as request messages into the subrings appropriately according to the receipt of the request messages as well as token from the subrings.

In an interconnected ring network, processors are interconnected in such a way that processor P_i in the first subring is connected with processor $P_{i+\frac{n}{2}}$ by a bidirectional link in which both the token as well as the request messages can flow in either directions. We assume unidirectional movements for requests and token, over the ring edges and bidirectional over the edges connecting the two subrings. In an interconnected ring network, each processor in the second subring has a special mechanism to sensor the direction inwhich token has been received so as to send the

token from the processor in the second subring to the processor in the first subring.

The analysis of the proposed protocols uses two distinct measures.

The first measure is the average number of messages per request necessary to satisfy a sequence of requests, i.e., the worst case ratio between the total number of (token and request) messages and the number of requests in the sequence.

The second measure is the service traffic, defined as the worst case number of messages that are exchanged in the network between the time in which a processor sends a request message and the time in which it gets serviced. To obtain a feasible solution, the service traffic should be finite.

3. TOUCHING RING NETWORK

We consider unidirectional Touching Ring Network with the central touching processor as $P_{\lceil \frac{n}{2} \rceil}$. Incoming links to the central processor are from $P_{\lceil \frac{n}{2} \rceil - 1}$ and P_n and outgoing links are to P_1 and $P_{\lceil \frac{n}{2} \rceil + 1}$. Each processor P_i $[i = 1, 2, 3, ..., n]$ has a local variable M_p, used to stop further movements of the following request messages. The value of M_p is set to 1, if P_i has a pending request message of its own or a request message has been passed through it and 0, otherwise. Now assume that the central processor $P_{\lceil \frac{n}{2} \rceil}$ has a token locator T_l to trace the token location in the subrings and a request recorder V_r that helps to record the pending requests of one subring when token has passed through the other subring. The value for T_l is assigned to 0, if token has passed to the processor in the first subring from the central processor and 1, otherwise.

V_r is assigned to 1, if a request is received from P_n to $P_{\lceil \frac{n}{2} \rceil}$ [similarly from $P_{\lceil \frac{n}{2} \rceil - 1}$ to $P_{\lceil \frac{n}{2} \rceil}$] when $T_l = 0(1)$ and 0, otherwise. Initially the values of M_p and V_r are assumed to be zero and token is placed at the central processor. It can be observed that whenever $T_l = 0(1)$ then V_r indicates the receipt of a request message received from second (respectively first) subring. Token maintains a two cell flag bit structure and sets as and when it recognizes either V_r=1 at $P_{\lceil \frac{n}{2} \rceil}$ or the receipt of a request message. The cells are indexed as T_1 and T_2 which takes 1, if a request is received from first and second subrings respectively and 0, otherwise. The value of the two cell flag bit helps token to move through the processors in the next subring whose M_p might be zero. At the central processor, token moves according to the value of M_p and V_r. Token stops only if the values of M_p, T_1 and T_2 are zero.

ALGORITHM : UTRN

(a) When P_i needs resource then
  If P_i has token then it enters the critical section.
  If P_i has no token then
    if P_i is the central processor then
      if T_l =0 (1) then set $M_p = 1$ and send the request message to P_1 ($P_{\lceil \frac{n}{2} \rceil + 1}$)
    else (if P_i is not the central processor then)
      if $M_p = 0$ then
        set $M_p = 1$ and send the request message to the next processor

else no request message is sent.

(b1) When P_i receives a request message
If P_i has token then
if P_i is the central processor then
if the request message is received from $P_{\lceil \frac{n}{2} \rceil - 1}$ (P_n) then
set $T_l = 0(1)$; set $T_1(T_2) = 1$; set $M_p = 0$ and
send token to the processor $P_1(P_{\lceil \frac{n}{2} \rceil + 1})$ in the current subring
else(if P_i is not the central processor then)
if $1 \leq i < \lceil \frac{n}{2} \rceil$ then set $T_1 = 1$;
else (if $\lceil \frac{n}{2} \rceil < i \leq n$ then) set $T_2 = 1$;
send token to the processor P_{i+1} in clockwise direction.

(b2) When P_i receives a request message and has no token then
If P_i is the central processor then
if the request message is received from $P_{\lceil \frac{n}{2} \rceil - 1}$ (P_n) then
if $T_l = 0(1)$ then set $M_p = 1$;
if $V_r = 1$ then stop the request at P_i;
else send the request to the next processor P_1 $(P_{\lceil \frac{n}{2} \rceil + 1})$
in the current subring.
else(if $T_l = 1(0)$ then) set $V_r = 1$.
if $M_p = 1$ then stop the request message at P_i;
else (if $M_p = 0$ then)
send the request to the next processor $P_{\lceil \frac{n}{2} \rceil + 1}$ (P_1) in the next subring;
else (if P_i is not the central processor then)
if $M_p = 0$ then set $M_p = 1$ and send the request to P_{i+1};
else stop the request at P_i.

(c1) When P_i receives token then
If P_i has a pending request message then
if P_i is the central processor then
if $V_r = 0$ then enter the critical section; At the end of the critical
section, set $M_p = 0$; set $T_{1(2)} = 0$; set $T_l = 0(1)$;
and send token to P_1 $(P_{\lceil \frac{n}{2} \rceil + 1})$ in the current subring
else(if $V_r = 1$ then)
if token is received from $P_{\lceil \frac{n}{2} \rceil - 1}$ (P_n) then
enter the critical section; At the end of the critical section,
reset $M_P = 0$; set $T_{2(1)} = 1$; set $T_l = 1(0)$; set $V_r = 0$ and
send token to $P_{\lceil \frac{n}{2} \rceil + 1}$ (P_1) in the next subring
else (if P_i is not the central processor then)
enter the critical section; reset $M_P = 0$;
if $1 \leq i < \lceil \frac{n}{2} \rceil$ then set $T_1 = 0$; otherwise set $T_2 = 0$;
At the end of the critical section, send token to the next
processor P_{i+1} in the current subring.

(c2) When P_i receives token and has no pending request then
if P_i is the central processor then
if $V_r = 1$ then
if token is received from $P_{\lceil \frac{n}{2} \rceil - 1}$ (P_n) then
set $T_l = 1(0)$; reset $V_r = 0$;
if $M_p = 1$ then set T_1=$T_2 = 1$; otherwise set $T_2(T_1) = 1$;
and send token to $P_{\lceil \frac{n}{2} \rceil + 1}$ (P_1) the next subring
else (if $V_r = 0$ then)
If $M_p = 1$ then set $M_p = 0$; and send token
to the next processor in the current subring
else (if $M_p = 0$ then)
if T_1 AND $T_2 = 0$ then stop token at P_i
else send token to next processor in the current subring
else (if P_i is not the central processor then)
if M_p OR T_1 OR T_2= 1 then reset $M_p = 0$ and pass token to P_{i+1}
else if T_1 AND $T_2 = 0$ then stop the token at P_i itself.

Theorem : 1

The algorithm UTRN serves all the request messages.

Proof:

We assume that token is at the central processor. Each request message will eventually reach the token and informs it to access the shared resource. On receipt of a request message, token moves through the appropriate subring by setting T_l value at the central processor accordingly. Meantime V_r and M_p are maintained at the central processor to record the request messages received from another subring and the current subring respectively.

Now let token be at any one of the processors in the subrings. Then the central processor when it receives a request message checks the values of T_l, V_r and M_p and sets them as in steps (b1) and (b2) and then the request message is sent to the corresponding subring based on the availability of the token. Whenever token reaches the central processor, it checks the values of V_r and M_p. If there is a pending request message of the next subring then token is forwarded to the next subring; otherwise token is forwarded to the next processor in the current subring. As and when token switches over subrings, V_r and M_p are dynamically maintained. Token moves through the processors until T_1 AND $T_2 = 0$ and $M_p = 0$. Thus request messages are not lost and token is aware of all the request messages. Thus the algorithm UTRN serves all the request messages. ∎

Theorem : 2

The algorithm UTRN requires atmost (n-1)messages per request and service traffic is bounded by $\frac{5n-2}{2}$.

Proof:

Assume that token is at the central processor. Now suppose that a request message has been originated at some processor in a subring. Infact, regardless of n, the total number of processors, atmost $\frac{n}{2}-1$ movements for each subring are required in the worst case to inform the token. If token has passed through the first subring then the request coming from second subring moves to the first subring if $M_p = 0$ at the central processor, else the request message stops. Meantime, atmost (n-1) processors may generate request messages. Thus in addition, token may have to move over atmost n processors. Thus the number of messages per request amounts ($2n$-1) in the worst case.

To prove the second measure, two types of request messages may be sent between the time in which P_i generates a request message and the time in which it has been served. The first type includes the request messages that require atmost (n-1) movements to inform the token. In response to this request, token needs to move atmost $\frac{n}{2}$ moves for switching it into the next subring; $\frac{n}{2}$ moves for the next subring and $\frac{n}{2}$ movements of either the current or the next subring. Thus in total $\frac{3n}{2}$ messages are needed for the token to reach the processor with a pending request. The last $\frac{n}{2}$ movements quarantee that token is aware of all the request messages of one subring when token is passing through another subring. Thus the service traffic in the worst case amounts to $\frac{5n-2}{2}$. ∎

Instead of assuming unidirectional movement over the touching ring network, we can assume opposite directional movement for both the token as well as the request messages. Such a touching ring network is termed as a *Bidirectional touching ring network* inwhich token flows in clockwise and request messages flow in anticlockwise direction. The algorithm UTRN still holds for a bidirectional touching ring network, provided if we keep track of the change in the direction of request messages that are allowed to move only in anticlockwise direction. The resulting algorithm with bidirectional movements, we call it as BTRN, ensures the fastest servicing of the request messages in the best, average case.

An immediate result follows in the sequel.

Theorem : 3

Algorithm BTRN for a bidirectional touching ring network serves all the request messages, requires atmost (n-1) messages per request and service traffic is bounded by ($2n$-1).

Proof

As the request messages move in anticlockwise direction and token in clockwise direction, no request message would be skipped by the token and hence all request messages are served. Now to prove the first measure, we consider the variables that are dynamically maintained according to the opposite movements of the token as well as the request messages. So (n-1) movements are necessary to satisfy a sequence of requests and hence average number of messages per request amounts to (n-1). Now

we prove the second measure. As the token moves in clockwise direction and request messages in anticlockwise direction, it can easily sensor and serve all the pending requests in its way to the destination. Token stops moving only when it recognizes a processor with no pending request message. Hence the service traffic amounts to (2n-1). ∎

The algorithm UTRN can also be generalized for n-touching rings network$(n, m[k])$ where n is the number of processors in the central ring and $m[k]$ is the number of subrings with k processors each and $k \geq 2$ [excluding the touching processor]. The n-touching rings network is formed in such a way that each subring with k processors is connected with one processor in the central ring network. Thus there are totally $(n + mk)$ processors.

We assume a fault-free n-touching ring network with $(n + mk)$ prcoessors. Now we can extend the proposed algorithm UTRN to the n-touching rings network by considering each subring with the remaining part of the network and for each processor in the central ring network. By this way, each processor in the central ring assumes all the variables as in UTRN and operates in parallel. Still we follow the basic assumptions that they have neither a shared memory nor a common clock and their speeds are not related. Thus the generalized algorithm amounts $(n + mk)$ messages per request and $(3n + 2mk - 1)$ service traffic in the worst case. It can be quaranteed that this generalized algorithm works perfectly for variable k also. But as n and k grow, the transmission delay for servicing the pending requests would be unavoided.

4. INTERCONNECTED RINGS

In this section, we describe request message based token control mechanism for shared resource allocation in an interconnected rings network. In this architecture, there are two subrings with $\frac{n}{2}$ processors each, where n is the total number of processors(assumed to be even). The interconnections between the subrings are made in such a way that processor P_i (i=1,2,3,...,$\frac{n}{2}$) in the first subring is connected to the processor $P_{i+\frac{n}{2}}$ in the second subring by a bidirectional channel in which the token as well as the request messages can pass through in either directions. The following assumptions would be useful in the sequel:

Each processor has a local variable M_p that takes 1, if it has a pending request(may be of its own or of the other processor's) and 0, otherwise. Each processor in the first subring has a flag variable V_r whose value is 1, if a pending request has been received from $P_{i+\frac{n}{2}}$ by P_i and 0, if there is a non-receipt of the request messages. The value of V_r will be checked by the token to identify the request messages received from the second subring. Each processor P_i resets V_r value from 1 to 0 only on receipt of a request message and not by the movement of the token through it. Token has a counter to record the receipt of the request messages and token stops moving only when the values of the token counter and M_p are zero.

ALGORITHM : ICRN

(a) When P_i needs resource then
If P_i has token then enter the critical section.
If P_i has no token then
If $1 \leq i \leq \frac{n}{2}$ then
if $M_p = 0$ then set $M_p = 1$;
if $V_p = 1$ then stop the request at P_i
else (if $V_p = 0$ then) send the request to P_{i+1}.
else(if $M_p = 1$ then) stop the request at P_i.
else(if $\frac{n}{2} + 1 \leq i \leq n$ then)
send the request from P_i to $P_{i-\frac{n}{2}}$.

(b1) When P_i receives a request message then
If P_i has token then
increase the token counter by one and
if $V_p = 1$ then
set $V_p = 0$ and send token to the processor $P_{i+\frac{n}{2}}$
else set $M_p = 0$ and send token to the next processor P_{i+1}.

(b2) When P_i receives a request message then
If P_i has no token then
if the request is received from $P_{i+\frac{n}{2}}$ then set $V_p = 1$
if $M_p = 1$ then stop the request at P_i
else send the request to P_{i+1}.
else (if the request is received from P_{i-1} then)
if $M_p = 1$ then stop the request at P_i
else set $M_p = 1$
if $V_p = 1$ then stop the request at P_i
else send the request to P_{i+1}.

(c1) When P_i receives token then
If P_i has a pending request message then
if $1 \leq i \leq \frac{n}{2}$ then
enter the critical section; decrease the token counter by 1; reset $M_p = 0$;
if token is received from P_{i-1} then
if $V_p = 1$ then
increase the counter by 1; set $V_p = 0$; exit the critical section
and send token to $P_{i+\frac{n}{2}}$; otherwise else send token to P_{i+1}.
else (if token is received from $P_{i+\frac{n}{2}}$ then)
reset $V_p = 0$; if token counter is bigger than zero then,
send token to P_{i+1}; otherwise stop token at P_i.
else (if $\frac{n}{2} + 1 \leq i \leq n$ then)
enter the critical section; decrease the token counter by one.
if token is received from P_{i-1} then send token to $P_{i-\frac{n}{2}}$
else send token to P_{i+1}.

(c2) When P_i receives token then
If P_i has no pending request message then
if $1 \leq i \leq \frac{n}{2}$ then
if token is received from P_{i-1} then
if $M_p = 1$ then set $M_p = 0$
if $V_p = 1$ then
increase the token counter by one; set $V_p = 0$
and send token to $P_{i+\frac{n}{2}}$.
else send token to P_{i+1}.
else (if $M_p = 0$ then)
if $V_p = 1$ then increase the token counter by one; set $V_p = 0$
and send token to $P_{i+\frac{n}{2}}$.
else if token counter is greater than zero then send token to P_{i+1};
otherwise stop the token at P_i.
else (if token is received from $P_{i+\frac{n}{2}}$ then)
if $M_p = 1$ then
reset $M_p = 0$; $V_p = 0$; if token counter is greater than zero then,
send token to P_{i+1}; otherwise stop the token at P_i.
else (if $M_p = 0$ then)
send token to the next processor if token counter is bigger than zero.
else (if $\frac{n}{2} + 1 \leq i \leq n$ then)
send token to $P_{i-\frac{n}{2}}$.

Theorem : 4

The algorithm ICRN serves all the request messages.

Proof:

Assume an interconnected rings network with two subrings. If there is no request message then token resides at any one of the first subring processors. So each originated request message from the processor in the first subring will eventually find out the token and push it till the next processor with a pending request. Infact each request message generated at the inner ring processors needs atmost $n-1$ moves to reach the destination in the worst case. Meanwhile each processor in the second subring requires just one move from processor P_i to $P_{i-\frac{n}{2}}$ and then moves over the processors in the first subring. In the worst case, as all the processors may originate request messages for the token, the zig-zag movements of the token between the subrings help to serve all the request messages. Thus the algorithm ICRN is aware of all the requests and hence serves all the request messages. ∎

Theorem : 5

The algorithm ICRN requires $(n-1)$ messages per request and service traffic is bounded by $(3n-2)$.

Proof:

To prove the first measure, let the token be initially at any one of the processors P_i in the first subring. Now each request at P_{i+1} has to traverse atmost $\frac{n}{2} - 1$ movements before it reaches the token or it has been stopped by the preceding request message. Meanwhile each processor in the second subring may originate and send requests for token. Thus in the worst case, request messages generated at the second subring require one move to reach their corresponding processors in the first subring and then moves $\frac{n}{2} - 1$ processors in the first subring. This overall results in $\frac{n}{2} + (\frac{n}{2} - 1)=(n - 1)$ moves required to inform the token. Hence in the worst case the number of messages per request amounts to $(n - 1)$.

In order to prove the second measure, service traffic, consider two types of movements (request as well as token) between the two subrings. The first type includes requests from the first subring and in the absence of the second subring, atmost $(\frac{n}{2} - 1)$ messages may be sent to inform the token. But by the interconnection of the second subring, each request originated from second subring requires one move ahead per processor to reach their corresponding interconnected processor in the first subring resulting $\frac{n}{2}$ moves in total and then moves over the processors in the first subring provided if $M_p = 0$. Thus totally $(n-1)$ messages are needed to inform the token in the worst case. Token starts from the processor in the first subring according to the value of V_p. The zig-zag movements of the token guarantee that it serves all the requests by visiting each processor atleast once. Thus n messages are required to visit each processor atleast once. So atmost $(2n - 1)$ messages are needed to service the processors with a pending request, regardless of the position of the processors in the subrings. Additionally $(n - 1)$ movements are needed, as each processor is again eligible to generate a request message after its previous request message has been served.

Hence the service traffic in the worst case amounts to $(3n-2)$ and it is guaranteed that token skips no request messages in its way to the destination. ∎

5. CONCLUSION

In this paper, we have considered the problem of controlling the allocation of a shared resource in ring extension topologies link touching rings and interconnected rings using request message based token control strategy. We have first presented an algorithm UTRN for the allocation of a shared resource in an unidirectional ring network. This could be extended to a bidirectional touching ring network. Finally we have presented an algorithm ICRN for unidirectional interconnected rings network with two interconnected subrings. Also this algorithm could be extended to the possible architectures of more interconnected subrings. But the communication delay would be higher as the total number of the interconnected subrings increases. It would be quiet interesting to look for similar problems in other ring extension topologies.

REFERENCES

[1] D.Andrews and G.Schulz, A token-ring architecture for local area network: an update, in: *Proc. IEEE COMPCON*, Fall 1982.

[2] D.Bertsekas and R.Gallager, *Data Networks*, 2^{nd} Ed, Prentice Hall, 1992.

[3] G.Le.Lann, Distributed Systems - Towards a formal approach in B.Gilchrist(ed.): *Inform. Proc. Lett*: **(77)** 155-160, 1977.

[4] M.Raynal, *Distributed Algorithms and Protocols*: John Wiley and Sons, 1988.

[5] F.E.Ross, FDDI- A tutorial, *IEEE Communication magazine*, **(24)** 10-17, 1986.

[6 W.Stallings, *Data and Computer Networks*, 5^{th} Ed., Prentice Hall, 1997.

[7] E.Feuerstein, S.Leonardi, A.Marchetti-Spaccamela and N.Santoro, Efficient Token Based Control in rings, *Inform. Proc. Lett*: **(66)** 175-180, 1998.

[8] V.Rego and L.M.Ni, Analytic models of cyclic service systems and their applications to token-passing local networks, *IEEE Trans. on Computers*, **C-37** (10) 1224-1234, 1988.

[9] J.E.Burns and J.Pachl, Uniform self-stabilizing rings, *ACM Trans. on Prog. Languages and Systems*, **(11)**(2) 330-334, 1989.

[10] N.Chalamaiah and B.Ramamurthy, Finding shortest paths in distributed loop networks, *Inform. Proc. Lett*: **(67)** 157-161, 1998.

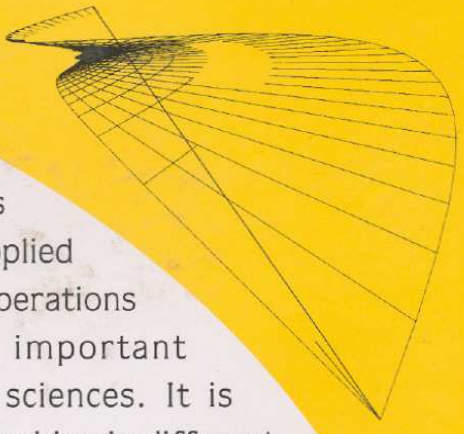
[11] K.Mukhopadhyaya and B.P.Sinha, Fault-Tolerant Routing in Distributed loop networks, *IEEE Trans. on Computers*: **(44)**(12) 1452-1456, 1995.

Recent Trends in MATHEMATICAL SCIENCES

Editors

J.C. Misra
Professor & Head, Department of Mathematics
Indian Institute of Technology, Kharagpur, INDIA

S.B. Sinha
Professor, Department of Mathematics
Indian Institute of Technology, Kharagpur, INDIA



This book deals with some important problems covering various important areas of pure and applied mathematics, statistics and operations research as well as some important applications to engineering sciences. It is intended for the researchers working in different areas of mathematical sciences and addresses a wide spectrum of problems related to theory as well as applications. This book will be highly useful for postgraduate science/engineering students interested to take up research career.

ISBN 81-7319-405-X

9 788173 194054

Narosa
Publishing House