Journal of Madras University - Section B: Sciences

World Mathematical Year 2000 - Special Issue

Volume 52 Year 2000

CONTENTS

Journal of the Madras University(WMY-2000 Special Issue)
-Section B : Sciences. Vol. **52** (2000) pp. 145-159

TOKEN BASED MESSAGE PASSING IN BIDIRECTIONAL RING EXTENSIONS

R. Rajendra Prasath and **P. Thangavel**

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

E-mail: rrajprasath@yahoo.com and
thangavelp@yahoo.com

Abstract

We propose a request message based token passing mechanism for the shared resource allocation in bidirectional touching rings and intersecting rings. In all the above interconnection networks, we assume that request messages as well as token move in opposite directions. In this message passing strategy, token moves only if it is informed of a request message generated for the single shared resource. After initiated by a request message, token searches the processor that generated the request message and then serves. This control strategy guarantees that the proposed protocols allow only a finite number of message exchanges per request and this complexity would be unfair for other classical solutions. Also this strategy ensures that the request messages would be served with in a finite delay.

AMS (2000) Mathematics Subject Classification : 68M10, 68W15.

Keywords : Distributed computing, message passing, shared resource allocation, 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 logically structured ring

extension topologies like bidirectional touching rings and intersecting rings. We propose a solution by means of a small frame called *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[1-7,10,11]. 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 [10]. 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* [5] in which token is circulated only if it is informed of a request message generated for the shared resource.

Although a vast amount of literature[2,5,6,9,11-13] 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[3,10]. It is assumed that the processors do not include any information to the request messages generated for the shared resource. Suppose if we include any information along the request messages then one can easily find out the shortest path[4,8] on which routing from source to destination is made and can be repeated for every processor.

Feuerstein *et al* [5] 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 $3n-3$. Then this request message based token control strategy for shared resource allocation problem has been extended to bidirectional rings, touching rings and two interconnected rings network [9]. In this paper, we propose two efficient algorithms for shared resource allocation in a bidirectional touching rings and intersecting tings networks. In the following section, we present preliminaries. Section 3 describes a protocol for bidirectional touching rings network.

In section 4, we present an algorithm for bidirectional intersecting rings network. Finally Section 5 concludes the paper.

2. Preliminaries

Consider a bidirectional touching rings 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 in opposite directions. In the bidirectional touching rings network, processor next to the common touching processor is indexed as 1 and running over the remaining processors in clockwise direction; crossing the common 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 common processor. The role of common processors is significant as it is assumed to have its own switching subsystem with a special mechanism to switch the token as well as request messages into the subarcs appropriately according to the receipt of the request messages as well as token from the subarcs.

Next we consider an intersecting rings network in which $N[=(n+m)]$ processors are logically arranged in two rings each with n and m processors that intersect at two common processors through which routing of the request messages as well as token from one subarc of one ring to another subarc of another ring takes place. In this network, the processor next to the common processor is indexed as P_1^2[or P_1^1] , then indexing runs over the remaining processors in the clockwise direction, crossing the common processor; running over the processors in the next subarc of the same ring in clockwise direction and finally ends up with the processor before the next common processor. Similarly the second ring is numbered starting from the processor next to the common processor which is part of the first ring. In this scheme, one common processor is counted in the first ring and the other in the second ring.

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. Bidirectional Touching Rings Network

We consider Bidirectional Touching Rings Network with the common touching processor as $P_{\lceil \frac{n}{2} \rceil}$. 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 common 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. We assume that token moves in anticlockwise direction and requests in clockwise direction. The value for T_l is assigned to 0, if token has passed in anticlockwise direction to the processor in the first subring from the common processor and 1, otherwise.

V_r is assigned to 1, if a request is received from P_n by $P_{\lceil \frac{n}{2} \rceil}$ [similarly from $P_{\lceil \frac{n}{2} \rceil - 1}$ by $P_{\lceil \frac{n}{2} \rceil}$] 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 common 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 common 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 : BTRN

(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 common processor then

if M_p =1 then stop the request message at P_i

else (if M_p =0 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 common processor then)

if M_p = 0 then set M_p = 1 and

send the request message to the next processor in clockwise direction.

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 common 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 $P_{\lceil \frac{n}{2} \rceil - 1}$ (P_n)

in the current subring

else (if P_i is not the common 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 anticlockwise direction.

(b2) When P_i receives a request message and has no token then

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

of the current subring

else (if P_i is not the common 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* BTRN *serves all the request messages.*

Proof. We assume that token is at the central processor. Each request message traversing in clockwise direction will eventually reach the token and informs it to access the shared resource. On receipt of a request message, token moves in anticlockwise direction 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. Thus token sensors all the request messages and no request message is skipped.

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 first and then checks 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 $M_p = 0$. As the request messages move in

clockwise direction and token in anticlockwise direction, token is aware of all the request messages. Thus the algorithm BTRN serves all the request messages. ■

Theorem 2. *The algorithm* BTRN *for a bidirectional touching rings network in the worst case requires* $(N-1)$ *messages per request and service traffic is bounded by (2 N -1).*

Proof. 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. Thus token can easily sensor and serve all the pending requests in its way to the destination. So, in the worst case, (N -1) movements are necessary to satisfy a sequence of requests and hence the number of messages per request amounts to (N -1).

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 at most (N -1) movements to inform the token. In response to this request, token needs to move at most $\frac{N}{2}$ moves for switching it into the next subring and $\frac{N}{2}$ moves for the next subring. The second $\frac{N}{2}$ movements guarantee that token is aware of all the request messages of one subring when token is passing through another subring. Thus in total N messages are needed for the token to reach the processor with a pending request. Hence the service traffic in the worst case amounts (2 N -1). ■

4. Bidirectional Intersecting Rings Network

We consider a fault-free Bidirectional intersecting rings network with two common processors say P_f^1 and P_s^2. The first ring consists of an arc l_1 with processors P_1^1, P_2^1, P_3^1 ,..., P_{f-1}^1; the common processor P_f^1 and an arc l_2 with processors $P_{f+1}^1, P_{f+2}^1, ..., P_n^1$ ($(f-1) > (n-f)$). Similarly the second ring consists of an arc r_1 with processors $P_1^2, P_2^2, P_3^2, ..., P_{s-1}^2$; the common processor P_s^2 and an arc r_2 with processors $P_{s+1}^2, P_{s+2}^2, ..., P_m^2$ ($(s-1) > (m-s)$). The superscripts 1 and 2 denotes the first and the second rings respectively. We assume that the request messages move in clockwise direction and token moves in anticlockwise direction. Each processor has a local variable M_p which assumes 1, if it has a pending request message of its own or already a

request message has been passed through it and 0, otherwise.

Each common processor has a token locator $L_{f[\text{or } s]}$ (L_f for P_f^1 and L_s for P_s^2) which is used to trace the arc through which the token has been sent and request indicators $V1_f$ and $V2_f$ [or $V1_s$ and $V2_s$] which are used to record the request messages received from P_{f-1}^1 and P_m^2 [or P_{s-1}^2 and P_n^1]. The token indicator $L_{f[\text{or } s]}$ is set to 1, if token has been passed through P_{f-1}^1 [or P_{s-1}^2] and 0, if token has been passed through P_m^2 [or P_n^1] or otherwise. $V1_f$ ($V2_f$) assumes 1, if a request is received from P_{f-1}^1 (P_m^2) and 0, otherwise. Similarly $V1_s$ ($V2_s$) assumes 1, if a request is received from P_{s-1}^2 (P_n^2) and 0, otherwise. Token maintains a four cell flag bit structure, say T_{l_1} , T_{l_2} , T_{r_1} and T_{r_2} . When token arrives at one of the common processors, the value of $T_{l_1(r_2)}$ [or $T_{r_1(l_2)}$] is set to 1, if there is a pending request received from P_{f-1}^1 (P_m^2)[or P_{s-1}^2 (P_n^1)] and 0, otherwise. The values of the these flag bit cells help token to move through the processors in the subarcs. Initially the values of M_p , $L_{f[\text{or } s]}$, $V1_{f[\text{or } s]}(V2_{f[\text{or } s]})$ and $T_{l_1(r_2)}[\text{or } T_{r_1(l_2)}]$ are assumed to be zero. Token stops moving only if the values of M_p and $T_{l_1} OR\ T_{l_2} OR\ T_{r_1} OR\ T_{r_2}$ are zero. The description of the algorithm : InteRN - B is as follows:

ALGORITHM : InteRN - B

(a) When $P_i^1[\text{or } P_i^2]$ needs resource then

If $P_i^1[\text{or } P_i^2]$ has token then enter the critical section.

If $P_i^1[\text{or } P_i^2]$ has no token then

If $i = f[\text{or } s]$ then set $M_p = 1$ and send the request message to P_{f+1}^1 [or P_{s+1}^2] if $L_{f[\text{or } s]} = 1(0)$

else if $M_p = 0$ then set $M_p = 1$ and send the request to $P_{i+1}^1[\text{or } P_{i+1}^2]$

else (if $M_p = 1$ then) no request message is sent.

(b1) When $P_i^1[\text{or } P_i^2]$ receives a request message then

If $P_i^1[\text{or } P_i^2]$ has token then

If $i = f[\text{or } s]$ then

if the request is received from P_{f-1}^1 (P_m^2) [or P_{s-1}^2 (P_n^1)] then

set $L_{f[\text{or } s]} = 1(0)$; $M_p = 0$; $T_{l_1(r_2)}[\text{or} T_{r_1(l_2)}] = 1$

and send token to $P^1_{f-1}(P^2_m)$[or $P^2_{s-1}(P^1_n)$]

else (if i is neither f nor s then)

if P^1_i[or P^2_i] $\in l_1\,(\,r_2\,)$[or $r_1\,(\,l_2\,)$] then

set $T_{l_1(r_2)}$ [or $T_{r_1(l_2)}$] $= 1$ and

send token to $P^1_{i-1}(P^2_{i-1})$[or $P^2_{i-1}(P^1_{i-1})$]

(b2) When P^1_i[or P^2_i] receives a request message then

If P^1_i[or P^2_i] has no token then

if $i = f$[or s] then

if the request is received from P^1_{f-1} (P^2_m) [or $P^2_{s-1}(P^1_n)$] then

set $V1_f(V2_f)$[or $V1_s(V2_s)$] $= 1$;

if $V2_f(V1_f)$[or $V2_s(V1_s)$] $= 1$ then

stop the request message at P^1_i[or P^2_i]

else (if $V2_f(V1_f)$[or $V2_s(V1_s)$] $= 0$ then)

send the request message to P^1_{f+1}[or P^2_{s+1}]

else if $M_p = 0$ then

set $M_p = 1$ and send the request

to P^1_{i+1}[or P^2_{i+1}]

else stop the request message at P^1_i[or P^2_i] .

(c1) When P^1_i[or P^2_i] receives token and has a pending request then

enter the critical section; reset $M_p = 0$;

if $i = f$[or s] then

if token has been received from P^2_1 [or P^1_1] then

set $V1_s$[or $V1_f$] $= 0$;

if $V2_s$[or $V2_f$] $= 1$ then set T_{l_2}[or T_{r_2}] $= 1$; $V2_s$[or $V2_f$] $= 0$;

else (if token has been received from P^1_{f+1} (P^2_{s+1}) then)

reset $V2_f$[or$V2_s$] $=0$;

if $V1_f$[or $V1_s$] $= 1$ then

set $T_{l_1[\text{or } r_1]} = 1$; $V1_f$[or $V1_s$] $= 0$;

send token according to selectPath()

else (if i is neither f nor s then)

if P_i^1[or P_i^2] $\in$ $l_1(l_2)$[or $r_1(r_2)$] then

reset $T_{l_1(l_2)}$[or $T_{r_1(r_2)}$] $= 0$

and send token to P_{i-1}^1[or P_{i-1}^2] in anticlockwise direction.

(c2) When P_i^1[or P_i^2] receives token then

If P_i^1[or P_i^2] has no pending request message then

if $i = f$[or s] then

if token is received from P_1^2[orP_1^1] then

reset $V1_f$[or $V1_s$] $= 0$;

if $V2_f$[or $V2_s$] $= 1$ then reset $T_{r_2[\text{or } l_2]} = 1$; $M_p = 0$;

send token according to selectPath();

else [if $V2_f$[or $V2_s$] $= 0$ then]

if T_{l_1} AND T_{l_2} AND T_{r_1} AND $T_{r_2} = 0$ then

stop token at P_i^1[or P_i^2]

else send token according to selectPath();

else (if token is received from P_{f+1}^1[or P_{s+1}^2] then)

reset $V2_f$[or $V2_s$] $= 0$;

if $V1_f$[or $V1_s$] $= 1$ then reset T_{l_1} [or T_{r_1}] $= 1$; $M_p = 0$ and

send token according to selectPath();

else [if $V1_f$[or $V1_s$] $= 0$ then]

if T_{l_1} AND T_{l_2} AND T_{r_1} AND $T_{r_2} = 0$ then

stop token at P_i^2[or P_i^1]

else send token according to selectPath();

else (if $i \neq f(s)$ then)

if $M_p = 1$ then reset $M_p = 0$;

send token to the processor P_{i-1}^1[or P_{i-1}^2] in

anticlockwise direction.

else (if $M_p = 0$ then)

if $T_{l_1}\ AND\ T_{l_2}\ AND\ T_{r_1}\ AND\ T_{r_2} = 0$ then

stop the token at P_i^1[or P_i^2] itself

else send the token to the next processor P_{i-1}^1[or P_{i-1}^2] .

Procedure selectPath();

/* This procedure selectPath() sends the token through an appropriate outgoing link alternatively if there is a pending request.*/

if $L_{f[\text{or } s]} = 0$ then

if $T_{l_1[\text{or } r_1]} = 1$ then set $L_{f[\text{or } s]} = 1$ and send token to P_{f-1}^1[or P_{s-1}^2]

else send token to P_m^2[or P_n^1]

else (if $L_{f[\text{or } s]} = 1$ then)

if $T_{r_2[\text{or } l_2]} = 1$ then set $L_{f[\text{or } s]} = 0$ and send token to P_m^2[or P_n^1]

else if $T_{l_1[\text{or } r_1]} = 1$ then send token to P_{f-1}^1[or P_{s-1}^2]

else set $L_{f[\text{or } s]} = 0$ send token to P_m^2[or P_n^1]

Theorem 3. *The algorithm* InteRN - B *serves all the request messages generated for the single shared resource.*

Proof. Let the token be initially at any one of the common processors [either at P_f^1 or at P_s^2]. Each request message in clockwise direction will eventually reach the token and informs it to access the single shared resource. After initiated by a request message, token moves in anticlockwise direction through the appropriate outgoing subarc of any one of the rings by setting the token locator L_f[or L_s] value in the common processor. Meantime $V1_f$, $V2_f$, $V1_s$, $V2_s$ and M_p values are maintained at the common processors to record the request messages received from two different arcs of the different rings respectively. This would be achieved by a switching subsystem of the respective common processor. Whenever token receives a request message at a common processor or reaches a common processor, it will reset $T_{l_1}(T_{r_2})$[or $T_{r_1}(T_{l_2})$] according to the values of the local variables $V1_f$, $V2_f$, $V1_s$, $V2_s$ and M_p. The switching subsystem records the receipt of request messages from two subarcs and hence the receipt of the request messages are not lost.

On the other hand, assume that token is at any one of the processors in

any one of the subarcs of a ring. The common processor, when it receives a request message, checks the values of $L_{f[\text{or } s]}$, $V1_f$, $V2_f$, $V1_s$, $V2_s$ and M_p and sets them accordingly as in steps $(b1)$ and $(b2)$. Then the switching subsystem of the respective common processor forwards the request message in clockwise direction to the corresponding subarc based on the direction through which the token has already been sent. In contrast, whenever token reaches the common processor in anticlockwise direction, it immediately checks the values of $V1_f$, $V2_f$, $V1_s$, $V2_s$ and M_p. After checking these values, procedure selectPath() is performed for selecting an appropriate outgoing link. As and when token switches over the subarcs, $V1_f$, $V2_f$, $V1_s$, $V2_s$ and M_p are dynamically maintained. Token moves through the processors until all the cell entries of the flag bit cells and M_p are zero. Thus request messages generated are not lost and token is aware of all the request messages. Thus the algorithm : InteRN - B serves all the requests. ■

Theorem 4. *The algorithm* InteRN-B *requires at most* N *messages per request and service traffic is bounded by* $2N$.

Proof. To prove the first measure, we look at the opposite movements of the token as well as the request messages. The associated variables are dynamically maintained at the processors. In the worst case, token is at a processor in the largest subarc of the first ring and the request message generated at a processor in the largest subarc of the second ring requires at most $(n+m) = N$ movements necessary to satisfy a sequence of requests. This N movements include the request messages generated by the remaining processors. Hence the average number of messages per request amounts to N in the worst case. Next we prove the second measure, service traffic. As the token moves in clockwise direction and request messages in anticlockwise direction, token can easily sensor and serve all the pending request messages in its way to the destination. In the worst case, token makes $(n+m) = N$ movements. Hence the service traffic amounts to $2N$, is a tight bound for the worst case. ■

5. Conclusion

In this paper, we have considered the problem of controlling the allocation of a single shared resource in ring extension topologies like bidirectional touching rings network and bidirectional intersecting rings network using request message based token passing strategy. First we have proposed an algorithm for the allocation of a shared resource in bidirectional touching rings network. It is shown that all the request messages generated are served and the algorithm - BTRN, requires at most $(N-1)$ messages per request and the service traffic is bounded

by $(2N-1)$. Then we have proposed another algorithm : InteRN - B for bidirectional intersecting rings network which requires at most N messages per request and its service traffic is bounded by $2N$. This could be generalized for higher order intersecting rings network. In future, we are planning to extend this problem for higher order mesh networks with fault tolerance. It would be quite interesting to look similar problems in multiport message passing systems.

Acknowledgments

The authors wish to acknowledge the Council of Scientific and Industrial Research (CSIR), New Delhi, INDIA for its partial financial support.

References

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

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

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

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

[5] **Feuerstein, E., S. Leonardi, A. Marchetti-Spaccamela** and **N.Santoro.** Efficient token based control in rings, *Inform. Process. Lett.* **(66)** (1998) 175-180.

[6] **Jie Wu.** *Distributed System Design*, CRC Press, 1999.

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

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

[9] **Rajendra Prasath, R.** and **P. Thangavel.** Shared resource allocation using token based control strategy in ring extension topologies, *In*: (Eds.) **Misra, J.C.** and **S.B.Sinha.** *Recent Trends in Mathematical Sciences*, Narosa Publishing House, Newn Delhi, Dec. 2000, pp. 53-63.

[10] **Raynal, M.** *Distributed Algorithms and Protocols,* John Wiley and Sons, New York, 1988.

[11] **Rego, V.** 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) (1988) 1224-1234.

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

[13] **Stallings, W.** *Data and Computer Networks*, 5^{th} Ed., Prentice Hall, New York, 1997.