INFORMATION
Volume 6, Number 2, pp. 197-206
ISSN 1343-4500

Shared resource allocation using token passing strategy in interconnected networks

R.Rajendra Prasath and P.Thangavel
Department of Computer Science, University of Madras,
Chepauk, Chennai - 600 005, India.

{rrajprasath, thangavelp}@yahoo.com

Abstract

This paper presents solutions based on *request based token control strategy* for controlling the allocation of a single shared resource in logically interconnected networks such as single side wrap around mesh and regular mesh. In this strategy, a processor which is in need of the shared resource, sends a request to inform the token of the access request. Token, on receipt of a request, continues searching the processor that has a pending request and stops after serving all requests. The proposed token passing strategies assume no source and destination identities and require only a finite number of message exchanges per request. It is observed that these protocols permit a bounded service time and tolerate node or link faults that are traced before the execution of the algorithms.

Keywords: Distributed computing, shared resource allocation, message passing, interconnection networks.

1. INTRODUCTION

We study the problem of controlling the allocation of a single shared resource among a set of processors that are arranged in logically interconnected networks like single side wrap around mesh and regular mesh. We propose solutions by means of a small frame called *control token*. A processor which needs the shared resource must get a free token and then transfers it into a busy token. After utilizing the shared resource, the busy token becomes a free token which is again circulated in the network for further allocation.

This type of message based token passing communication model is used in various combinatorial problems such as election problem, mutual exclusion problem, hub polling systems, etc [1, 2, 6, 10, 11, 12, 13]. All these problems use the same procedure: a processor having the token will pass it along the circular communication model, as soon as it no longer needs it. But the uncontrolled movement of the token results in an unbounded number of message exchanges even for a finite set of requests. To avoid the unnecessary traversals of the token and requests, we present protocols based on token passing mechanism [5]

in which token circulates only if it is informed of a request generated for the shared resource.

A vast literature[2, 11] on the performance of message based token passing strategies exists and only little is known about the use of token based control mechanism. In distributed computing, the research has been mainly focused on the detection of token loss and on self stabilizing aspects[3]. In this paper, we assume single shared resource and single token. We strictly assume that processors do not include any information along requests generated for the shared resource. Hence neither the origin nor the destination of generated requests is known in *priori*. Suppose if we include any information along requests then, one can easily find out the shortest path[4, 7] through which messages are routed from source to destination and this can be repeated for every processor by adding a queue of pending requests.

Feuerstein *et* al.[5] have proposed request based token passing protocols for controlling the allocation of a shared resource in unidirectional fault-free ring network in which the token as well as requests move in single direction. Then the request based token passing strategy has been extended to bidirectional rings, linear arrays and touching rings[8, 9, 15]. In this paper, we propose solutions for single side wrap around mesh and regular mesh network. Our objective is not only to minimize the total number of message exchanges per request necessary to allocate the single shared resource to a given set of requests, but also to accelerate the servicing of the pending requests with in a finite amount of time.

In the sequel, we present preliminaries. Then in section 3, we present an algorithm for a shared resource allocation in single side wrap around mesh network. In section 4, bidirectional token passing algorithm for shared resource allocation in regular mesh is presented. Finally, section 5 concludes the paper.

2. PRELIMINARIES

We suppose a single side wrap around mesh in which $N[= km]$ processors are arranged in k rings each with m processors ($k < m$) that are interconnected by a bidirectional channel in such a way that processor P_i^j ($i = 1, 2, 3, ..., k$ and $j = 1, 2, 3, ..., m$), of a ring is connected to P_{i-1}^j and P_{i+1}^j apart from P_i^{j+1} and P_i^{j-1} and processor P_i^m is connected to P_i^1. We assume unidirectional movements for token and requests over the ring edges and bidirectional movements for token over the edges connecting the processors in the adjacent rings. Each processor in this network has a switching subsystem to sensor the direction from which the request or token has been received. Also we assume that the token has a flag bit to control its movement over the processors in the adjacent columns of mesh.

Next we consider a regular mesh without wrap around connections in which N processors are arranged in two dimensions. In this network, token starts searching pending requests in one direction. When token reaches the end processor in that direction, then its direction is reversed and serves requests in other direction. Meantime it also serves all pending requests of column processors.

The analysis of the proposed protocols uses two distinct measures[5, 14]. 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 and the time in which the processor receives the token.

3. ALLOCATION IN SINGLE SIDE WRAP AROUND MESH

In this section, we describe single shared resource allocation using token passing approach in a single side wrap around mesh which is depicted as in Figure 1. In this architecture, we term the ring for which $i = 1$ (with processors $P_1^1, P_1^2, P_1^3, ..., P_1^m$) as *base ring*. Initially token resides at a processor in the base ring. It is assumed that if token is in a moving state over processors in the column of adjacent rings, no request from neighboring processors would be registered by that processor.

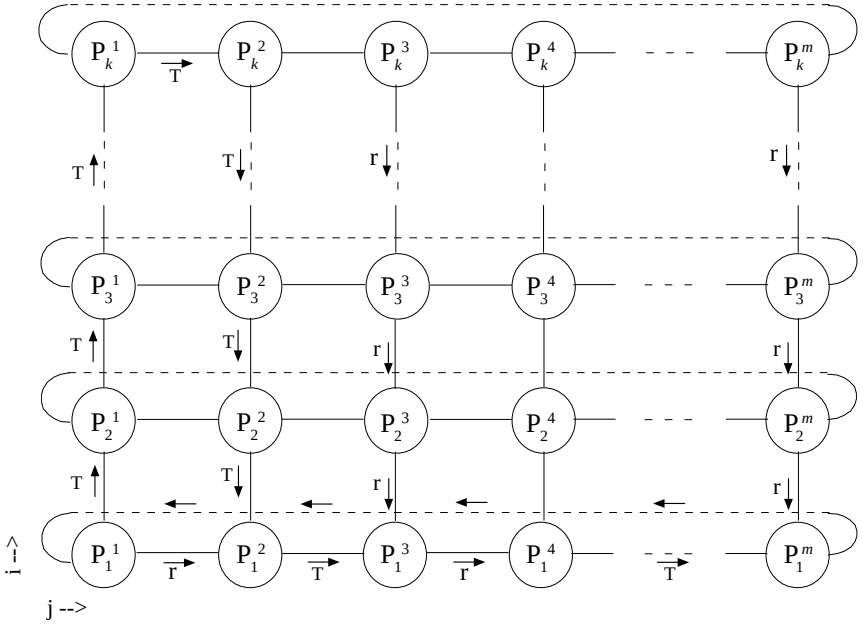

Figure 1: Single side wrap around mesh network

Each processor has local variables M_p and V_r. M_p assumes 1, if the processor has a pending request or a request has been passed through it and 0, otherwise. V_r assumes 1, if a request has been received from the processor P_{i+1}^j and 0, otherwise. Token has a flag bit L to control the token movement over processors in the adjacent columns. This flag bit L assumes 1, whenever token is received from processor P_i^{j-1} $(i > 1)$ by P_i^j. The value of L will be reset to zero as and when token reaches the processor in the base ring. When token recognizes both the values of M_p and V_r as zero at any one of the processors in the base ring, then it starts making a *check tour* to trace all remaining pending requests. For this purpose, we assume that token additionally has a variable C for recording the number of token movements in the check tour over the processors in the base ring. During the check tour, id the token recognizes V_r of any one of the processors as 1, then the check tour will be terminated; the value of C will be set

to zero and the corresponding pending request(s) from the column processor(s) will be served. Token stops moving when it recognizes the value of C as m and the values of M_p and V_r as zero.

Algorithm - MICRN

// Left arrow($\leftarrow$) indicates the processor from which request or token is received.

(a) When P_i^j needs the shared resource then

- $\star$ If P_i^j has token then enter the critical section.
- $\star$ If P_i^j has no token then

  if ($i = 1$; $M_p = 0$) then set $M_p = 1$ and send request to P_i^{j+1} if $V_r = 0$;
  else (either M_p= 1 or $V_r = 1$) stop the requests at P_i^j

  else (if $i > 1$ then) set $M_p = 1$ and send the request to P_{i-1}^j if V_r=0;
  otherwise stop the request at P_i^j

(b) When P_i^j receives a request then

- $\star$ If P_i^j has token then reset $C = 0$, $M_p = 0$;

  if request is from P_i^{j-1} then send token to P_i^{j+1}
  else (if request is from P_{i+1}^j then) send the token to P_{i+1}^j.
- $\star$ If P_i^j has no token then

  if $i = 1$ then if the request is received from P_i^{j-1} then
  if $M_p = 1$ then stop the request at P_i^j
  else (if $M_p = 0$ then) set $M_p = 1$;
  if $V_r = 0$ then send request to P_i^{j+1}; else stop request at P_i^j
  else (if the request is received from P_{i+1}^j then) set $V_r = 1$;
  if $M_p = 1$ then stop the request at P_i^j;
  else set $M_p = 1$ and send the request to P_i^{j+1}

  else (if $i > 1$ then) set $V_r = 1$;
  if $M_p = 1$ then stop the request at P_i^j
  else set $M_p = 1$ and send the request to P_{i-1}^j.

(c1) When P_i^j receives token and has a pending request then

enter the critical section; set $M_p = 0$;
if $i = 1$ then reset $C = 0$;
if token is from P_i^{j-1} then send token to P_i^{j+1} if V_r=0;
otherwise set $V_r = 0$ and send token to P_{i+1}^j
else (token $\leftarrow P_{i+1}^j$) reset $V_r = 0$; set $L = 0$ and send token to P_i^{j+1}
else(if $i > 1$ then) if token is received from P_{i-1}^j then
if $V_r = 1$ then reset $V_r = 0$ and send token to P_{i+1}^j
else if $L = 0$ then send token to P_i^{j+1}; else send token to P_{i-1}^j
else if token is received from P_i^{j-1} then set $L = 1$;
if $V_r = 1$ then reset $V_r = 0$ and send token to P_{i+1}^j
else send token to P_{i-1}^j
else (token $\leftarrow P_{i+1}^j$) reset $V_r = 0$ and send token to P_{i-1}^j

(c2) When P_i^j receives token and has no pending request then

if (i=1; token is received from P_i^{j-1}) then
if $V_r = 1$ then reset $M_p = 0$; $V_r = 0$; $C = 0$; and send token to P_{i+1}^j

else if $M_p = 1$ then reset M_p=0; and send token to P_i^{j+1}
else if C = m then stop the token at P_i^j
else set $C = C + 1$ and send token to P_i^{j+1}
else (i=1; token $\leftarrow P_{i+1}^j$)
set $L = 0$; $V_r = 0$; $M_p = 0$ and send token to P_i^{j+1}
else ($i > 1$) set $M_p = 0$;
if (token is received from P_i^{j-1} then) set $L = 1$;
if $V_r = 1$ then reset $V_r = 0$ and send token to P_{i+1}^j;
else send token to P_{i-1}^j
else (token $\leftarrow P_{i-1}^j$) set $V_r = 0$ and send token to P_{i+1}^j
else (token $\leftarrow P_{i+1}^j$) set $V_r = 0$ and and send token to P_{i-1}^j.

Theorem : 1

Algorithm - MICRN serves all requests generated for single shared resource.

Proof:

Initially let the token be at any one of the processors in the base ring. All processors other than those in the base ring are allowed to send requests towards the corresponding processor in the base ring and processors in the base ring can send requests only in clockwise direction. Therefore a request generated by processor P_i^j in the i^{th} ring is projected towards the corresponding processor P_i^j in the base ring. If the token is not available then, it will be passed to the next processor in the clockwise direction by setting the variables appropriately. In the meantime, other processors may also generate and project requests to inform token for accessing the shared resource.

The generated request will be forwarded only if it recognizes the values of M_p and V_r as zero. Otherwise, it will be stopped as another request has already been sent through that processor. Thus the generated request reaches the token and informs it to access the single shared resource. Now token starts moving to serve the pending requests. Token searches for all pending requests as in steps (c1) and (c2) and stops moving after completing a check tour in the base ring. Thus algorithm MICRN is aware of all requests and hence token serves all processors. ∎

Theorem : 2

The algorithm MICRN requires $(N - 1)$ messages per request and service traffic is bounded by $(2N + m - 1)$.

Proof:

Let the token be initially at any one of the processors in the base ring. Now each request originated at P_i^{j+1} has to traverse atmost $(m-1)$ processors before it reaches the token or it has been stopped by a preceding request. Meanwhile processors in adjacent rings may also generate and send requests for the token. Thus in the worst case, a request generated in the adjacent ring processor requires $(k - 1)$ messages to reach their corresponding processor in the base ring and then makes $(m - 1)$ moves over processors in the base ring. In the mean time, $m(k - 1)$ messages may be generated by column processors. Hence in the worst case, the total number of message exchanges per request amounts to $(m(k - 1) + m - 1) = (N - 1)$.

To find the service traffic, we consider both request and token movements.

The request originated from the base ring processor requires atmost $(m-1)$ messages to inform the token. Now each request originated from column processors may require $(k-1)$ moves ahead per column to reach their corresponding processor in the base ring and then moves over the processors in the base ring provided $M_p = 0$. Thus in total, atmost $(N-1)$ messages are required in the worst case. Token starts, according to the value of V_r and then M_p, from the processor in the base ring. So atmost $(N+m)$ token movements are required to service the processors with pending request. Hence in the worst case, service traffic amounts to $(2N+m-1)$ and token skips no requests. ∎

4. ALLOCATION IN MESHES

Next we consider a regular mesh network in which bidirectional movements for both token and requests are assumed. Here token needs two flag bits T_l and T_r to record the pending requests from the left and right end processors of the base row respectively and T_d to regulate the direction of the token over processors in i^{th} $(1 < i \leq k)$ row. Token maintains T_l (T_r) as 1, if request is received from the left end (right end) processor and 0, otherwise. In the base row (i=1), we require another local variable $M_p^{l(r)}$ that stores 1, if a request is received from processor P_i^{j-1} (P_i^{j+1}). Each processor in the base row (i=1) has a direction bit d that takes value 0(1) if token is sent to left (right) processor. We assume all other variables and assumptions as in the algorithm-MICRN except the token counter C, used in the check tour, which is not needed in meshes without wrap around connections. In this algorithm token moves from the base row processor to column processors and serves all requests in that column. Then, token is sent to the next processor in the adjacent column. From this column processor, token reaches the processor in the base row only after serving all the processors in that column. Whenever there is no pending request, then token is idle in any one of the base row processors. Similarly request from column processors first reaches the base row and then forwarded towards left or right port appropriately.

Algorithm : SRA_Mesh

// Left arrow(←) indicates the processor from which request or token is received.

(a) When P_i^j needs resource then
  If P_i^j has token then it enters the critical section.
  If P_i^j has no token then
    If $i = 1$ then if $M_p = 0$ then set M_p =1
      if $V_r = 0$ then send the request along the left port if d= 0
        otherwise through the right port
      else (either $V_r = 1$ or $M_p = 1$) no request is sent
    else(if $i > 1$ then) set $M_p = 1$ and send the request to P_{i-1}^j if $V_r = 0$;
      otherwise stop the request at P_i^j.

(b1) When P_i^j receives request and has token then set $M_p = 0$;
  If $j = 1[or\ m]$ then set $d = 1[or\ 0]$; reset $M_p^{r[or\ l]} = 0$;
    if request is from P_i^{j+1}[or P_i^{j-1}] then
      set $T_{r[or\ l]} = 1$ and send token to P_i^{j+1}[or P_i^{j-1}]
    else (request ← P_{i+1}^j) reset $V_r = 0$ and send token to P_{i+1}^j

else $(1 < j < m)$
if request is from P_i^{j-1}[or P_i^{j+1}] then set $T_{l[or\ r]} = 1$; $M_p^{l[or\ r]} = 0$;
if $d = 0$ [or 1] then
if $T_{r[or\ l]} = 1$ then set $d = 1$[or 0] and send token to P_i^{j+1} [or P_i^{j-1}]
else send token to P_i^{j-1}[or P_i^{j+1}]
else (request $\leftarrow P_{i+1}^j$) set $V_r = 0$ and send token to P_{i+1}^j

(b2) When P_i^j receives request and has no token then
if ($i = 1$ and M_p=0) then
if the request is received from P_{i+1}^j then set $V_r = 1$;
else (request $\leftarrow P_i^{j-1}$[or P_i^{j+1}]) set $M_p^{l[or\ r]} = 1$; $M_p = 1$
if $j = 1$[or m] then send the request to P_i^{j+1}[or P_i^{j-1}]
else $(1 < j < m)$ send the request to P_i^{j-1} if d=0; else to P_i^{j+1}
else(i=1 and M_p=1) stop the request at P_i^j
else $(i > 1)$ set $V_r = 1$ and stop the request at P_i^j if M_p=1
otherwise set $M_p = 1$ and send the request to P_{i-1}^j

(c1) When P_i^j receives token and has a pending request then
it enters the critical section; reset $M_p = 0$; at the end of critical section,
if ($i = 1$; $j = 1$[$or\ m$]) then set $T_{l[or\ r]} = 0$
if token is received from $P_2^{1[or\ m]}$ then reset $L = 0$; $V_r = 0$;
if $T_{r[or\ l]} = 1$ then set $d = 1$ [or 0] and send token to P_1^2[or P_1^{m-1}]
else if token is received from P_i^{j+1}[or P_i^{j-1}] then
if $V_r = 1$ then set $V_r = 0$; $T_d = 1$[or 0] and send token to P_{i+1}^j
else if $T_{r[or\ l]}$=1 then set d=1[or 0] and send token to P_i^{j+1}[or P_i^{j-1}]
else token stays at P_i^j
else (i= 1; $1 < j < m$)
if token is received from P_i^{j-1} [or P_i^{j+1}] then
if $V_r = 1$ then set $d = 1$ [or 0]; $T_d = 1$ [or 0];
$V_r = 0$ and send token to P_{i+1}^j
else set d=1[or 0] send token to P_i^{j+1}[or P_i^{j-1}]
else(token is received from P_{i+1}^j) reset V_r= 0; L=0;
if $d = 0$ [or 1] then if $M_p^{l[or\ r]}$=1 then reset $T_{l[or\ r]}$=1
if $T_{r[or\ l]} = 1$ then set $d = 1$[or 0] and send token P_i^{j+1} [or P_i^{j-1}]
else if $T_{l[or\ r]} = 1$ then send token P_i^{j-1}[or P_i^{j+1}]
else ($T_{l[or\ r]} = 0$) token stops at P_i^j
if ($i > 1$; j=1[or m]) then
if token is received from P_{i-1}^j then
if $V_r = 1$ then reset $V_r = 0$ and send token to P_{i+1}^j
else if $L = 0$ then send token to P_i^{j+1} [or P_i^{j-1}]
else send token to P_{i-1}^j
else if token is received from P_i^{j+1} [or P_i^{j-1}] then set $L = 1$;
send token to P_{i+1}^j if $V_r = 1$; otherwise send token to P_{i-1}^j
else (token $\leftarrow P_{i+1}^j$) reset $V_r = 0$ and send token to P_{i-1}^j
else ($i > 1$; $1 < j < m$) if (token $\leftarrow P_{i-1}^j$) then
if $V_r = 1$ then reset $V_r = 0$ and send token to P_{i+1}^j

else if $L = 0$ then send token to P_i^{j-1}[or P_i^{j+1}] if $T_d = 0$ [or 1]
else send token to P_{i-1}^j
else if token is from P_i^{j+1} [or P_i^{j-1}] then set $L = 1$;
if $V_r = 1$ then set $V_r = 0$ and send token to P_{i+1}^j
else send token to P_{i-1}^j
else if token $\leftarrow P_{i+1}^j$ then reset V_r= 0 and send token to P_{i-1}^j

(c2) When P_i^j receives token and has no pending request then
if ($i = 1$; $j = 1$[or m]) then
if token is received from P_i^{j+1}[or P_i^{j-1}] then set $T_{l[or\ r]}$=0;
if $V_r = 1$ then
set $M_p^{r[or\ l]} = 0$; $M_p = 0$; $V_r = 0$ and send token to P_{i+1}^j
else if $T_{r[or\ l]} = 1$ then set $d = 1$[or 0] and send token to P_i^{j+1}[or P_i^{j-1}]
else token stays at P_i^j
else(token $\leftarrow P_2^{1[or\ m]}$)
if $T_{r[orl]} = 1$ then reset $M_p^{r[orl]}$ =0; M_p=0; V_r=0;
d=1[or 0] and send token to P_1^2[or P_1^{m-1}]
else token stays at $P_1^{1[or\ m]}$
else ($i = 1$; $1 < j < m$) if token $\leftarrow P_{i+1}^j$ then
if d=0[or 1] then if $M_p^{l[or\ r]} = 1$ then set $T_{l[or\ r]} = 1$;
if $T_{r[or\ l]} = 1$ then set $d = 1$[or 0] and send token to P_i^{j+1}[or P_i^{j-1}]
else if $T_{l[or\ r]} = 1$ then send token to P_i^{j-1}[or P_i^{j+1}]
else ($T_r = 0$; $T_l = 0$) token stays at P_i^j
else if token is received from P_i^{j-1}[or P_i^{j+1}] then
if $V_r = 1$ then set $V_r = 0$; $d = 1$[or 0];
$T_d = d$ and send token to P_{i+1}^j
else if $T_{r[or\ l]} = 1$ then reset $M_p = 0$; $M_p^{r[or\ l]}$=0; d=1[or 0] and
send token to P_i^{j+1}[or P_i^{j-1}]
else if $T_{l[or\ r]} = 1$ then send token to P_i^{j-1}[or P_i^{j+1}]
else (T_l=T_r=0) token stays at P_i^j
If ($i > 1$; $j = 1$[or m]) then
if token $\leftarrow P_{i-1}^j$ then if $V_r = 1$ then reset $V_r = 0$ and send token to P_{i+1}^j
else(V_r=0) if L=0 then send token to P_i^{j+1}[or P_i^{j-1}]
else send token to P_{i-1}^j
else if token $\leftarrow P_i^{j+1}$[or P_i^{j-1}] then set $L = 1$; if $V_r = 1$ then set V_r=0
and send token to P_{i+1}^j; otherwise send token to P_{i-1}^j
else if token $\leftarrow P_{i+1}^j$ then reset $V_r = 0$ and send token to P_{i-1}^j
else ($i > 1$; $1 < j < m$)
if token $\leftarrow P_i^{j-1}$[or P_i^{j+1}] then set $L = 1$; if $V_r = 1$ then set $V_r = 0$ and
send token to P_{i+1}^j; otherwise(V_r=0) send token to P_{i-1}^j
else(token $\leftarrow P_{i+1}^j$) reset $V_r = 0$ and send token to P_{i-1}^j
else(token $\leftarrow P_{i-1}^j$)
if V_r=1 then set V_r=0; M_p=0 and send token to P_{i+1}^j
else if L=0 then send token to P_i^{j+1}[or P_i^{j-1}] if T_d=1[or 0];
else(L=1) send token to P_{i-1}^j

Theorem : 3

The algorithm **SRA_Mesh** serves all requests generated for the single shared resource and requires $(N-1)$ messages per request and service traffic is bounded by $(2N+1)$.

Proof: In algorithm **SRA_Mesh**, token serves all requests in one direction and then its direction is reversed. Token serves requests according to the value of flag bits T_l and T_r. Token from an intermediate processor takes diversion to the next column as per the value of T_d. Token, while reaching the base row processor, it checks $M_p^{l[or\ r]}$ value to store the pending request received from the left[or right] processor. All other arguments are similar to the proof of Theorems 1 and 2. ∎

In both algorithms, as processors are aware of incoming and outgoing requests and token movement, we can detect the faulty nodes or links that occur other than the base row (which is assumed to be fault-free) before starting the execution of the algorithms.

5. CONCLUSION

In this paper, we have considered the problem of controlling the allocation of a single shared resource in interconnected networks like a single side wrap around mesh and regular meshes using request based token passing strategy. First we have proposed an algorithm that allocates the single shared resource in a single side wrap around mesh and requires atmost $(N-1)$ messages per request and the service traffic is bounded by $(2N+m-1)$. Then we have proposed an allocation algorithm which requires atmost $(N-1)$ messages per request and $(2N+1)$ service traffic in regular meshes. It would also tolerate node or link faults which can be detected before the execution.

ACKNOWLEDGEMENTS

This work is partially supported by Council of Scientific and Industrial Research(CSIR) and All India Council for Technical Education(AICTE) Government of India.

References

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

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

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

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

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

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

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

[8] Rajendra Prasath R., Thangavel P., Token based message passing in bidirectional ring extensions, *Journal of Madras University*, 52(2000), 145-159.

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

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

[11] Rego V, Ni L.M., 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, 1997.

[14] Tarjan, R.E., Amortized computational complexity, *SIAM J. Alg. Discrete Math.* 6(2)(1985), 306-318.

[15] Thangavel P, Rajendra Prasath R., A note on token based control in rings and linear arrays, *Journal of the CSI*, 32(3)(2002), 62-65.