ELSEVIER

Journal of Systems Architecture 45 (1998) 409–420

JOURNAL OF SYSTEMS ARCHITECTURE

# An $O(n^{1/3})$ algorithm for distributed mutual exclusion

Pranay Chaudhuri *, Mehmet Hakan Karaata [1]

*Department of Electrical and Computer Engineering, Kuwait University, PO Box 5969, Safat, Kuwait*

Received 16 January 1997; received in revised form 30 September 1997; accepted 21 October 1997

**Abstract**

In this paper, a distributed algorithm is proposed that realizes mutual exclusion among $n$ nodes in a computer network. No common or global memory is shared by the nodes and there is no global controller. The nodes of the network communicate among themselves by exchanging messages only. The best-known algorithms so far, for the distributed mutual exclusion problem, require $O(\sqrt{n})$ messages per mutual exclusion invocation. The proposed algorithm is the first to cross this $O(\sqrt{n})$ barrier and the message complexity achieved by our algorithm is $O(n^{1/3})$ per mutual exclusion. 

*Keywords:* Distributed algorithm; Computer network; Mutual exclusion; Critical sections; Message complexity

## 1. Introduction

The *mutual exclusion* problem consists of ensuring that at most one process can access a shared object at any time. For example, one process may not write to a file while another is reading it. Segments of code to be executed mutually exclusively are referred to as *critical sections* (or critical regions) in the literature [1–4] It is not too difficult to implement mutually exclusive use of objects if the use of such objects is under a global or centralized controller. However, in a distributed or network environment, due to the absence of a global controller, the mutual exclusion algorithms become far more complex than those for systems with centralized control.

In this paper, we are concerned with a computer network consisting of $n$ nodes where the nodes communicate solely by exchanging messages and do not share any common or global memory. There are $n$ processes, each residing at a different

* Corresponding author. E-mail: pranay@eng.kuniv.edu.kw
[1] E-mail: karaata@eng.kuniv.edu.kw

1383-7621/98/$ – see front matter 
PII: S1383-7621(97)00087-8

node, wanting to invoke mutual exclusion. For convenience of presentation, we shall not distinguish between a node and its resident process. The problem is to design a distributed algorithm that realizes mutual exclusion among the nodes, i.e. at any moment at most one node may be allowed to remain in its critical section.

Lamport's algorithm, for this problem, requires approximately $3(n-1)$ messages per mutual exclusion invocation [5]. Later Ricart and Agrawala [6] proposed an algorithm for this problem that requires $2(n-1)$ messages, and then Carvalho and Roucairol [7] presented an algorithm that fulfills similar requirements as Ricart and Agrawala's solution, but it requires between 0 and $2(n-1)$ messages. A second algorithm by Ricart and Agrawala uses either 0 or $n$ messages for the same problem [8]. Suzuki and Kasami's algorithm, based on the concept of node privilege, requires $n$ messages per mutual exclusion invocation [9]. Maekawa [10] further reduces the number of messages per mutual exclusion to $c\sqrt{n}$, where $c$ is a constant between 3 and 5. A distributed mutual exclusion algorithm is presented by Helary et al. that works for any arbitrary network [11]. The number of messages required by the algorithm due to Helary et al. varies from $n$ to $2e+n-1$ depending on network topology, where $e$ is the number of links in the network. Raymond proposed a tree-based algorithm for distributed mutual exclusion which uses a spanning tree of the computer network and the average number of messages is $O(\log n)$ per mutual exclusion [12]. An optimal algorithm for mutual exclusion in mesh-connected computer networks has been proposed by Chaudhuri [13]. This algorithm requires $3.5\sqrt{n}$ messages per mutual exclusion invocation under light demand which reduces to approximately four messages only under heavy demand. Very recently, Chaudhuri has proposed a distributed mutual exclusion algorithm for arbitrary networks which requires $3\sqrt{n}$ messages per mutual exclusion and the message requirement reduces to only $\sqrt{n}$ under heavy demand [14].

It appears that the best available algorithms so far, for the distributed mutual exclusion problem, achieved an upper bound of $O(\sqrt{n})$ for message complexity [10,14]. In this paper, we propose an algorithm for distributed mutual exclusion that achieves a message complexity of $O(n^{1/3})$ per mutual exclusion invocation.

## 2. Distributed mutual exclusion algorithm

We assume that the $n$ node network is available in the form of a three-dimensional mesh, i.e., $(p \times q \times r)$ mesh. Although our algorithm works for any values $p$, $q$, and $r$, we assume, for convenience, $p=q=r=n^{1/3}$. Each node of the network is identified by a triple (i.e. row number, column number, plane number). The bidirectional links of this three-dimensional mesh are used for message passing. However, all these links are not used for message passing. Such a three-dimensional mesh is shown in Fig. 1. The links which are shown by bold lines are actually useful for message passing. Therefore, the total number of necessary bidirectional links for message passing in order to realize mutual exclusion among $n$ nodes is $13(n^{1/3}-1)$. If the network is fully connected, we assume it as an equivalent mesh by logically grouping the nodes to form the three-dimensional mesh. In this case, all the links other than those $13(n^{1/3}-1)$ links, as mentioned earlier, are redundant so far as the realization of mutual exclusion by the proposed algorithm is concerned. From now on, we shall refer to the network without considering the redundant links as a *three-dimensional equivalent mesh* (TDEM). The proposed algorithm relies on some kind of token circulation and is also based on the concept of plane privilege. These are explained in Section 2.1.

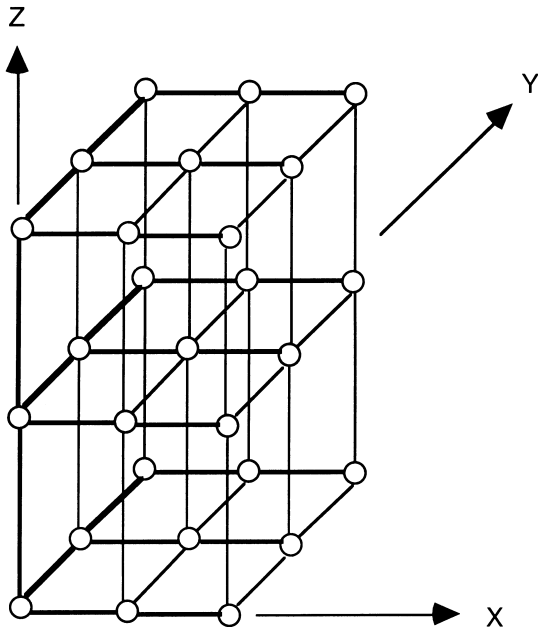



Fig. 1. A three-dimensional mesh with $n = 27$ nodes. The links shown by bold lines are only used by the proposed algorithm for message passing.

### 2.1. Token circulation strategy

In this subsection, we establish the basis of the algorithm by explaining the rules for token circulation among the nodes and introducing the notions of privilege state and privilege plane.

To each node $(0,0,k)$, $k = 0,1,\ldots,n^{1/3} - 1$, we assign a token which can assume one of the three colours: green, red, and yellow. Initially, node (0,0,0) of the TDEM is assigned with a token of colour green and each other node $(0,0,k)$, $k = 0,1,\ldots,n^{1/3} - 1$, is assigned with a token of colour red. No token is assigned, initially, to the remaining nodes. Fig. 2 shows a typical TDEM with $n = 27$ and the initial token assignment. A plane of the TDEM can be in either of the following two states.

*Privileged state*: A plane $x$ (i.e., all the nodes with $k = x$) of the TDEM is said to be in the privileged state if node $(0,0,x)$ has either a green or a yellow token. Such a plane is referred to as a *privileged plane*. The initial token assignments to each node, as indicated above, ensures that at the beginning there exists exactly one privileged plane in a TDEM. Later in this paper, it is shown that at any time during the execution of the algorithm there cannot exist more than one privileged plane in a TDEM.

*Nonprivileged state*: Any plane of the TDEM other than the privileged plane is said to be in the nonprivileged state. Such a row will be referred to as a *nonprivileged plane*.

The main idea behind our algorithm is that a node $(i, j, k)$ can invoke mutual exclusion if node $(i, j, k)$ has received a PERMIT message (defined in Section 2.2), i.e., plane $k$ is the privileged plane and node $(i, j, k)$ has got its turn among all the nodes of plane $k$. If plane $k$ is not the privileged plane, then one and only one plane $q$ $(q \neq k)$ must be the privileged plane. In this case, plane $k$ will become the privileged plane first, upon interchanging the tokens of node $(0,0,k)$ and node $(0,0,q)$. Node $(0,0,k)$ upon receiving the green token from node $(0,0,q)$ sends a PERMIT message to node $(i,0,k)$ and changes its colour to yellow indicating that this plane has the privilege and a node of this plane is now in its critical section. When no node in this plane would be in its critical section, the colour of the token in node $(0,0,k)$ would be turned back to green. The PERMIT message eventually reaches to node $(i,j,k)$, since it is transmitted through all the nodes of row $i$. Clearly, as long as a PERMIT message exists in a row of the privileged plane $k$, the colour of the token with node $(0,0,k)$ remains yellow. Thus, one of the assumptions on which our algorithm is based is as follows.

**Assumption A1.** A node $(i,j,k,)$ for $i,j,k \in \{0,1\ldots, n^{1/3} - 1\}$ can be inside its critical section if plane k is the privileged plane.

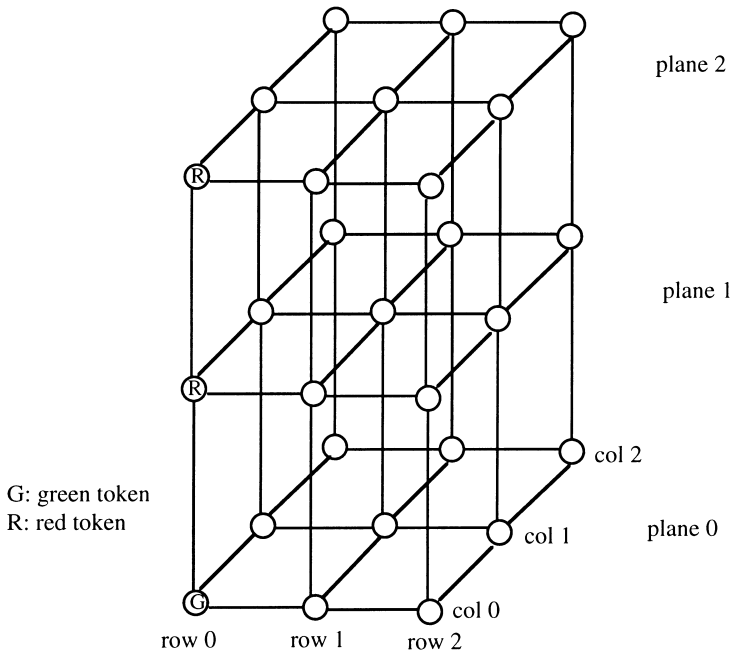



Fig. 2. A 27-node TDEM with initial token distribution to nodes (0,0,$k$), $k = 0, 1, 2$. Initially, plane 0 is the privileged plane.

From the above discussion, we have seen that token interchange can take place only in the $z$-direction, and between a pair of nodes – one of which has a green token and the other with a red token. The token distribution and the token interchange (or movement) among nodes (0,0,$k$), $k = 0,1,\ldots,n^{1/3} - 1$, in the TDEM can be restricted by ensuring that the algorithm must maintain the following set of invariants at any time during the execution.

**Invariant I1.** There cannot exist more than one node in the TDEM with green or yellow token at any time. The plane to which this node belongs is the privileged plane.

**Invariant I2.** There cannot exist both green and yellow tokens in TDEM at the same time.

The token state of node (0,0,$k$), $k = 0$, $1,\ldots,n^{1/3} - 1$, and possible transitions is shown in Fig. 3. In order to develop the distributed algo-

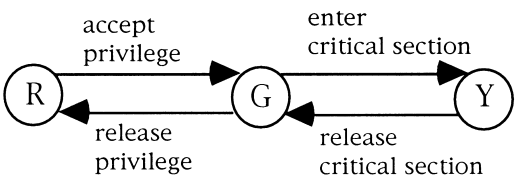



Fig. 3. The token state of node $(0, 0, k), k = 0, 1 \ldots, n^{1/3} - 1$, and possible transitions.

rithm, in addition to assumption A1, we need the following assumption.

**Assumption A2.** The messages are received without errors in the order sent and require a finite time to move from a node to its neighbour.

### 2.2. Messages used

In this subsection, first we describe the structures of the messages and the type of the variables used by the algorithm, and then present a more formal description of the algorithm based on the ideas of Section 2.1.

A set of three different types of messages is used by our algorithm. A general syntactic description of these messages is as follows.

⟨message⟩ ::= ⟨⟨type⟩, ⟨node id⟩⟩
⟨type⟩ ::= REQUEST | PERMIT | RELEASE
⟨node id⟩ ::= triple (row no., column no., plane no.)

The meaning of the different types of messages are explained below.

REQUEST: The parameter field of a REQUEST message carries only the ⟨node id⟩ of the sender. A REQUEST message from node $(i,j,k)$, whose final destination is always node $(0,0,k)$, implies that the sender wants to enter into its critical section and is waiting to receive the PERMIT message from node $(0,0,k)$. A node $(x,y,z) \neq (0,0,k)$ upon receiving the REQUEST message from a neighbour node simply passes it to its neighbour which is closer to node $(0,0,k)$, i.e., to node $(x,y-1,k)$ when $x=0$, and to node $(x-1,y,k)$, when $x \neq 0$. On the otherhand, node $(0,0,k)$ upon receiving a REQUEST message from its neighbour does one of the following: (i) If the colour of its token is green then it changes the token colour to yellow and sends a PERMIT message to the initiator of the REQUEST message, (ii) If the colour of its token is yellow then it inserts the REQUEST into its queue, (iii) If the colour of its token is red then it sends REQUEST messages to its neighbours in other planes.

PERMIT: The parameter field of a PERMIT message carries only the ⟨node id⟩ of the final destination node which has been granted the permission for its critical section entry. A PERMIT message can be initiated by a node having a green token. Assuming node $(0,0,k)$ as the initiator of a PERMIT message, its final destination may be either a node $(i,j,k)$ in the same plane or a node $(0,0,z)$ in a different plane $z \neq k$. A node $(p,q,k)$ upon receiving a PERMIT message from its neighbour, whose final destination is $(u,v,k)$ passes it to node $(p+1,q,k)$ when $q=v$, and to node $(p,q+1,k)$, when $q \neq v$. A node $(0,0,z)$ upon receiving a PERMIT message either passes it to its other neighbour in a different plane if the final destination of the PERMIT message is not itself, or changes the colour of its token to green.

RELEASE: The parameter field of a RELEASE message is redundant; however, to keep the uniformity of the message body the sender ⟨node id⟩ is included in it. The RELEASE message is initiated by a node, other than the node having a yellow token, upon completion of its critical section processing. If node $(i,j,k)$ initiates a RELEASE message then its final destination is always node $(0,0,k)$. Every other node except node $(0,0,k)$ simply passes the RELEASE message to its neighbour that is closer to node $(0,0,k)$, whereas node $(0,0,k)$ changes the colour of its token from yellow to green.

Other than the set of triples for various ⟨node id⟩'s, the following variables are used by the algorithm.

token $(i,j,k)$ It can assume one of the three different values, denoted in the algorithm as three different colours: green, yellow, and red.

| | |
|---|---|
| mutex | A local Boolean variable which is TRUE if corresponding node has got its turn to enter into its critical section and mutex remains TRUE until the critical section operation is completed. |
| $Q$ | Queue of ⟨node id⟩s of the REQUEST messages |

We also require the following operations and assume that standard procedures corresponding to them are available.

| | |
|---|---|
| enqueue ($Q$,($x$,$y$,$z$)) | Inserts ($x$,$y$,$z$) into the rear of queue $Q$ |
| dequeue ($Q$,($x$,$y$,$z$)) | Deletes an element from the front of queue $Q$ and returns it as ($x$,$y$,$z$) |
| delete ($Q$,($x$,$y$,$z$)) | Deletes element ($x$,$y$,$z$) from queue $Q$ |

### 2.3. The algorithm

Each node has four procedures. Procedure $P1$ is called when a node attempts to invoke mutual exclusion. Each of the procedures $P2$, $P3$, and $P4$ is executed indivisibly whenever a REQUEST, PERMIT, and RELEASE message, respectively, arrives. The detailed algorithm for node ($i$,$j$,$k$), $i$,$j$,$k$ $\in \{0,1,\ldots,n^{1/3}-1\}$ is given below.

**Algorithm DISTRIBUTED_MUTEX:**

```
/*Initialize*/
mutex := FALSE
if i=j=0 then do
 Q := φ
 if k=0 then token (i,j,k) := GREEN
 else token (i,j,k) := RED
 fi od fi
Procedure P1;
/*This procedure invokes mutual exclusion for
this node*/
/*Request entry for critical section*/
if i=j=0 then
 if token (i,j,k)=GREEN then do
  token (i,j,k) := YELLOW
  mutex := TRUE
 od
 else if token (i,j,k):= RED then do
  enqueue (Q,(i,j,k))
  send <REQUEST, (i,j,k)> to (i,j,q),
  q=k ± 1
   od
  else enqueue (Q,(i,j,k)) fi fi
else if i=0 then send <REQUEST, (i,j,k)> to
(i,j − 1,k)
 else send <REQUEST, (i,j,k)> to (i − i,j,k)
 fi fi
wait for (mutex=TRUE)
/*Critical section processing can be performed
at this point*/
/*Release critical section*/
mutex := FALSE;
if i=j=0 then do
 token (i,j,k)=GREEN
 if Q ≠ φ then do
  dequeue (Q,(x,y,z))
  grant_permission (x,y,z)
  od fi od
else send <RELEASE, (i,j,k)> to (0,0,k)
fi
Procedure P2;
/*<REQUEST, (u,v,w)> is received*/
if i=j=0 then do
 enqueue (Q,(u,v,w))
 if u=v=0 then
  if w < k then send <REQUEST>, (u,v,w)>
  to (i,j,k + 1)
  else send <REQUEST,(u,v,w)> to
  (i,j,k − 1) fi
 else if token (i,j,k)=RED then
    send <REQUEST, (i,j,k)> to (i,j,q),
  q=k ± 1 fi fi
 if token (i,j,k)=GREEN then do
```

```
  dequeue (Q,(x,y,z)
  grant_permission (x,y,z)
  od fi od
else if i = 0 then send <REQUEST, (u,v,w)>
to (i,j − 1,k)
   else send <REQUEST, (u,v,w)> to
 (i − 1,j,k)
 fi fi
Procedure P3;
/*<PERMIT, (u,v,w)> is received*/
if (u,v,w) = (i,j,k) then
 if i = j = 0 then do
  token (i,j,k) := GREEN
  if Q ≠ φ then do
   dequeue (Q,(x,y,z))
   grant_permission (x,y,z)
   od fi od
 else mutex:= TRUE fi
else if u = v = i = j = 0 then do
   delete (Q,(u,v,w))
  if w < k then send <PERMIT, (u,v,w)> to
  (i,j,k − 1)
      else send <PERMIT, (u,v,w)> to
   (i,j,k + 1) fi od
 else if v = j then send <PERMIT, (u,v,w)> to
 (i + 1,j,k)
         else send <PERMIT, (u,v,w)> to
  (i,j + 1, k)
  fi fi fi
Procedure P4;
/*<RELEASE, (u,v,w)> is received*/
if i = j = 0 then do
 token (i,j,k):= GREEN
 if Q ≠ φ then do
  dequeue (Q,(x,y,z)
  grant_permission (x,y,z)
  od fi od
else if i = 0 then send <RELEASE (u,v,w)> to
(i,j − 1,k)
 else send <RELEASE, (u,v,w)> to (i − 1,j,k)
 fi fi
```

The following is the details of procedure grant_permission which has been called by each of the above four procedures.

```
Procedure grant_permission (x,y,z);
/* This procedure grants permission to node
(x,y,z) either for mutual exclusion entry
(when the request is from the same plane as
node (i,j,k) or for changing the colour of
its token to green (when the request is from
another plane) */
if z = k then do
 token (i,j,k):= YELLOW
 if x = y = 0 then mutex:= TRUE
 else if y = j then send <PERMIT, (x,y,z)> to
 (i + 1,j,k)
  else send <PERMIT, (x,y,z)> to (i,j + 1, k)
 fi fi od
else do
 token (i,j,k):= RED
 if z < k then send <PERMIT (x,y,z)> to
 (i,j,k − 1)
 else send <PERMIT, (x,y,z)> to (i,j,k+1)
 fi od fi
```

## 3. Correctness of the algorithm

Correctness of algorithm DISTRIBUTED_MUTEX is established through the following lemmas.

**Lemma 1.** *Algorithm DISTRIBUTED_MUTEX ensures that a node can be inside its critical section if it is in the privileged plane.*

**Proof.** According to algorithm DISTRIBUTED_MUTEX a node $(i,j,k)$ can enter its critical section if and only if it receives a PERMIT message from node $(0,0,k)$. It can be verified from the algorithm that when node $(0,0,k)$ has a green token then only it can initiate a PERMIT message.

If node $(i,j,k)$'s request is the oldest then the final destination of this PERMIT message would be node $(i,j,k)$. Once node $(0,0,k)$ generates a PERMIT message it changes the colour of its token to yellow thereby ensuring that it would not generate any more PERMIT message until the colour of its token becomes green again. This change of token colour from yellow to green can happen only after node $(i,j,k)$, upon completion of its critical section processing, sends a RELEASE message to node $(0,0,k)$. By definition, a plane $k$ is a privileged plane if node $(0,0,k)$ has either a green token or a yellow token. Therefore, algorithm DISTRIBUTED_MUTEX ensures that a node can be inside its critical section if it is in the privileged plane. □

**Lemma 2.** *Algorithm DISTRIBUTED_MUTEX ensures that there cannot be more than one privileged plane in the TDEM at any time.*

**Proof.** Algorithm DISTRIBUTED_MUTEX initializes the token colour for each node of the TDEM in such a manner that at the beginning of the algorithm plane 0 becomes the privileged plane. Assume that at some point in time plane $k$ has become the privileged plane, i.e., node $(0,0,k)$ has a green token. Now, if a node $(x,y,z)$ in a nonprivileged plane wants to invoke mutual exclusion then it will have to wait until plane $z$ becomes the privileged plane (by Lemma 1). We know that in a nonprivileged plane (say, plane $z$) the token of node $(0,0,z)$ has colour red. Therefore, in order to become privileged, plane $z$ must interchange the token of node $(0,0,z)$ with that of node $(0,0,k)$. For this purpose, node $(0,0,z)$ has to receive a PERMIT message from node $(0,0,k)$ in response to its REQUEST message. Node $(0,0,k)$ upon finding that the REQUEST message received from node $(0,0,z)$ is the oldest among all the pending REQUEST messages in its queue, first changes the colour of its token to red and then only sends a PERMIT message to node $(0,0,z)$. Upon receiving this PERMIT message, node $(0,0,z)$ changes the colour of its token to green. During this transition, i.e., the period after node $(0,0,k)$, changes its token colour to red and node $(0,0,z)$ upon receiving the PERMIT message from node $(0,0,k)$ changes its token colour to green, all the planes of the TDEM remain nonprivileged. This does not violate any of the invariants I1 and I2. Therefore, algorithm DISTRIBUTED_MUTEX ensures that there cannot be more than one privileged plane in the TDEM at any time. □

**Lemma 3.** *Algorithm DISTRIBUTED_MUTEX ensures mutual exclusion.*

**Proof.** On the contrary, assume that two nodes $(i,j,k)$ and $(x,y,z)$ are both in their critical sections at the same time. Now, there are only two cases possible.

*Case 1* $(k=z)$: Both the nodes are in the same plane of the TDEM. By Lemma 1, plane $k$ must be the privileged plane. According to algorithm DISTRIBUTED_MUTEX, a node $(i,j,k)$ can enter into its critical section if it receives a PERMIT message from node $(0,0,k)$. Only one PERMIT message can be generated by node $(0,0,k)$ at a time addressing a specific node whose REQUEST message is the oldest. To ensure that node $(0,0,k)$ does not generate more than one PERMIT message at a time, it changes the colour of its token from green to yellow at the time of generating the PERMIT message and remains yellow until it receives a RELEASE message. The REQUEST messages of node $(i,j,k)$ and node $(x,y,k)$ are stored in the queue of node $(0,0,k)$ as per their arrival order (Assumption A2) and hence it is ensured at node $(0,0,k)$ that the node whose REQUEST is the oldest receives the PERMIT message first. When this node is in its critical section then the other node cannot enter in its critical section, since it has to wait for the PERMIT message. This can happen only after the node that is currently in the critical

section completes its critical section operation and then sends a RELEASE message to node $(0,0,k)$. Node $(0,0,k)$ then can change its token colour from yellow to green and then only proceed to generate a PERMIT message. Hence, node $(i,j,k)$ (or $(x,y,k)$) receives the PERMIT message and can enter its critical section only when no other node of this plane is in the critical section. Therefore, both the nodes of $(i,j,k)$ and $(x,y,z)$ cannot be in their critical sections which contradicts the initial assumption. Hence, the lemma holds if both nodes $(i,j,k)$ and $(x,y,z)$ are in the same plane.

*Case 2* ($k \neq z$): The nodes $(i,j,k)$ and $(x,y,z)$ are in the different planes of the TDEM. By Lemma 1, a node $(i,j,k)$ can be inside its critical section if plane $k$ is the privileged plane. Now, if nodes $(i,j,k)$ and $(x,y,z)$ are both in their critical sections at the same time then both planes $k$ and $z$ are privileged. This means the violation of invariant I1 and Lemma 2 shows that this is not possible. Hence, nodes $(i,j,k)$ and $(x,y,z)$ both cannot remain inside their critical sections at the same time if $k \neq z$ which contradicts the initial assumption about nodes $(i,j,k)$ and $(x,y,z)$.

Combining both, cases 1 and 2, the lemma clearly holds. □

**Lemma 4.** *Algorithm DISTRIBUTED_MUTEX is deadlock-free.*

**Proof.** Assume the contrary, that deadlock is possible. This means that all requesting nodes are unable to proceed to enter into their critical sections because none of them is receiving the PERMIT message. In the first part of the lemma, we prove that a plane of the TDEM trying to become the privileged plane cannot be delayed indefinitely by other planes. Finally, we prove that once a plane becomes privileged, there cannot be a circular waiting among the nodes requesting mutual exclusion in the same plane.

According to algorithm DISTRIBUTED_MUTEX, a node in a nonprivileged plane (say, plane $z$) whose node $(0,0,z)$ has a token of colour red must send a REQUEST message to each node $(0,0,k)$, $k \in \{0,1,\ldots,n^{1/3}-1\}$. Each node $(0,0,k)$, $k \in \{0,1,\ldots,n^{1/3}-1\}$, upon receiving this REQUEST message preserves the identity of the requesting node in its queue. Assume that plane $w$ be the current privileged plane. Now node $(0,0,w)$ in the privileged plane upon receiving RELEASE message from a node $(u,v,w)$ say, finds the oldest REQUEST from its queue and proceeds to send the PERMIT message. Since node $(0,0,z)$ of plane $z$ seeking the token interchange permission sends REQUEST message to node $(0,0,k)$, $\forall k \in \{0,1,\ldots,n^{1/3}-1\}$, within a finite amount of time a privileged plane will find this REQUEST as the oldest and accordingly PERMIT message will be issued granting the token interchange permission to change the token colour of node $(0,0,z)$ to green. Therefore, a plane trying to enter into the privileged state (and hence become the privileged plane) cannot be delayed indefinitely by other planes of the TDEM.

To complete the proof, we now prove that there cannot be a circular waiting among the nodes requesting mutual exclusion in the privileged plane. This follows immediately because in the privileged plane, the PERMIT message is issued to the oldest REQUEST and whenever the PERMIT message reaches the specified node it can enter into its critical section. Since a node cannot remain into its critical section indefinitely, a requesting node will find its REQUEST as the oldest within a finite amount of time and eventually receive the PERMIT message. Therefore, there cannot exist a waiting cycle with respect to the privileged plane and hence the lemma follows. □

**Lemma 5.** *Algorithm DISTRIBUTED_MUTEX is starvation-free.*

**Proof.** The starvation of a node occurs when it is waiting indefinitely for the PERMIT message to enter into its critical section although other nodes are receiving the PERMIT message and performing their critical section operations. From Lemma 4, it is clear that a plane trying to enter into the privileged state cannot be delayed indefinitely by other planes and whenever a plane becomes privileged the PERMIT message is issued to the oldest REQUEST in this plane. Since a node cannot remain into its critical section indefinitely, if a node in the same plane of the starving node receives the PERMIT message then within a finite amount of time the REQUEST of the starving node will be the oldest. Hence the PERMIT message will eventually reach the starving node and it can no longer starve. Therefore, the lemma holds. □

**Theorem 1.** *Algorithm DISTRIBUTED_MUTEX is correct.*

**Proof.** Follows directly from Lemma 3–5. □

## 4. Algorithm complexity

According to algorithm DISTRIBUTED_MUTEX, a node $(i,j,k)$ requesting critical section entry sends a REQUEST message to node $(0,0,k)$ and waits for the PERMIT message from node $(0,0,k)$. Node $(i,j,k)$ upon completing its critical section processing sends the RELEASE message to node $(0,0,k)$. If node $(i,j,k)$ is already in the privileged plane then no more messages, other than those indicated above, will be required for node $(i,j,k)$ to enter into its critical section. In the worst-case, each of these messages will be required to travel a distance of $2(n^{1/3} - 1)$. So, in this case the total number of messages required is $6(n^{1/3} - 1)$ in the worst-case. If node $(i,j,k)$ is in a nonprivileged plane then node $(0,0,k)$ has a red token. Therefore, in order to become the privileged plane, node $(0,0,k)$ sends REQUEST message for token interchange permission to each node $(0,0,z)$, $z \in \{0, 1,\ldots,n^{1/3} - 1\} - \{k\}$. Finally, a node $(0,0,w), w \in \{0,1,\ldots,n^{1/3} - 1\} - \{k\}$, granting the token interchange permission sends the PERMIT message to node $(0,0,k)$. Node $(0,0,k)$ upon receiving the PERMIT message changes the colour of its token to green, while other nodes which have received the same PERMIT message remove the REQUEST of node $(0,0,k)$ for token interchange permission from their queues. Thus, an additional $2(n^{1/3} - 1)$ messages are required, in the worst-case, for node $(i,j,k)$ for entering and completing its critical section processing when plane $k$ is nonprivileged. Therefore, in all $8(n^{1/3} - 1)$ messages are required per mutual exclusion invocation in the worst-case.

**Theorem 2.** *Algorithm DISTRIBUTED_MUTEX requires only $8n^{1/3}$ messages per mutual exclusion invocation, where n is the number of nodes in the network.*

**Proof.** Follows from the above discussion. □

**Theorem 3.** *Under heavy demand algorithm DISTRIBUTED_MUTEX requires only $3n^{1/3}$ messages per mutual exclusion invocation.*

**Proof.** Under heavy demand, we may assume that whenever a plane becomes privileged almost all of its $n^{2/3}$ nodes invoke mutual exclusion (of course, one at a time) before this plane again becomes nonprivileged. A node $(i,j,k)$ in the privileged plane $k$ requires a total $3(i + j)$, $i,j, \in \{0,1,\ldots,n^{1/3} - 1\}$ REQUEST, PERMIT, and RELEASE messages to enter and complete its critical section processing. So, the total number of messages required by all $n^{2/3}$ nodes in plane $k$ to perform their critical section processing is given by $\sum_i \sum_j (i + j) = 3n^{2/3}$

$(n^{1/3} - 1)$, where $0 \leqslant i, j \leqslant n^{1/3} - 1$. As discussed earlier, another $2(n^{1/3} - 1)$ messages are required for plane $k$ to become privileged. So, under this condition, the number of messages required per mutual exclusion invocation is given by $(3n^{2/3} (n^{1/3} - 1) + 2(n^{1/3} - 1)) / n^{2/3} \cong 3 (n^{1/3} - 1)$. This proves the theorem. □

The following theorem provides the upper bounds to the message length and the queue length used by the algorithm.

**Theorem 4.** *To implement algorithm DISTRIBUTED_MUTEX*, (i) *the maximum length of the messages required is bounded from above by* $O(\log n)$ *bits;* (ii) *the maximum space required by a node is bounded from above by* $O(n^{2/3} \log n)$ *bits.*

**Proof.** (i) In algorithm DISTRIBUTED_MUTEX, three types of messages are used. Each of these messages has only the "type" field and the "node id" field. For the "type" field 2 bits are sufficient, while for the "node id" field to specify three integers $(i,j,k)$, each in the range from 0 to $n^{1/3} - 1$, $3 \log n^{1/3}$ bits are required. Therefore, the message length is bounded from above by $(3 \log n^{1/3} + 1)$ bits $\cong O(\log n)$ bits.

(ii) According to algorithm DISTRIBUTED_MUTEX, each of the $n^{1/3}$ nodes with node number $(0,0,k)$, $k \in \{0,1,\ldots,n^{1/3} - 1\}$, requires to maintain a queue to store the node numbers corresponding to the pending REQUEST messages. Each of the $n$ nodes of the network also requires 1 bit for the Boolean variable *mutex*. The maximum possible number of REQUEST messages received by a node $(0,0,k)$, $k \in \{0,1,\ldots,n^{1/3} - 1\}$ is $n^{1/3}(n^{1/3} + 1)$. Therefore, the length of the queue at each node $(0,0,k)$, $k \in \{0,1,\ldots,n^{1/3} - 1\}$, is bounded from above by $3n^{1/3} (n^{1/3} + 1) \log n^{1/3}$ bits. Therefore, to implement algorithm DISTRIBUTED_MUTEX, the maximum space required by a node is bounded from above by $(3n^{1/3} (n^{1/3} + 1) \log n^{1/3}) + 1$ bits $\cong O(n^{2/3} \log n)$ bits. Hence, the theorem follows. □

## 5. Conclusion

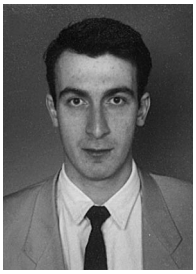
A non probabilistic algorithm is presented that implements mutual exclusion among $n$ nodes in a computer network. In the worst-case, algorithm DISTRIBUTED_MUTEX requires $8n^{1/3}$ messages per mutual exclusion. However, under heavy demand the message requirement reduces to $3n^{1/3}$ only. Thus, our algorithm outperforms all the existing algorithms for distributed mutual exclusion. The algorithm is shown to be free from deadlock as well as from starvation and exhibits fully distributed control. We also claim our algorithm to be symmetrical in the sense that all the $n$ processes residing at $n$ different nodes are identical, i.e., each node $(i,j,k)$, for $i,j,k \in \{0,1,\ldots,n^{1/3} - 1\}$, has an identical copy of the process described by algorithm DISTRIBUTED_MUTEX.

## References

[1] E.G. Coffman Jr., P.J. Denning, Operating Systems Theory, Prentice-Hall, New York, 1973.
[2] P.B. Hansen, Operating System Concepts, Prentice Hall, New York, 1973.
[3] M. Maekawa, A.E. Oldehoeft, R. Oldehoeft, Operating Systems: Advanced Concepts, The Benjamin/Cummings, Menlo Park, CA, 1987.
[4] J.L. Peterson, A. Silberschatz, Operating Systems Concepts, Addson-Wesley, New York, 1986.
[5] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM 21 (1978) 558–565.
[6] G. Ricart, A.K. Agrawala, An optimal algorithm for mutual exclusion in computer networks, Communications of the ACM 24 (1981) 9–27.
[7] O. Carvalho, G. Roucairol, On mutual exclusion in computer networks, Communications of the ACM 26 (1983) 147–148.

[8] G. Ricart, A.K. Agrawala, Author's response to On mutual exclusion in computer networks by Carvalho and Roucairol, Communications of the ACM 26 (1983) 147–148.
[9] I. Suzuki, T. Kasami, A distributed mutual exclusion algorithm, ACM Transactions on Computer Systems 3 (1985) 344–349.
[10] M. Maekawa, A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems, ACM Transactions on Computer Systems 3 (1985) 145–159.
[11] J.M. Helary, N. Plouzeau, M. Raynal, A distributed algorithm for mutual exclusion in an arbitrary network, Computer Journal 31 (1988) 289–295.
[12] K. Raymond, A tree-based algorithm for distributed mutual exclusion, ACM Transactions on Computer Systems 7 (1989) 61–77.
[13] P. Chaudhuri, Optimal algorithm for mutual exclusion in mesh-connected computer networks, Computer Communications 14 (1991) 627–632.
[14] P. Chaudhuri, An algorithm for distributed mutual exclusion, Information and Software Technology 37 (1995) 375–381.

**Pranay Chaudhuri** received his B.Sc.(Hons) and B.Tech. (in Electronics) from Calcutta University. He obtained his M.E. and Ph.D., both in Computer Science and Engineering, from Jadavpur University, Calcutta. At present he is an Associate Professor in the Department of Electrical and Computer Engineering, Kuwait University. Dr. Chaudhuri has held faculty positions at the Indian Institute of Technology, Kharagpur, and the University of New South Wales, Australia, prior to his joining the Kuwait University in February, 1993. Dr. Chaudhuri's research interests include Parallel and Distributed Algorithms, Graph Theory, and Distributed O/S. He is the author of over 40 research papers in these areas and a book entitled, Parallel Algorithms: Design and Analysis (Prentice Hall, 1992). Dr. Chaudhuri is the recipient of several international awards for his research contribution.



**Mehmet Hakan Karaata** received his Ph.D. degree in Computer Science in 1995 from the University of Iowa. He joined Bilkent University, Ankara, Turkey, as an Assistant Professor in 1995. He is currently working as an Assistant Professor in the Department of Electrical and Computer Engineering, Kuwait University. His research interests include distributed systems, fault tolerant computing, self stabilization and algorithm complexity.