Distributed algorithms for mobile hosts

L.M. Patnaik
A.K. Ramakrishna
R. Muralidharan

Indexing terms: Mobile environment, Physical movement, Computer communications, Proxies, Static hosts

Abstract: In a mobile environment the physical movement of hosts causes changes in the physical topology of the network with time. Therefore, the direct execution of the existing distributed algorithms in a mobile environment renders them inefficient. Distributed algorithms are therefore modified by assigning proxies, which are static hosts in the network, to each of the mobile hosts. Proxies have been used at a level $d (d \geq 1)$ in the logical hierarchy of location servers. A mutual exclusion algorithm has been redesigned, and the results of simulations of the original and modified versions are presented. General purpose procedures have been developed which can be used in any distributed algorithm designed for a mobile environment and which uses static hosts as proxies.

1 Introduction

As computer networks such as the Internet have spread worldwide and computers have become smaller and smaller, a new environment has been created in which users carry portable computers and use them on the move to access networks. This is called 'mobile computing', which of late has become a buzzword in the area of computer communications.

Since mobile computers might not remain at a fixed location, a fixed wired physical connection cannot be maintained between a mobile computer and a communication network. So, communication from and to a mobile computer takes place through a wireless medium in the form of wireless messages.

Because of the mobility of hosts, the geographical distribution of mobile hosts keeps changing. This mobility of hosts makes distributed algorithms, which were designed assuming static hosts, of hosts, inefficient in a mobile environment. Thus, distributed protocols that run on mobile hosts might have to be modified so that they can be executed efficiently in a mobile environment using the available resources.

The only work that is directly related to redesigning distributed algorithms for mobile environments was reported by Badrinath *et al.* [1] . They raised some of the issues resulting from host mobility, and use level one proxies in the logical hierarchy of location servers to modify Lamport's mutual exclusion algorithm [2] and Lann's algorithm for mutual exclusion [3]. Badrinath *et al* discuss the creation of a logical hierarchy of location servers but use only level one proxies for modifying distributed algorithms.

If we use higher level proxies in the logical hierarchy of location servers, we expect that a reduced number of messages will be sent regarding the setting up of proxies compared to when lower level proxies are used. Thus, in our work we have used proxies at a level $d (d \geq 1)$ in the logical hierarchy of location servers. We have redesigned Raymond's tree-based algorithm for distributed mutual exclusion [4], and have simulated the modified algorithm and compared it with the original one in terms of communication costs incurred per critical section entry, average messages required per critical section entry and average response time.

2 System model and methodology

The entire geographical area is divided into wireless cells. There is a base station, called a hub or mobile support station (MSS) for each cell. These base stations are static. Each mobile host (MH) is part of one and only one cell at a time.

The hub is connected to an existing wired backbone transport network on one side and on the other side has a hub antenna with which it communicates, using radio waves or infrared waves, with the mobile hosts in its cell. Value added networks are connected to this wired backbone transport network. Other static hosts are also connected onto the wired network.

2.1 Message passing, terminology and assumptions

2.1.1 *Message passing:* An MH, to send a message to another host (mobile or static), first sends it through the wireless medium to the mobile support station. The MSS takes care of routing the message to its destination. Since the destination MH can be moving, locating the mobile host is a problem. Some routing protocols *[5, 6]* have been designed to address this problem.

2.1.2 *Terminology:* From the topology given above, we can see that the number of MHs, N, is far greater than the number of hubs (or MSSs), M, i.e. $N >> M$.

To make sure that performance measures are not dependent on the routing protocol used, a constant fixed search cost is assumed to be incurred whenever a

search for the MSS in whose cell the destination MH is present is done.

We use the terminology given in [1] for the purpose of computing the communication costs incurred per each critical section entry. The terminology used is as follows:

C_F: cost of sending a point-to-point message between any two fixed hosts

C_W: cost of sending a wireless message from an MH to its local MSS (and vice versa)

C_S: cost incurred in locating an MH and forwarding a message to its current local MSS, from a source MSS.

2.1.3 Assumptions: Some of the important assumptions made about the network are:

(a) The network is assumed to be fully connected.

(b) In a wireless cell, messages will be received in the order sent.

(c) An MH that leaves a cell will eventually enter some cell in the system after some time.

Each MSS has a list of identifiers of mobile hosts that are currently in its cell. This list is correspondingly modified when an MH sends a MOVING or ENTERED message to a relevant MSS.

Disconnection of an MH is handled similarly to MH switching cells. When an MH disconnects, there is **no** guarantee that it will eventually reconnect. This is not the case when an MH is moving off from a cell. An MH disconnects by sending a DISCONNECT message and can reconnect by sending a RECONNECT message.

2.2 Methodology

A logical hierarchy of location servers (LHLSs) in the form of a tree is created on the network to which hosts are connected. Location servers act as proxies for mobile hosts during execution of the distributed algorithm.

Whenever a logical structure has to be imposed on the mobile hosts, we instead impose the logical structure **only** on the proxies which are at level d in the LHLS. Each distributed algorithm is designed to work for a particular d level but can be modified to work with different d level proxies by imposing the logical structure on a set of proxies at the different d levels.

Since a proxy for an MH is at level d in the LHLS, whenever an MH moves into an an area with a different d level proxy, a new d level proxy will be assigned to the MH. The relationship between a MH and a d level proxy remains invariant otherwise. When $d = 1$ the mobile support stations act as proxies for mobile hosts. So, the association between an MH and its proxy keeps changing whenever the MH moves into a new cell. When d = maximum levels in the LHLS, there will be only one static host which acts as a proxy for all the MHs in the network.

3 Tree-based algorithm for distributed mutual exclusion

In this Section we modify the tree-based algorithm for distributed mutual exclusion [4]. We use the system model described in Section 2 and give general purpose procedures and general purpose data structures which can be used in **any** distributed algorithm that is designed using proxies. The proxies used for modified algorithm at a level d in the LHLS.

Informal description of algorithm

This algorithm is token based, with only the holder of the token being able to execute critical section code.

The nodes in the distributed system are arranged in an unrooted tree structure. All messages are sent along the undirected edges of the tree. The tree may either be a minimal spanning tree of the actual network topology, or merely a logical structure imposed on the network assumed. Each node knows only of the existence of its neighbours. Every time the token passes between nodes, the tree becomes directed with the new holder of the token being the root. Each edge is made directed and points towards the root. Each node maintains a requests queue into which requests from the neighbouring nodes and requests from itself are queued. When a node obtains privilege it is sent to the first node in its request queue.

3.1 Modified algorithm

In this algorithm a STATIC is assigned as a proxy for each MOBILE host.The proxy assigned is at level d ($d \geq 1$). The proxy for an MH remains the same as long as the MH moves between locations having the same d level proxy. Every MSS keeps information about the d level proxy in the logical hierarchy of location servers (which is at $d - 1$ levels above the current MSS).

In the original algorithm [4], a node needs to have PRIVILEGE (token) to execute the critical section. A REQUEST message indicates that the sender is nonprivileged and wants PRIVILEGE (in order to execute the critical section) either for itself or one of its immediate neighbours. A PRIVILEGE message indicates the transfer of PRIVILEGE from the sender to receiver.

One node is chosen as the initial holder of PRIVILEGE. A variable holder at node i indicates the holder of PRIVILEGE according to node i. The initial holder of PRIVILEGE informs all the other nodes that it is holding the PRIVILEGE, by sending an INITIALISE message to all its neighbouring nodes. When node i receives an INITIALISE message from the node j, holder *(i)* is set to j and the INITIALISE message is sent to all its neighbours other thanj.

In the modified algorithm we introduce three new messages in addition to the INITIALISE, REQUEST and PRIVILEGE messages. The new messages introduced are:

(a) PASSPRIV(i). This message, sent by node i, indicates something similar to granting of PRIVILEGE, when the receiving node is MOBILE, but is not exactly the same. If the receiving node is STATIC, this means that PRIVILEGE is passed back from an MH to its proxy.

(b) MYPROXY (i, CurrProxy, d). When an MH, MH_i moves from one cell to another, it sends a MYPROXY(i, CurrProxy, d) message to the MSS of the new cell after sending an ENTERED message. By sending this message, MH_I indicates to the MSS that CurrProxy is the proxy of MH_I at level d.

(c) SETPROXY (SavedProxy). When an MH, MH_I, sends a MYPROXY (i, Currproxy, d) message to an MSS, the MSS checks to see if CurrProxy is same as the proxy at level d for this MSS. If not, the MSS sends a SETPROXY (SavedProxy) message to MH_I. Here, SavedProxy is the proxy at level d for this MSS. The

MH is expected from then on to communicative with SavedProxy regarding its distributed algorithm.

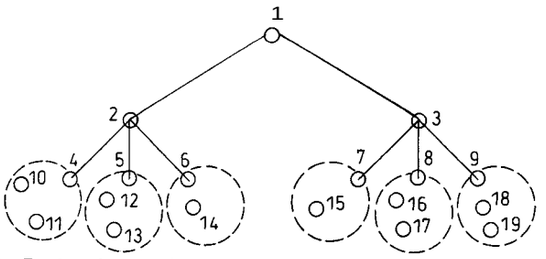


Fig. 1 *Sample network*

Fig. 1 gives a sample network structure. Nodes 10, 11, 12, 13, 14, 15, 16, 17, 18 and 19 are mobile and others are static. Dotted circles indicate wireless cells. If the LHLS imposed on the network has nodes 4, 5, 6, 7, 8 and 9 at level one, nodes 2 and **3** at level two, and node 1 at level three, then:

In the original algorithm the tree is imposed among nodes 10, 11, 12, 13, 14, 15, 16, 17, 18 and 19.

But in the modified algorithm,

if $d = 1$, it is imposed among nodes 4, 5, 6, 7, 8 and 9

if d = 2, it is imposed among nodes 2 and 3 only

if d = 3, it is imposed among node 1 (only one node is there in the tree).

There are many different trees that are possible for a given set of nodes. Which tree to be imposed is left as a choice.

3.2 General purpose data structures for modified distributed algorithms

```
At each Node (MOBILE or STATIC):
  NodeType : [MOBILE, STATIC]
At each Mobile Host:
  struct DAlgoInfo
  {
    dLevel: integer;
    dLevelProxy: integer;
  3
At each Mobile Support Station:
  struct ProxyInfo
  {
    ProxyLevel: integer;
    StoredProxy: integer;
  3
  struct ProxyInfo * ProxyInfoQ;
Message Types: MOVING, ENTERED, SETPROXY, MYPROXY
```

3.3 General purpose procedures for modified distributed algorithms

```
Procedure InitStoredProxy(i : integer)
{
  for d = 1 to MaxLevels in LHLS
  {
    ProxyInfo.dLevelProxy = Computeddlevelproxy
                  using the LHLS being used for
                  the Distributed Algorithm
    ProxyInfo.dLevel = d;
    file this ProxyInfo in ProxyInfoQ(i);
  3
3

Procedure InitDAlgoInfo(i, dLevel: integer)
{
  DAlgoInfo(i).dLevelProxy = Computed d level proxy
                  using the LHLS being used for
                  the Distributed Algorithm
  DAlgoInfo(i).dLevel = dLevel;
3
Procedure InitGeneral (d: integer)
{
  InitNodeTypes();
  for each Mobile Host, MHi
    InitDAlgoInfo(MHi, d);
  for each Mobile Support Station, MSSj
    InitStoredProxy (MSSj);
3
At a MSS, MSSi:
  Upon receipt of MYPROXY(MHj, MSSi, Proxy, dLev)
                  message
  {
    Get ProxyInfo x from ProxyInfoQ(MSSi) with
                  x.ProxyLevel = dLev
    SavedProxy = x.StoredProxy;;
    If (SavedProxy <> Proxy)
      Send a INITHANDOVER(MHj, Proxy) message to
                  SavedProxy;
  3
At a Proxy, Proxyi:
  Uponreceipt of INITHANDOVER(MHj, Proxy) message
  {
    Send SETPROXY(Proxyi) message to MHj;
    Send HANDMEOVER(MHj, Proxyi) message to Proxy;
  3
  Upon receipt of HANDMEOVER(MHj, k) message
  {
    Send Data Structures held for MHj to k;
  3
At a MH, MHi:
  If Movingoff fromacellj
  {
    Send a MOVING(MHi) message to cell j's MSS,
                  MSSj;
  3
  If enteredanewcellj
  {
    Send a ENTERED(MHi) message to cell j's MSS,
                  MSSj;
    proxy = DAlgoInfo.dLevelProxy;
    dLev - DAlgoInfo.dLevel;
    Send a MYPROXY(MHi, proxy, dLev) message to
                  MSSj;
  }
  Upon receipt of SETPROXY(SavedProxy) message
  {
    DAlgoInfo.dLevelProxy = SavedProxy;
  3
```

3.4 Algorithm-specific data structures used

At each d level proxy node, Proxyi, which is at $d - 1$ levels above the MSS_i in the network:

Holder. Contains the number of one of its immediate neighbours in the tree or Proxyi. This gives the relative position of the proxy node which is holding PRIVILEGE or has given the PASSPRIV to one of the

mobile hosts, for which it is currently acting as proxy.

Using. This variable indicates whether or not a PASSPRIV has been passed on to one of the MHs the proxy is supporting. This is equal to TRUE even when a MH upon finishing its critical section passes back the PASSPRIV message and the message is currently in transit.

ReqQ. This is a queue containing identifiers of the neighbouring STATIC nodes, or MHs for whom this node is acting as a proxy, that sent a REQUEST message to Proxyi but haven't been sent the PRIVILEGE or PASSPRIV messages in reply. If the REQUEST message comes from one of the MHs for whom Proxyi is acting as a proxy, then depending on whether a REQUEST has already been made, its own identifier might be put in the ReqQ.

Asked. A Boolean, is TRUE if node Proxyi does not have a privilege and also it did not send a PASSPRIV to one of the MHs for whom it is acting as proxy and had sent a REQUEST message to holder; otherwise it is FALSE.

3.5 Algorithm-specific procedures

```
Procedure AssignPriv_Mod(Proxyi: integer)
{
  if ( (Holder(Proxyi) = Proxyi) ^ (Using(Proxyi) =
                  FALSE) ^ (ReqQ(Proxyi) <>
                  Empty))

    Asked(Proxyi) =FALSE;
    ReqNode = dequeue (ReqQ(Proxyi));
    if (NodeType(ReqNode) = MOBILE)
    {
      Using(i) = TRUE;
      Send a PASSPRIV(Proxyi) message to ReqNode;
    }
    else
    {
      Holder(Proxyi) = ReqNode;
      Send PRIVILEGE message to ReqNode;
    }
  }
}
Procedure ProcessPassPriv(i, Source: integer)
{
  if (NodeType(i) = MOBILE)
  {
    <Execute Critical Section>
    Send PASSPRIV(i) to Source;
  }
  else
  {
    Using(i) =FALSE;
    AssignPriv_Mod(i)
    MakeRequest(i);
  }
3
Procedure InitialiseHolder
{
  Choose one of the proxy node Proxyi;
  Holder (Proxyi) = Proxyi;
  for each neighbor Proxyj of Proxyi in the tree
                  imposed
    Send INITIALISE(Proxyi) message to Proxyj;
}
```

```
At a MM, MHi:
  If decide to execute Critical section
  {
    proxy = DAlgoInfo.dLevelProxy;
    Send a REQUEST(MHi) messagetoproxy;
    wait (PASSPRIV(Source) message);
    ProcessPassPriv(MHi, Source);
  }
At a Proxy, Proxyi:
  Upon receipt of REQUEST(k) message
  {
    enqueue (ReqQ(Proxyi), k);
    AssignPriv_Mod(Proxyi);
    MakeRequest (Proxyi);
  }
  Uponreceipt of PRIVILEGE message
  {
    Holder(Proxyi) = Proxyi;
    AssignPriv_Mod(Proxyi);
    MakeRequest (Proxyi);
  3
  Upon receipt of PASSPRIV(k) message
  {
    ProcessPassPriv(Proxyi, k);
  3
  Uponreceipt of INITIALISE(Proxyj) message
  {
    Holder(Proxyi) = Proxyj;
    for each neighbor Proxyk of Proxyi in the tree
                  imposed
      if (Proxyk <> Proxyj)
        Send INITIALISE(Proxyi) message to Proxyk;
  }
```

The algorithm is started by calling the InitialiseHolder procedure at one of the Proxy nodes, $Proxy_i$.

3.6 Proof

First, the assumption that all messages are received reliably and eventually should be noted. This assumption is generally made in all distributed algorithms.

Theorem 1. Mutual exclusion is assured at all times.

Proof. To prove that mutual exclusion is assured, we will show that at any point of time, at most one host, mobile or static is executing in the critical section. From the algorithm, we can see that holding PRIVILEGE or PASSPRIV is a prerequisite for executing in the critical section. Calling a host holding PRIVILEGE or PASSPRIV, a privileged host, initially only one host is privileged. Afterwards, only a privileged host can send PRIVILEGE or PASSPRIV to another host. Once a privileged host sends a PRIVILEGE or PASSPRIV to another host it immediately becomes nonprivileged. So, at any point of time at most one host is executing in the critical section and hence mutual exclusion is assured.

Theorem 2. This algorithm is deadlock free

Proof. Deadlock can occur if (1) PRIVILEGE or PASSPRIV is lost, or does not eventually reach the requested node and there are one or more nodes wishing to enter the critical section, or (2) a privileged node is not aware that other nodes require the privilege.

Because a request queue is maintained at each proxy node and the tree is imposed on the proxy nodes, an

ordering of all the requests that are generated in the system is maintained at all times. This, combined with the collective asked variables, ensures that there is a sequence of REQUEST messages, for which no PRIVILEGE or PASSPRIV message has been received in reply, between each node requiring the privilege and the privileged node, using the collective holder variables to route these REQUEST messages towards the privileged node. So condition (2) cannot occur, assuming that messages are received at the destination reliably and eventually.

Further, because of the assumption about the reliability and eventuality of reception of messages, part of condition (1) stating that 'PRIVILEGE or PASSPRIV is lost' does not occur.

Because once a REQUEST message is sent from a proxy node, it cannot be sent again until it receives PRIVILEGE, uses or sends PASSPRIV to an MH and gets it back, and then sends it to some other proxy node (because of the asked variable), a PRIVILEGE message destined for a node j will be received at j eventually. Also a PASSPRIV message sent by a proxy node to an MH or vice versa is not lost (because of reliable message transmission). So part of condition (1) stating that a PRIVILEGE or PASSPRIV message does not eventually reach the requested node does not hold. So, no deadlocks are possible in the modified algorithm.

Theorem 3. This algorithm is starvation free.

Proof. First, an ordering of requests takes place as described in the proof for impossibility of deadlock. Secondly, PRIVILEGE or PASSPRIV is sent according to the order of requests queued up in the Request queue at proxy nodes, and an MH that receives a PASSPRIV from a proxy node k sends back PASSPRIV to k (and not to the current proxy, which could be different from the old proxy because the MH could have moved in such a way that it is now in an area with a different d level proxy from the one it had when it received the PASSPRIV message), after it finishes executing in critical section. So, no starvation can take place.

3.7 Communication costs

For a general $d > 1$), it is not possible to compute the communication costs unless a simulation is performed. But for $d = 1$, it is possible to compute the communication costs. So we take $d = 1$, and compute the costs incurred by both the original algorithm and the modified algorithm and compare them with respect to the communication costs incurred.

When $d = 1$ the proxies are the MSSs for each cell. Every time an MH goes into a different cell, the MSS for the new cell is different, and so a new proxy (which will be the new cell's MSS) is assigned to the MH.

As stated earlier, the number of MHs is very much greater than the number of MSSs (i.e. $N >> M$). Also $C_S > C_F$. When a message has to go from one MH to another MH, the communication cost incurred is 2 C_W + C_S, one C_W for the cost incurred for the message to go from MH_i to the MSS of the cell in which MH_i is currently residing, and C_S for locating MH_j. Later a C_W is incurred for the message to go from the MSS in which MH_j is currently residing to MH_j.

The performance of the algorithms, original and modified, depends on the structure of the tree imposed. We will compute communication costs incurred for the worst case, best case, and average case, that can occur. The performance measure chosen is the cost incurred by an MH for gaining access to a critical section.

Let TC_O represent the total cost incurred by original algorithm, TC_M the total cost incurred by the modified algorithm, C_{req} the cost of request messages, and C_{Priv}, the cost of privilege messages. The total cost incurred will be sum of C_{req} and C_{Priv}.

Worst case analysis

The worst case performance occurs when the tree imposed is a straight line and the privilege message keeps shuttling between the ends of straight line.

Cost of original algorithm. Since the straight line segment will have N nodes (all mobile), the path length between the ends of the line segment is $N - 1$.

$$C_{\mathrm{Req}} = (N-1)(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (1)$$

$$C_{\mathrm{Priv}} = (N-1)(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (2)$$

Thus

$$TC_{\mathrm{O}} = 2(N-1)(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (3)$$

Cost of modijiied Algorithm

$$C_{\mathrm{Req}} = C_{\mathrm{W}} + (M-1)C_{\mathrm{F}} \quad (4)$$

$$C_{\mathrm{Priv}} = C_{\mathrm{W}} + C_{\mathrm{F}} + (M-1)C_{\mathrm{F}} + C_{\mathrm{S}} + C_{\mathrm{W}} \quad (5)$$

Thus

$$TC_{\mathrm{M}} = 3C_{\mathrm{W}} + (2M-1)C_{\mathrm{F}} + C_{\mathrm{S}} \quad (6)$$

Average case in a straight line topology

This occurs when each node is equally likely to request the privilege. So, the the average distance between requesting node and privileged node is $(N + 1)/3$.

Cost of original algorithm

$$C_{\mathrm{Req}} = ((N-2)/3)(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (7)$$

$$C_{\mathrm{Priv}} = ((N-2)/3)(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (8)$$

$$TC_{\mathrm{O}} = 2((N-2)/3)(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (9)$$

Cost of modified algorithm

$$C_{\mathrm{Req}} = C_{\mathrm{W}} + ((M-2)/3)C_{\mathrm{F}} \quad (10)$$

$$C_{\mathrm{Priv}} = C_{\mathrm{W}} + C_{\mathrm{F}} + ((M-2)/3)C_{\mathrm{F}} + C_{\mathrm{S}} + C_{\mathrm{W}} \quad (11)$$

$$TC_{\mathrm{M}} = 3C_{\mathrm{W}} + ((2M-1)/3)C_{\mathrm{F}} + C_{\mathrm{S}} \quad (12)$$

Best case

This occurs when the tree has a star structure.

Cost of original algorithm

$$C_{\mathrm{Req}} = 2(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (13)$$

$$C_{\mathrm{Priv}} = 2(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (14)$$

$$TC_{\mathrm{O}} = 4(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (15)$$

Cost of modijiied algorithm

$$C_{\mathrm{Req}} = C_{\mathrm{W}} + C_{\mathrm{F}} \quad (16)$$

$$C_{\mathrm{Priv}} = C_{\mathrm{W}} + C_{\mathrm{F}} + C_{\mathrm{F}} + C_{\mathrm{S}} + C_{\mathrm{W}} \quad (17)$$

$$TC_{\mathrm{M}} = 3C_{\mathrm{W}} + 3C_{\mathrm{F}} + C_{\mathrm{S}} \quad (18)$$

Average case

We can see that the average diameter in the original algorithm, D_o is

$$D_{\mathrm{O}} = O(\log N)$$

The average diameter in the modified algorithm, D_M is

$$D_{\mathrm{M}} = O(\log M)$$

So in the worst case, the privilege shuttles between the ends of the arbitrarily chosen tree. So

$$TC_{\mathrm{O}} = 2(k_1 \log(N) + k2)(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (19)$$

and

$$TC_{\mathrm{M}} = 3C_{\mathrm{W}} + (2(k_1 \log(M) + k2) + 1)C_{\mathrm{F}} + C_{\mathrm{S}} \quad (20)$$

Performance under heavy demand

$$TC_{\mathrm{O}} = (2 \cdot 2(2C_{\mathrm{W}} + C_{\mathrm{S}})(N-1))/N = 4(2C_{\mathrm{W}} + C_{\mathrm{S}}) \quad (21)$$

$$TC_{\mathrm{M}} = (NC_{\mathrm{W}} + N(C_{\mathrm{S}} + C_{\mathrm{W}}) + N(C_{\mathrm{W}} + C_{\mathrm{F}}) + 4(M-1)C_{\mathrm{F}})/N \quad (22)$$

$$= 3C_{\mathrm{W}} + C_{\mathrm{S}} + (4(M-1)/N + 1)C_{\mathrm{F}} \quad (23)$$

3.8 Comparison of original and modified algorithms

3.8.1 Worst case: The original algorithm incurs a search cost which is proportional to N, while the modified algorithm incurs a search cost of C_S, which is a constant independent of N. Also, wireless message cost in the modified algorithm is $3C_W$ which is a constant, but in original algorithm it is proportional to N. Since $C_S > C_F$ and $N >> M$, $TC_M << TC_O$.

Some of the problems that the original algorithm faces and which do not occur in the modified algorithm are:

(I) If any of the MHs disconnect, then the tree structure might have to be restructured (unless the disconnected MH was a leaf in the tree). If the disconnected MH holds a token with it, then deadlock occurs in the original algorithm. This is not the case in the modified algorithm. In the modified algorithm, if the MH is holding privilege and disconnects then the proxy for the MH after waiting for some time (if it does not get back the privilege) regenerates a privilege and then starts using it.

(2) In the original algorithm, an MH that is operating in doze mode has to be interrupted if it receives requests or privilege from neighbouring MHs in the tree, irrespective of whether it is interested in the distributed algorithm or not.This case does not occur in the modified algorithm where an MH is interrupted only when it has requested entry into the critical section and has not yet been granted privilege.

(3) In the original algorithm, all the MHs which are up (powered on) have to be involved in the distributed algorithm all the time. Request and privilege messages have to be transmitted by the MHs to the holders and to the MHs which are in front of their request queue, irrespective of own interest in the distributed algorithm. These request and privilege messages consume battery power. This is not the case in the modified algorithm where battery power is consumed only when an MH is interested in participating in the algorithm. The power consumed is for it to send two messages, request and privilege, and to receive one privilege message.

Average case

This case is similar to the 'worst case', except that instead of N in TC_O we have $(k_1 \log(N) + k_2)$ and instead of M in TC_M, we have $(k_1 \log(M) + k_2)$. Since $\log(M) < \log(N)$ for $N >> M > 1$, $TC_M < TC_O$.

So, the modified algorithm is better than the original algorithm for the reasons explained in the earlier case.

Best case and heavy demand case

These two cases are similar to the 'worst case' and similar reasoning can be used to show that, $TC_O > TC_M$ and hence the modified algorithm is better than the original algorithm.

4 Simulation experiments and results

The initial physical network that is used during simulation is given in Fig. 2. Dotted circles are wireless cells.

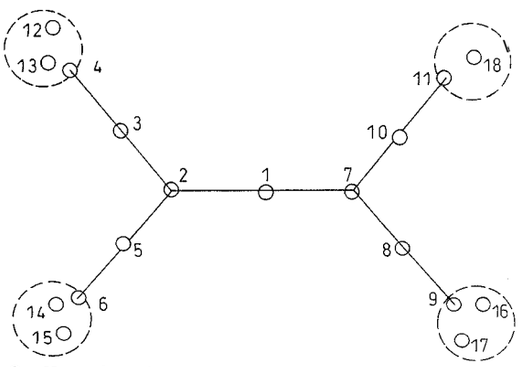


Fig.2 *Network used*

In Fig. 2, nodes 1 – 11 are static and nodes 12 – 18 are mobile. Nodes 4, 6, 9, and 11 act as mobile support. stations. The LHLS imposed on the network has a tree structure and is the same as the network shown in Fig. 3 but with only static hosts. Nodes 4, 6, 9 and 11 form the leaves of the hierarchy, i.e. they are at level one in the LHLS. Nodes 3, 5, 9 and 10 are at level two, nodes 2 and 7 are at level three, and finally node 1 is at level four in the LHLS.

The modified algorithms simulated have proxies which are at level three in the LHLS. So, in the modified version of [4], we impose a tree structure on nodes 2 and 7 only.

We compute: (1) average number of messages required per critical section entry; (2) average cost of communication per critical section entry; and (3) average response time (response time is the time interval between the instant a request for entry into the critical section is made and the instant permission to enter the critical section is received) as a function of mean time of MHs movement (will also be called mean time to move, and Narrtime) and also as a function of mean arrival rate of requests (lambda) for entry into critical section.

The details of the algorithm are given in Section 3. We present below only the results.

4.7 Results

We can see very clearly from Fig. 3, that the average number of messages required per critical section entry for the modified algorithm is lower than of the original algorithm for any lambda. Similarly, from Figs. 4 and 5, we can see clearly that the modified algorithm is better than the original algorithm in terms of communication costs and average response time for all values of lambda.

We can see very clearly from Fig. 6, that the average messages required per critical section entry for the modified algorithm is lower than that of the original algorithm for any mean node movement time. Similarly, from Figs. 7 and 8, we can see clearly that the modified algorithm is better than the original algorithm in terms of communication costs and average response

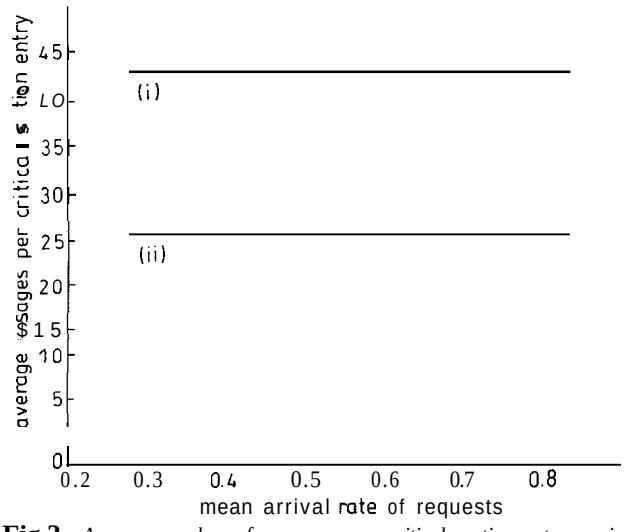


Fig.3 *Average number of messages per critical section entry against mean arrival rate of requests for tree-based algorithm*
(i) original algorithm
(ii) modified algorithm
Mean time to move = 100

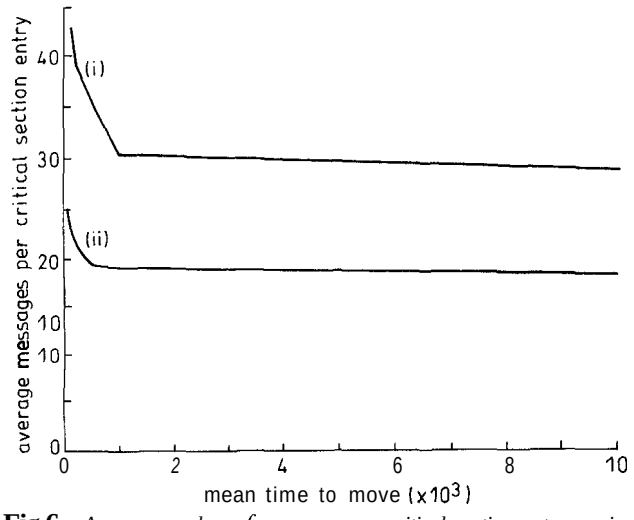
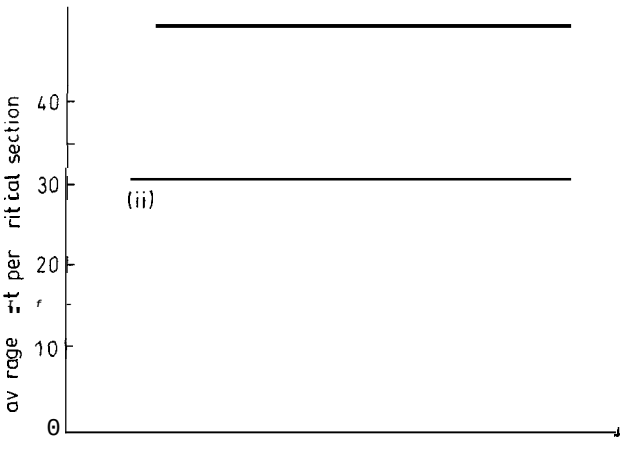
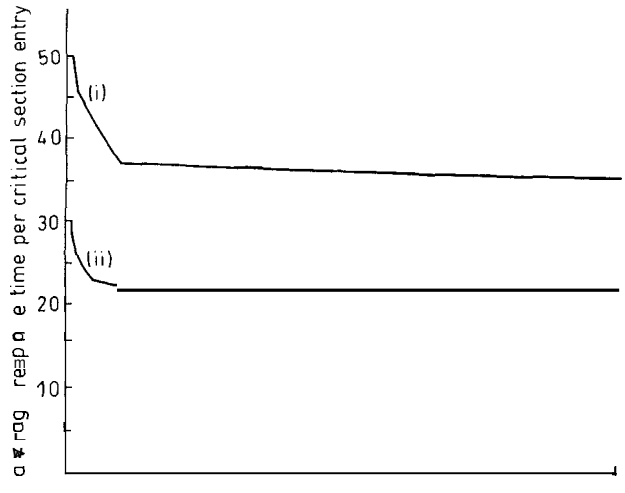


Fig.6 *Average number of messages per critical section entry against mean time to move for tree-based algorithm*
(i) original algorithm
(ii) modified algorithm
Mean arrival rate of requests 0.63





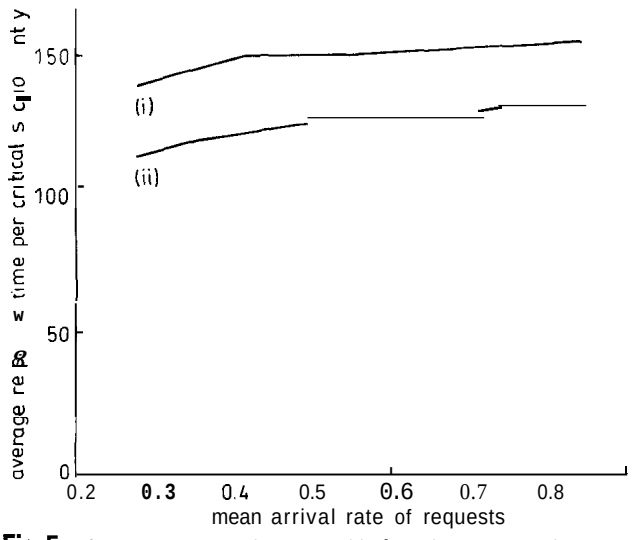
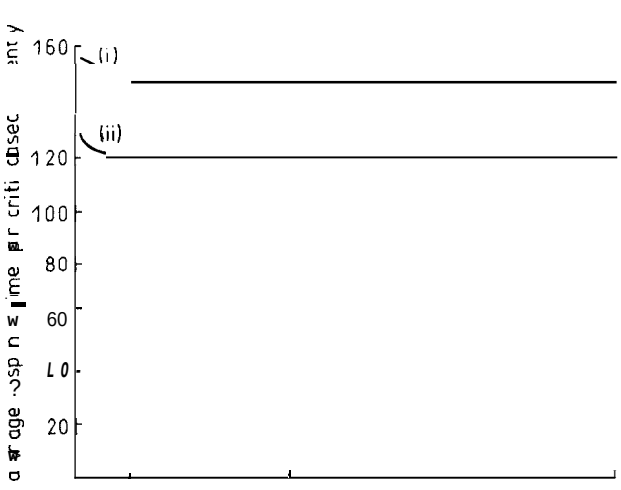


Fig.5 *Average response time per critical section entry against mean arrival rate of requests for tree-based algorithm*
(i) original algorithm
(ii) modified algorithm
Mean time to move = 100



time for all values of mean time to move. As node movement time increases, it can be seen that all the performance measures for both the original and modified algorithms improve. This is to be expected since the increase in node movement time means reduced mobility of MHs and we know that a distributed algorithm executed only by static hosts is better than one executed by some mobile hosts. If mean time to move equals infinity, all MHs are static.

5 Conclusion

In our work, we have modified the tree-based algorithm for distributed mutual exclusion Fig. 4, so that the modified version executes efficiently in a mobile environment. Also, our modified algorithms do not face the problems faced by the original versions such as limited life of battery, disconnections of MHs and doze mode operation of MHs. We have used logical hierarchy of location servers to assign proxies to MHs in the modified version.

In terms of the communication costs incurred per critical section entry, we have shown theoretically that the modified version is better than the original version of the algorithm, where proxies assigned are at level one. We have simulated the original and modified versions of this algorithm, where proxies assigned are at level three. We have computed the performance of these versions in terms of average messages per critical section entry, average cost incurred per critical section entry, and average response time. The above performance measures were computed by varying mean movement time of MHs (also called mean time to move) and mean arrival rate of requests. In all cases the modified algorithm was shown to be better than the original algorithm.

There are a number of distributed algorithms which can be modified by imposing an LHLS and assigning proxies for MHs from among the nodes in the LHLS. The general purpose procedures and data structures developed can be used in all these modified versions of distributed algorithms. One can use different costs for a message transmission on a communication link, depending on the link on which it is sent, and work out the communication costs incurred, average number of messages required, and response time per critical section entry, for the modified algorithms. The distributed algorithms can be modified to work in an environment where MHs arbitrarily disconnect from the network or suddenly go into doze mode operation.

6 Acknowledgments

The authors would like to thank Professor Dagless and the two anonymous referees for their suggestions.

7 References

1 BADRINATH, B.R., ACHARYA, A., and IMIELINSKI, T.: 'Structuring distributed algorithms for mobile hosts'. Proc. 14th Int. Conf. on *Distributed computing systems*, June 1994, pp. 1–12
2 LAMPORT, L.: 'Time, clocks, and the ordering of events in a distributed system', *Commun. ACM,* 1978, **21,** (7), pp. 558–565
3 LANN, G.LE.: 'Distributed systems, towards a formal approach'. IFIP Congress, Toronto, 1977
4 RAYMOND, K.: 'A tree-based algorithm for distributed mutual exclusion', *ACM Trans. Comput. Syst.,* 1989, **7,** (1), pp. 61–77
5 TERAOKA. F.. UEHARA. K.. SUNAHARA. H., and MURAI, J.:' 'VIP: A protocol providing host mobility', *Commun. ACM*,, **37,** (8), pp. 67–113
6 IOANNIDIS. J., DUCHAMP, D., and MAGUIRE, G.Q.: 'IP-based protocols 'for mobile internetworking'. Proc. SIGCOMM, September 1991, Vol. 21, pp. 235–245 (no. 4)