5.1 Introduction

Mobile Computing Environment (MCE) essentially provides a Distributed Computing Environment (DCE) wherein some of the computers communicate wirelessly and have low capabilities compared to other nodes. To appericiate the differences and similarities between MCE and DCE, let us first focus on distributed computing environment.

Loosely speaking, a distributed algorithm partitions a problem into smaller parts for execution on two or more computers, referred to as nodes, that communicate over a network. The nodes execute local algorithms, and are unaware of global state. To be aware of global state they exchange local state information periodically over the network through messaging. Three fundamental problems in this settings are communication, resource allocation, and synchronization.

Algorithms for sending information from one node to other are collectively referred to as communication protocols. These protocols accomplish two important tasks, viz., (i) broadcasting data or local state information from one node to all others, or (ii) route (send) data/information about local state from one node to another. The resource allocation algorithms can be viewed in a more abstract form as a serialization technique, for example, the access of a critical section by a set of competing processes. The basic question here is: who among a set competing entities should access the common resource. This problem is popularly known as mutual exclusion problem. It permit or provides a ticket to an entity for performing certain task in isolation from other competing entities. Another important problem that arise in distributed settings is to determine whether a computation has terminated or not. Termination is a stable property of the global state of a computation, because when termination is detected computation ceases. While there is there is not much of a paradigm shift in design of algorithm for mobile computing environment, certain issues typical to implementation do arise.

So, our plan in this chapter is to first examine the message complexity when distributed algorithms are executed directly on mobile distributed environment in a straightforward manner. Then we examine how structuring the algorithms for execution in mobile distributed environment could help us to reduce the message complexity. The focus here is mainly in minimizing exchange of messages on wireless channel by localizing computational task to static hosts as much as possible.

5.2 Mobile Computing Environment

Some of the basic problems encountered in distributed algorithms for execution in mobile environment are in synchronization of computation on fixed network. These issues are dependent on the abstract notion of coordination and control in distributed algorithms. As an analogy, consider the issue of quality in software design process. Efficiency of algorithms addresses the quality in the problem domain. But when algorithms are converted to software processes, the issue of quality no longer remains in the problem domain alone; it becomes linked to the choice of technologies for implementation such as computers and their capabilities, underlying network, storage, programming tools and languages, etc.

In a distributed computing environment control is not exercised by a single computer. So the efficiency issue, i.e, the issue of quality in the problem domain must consider how control and coordination are exercised in the execution of distributed algorithms. Therefore, in order to structure distributed algorithm for execution in mobile environment we need to consider the problem of coordination and control.

5.2.1 Messaging cost

One of the important issue that remains unaddressed in this context is how the efficiency of an algorithm could be evaluated when it executed in a mobile distributed environment. Obviously, the message complexity remains the fundamental measure of the efficiency algorithms as in the case of a distributed computing environment. However, in a mobile computing environment, the cost of message is variable. To concretize the cost model, we follow the assumptions made in [1]:

- $C_w \Rightarrow$ cost of sending a message from MH to BS over wireless channel (and vice versa)
- $C_f \Rightarrow$ cost of sending a message from a point to another point linked by the wired N/W.
- $C_s \Rightarrow$ cost of searching/locating a MH and forwarding a message to its current BS from a source BS.

With the above assumptions, the cost of delivering message to a mobile host consists of two part: (i) first locating the destination node, and (ii)

then delivering the message to the destination. A simplest search strategy to locate a mobile host is to let the searching base station query all other base stations and determine if a mobile host belongs to a cell under one of the set of queried base stations. The base station responding to the query is the one which has the mobile host in the cell under it. The querying base station can then forward the message meant for the mobile host to the responding base station. So, message exchanged for location search will be as under.

1. In the first round all base stations except one receive message from the querying base station. So total of $(N_{BS} - 1) \times C_f$ messages are exchanged.

2. The base station having the searched mobile host responds. This incurs a cost of C_f.

3. Finally querying base station forwards a message (data packet) to the responding base station. This incurs a cost of C_f.

Adding all three costs, the worst case cost for search is equal to $(N_{BS}+1)\times C_f$.

The cost of sending message from a mobile host MH to another mobile host MH$'$ can be determined by analyzing the steps involved in forwarding of the message:

1. MH sends the message to its own base station BS. The cost incurred is C_w

2. BS then searches for the destination MH$'$ to find the base station BS$'$ under whose cell area MH$'$ is currently located and deliver the message to BS$'$. The cost incurred for this action is C_s.

3. BS$'$ sends the data packet to final destination MH$'$. An action incurring a cost of C_w.

Adding the cost of each step provides the overall cost of sending a message from a mobile host MH to another mobile host MH$'$. Therefore, the cost of sending a message from one mobile host to another is most $2C_w + C_s$.

It will be interesting to examine how the message complexity of various distributed algorithms get affected when they are executed on mobile computing environment without any structuring. This can be appreciated best by analyzing a few well known distributed algorithms over the different models of distributed systems.

5.3 Executing Distributed Algorithms on MCE

The class of distributed systems can categorized as follows.

- Non-coordinator based systems:
  - Systems with all machines equivalent
  - Systems with exception machines
- Coordinator based systems:
  - Fixed coordinator systems
  - Moving coordinator systems

5.3.1 Non-coordinator systems with all machines equivalent

In these systems all machines are equivalent, so it is actually peer-to-peer system with all peers having equal power. The amount of computational and communication load shared by each machine is the same. Such systems can easily be modified to work in a mobile computing environment. We illustrate this with Lamport's Bakery algorithm [4] for mutual exclusion.

In the Bakery algorithm, a process waiting to enter critical section chooses a number. This number must be greater than all other numbers currently in use. There is a global shared array of the current numbers for each process. The entering process checks all other processes sequentially, and waits for each one which has a lower number. Ties are possible, and are resolved using process IDs. To appreciate the impact of executing this directly on the mobile hosts we need not look into the details of the algorithm execution, but focus only on the communication aspects of the algorithm.

Since the computation load of each machine is equivalent there is not much one can do to improve the same. In the algorithm, each machine has to access the registers of the other machines. The drawbacks of non-restructuring (i.e., whe the algorithm is executed without restructuring) include:

1. High search cost: Since every message exchange is between mobile hosts, every message incurs a search cost. This leads to a overall search cost of
$$3(N_{MH} - 1) \times (2C_w + C_s)$$

which is proportional to the number, N_{MH}, of mobile hosts in the system. We use the same cost model as the one used by [1].

2. Battery consumption at mobile hosts will also be high. In each step the overall energy consumption would be proportional to the number of mobile hosts.

3. Since the algorithm requires the participation of each mobile host, it does not allow any host to disconnect or doze off.

4. The correctness of the algorithm requires that messages be delivered in FIFO order. This places a burden on the underlying network to maintain a logical FIFO channel between every pair of mobile hosts.

In order to improve the overheads of the bakery algorithm, the burden communication should be shifted to the fixed hosts as suggested by two-tier principle. So instead of N_{MH} mobile hosts executing the algorithm, the base stations (BSs) maintain the necessary data structures for the algorithm (*request queue, request and reply messages*) and secure mutual exclusion on behalf of the mobile hosts. The necessary modifications are as follows:

1. Time stamping is used only for messages exchanged between the BSs – which now form the components of bakery algorithm – and not for messages exchanged between the mobile hosts.

2. A mobile host initiates the algorithm by sending an initialization message to its current local BS. The BS executes the algorithm on behalf of the mobile host. Thus the appropriate request, reply and release messages are marked with the id of the mobile host for which it is sent by the BS.

3. When a BS secures mutual exclusion for a MH, it informs the MH. But the mobile host might have changed its location in the meantime. Therefore, potentially a search may be required.

4. The mobile host then accesses the critical section and informs the BS after need for access is over. A release message is sent to every other BS.

5. If the mobile host disconnects prior to the requested access to the critical section, then every disconnection has to be informed to the BS. On

receiving the grant for access, the BS can instantly initiate a release response in this case and, thus, other requests are not delayed.

Since time stamping of messages is used and each BS maintains a queue of pending requests, FIFO ordering of messages is easy to ensure. Also as the BSs execute the algorithm without any modifications and messages for different MHs are distinguished by tagging the host ids to them, the correctness of the algorithm is ensured. The overall communication cost of this algorithm will be as follows:

- Initialization = C_w,
- Granting a request = $C_w + C_s$, and
- Release of lock = $C_w + C_f$.

The overall cost along with the cost for executing the algorithm between BSs is, thus,

$$3C_w + C_f + C_s + 3(N_{BS} - 1) \times C_f,$$

where N_{BS} is the number of BSs participating in the algorithm execution. Note that comparing with the previous approach where the bakery algorithm was executed directly by the mobile hosts, since $C_s > C_f$ and $N_{MH} \gg N_{BS}$, the overall communication cost is much cheaper.

The mutual exclusion can be achieved using a logical ring structure in which the hosts access the critical section only when they receive the token in its due course of traversal in the ring as discussed in the previous section. This algorithm translates to distributed case in exactly in the same manner as above, along with the same associated benefits in terms of communication cost savings.

5.3.2 Non-coordinated systems with exception machines

Such systems are similar to the previous category, and differ only in the sense that the code executed by one machine (or a few of them) is different than that executed by the most of the other machines. The machines executing different code constitute the set of *exception* machines. An example of this category is Dijkstra's self stabilizing algorithm [2].

Consider a system consisting of a set of n finite state machines connected in the form of a ring, with a circulating *privilege* which denotes the ability

of a machine to change its state. when a machine has the privilege, it is able to change its current state which is termed a *move*. Thus, the system is self-stabilizing when regardless of the initial state and privilege selected each time, the system always converges to a legal configuration in a finite number of steps. On the presence of multiple privileges in the system, the choice of which machine is entitled to make the move can be decided arbitrarily. A legal state of the system has the following properties:

- No deadlock: there must be at least one privilege in the system.
- Closure: every move from a legal state must place the system into a legal state.
- No starvation: during an infinite execution, each machine should enjoy the privilege an infinite number of times
- Reachability: given any two legal states, there is a series of moves that change one legal state to another.

Let us now look at Dijkstra's algorithm involving K states where $K \geq n$, and n is the total number machines. For any machine, we use the symbols S, L, R to denote its own state, the state of left neighbor, and the state of the right neighbor respectively.

- **the exception machine**

  if $L = S$ then $S = (S + 1) \mod K$.

- **the other machines**

  if $L \neq S$ then $S = L$.

Similarly, a better algorithm (also by Dijkstra) using 3 states and two exception machines is as follows:

- **the bottom machine, machine 0**
  if $(S + 1) \mod 3 = R$ then $S = (S - 1) \mod 3$.
- **the top machine, machine n-1**
  if $L = R$ and $(L + 1) \mod 3 \neq S$ then $S = (L + 1) \mod 3$.
- **the other machines**
  if $(S + 1) \mod 3 = L$ then $S = L$
  if $(S + 1) \mod 3 = R$ then $S = R$.

The details of how the self-stabilization works is, however, not important so far as restructuring the algorithms to mobile environment is concerned. Instead we need to look at the communication that occurs between two machines.

In both the algorithms cited and also other self-stabilization algorithms, the essential communication perspective is to access the registers for the left and right neighbors. Thus, clearly these class of algorithms are very similar to the previous class of algorithms and the presence of one or more exception machines really does not make much of a difference to the communication costs involved. Thus, depending on the mobility of the hosts, either of search, inform or proxy strategies can be used to adapt to the mobile environment.

Since only the neighbor's values are needed, the overhead of wireless communication would be even less, as most neighboring hosts are expected to be under the same BS except for two at the edges.

5.3.3 Fixed coordinator based system

Distributed algorithms which involve a coordinator for running the algorithm properly, induce a greater communication overhead on the coordinator. The first class of such algorithms is the fixed coordinator set of algorithms which involves one particular host being permanently assigned to be the coordinator in the set of machines or processes. Thus apart from the normal optimization possible for the mobile hosts, the communication pattern for the coordinator has to be specifically optimized, which will yield better dividends in terms of reducing the communication cost, because a large portion of the communication in a coordinator based system will be centered around the coordinator. Apart from increased communication load, it increases the probability of failure as the coordinator is a single point of failure.

An example of the fixed coordinator system is encountered in the case of total order atomic broadcast algorithms. The system consists of N hosts, each broadcasting messages to the other hosts. Following a broadcast, the hosts time stamp the received messages and send them coordinator thread. On receiving the relayed broadcast message from all the threads, the coordinator sets the time stamp of the message to the maxima of the received time stamps. This is then broadcast back to the hosts. This way a total ordering on the broadcast messages is induced by the coordinator. Figure 5.1 illustrates this with a timeline diagram. In the case of fixed coordinator algorithms, the coordinator is always fixed to a particular host, e.g., host 3 in

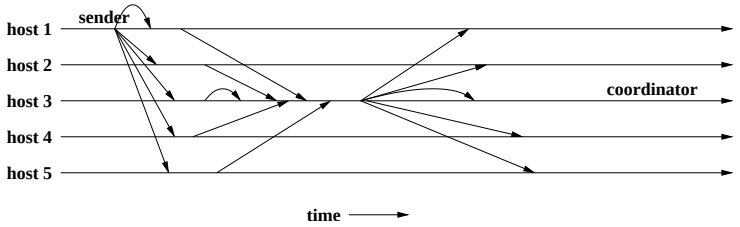

Figure 5.1: Total Order Atomic Broadcast using fixed/moving coordinator.

figure 5.1. To modify such algorithms for execution in a mobile environment, we give special attention to the coordinator. The modifications required is as follows:

1. If possible, the fixed coordinator is executed on a static host, that is a base station.

2. The other hosts are mobile – the normal search, inform and proxy strategies can be applied depending on the mobility characteristics of the system.

In the case where the algorithm is directly executed on the mobile hosts without any change, the total cost incurred for each broadcast will be,

1. cost of broadcast: $(N_{MH} - 1) \times (C_s + C_w)$,

2. cost of sending to the coordinator: $(N_{MH} - 1) \times (C_s + C_w)$,

3. cost of broadcasting time stamped messages back: $(N_{MH} - 1) \times (C_s + C_w)$

This leads to a overall cost of

$$3(N_{MH} - 1) \times (C_s + C_w)$$

This can be improved marginally if the location of the coordinator cached by each base station. The cost in that case would be

$$(2(N_{MH} - 1) \times (C_s + C_w) + (N_{MH} - 1) \times (C_f + C_w)$$

However, if the coordinator is placed on a BS, the cost is revised as follows:

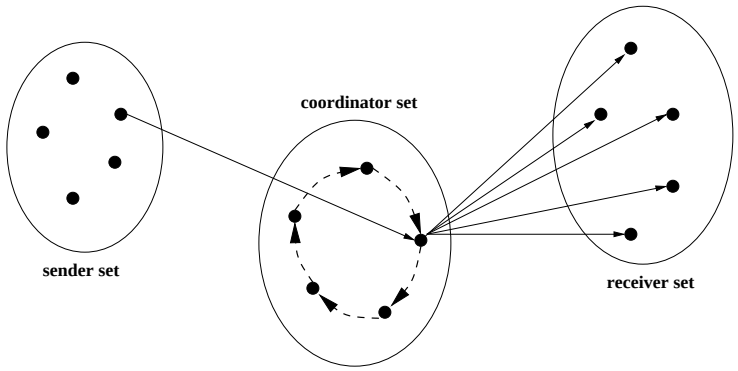

Figure 5.2: Conceptual model of a moving coordinator system, the sender, coordinator and receiver sets are the same, that is comprise of the same hosts, but are shown distinctly for the sake of clarity.

1. cost of broadcast: $C_w + (N_{BS} - 1) \times C_f$,
2. cost of sending to coordinator: $(N_{BS} - 1) \times C_f$,
3. cost of broadcasting timestamped message back: $(N_{BS} - 1) \times C_f + (N_{MH} - 1) \times C_w$

making it a total cost of

$$N_{MH} \times C_w + 3(N_{MH} - 1) \times C_f$$

Thus simple restructuring by placing the coordinator on a base station leads to substantial savings.

5.3.4 Moving coordinator based system

The difference of a moving coordinator system from the fore-mentioned fixed sequencer is in that the role of coordinator can be performed by different mobile hosts at different times. Thus who is the coordinator is a function of time as shown in figure 5.2. In this case normal algorithm execution at mobile hosts again has the same complexity as in the previous case. However, we can modify the system as follows:

1. The normal search, inform or proxy strategies are used for the all the hosts in the system.

2. As soon as a mobile host becomes a coordinator, then the number of accesses to it rises drastically in a short space of time. Hence the MH informs its BS about change of status to coordinator, which is then broadcast to all base station. Also the MH during the duration of its tenure as coordinator uses the inform strategy, while other hosts use the search strategy.

Using these modifications, each step of the algorithm now requires

1. cost of broadcast: $C_w + (N_{BS} - 1) \times C_f$,
2. cost of sending to coordinator: $(N_{BS} - 1) \times C_f + C_w$,
3. cost of broadcasting back the time stamped message: $(N_{BS} - 1) \times C_f + (N_{MH} - 1) \times C_w$, and
4. additional overhead associated with change of coordinator: $C_w + (N_{BS} - 1) \times C_f$.

Thus the total cost becomes: $(N_{MH}+1) \times C_w + 3(N_{BS}-1)C_f + \alpha(C_w + (N_{BS}-1) \times C_f)$ where a change of coordinator occurs every α broadcasts. Comparing this with the normal non-restructured cost $3(N_{MH} - 1) \times (C_s + C_w)$, the savings are significant.

5.4 Structuring Distributed Algorithms for MCE

Obvious from the characteristics of a mobile computing environment that the efficiency requirements of distributed algorithms suited for such an environment should be to

- Minimize communication cost,
- Minimize bandwidth requirement,
- Meet the synchronization requirements, and
- Overcome resource constraints of mobile hosts.

Bandwidth is usually treated as a resource. Therefore, the impact of poor bandwidth can be examined along with the other resource constraints.

Synchronization is one of the key issue for the correct execution of any distributed algorithm. Unlike static clients, mobile clients can appear and disappear in any cell in a service area at any time. So, synchronization techniques should be adjusted to handle dynamicity of the peers. This makes the synchronization among mobile clients more complicated and far more expensive than that of clients in a static environment. There is, thus, a need to evolve of a new model for evaluating the cost of distributed algorithms in mobile environments. Apart from some easily identifiable cost criteria such as,

- Cost of computation on a mobile node versus that on a static node,
- Cost of relocating computation to static host, and
- Cost of communication on wireless channels,

there are other cost related elements such as *disconnection* of a mobile host from the fixed network and its subsequent reconnection to the fixed network. Furthermore, the cost model applicable for mobile infrastructured network is not directly extendible to infrastructureless mobile ad hoc networks. So, separate cost models need to be developed for different mobile networks.

5.4.1 Messaging for remote execution

Whenever an operation is to be executed on a remote object it is achieved by sending messages to the node that hosts that object. The operation then gets performed by the remote node on behalf of the initiating host. It is convenient to think that the mobile host *logically* executing operations directly on the remote node. Sending a message a mobile host itself is two step task:

1. First figure out where to send the message.
2. Next to actually send the message.

If the message is meant for a fixed host, the above two steps can be carried out by the base station (BS) of the mobile host within the fixed network. The BS being a part of fixed network would be able to forward the message to the destination node by leveraging the IP forwarding protocol. Sending

messages to a fixed host is thus, lot less expensive than sending messages to a mobile host, because no location search is required for delivering messages to a fixed host. Thus, it is preferable to avoid sending messages to mobile hosts except the case when both sender and the receiver are under the same BS. As mobile hosts are required to avoid sending messages to one another, a mobile host should avoid executing operations on objects resident in another mobile host. So, the first design principle is:

Principle 1 *To the extent possible, all remotely accessed objects should be resident on the fixed hosts.*

The above principle says to the extent possible, computation and communication costs of an algorithm is borne by the static portion of the network. Hence, the core objective of the algorithm is achieved through a distributed execution of computation on the fixed hosts, while performing only those operations at the mobile host as that are necessary for the desired results. In our case an object is associated with the entity that requires computing power and bandwidth, thus, we disallow such things on the mobile host. The role of the mobile host comes up, as being involved in the thread of execution which executes the operations on the object (in most case the operation is staged by sending message to the fixed remote node holding the object).

5.4.2 Location management issues

Whenever a centrally located resource is accessed from a number of remote locations there is contention. We can treat this centrally located resource as an object. Each sequence of operations that are being initiated from a specific place (a mobile host) can be called a *thread* of execution. Thus, execution scenario is that of many concurrently running threads trying to operate on an object. For the moment, let no assumptions be made about the location of this object. It may be on a fixed host or it may reside on a mobile host as well. The concurrent threads compete for gaining access to the object. It is undesirable to have the object in an inconsistent state due to concurrent operations by the competing threads. In other words, the access to on the object by mutually competing threads should be synchronized.

The general pattern of all distributed algorithms can be visualised as comprising a *communication* component and *computation* component. In a distributed execution, the computation component is limited to the individual hosts and during these computation it might require also to commu-

nicate with its neighbours or other nodes for exchanging results of partial computations and to synchronize for further progress of computation. The synchronization requirements among other things may involve parameter initializations for the next phase of computation. Thus a distributed system of hosts exhibit bursts of communication in between periods of local computation.

When hosts become mobile, two additional cost parameters, namely, *location management* and *lookup cost* and *cost of computational resources at mobile hosts* become dominant along with communication cost. To deal with the computational resource related cost, all the compute intensive parts of the algorithm are located on the fixed hosts to as much extent as possible. To address location maintenance and lookup cost, three different strategies, namely, the *search*, *inform* and *proxy* are proposed by Badrinath, Acharya and Imielinski [1] in the context of restructuring a logical token ring network. These principles are in fact more general and can be used in various other classes of distributed algorithms. We will examine these strategies

5.5 Two Tier Structure

As apparent from the previous section, a distributed system is viewed as a two tier structure, namely,

(i) a collection of mobile nodes, each of which may operate in connected or disconnected or doze mode, and a low bandwidth wireless connects it to the fixed network.

(ii) a network of fixed nodes each having more resources in terms of storage, CPU speed, power and bandwidth,

The guiding principle for structuring distributed algorithms for mobile applications is that communication and computing demands of an algorithm should be satisfied within static segment of the network to as much extent as possible which is already mentioned under principle **??**. Some important justifications for this principle are:

- The message sent from a base station to a non local mobile host will incur a search cost. The same is true for exchange of messages between mobile hosts located in cell areas under distinct base stations. In order

to reduce overall cost the search cost must be reduced. So it is desirable to limit the communication between a mobile node and a fixed node locally within the same cell.

- Mobile nodes operate with battery power. It has to expend power for disk accesses, cpu operations and also for sending and receiving message over wireless links. On the top of this wireless channels offer significantly low bandwidth compared to wired links, the quality of these channels also depend on vagaries of weather and channel interferences of various types. So, transmission over wireless interface will involve packet loss, retransmission and consequently large latency. Therefore, the number of wireless messages exchanged during execution of any algorithm should kept at a minimum even at the cost of a higher number of message exchanges over the wired network.

- A mobile node can operate in *disconnection* or *doze* mode. When a node operates in doze mode it shuts or slows down most of its system functions in order to reduce power consumption. It can listen to incoming messages in doze mode. In a disconnection mode, a mobile host operates autonomously and has no connectivity with the network. The major difference between two modes of operations is that a mobile host in doze mode is reachable from the rest of the system whereas in a mobile host in disconnection mode is unreachable. A node in doze mode may be forced into reconnection through paging and wakeup. As opposed to this a disconnection and subsequent reconnection are always initiated by the mobile host. However, if mobile hosts were to participate in execution of distributed algorithm then these nodes should operate in connected mode. Even a participating mobile host which neither initiated the computation nor requires the result is prevented from operating in doze or disconnection mode.

- Most distributed system employ a regular logical structure such a tree, ring or a grid among the participants to carry out the required communication. The purpose of providing a systematic structure of communication is to make it ordered and predictable among the participants. Messages exchanged will then follow only preselected logical paths. But if a participating mobile host decides to operate in disconnection mode during execution of a algorithm, then the communication structure has to be reconfigured resulting in additional overhead in terms of messages

traffic and possibly search. A selected logical path for communication predefines a sequence of nodes through the messages would traverse from a given source to a given destination. So, all intermediate nodes on the path are prevented from operation in doze or disconnected mode.

5.5.1 Mobility and Disconnection

It is assumed that a message will eventually reach destination mobile host MH possibly after search. There is no limitation on the number of moves that a mobile node can make. Let us assume that a FIFO channel exists from MH to local BS and another from BS to MH. A MH may leave a cell at any time. Therefore, if $m_1, m_2, \ldots, m_s$ is the sequence of messages sent from BS to a MH, then the sequence of messages received at MH may be $m_1, m_2, \ldots, m_r$, where $r < s$. It implies that the messages received by MH from BS is only a prefix of the entire message sequence.

Mobility is exhibited by a host leaving the area of an old cell to entering into the area of a new cell. Each base station maintains a list of active mobile hosts. When a mobile host MH moves away from a cell under a base station BS to a cell under a new base station BS′, it sends *leave*(r) to BS, where r is the sequence number of the last message MH received from BS before leaving its cell. On receipt of *leave(r)* BS neither sends on BS-to-MH channel nor receives any from MH-to-BS channel. A MH on entering the cell under BS′ sends a *join*(MH, BS) to BS′. BS′ can then update its list of mobile hosts. Technically a *handoff* process is to executed for maintaining connectivity for a mobile host leaving an old cell and entering a new cell. A discussion on handoff process is beyond the scope of this chapter. We defer it to chapter **??**.

Disconnection when voluntary, is same as leaving a cell. This is because MH operates autonomously in disconnection mode. The difference is that a MH leaving a cell will eventually become active in some adjoining cell. But a disconnected mobile host may not even surface. A MH deciding to operate in disconnected mode sends *disconnect(r)* to BS. BS deletes the MH but puts a disconnect flag on. When MH connects back it sends *reconnect*(*MH*, *BS*). Disconnection flag is removed as part of hand-off. MH may not always supply id of the base station at the time of reconnection, in that case a query must be executed to find the local base station. Query involving a disconnected MH is satisfied by a base station BS where the disconnect flag is set.

5.6 Mutual Exclusion

Consider the case of executing an algorithm over a set of participating mobile hosts. They need to cooperate and communicate for the progress in execution of the algorithm. Fundamental to this cooperation is the mutual exclusion on the use of certain common resources. For example, suppose underlying communication in a distributed algorithm is carried out using a logical ring structure. Then one of the participating host should have the opportunity to send message to its adjacent host in logical ring structure. Normally this is solved by circulating a token on the ring of participating nodes. In the token ring algorithm the host which receives the token gets opportunity to send. We will examine the token ring algorithm subsequently in more details. In general, the contention for a sharable resource in a distributed environment is solved by mutual exclusion principles which requires certain activity to be carried out only by one participant at a time. Such sharable resources are termed as critical resources.

The mutual exclusion in a distributed system can be solved as follows.

- A participating node which wishes to use a critical resource, sends a request message to all other nodes.
- Waits for a reply from each.
- If the critical resource is not used presently by any other node, then use the resource.
- Sends a release to all after completing the access to the critical resource.

Suppose the above mutual exclusion to be executed for by set of N_{MH} mobile hosts. The cost of execution can be found by counting messages that are exchanged.

1. In the first step a mobile host sends $(N_{MH} - 1)$ request messages to its local base station for $N_{MH} - 1$ other mobile hosts. So the message cost for this step is $(N_{MH} - 1) \times C_w$
2. The receiving base station have to search and deliver the message to the local base stations of the respective mobile hosts. This will cost $(N_{MH} - 1) \times (C_s + C_w)$.

3. In the next step each of $N_{MH}-1$ participants replies back to requesting mobile host. These participants collectively send $(N_{MH}-1)$ messages through their local base stations. Each message requiring a search and delivery through local base station of the requesting mobile host. So the cost of this step is $(N_{MH}-1) \times (2C_w + C_s)$.

4. Finally, after use of critical resource the initiating mobile host send release messages to $N_{MH}-1$ other mobile hosts. The cost for sending this message is again $(N_{MH}-1) \times (2C_w + C_s)$.

Adding the above costs together the cost of satisfying a mutual exclusion request is

$$3(N_{MH}-1) \times (2C_w + C_s)$$

Note that each request requires exchange of $6(N_{MH}-1)$ wireless messages. So the requirement of energy consumption at participating nodes is proportional to $6(N_{MH}-1)$ The energy requirement at requesting host alone is $3(N_{MH}-1)$. In addition to prohibitive cost, there are technical difficulties like reconfiguring set of active participants to permit some participants to operate in disconnected mode.

We can improve it by assuming that mutual exclusion is achieve by passing a token in a logical ring among the participating mobile hosts. The mutual exclusion is satisfied by having only mobile host that has the token, can access the critical resource. The token is passed from one mobile host to its successor in the logical ring after the former has used it. With this improvement the cost is reduced considerably. The analysis of cost of the above token ring algorithm, which we will refer to as TR-MH, is as follows.

- Both sender and recipient are mobile hosts. The message is sent first from sender to its local base station then from there to base station under which the recipient is found. So the cost of messages exchanged on wireless links is $2C_w$. We know that the cost of locating a mobile host and subsequently sending a message to its current base station is C_s. Therefore the cost of passing of token between two adjacent MHs in the logical is $2C_w + C_s$.

Assuming the ring to consist of N_{MH} mobile hosts, overall cost of circulating token on the logical ring is $N_{MH}(2C_w + C_s)$. Note that the above cost is independent of the mutual exclusion requests satisfied by one complete circulation of the token in the logical ring.

Each exchange of message requires power both at the recipient and the sender. Every MH accesses wireless twice (once for acquiring and once for releasing) for one circulation of the token through it. So, energy requirement for executing this algorithm is proportional to $2N_{MH}$.

5.6.1 Logical Ring Using Two-Tier Principle

According to this principle, a logical structure of ring is to be maintained with fixed hosts, i.e, base stations. A token is circulated among BSs in a predefined sequence. A mobile host MH which wishes to access the token has to submit the request to its current BS. When the token becomes available at BS, it is sent to MH at its current local base station, say BS′, The MH after using the token returns it to BS′ which in turn returns the same to BS.

As apparent the cost of servicing token will depend also on the way location of a migrant MH is managed. Badrinath, Acharya and Imielinski [1] had explored the token ring problem under two location management strategies, namely *search* and *inform*. They also investigated an alternative method where the token is circulated among *proxies*. In this method the set of all base stations are partitioned into regions and associated a designated fixed host called *proxy* with each region. So the number of proxies is much less than the number of base stations. The token circulates among proxies and each proxy services token requests arising out of all base stations under its region. A combination of search and inform is used to manage the location of a migrant MH. We discuss the proxy method subsequently.

5.6.2 Search Strategy

It involves searching the entire area of coverage to find a migrant MH. The algorithm can be best understood by fragmenting the actions performed by the component devices, namely, base station and mobile hosts. A base station BS performs the following actions.

case *event* **of**
    on receipt of a request for access of the token from MH:
        add MH's request to the rear of *request queue*
    on receipt of the token from the predecessor:
        move pending requests from *request queue* to *grant queue*
        repeat

```
            remove request at head of grant queue
            if (MH which made the reques is local to BS)
                deliver the token to MH over wireless link.
            else
                search and deliver token to MH at its current cell
            await return of token from the MH.
        until grant queue is empty.
        forward token to BS's successor in the logical ring.
end case
```

Actions performed by a MH are as follows:

```
case event of
    on requirement for an access of the token :
        submit request to current local BS.
    on receipt of token from local BS:
        use the token
        return the token to local BS.
end case
```

From the above sketch of the algorithms, it is clear that at any time only one MH holds the token. Therefore, mutual exclusion is trivially guaranteed. The maximum number of requests that can be serviced at any base station is bounded by the size of the *grant queue* at that base station when the token arrives. No fresh request can go to *grant queue* as they are added only to the *request queue*. So a token acquired by a base station will be returned after a finite time by servicing at most all the requests which were made before the arrival of the token. This number can not exceed N_{MH}, the total number of mobile hosts in the system. This means the token would eventually visit each base station in the ring.

The communication cost can be analyzed as follows:

1. Cost for one complete traversal of ring (of base stations) is equal to $N_{BS} \times C_f$, where N_{BS} is the number of base stations.

2. Cost for submission of a request from a MH to a base station is C_w.

3. If the requesting MH is local to BS that receives the token then the cost of servicing a request is C_w. But if the requesting MH has migrated to

another base station before the token reaches the base station where the request was initially made, then a location search will be needed for the migrant MH. The cost of this search plus the cost delivery of token to current base station of the migrant MH is C_s. So, the worst case cost of satisfying a request can be $C_s + C_w$.

4. The cost of returning the token from a MH to the sender base station is $C_w + C_f$ in the worst case. C_f component of cost comes from the fact that the requesting MH may have subsequently migrated from the base station where it made request.

Adding the cost components together we have: the cost of submitting a single request and satisfying it is equal to 3 C_w + C_f + C_s. If we have K requests met in a single traversal of the ring then the cost will be

$$K \times (3C_w + C_s + C_f) + N_{BS} \times C_f$$

It is expected that the number of mobile host requesting a service is much less than total number of mobiles in the system. So $K \ll N_{MH}$.

In order to evaluate the benefit of relocating computation to fixed network we have to compare it with token ring algorithm TR-MH described earlier in this section where the token circulates among mobile hosts.

- *Power consumption.* The power consumption in present algorithm is proportional to $3K$ as only $3K$ messages are exchanged over wireless links. In TR-MH algorithm it was $2N_{MH}$, we know that $K \ll N_{MH}$. So it is expected that $\frac{3K}{2N_{MH}} < 1$
- *Search cost.* For the present algorithm it is $K \times C_s$ whereas in the previous algorithm the cost is $N_{MH} \times C_s$ which is considerably more.

5.6.3 Inform Strategy

Inform strategy reduces the search cost. The idea is based on simple principle that the search becomes faster if we know where to search. In other words search is less expensive when we are better informed about the location of a migrant host. We can only know the location before hand if a migrant MH to notifies the BS (where it submitted its request) after every change in its location till it has received the token. The algorithm specifying the actions of a BS is provide below.

case *event* **of**
on receipt of a request from a local MH:
add the request $< MH, BS >$ to the rear of *request queue*.
on receipt of a *inform*(MH, BS') message:
replace the current value of *locn*(MH) with BS'
in the entry $< MH, locn(MH) >$ in *request queue*.
on receipt of the token:
move *request queue* entries to the *grant queue*.
repeat
remove the request $< MH, locn(MH) >$ at the head of
the grant queue
if $(locn(MH) == BS)$
deliver the token to MH over the local wireless link
else
forward token to *locn*(MH), i.e. to BS'
currently local to MH which will transmit
it to MH in the local wireless cell.
await return of the token from MH.)
until (grant queue is empty)
forward token to BS's successor in logical ring.
end case

The algorithm for actions to be executed by a mobile host MH is provided below.

case *event* **of**
on requirement of MH needing an access to the token:
submit request to its current local BS
stores current local BS in the local variable *req_locn*.
on receipt of the token from BS *req_locn*:
access the critical resource or region
return token to BS (*req_locn*)
set *req_locn* to $\perp$.
on each move made by MH:
sends *join*(MH, *req_locn*) message to local BS
on entering the cell under a BS:
if $(req\_locn \neq \perp)$
sends a *inform*(MH, BS') to BS (*req_locn*).
end case

To compare the search and inform strategies, let a MH submit a request at BS and receive the token at BS. Assume that it makes MOB number of moves in the intervening period between submission of the request and the receipt of the token. After each of these moves, a *inform*() message was sent to BS, i.e. the cost of inform is $MOB \times C_f$. Since the location of MH is known after each move it makes, there is no need to search for its location. So, when token becomes available at BS and it is MH's turn to use the token, then it is directly despatched to BS$'$ where MH currently found. On the other hand, in the algorithm employing simple search, BS must search for the current location of MH, incurring a cost C_s. Therefore, the inform strategy is cost effective compared to search strategy provided $MOB \times C_f < C_s$. In other words if MH is less mobile after submitting its request, then it is better for MH to inform BS about every change of location rather than BS searching for it.

5.6.4 Proxy Strategy

Proxy strategy combines advantages of both search and inform strategies. It is designed to exploit the imbalance between frequencies of local versus global moves that a mobile host makes. Usual mobility pattern is that a mobile host moves more frequently between adjacent cells and rarely between non-adjacent cells. The coverage area consisting of all base stations are partitioned logically into *regions*. BSs within the same region are associated with a common proxy. A proxy is a static host, but it may not necessarily be a base station. The token circulates in a logical ring comprising of proxies. Each proxy, when it receives the token, is responsible for servicing pending requests from the request queue maintained by it. Each request is an ordered pair $< MH, proxy(MH) >$, where MH is the mobile host that submitted the request, and $proxy(MH)$ represents the region where MH is currently located. If a mobile host makes a inter regional move, then its local BSs before and after the move, are associated with different proxies. On the other hand, when a mobile host MH makes an intra regional move the proxy does not change after a move. The implicit assumption here is that a MH is aware of the identity of the proxy associated with its current cell. This can be realised by having each BS include the identity of its associated proxy in the periodic beacon messages.

The actions executed by a proxy P is provided by the following algorithm.

case *event* **of**
on receipt of a token request from MH:
// request forwarded by a BS within P's local area),
request $< MH, P >$ is appended to the rear of *request queue*
on receipt of a *inform*(MH, P') message:
current value of $proxy(MH)$ in entry $< MH, proxy(MH) >$
is changed to P'.
on receipt of the token from the predecessor in logical ring:
requests from *request queue* moved to *grant queue*.
repeat
delete request $< MH, proxy(MH) >$ from head of
request queue.
if $(proxy(MH) == P)$ // MH located within P's area,
deliver the token to MH after searching for MH
within the BSs under P.
else // $proxy(MH)$ is different from P
forward the token to $proxy(MH)$ which delivers it
to MH after a local search for MH within its area.
await return of the token from MH.
until (*grant queue* is empty)
forward token to P's successor in ring.
end case

The actions executed by mobile host MH are:

case *event* **of**
on requirement for an access of the token:
submit request $< MH, proxy(MH) >$ to local BS
stores identity of local proxy in *init_proxy*.
on the receipt of the token from *init_proxy*:
use the token
return token to *init_proxy*
set *init_proxy* to $\perp$
on a inter regional move by MH:
send a *join*$(MH, init\_proxy)$ message to new BS.
if $(init\_proxy \notin \{P, \perp\})$
// P' is the proxy of new BS, MH has made a
// wide-area move
new BS sends a *inform*(MH, P') message

to *init_proxy*.

end case

Let N_{proxy} represent the number of proxies constituting the ring. Let N_{BS} be the number of BSs in the coverage area. Then there are N_{BS}/N_{proxy} base station in a region. Let MOB_{wide} be the number of inter regional moves made by a MH in the period between submitting a token request and receiving the token. Similarly, let MOB_{local} represent the total number of intra regional moves in the same period. So, the total number of moves MOB is equal to $MOB_{wide} + MOB_{local}$. Before delivering the token to a MH, a proxy needs to locate a MH amongst the BSs within its region. This as search is referred to as local search with an associated cost C_{ls}. With the above notations and assumptions, the communication costs can be analyzed as follows:

1. Cost of one token circulation in the ring: $N_{proxy} \times C_f$
2. Cost of submitting a token request from a MH to its proxy: $C_w + C_f$
3. Cost of delivering the token to the MH: $C_f + C_{ls} + C_w$

   Note that C_f term can be dropped from above cost expression if MH receives the token in the same region where it submitted its request.
4. Cost of returning the token from the MH to the proxy: $C_w + C_f$

The above costs add up to: $3C_w + 3C_f + C_{ls}$

If an inter regional move is made, then the current local BS of MH sends the identity of new proxy to the proxy where MH submitted the request initially. The cost for inform is, therefore, C_f. The worst overall cost for satisfying a request from a MH, including the inform cost, is then

$$(3C_w + 3C_f + C_{ls}) + (MOB_{wide} \times C_f)$$

Obviously, the cost of circulating the token amongst the proxies is less than circulating it amongst all BSs by a factor N_{proxy}/N_{BS}. The cost of circulating the token within the logical ring is a measure of distribution of the overall computational workload amongst static hosts. If all three schemes service the same number of token requests in one traversal of the ring, then this workload is shared by

- N_{BS} static hosts in search strategy,

- It is shared amongst N_{proxy} fixed hosts under the proxy method.

To compare the efficiency of each algorithm to handle mobility, we need to consider the communication cost of satisfying token requests. The three algorithms incur the following costs to satisfy a request for the token:

search: $3C_w + C_f + C_s$

inform: $3C_w + C_f + (MOB \times C_f)$

proxy: $3C_w + (3 + MOB_{wide}) \times C_f + C_{ls}$

The above cost expressions can be compared against one another to determine which strategy performs better than the other. For instance, proxy strategy performs better than search strategy, if

$$\begin{array}{lll} (3 + MOB_{wide}) \times C_f + C_{ls} & < & C_f + C_s \\ \equiv \quad MOB_{wide} + 2 & < & (C_s - C_{ls})/C_f \end{array} \tag{5.1}$$

Search strategy requires a BS to query all other BSs within a search region to determine if a MH is local to its cell. The BS currently local to the MH will respond, and the original BS can then forward the token to this BS. If N_{region} is the number of BSs within the search area, the search cost is equal to $(N_{region} + 1) \times C_f$. Using such a search strategy, where the search cost is linearly proportional to the number of locations within the search area, formula 5.1 above reduces to:

$$MOB_{wide} < N_{BS} - (N_{BS}/N_{proxy}) - 2$$

Indicating that the proxy scheme performs better than search when the number of inter regional moves made by a migrant MH is two less than the total number of BSs that is outside any given region.

Now let us compare proxy with inform. The proxy scheme incurs a lower cost to satisfy a token request when:

$$\begin{array}{l} (3 + MOB_{wide}) \times C_f + C_{ls} < (MOB + 1) \times C_f \\ \equiv C_{ls} < (MOB - MOB_{wide} - 2) \times C_f \\ \equiv C_{ls} < (MOB_{local} - 2) \times C_f \end{array} \tag{5.2}$$

The cost of a local search equals $(N_{BS}/N_{proxy} + 1) \times C_f$, since all base stations have to be queried by the proxy in its region and only the local base station of MH will reply. So, the formula 5.2 above reduces to:

$$N_{BS}/N_{proxy} + 2 < MOB_{local}$$

This indicates that if the number of intra regional moves made by a migrant MH is two more than the number of BSs under each proxy, then proxy scheme outperforms the inform scheme.

Fairness in Access of the Token

There are two entities whose locations vary with time namely, the token and the mobile hosts. So we may have a situation in which a mobile host MH

1. submits a request to its current local base station BS.
2. gets the token from BS and uses it,
3. moves to the base station BS' which is the next recipient of the token,
4. submits a request for the token access at BS'.

So, the mobile host MH by virtue of being more mobile than other mobile hosts can gain multiple accesses to the token during one circulation of the token through the fixed hosts. This violates the *fairness* property of the token access among mobile hosts. So, we need to put additional synchronization mechanisms in place to make the algorithms fair. Interestingly, the problem of fairness does not arise in the algorithm TR-MH, which maintains the logical ring amongst MHs. Therefore, we need to evolve ways to preserve the functionality of fairness of TR-MH algorithm in the token algorithms which maintain logical ring within the fixed network regardless of the mobility of hosts.

Of course, none of the algorithms, search, inform and proxy cause starvation. Because a stationary mobile host is guaranteed to get its request granted when the token arrives its local base station. We have already observed that the length the pending requests in request queue at a fixed host is always bounded. This means after a finite delay each fixed host will release the token. Therefore, each requesting mobile host at a base station will eventually gain an access to the token. In the worst case, a stationary MH may gain access to the token after every other mobile host has accessed the token once from every base station, i.e, after $(N_{MH} - 1) \times N_{BS}$ requests have been satisfied. A simple fix to the problem of fairness is as follows:

1. The token is associated with a loop counter (`token_val`) which is incremented every time it completes one traversal.

2. Each MH maintains a local counter `access_count` whose current value is sent along with the MH's request for the token to the local BS.

3. A pending request is moved from the request queue to the grant queue at the BS (or the proxy) holding the token, only if the request's `access_count` is less than the token's current `token_val`.

4. When a MH receives the token, it copies the current value of `token_val` into its `access_count`.

Since a MH's `access_count` is reset to the token's current `token_val` after each access of the token, a MH can access the token only once in a single circular traversal of the token even if a MH has a low `access_count`. With the modified algorithm, the number of token accesses K satisfied in one traversal of the ring is limited to N_{MH} (when the ring comprises of all BSs), while the value of K could be at most $N_{BS} \times N_{MH}$ otherwise. So the difference between the above modification and the original scheme essentially represents a trade-off between "fairness" of token access among the contending MHs and satisfying as many token requests as possible per circular traversal of the token.

Of course one can devise an alternative definition of "fairness" that requires a MH to be allowed to access the token multiple times in one traversal of the ring, under the limitation that its total number of accesses to the token does not exceed the current `token_val`. This criterion can be implemented by modifying step (4) above, as follows: the `access_count` of a MH is incremented on every access, instead of being assigned the current value of `token_val`.

5.7 Termination Detection

There is clear a evidence that the generic design principle of exploiting asymmetry of two-tier model would be very effective in structuring distributed algorithms for mobile environment. However, the scope of discussion in the previous section was restricted. It is just centered around design of mutual exclusion algorithm. Therefore, we need to examine further how the design principle built around the asymmetry approach could be useful for other problems as well. In this section we focus on termination detection.

Termination of a distributed computation represents one of global state of the computation. The recording global states of a distributed systems is known to be difficult. Therefore, it is difficult to record the termination state of a distributed computation. However, termination being one of the stable properties a distributed computation can be observed. When a computation terminates there can be no messages in transit. Therefore, it is possible to design an observing algorithm which does not interfere with the main computation but is able to detect termination of the computation. More precisely, the termination detection algorithm determines whether a distributed computation has entered a state of *silence*. In a silent state no process is active and all the communication channels are empty, taking into account unpredictable delays in message delivery.

An implicit assumption made by most termination detection algorithms is that the main computation never enters an incorrect state. In the case of mobile distributed systems, termination detection is more complex, because the detection algorithm should also handle the scenarios arising out of many mobile hosts operating in disconnected mode. Mobile hosts in disconnected mode should not be disturbed. Of course voluntary disconnection can be planned so termination algorithm can handle them. But, in the cases of involuntary disconnections, mobile hosts may not regain connectivity due to failure. In other words, an incorrect state at times is indistinguishable from a correct state of the computation in mobile distributed systems.

5.7.1 Two Known Approaches

In a termination state the channels are empty. Therefore, no message can reach any of the process and consequently, no processes can become active ever again under this state. There are two fundamentally different approaches to detect termination:

- Diffusion.
- Weight throwing.

A convenient model to visualize a distributed computation is a directed graph that grows and shrinks as the computation progresses [**?**]. If such a graph contains an edge from a node $n1$ to another node $n2$, then $n2$ is known as a successor of $n1$, and $n1$ a predecessor of $n1$. Every node in a distributed computation starts with a a *neutral state*. Dijkstra and Scholten [**?**] define

that a diffusion based distributed computation is initiated in a neutral state when the environment on its own generates a message and sends it to its successor. After the first message has been sent, an internal node is free to send messages to its successor. So, a diffusion computation grows as a directed graph. After an internal node has performed its node specific computation, it signals completion to its predecessors. In practice, though a two-way communication is possible, the flow of computation messages is only in the direction from a predecessor to a succcesor. The completion event can be viewed as an acknowledgement for some computation message received earlier by the node. So when completion event is eventually signaled back to environment, the distributed computation is assumed to have terminated.

In weight throwing scheme [3, 5, 6], environment or starting process in neutral state has a weight credit of 1 with it. Every message sent by any node is associated with a weight. The sending node partitions the weight available with it into two parts. One of the part is attached to the message before sending it while the other part is retained by the sending node. The computation is started by the environment generating a message of its own and sending it to its successor. Thereafter the internal nodes send messages to their successors as in a diffusion computation. When the node specific computations at a node is over, it signals completion of computation by a weight reportage message to its predecessor besides setting its local weight to 0. The weight reportage message carries the total weight left at a node at the time of completion of the local node specific computation.

5.7.2 Hybrid approach for MCE

Termination detection in mobile distributed system follows a a hybrid approach [7]. It consists of running a simplified diffusion based termination detection algorithm for the mobile part and a weight throwing based algorithm for the fixed part. The base stations act as bridges between the two parts.

Let us first introduce a few notations which will be convenient to understand the above protocol. A mobile distributed computation can be described by:

1. A special process P_c called weight collector.

2. A set of base station processes denoted by P_i, for $i = 1, 2, \ldots$.

3. A set of mobile processes, P_i^m, for $i = 1, 2, \ldots$.

4. A set of messages.

The computation is started by the weight collector process. However, this does not necessarily represent a limitation of the model. A computation may also be triggered by a mobile process. In that case the weight collection will be performed by the static process representing the current base station for the initiating mobile process. In effect, the starting base station process becomes the environment node.

A mobile process can roam and execute handoff from one base station to another. When a mobile process moves the distributed computation on that process is suspended. If a mobile process moves away from its current base station and unable to find a new base station to which it can connect, then the mobile process is said to be temporarily disconnected. Further, it is assumed that mobile process can not carry out any basic computation as long as it remains disconnected.

5.7.3 Message Types

Six different types of messages are needed for the algorithm. These are:

1. M_b: a basic message. If M_b is tagged with a weight w, then it is denoted by $M_b(w)$.

2. $M_{wr}(w)$: a reporting message with a weight w.

3. $M_{ack}(k)$: an acknowledgement for k basic messages.

4. M_{HF}: handoff message. It contains four subtypes: $M_{HF.req}$, $M_{HF.ind}$, $M_{HF.rep}$ and $M_{HF.ack}$.

5. M_{DIS}: message for temporary disconnection. It contains two subtypes: $M_{DIS.req}$, $M_{DIS.ind}$

6. M_{JOIN}: messages connected with rejoining of a disconnected mobile node. It consist of two subtypes: $M_{JOIN.ind}$ and $M_{JOIN.rep}$

Termination detection in a mobile distributed system is of two types:

(i) Strong termination, and

(ii) Weak termination.

A strong termination state is reached when all processes have turned idle, there are no disconnected mobile process or any basic message in transit. In a weak termination state all processes except disconnected mobile processes have reached the state as in strong termination.

The protocol for termination detection in mobile systems should allow for a disconnected process to rejoin the computation at a later point of time. It is possible that a mobile may be disconnected due to failure. In that case the mobile may not join back. If the termination detection protocol does not have a provision to handle such a situation, and then it will not work. Weak termination is an important indication of anticipated system failure due to disconnected mobile processes. So, roll back and recovery protocols can be planned around conditions involving weak termination.

The termination detection algorithm proposed by Tseng and Tan [7] divides the protocol into several parts and specifies it in form of the actions by a mobile process, a base station process and the weight collector process. Before we discuss these algorithms it is necessary to understand how the mobile processes communicate with static processes and while mobile hosts roams.

A mobile process always receives a message from a base station. But it can send a message either to its local base station or to another mobile process. The message, however, is always routed through the base station to which the mobile is connected.

When a mobile moves to another base station while being active then it requires a handoff. A handoff is initiated by sending a $M_{HF.req}$ message. The responsibility for carrying out the rest of the handoff process lies with the static part. Apart of sending handoff, a disconnected mobile node may send a rejoin message on finding a suitable base station to connect with. The rejoin message, denoted by $M_{JOIN.req}$, is sent to the base station with which mobile node attempts to connect. The only other message that a mobile node could send is a signal for completion of local computation. This message is sent to base station. After sending this the mobile turns idle. So the relevant message types handled by a mobile host (MH) are:

- M_b: a basic message which one MH sends to another. M_b is always routed through the base station of the sender MH. A base station after receiving a message M_b from a mobile host (MH), appends a weight w to M_b and sends $M_b(w)$ to the base station of the receiving MH.

- $M_{HF.req}$: This message is sent by an MH to its local base station for a handoff request.

- $M_{JOIN.req}$: This message is sent by a disconnected MH when it is able to hear radio beacons from a base station of the network.

A base station can send a basic message either to a mobile node or to another base station on behalf of a connected mobile. The base stations are also involved in handoffs to facilitate roaming of mobiles. A disconnected mobile may rejoin when it hears beacons from a base station over its radio interface. In order to perform its tasks, a base station has to process and send different message types of messages. These message and intended use of these messages can be understood in the context of the protocol discussed later.

5.7.4 Entities and Overview of Their Actions

The termination detection protocol is carried out by actions of following three entities:

- Mobile hosts,

- Base stations, and

- Weight collector.

The termination detection protocol requires support at protocol level for roaming and disconnected mobile processes. A roaming of mobile process is managed by handoff protocol. Handoff protocol handles the transfer of weights associated with basic computation messages to appropriate base station. Similarly, the concerned base stations transfers associated weights of disconnected mobile processes to the weight collector. The transfer of weights as indicated above ensures that every base station can correctly transfer the weights to the weight collector when completion of basic computation is signalled.

The protocol distinguishes between two sets of mobile processes:

- MP_i: set of active mobile processes under base station BS_i.

- DP: set of disconnected mobile processes in the system.

Each active mobile processes is associated with an active mobile node while each disconnected process is associated with one disconnected mobile nodes. The set of mobile processes involved in the computation is given by the union $DP \cup \{\cup_i MP_i\}$.

5.7.5 Mobile Process

A mobile process, which runs on a mobile host, can either receive a basic message from a base station or send a basic message to another mobile process. All messages either originating or terminating at a mobile host (MH) are routed through the current base station of MH. When a mobile process receives a basic message it keeps a count of the number of unacknowledged message received from the concerned base station. So, mobile process knows the number of acknowledgements to be sent to the base station when it has finished computation.

A mobile process P_j^m always routes a basic message M_b through its current base station process P_i to another mobile process P_k^m. The process P_i attaches a weight $w_i/2$ to M_b and sends $M_b(w_i/2)$ to the base station hosting P_k^m. P_i also keeps track of the acknowledgements it has received from all mobile processes. So no additional protocol level support is required at P_j^m for sending a basic message to any other process P_k^m.

P_j^m increments the number of unacknowledged messages by 1 when it receives a basic message from P_i. A local counter in is used by a mobile process to update the count of unacknowledged messages. For example, P_j^m knows that it has to send acknowledgements for in messages. Therefore, before turning idle, P_j^m signals the completion of basic computation by sending an acknowledgement for in messages to P_i. After the acknowledgement has been sent, in is set to 0. The summary of the rules applicable for a mobile node in the termination detection protocol are as follows:

Rule M1: P_j^m sends a basic message M_b through its current base station process P_i to another mobile process P_k^m.

Rule M2: P_j^M on receiving a basic message from P_i:

$in = in + 1$; // increments count of unacknowledged messages

Rule M3: On turning idle from active mode, P_j^m execute the following:

send $M_{ack}(in)$ to P_i;

$in = 0$;

5.7.6 Base Station Processes

The base station process P_i at BS_i attaches a weight x from its available weights to every message M_b from a mobile process P_j^m before sending to the destination base station process P_ℓ. P_ℓ extracts the weight x from $M_b(x)$ and adds x to its the current weight. After that the extracted basic message $M_b = M_b(x) - \{x\}$ is forwarded to destination mobile process P_k^m. In this manner, static processes at base stations control the weight throwing part of the protocol on behalf of all the mobile processes.

In order to control the diffusion part, a static process P_i keeps track of the messages it has sent to every mobile process P_j^m under it. A base station process P_i expects a mobile process P_j^m under it to send acknowledgement for each and every message the latter has received. P_j^m signals the completion of computation by sending the acknowledgements for all the messages of P_i. When all mobile processes P_j^ms involved in a computation have sent their final acknowledgements to P_i, diffusion part terminates.

The rules for termination detection executed by a base station process P_i are as follows:

Rule B1: On receiving a basic message $M_b(x)$ from wired network which is meant for P_j^m, P_i does the following:

$w_i = w_i + x$; // Extract the weight carried by $M_b(x)$
$M_b = M_b(x) - \{x\}$; // Form basic message without weight
$out_i[j] = out_i[j] + 1$; // Update number of messages sent to P_j^m
Forward M_b to P_j^m;

Rule B2: On receiving a basic message M_b from P_j^m on wireless network for a destination process P_k^m, P_i does the following:

if ($P_k^m \in MP_i$) {
    // P_k^m is local to P_i
    $out_i[k] = out_i[k] + 1$; // Increment unacknowledged message count
    Forward M_b to P_k^m;
}**else** {
    // Destination P_k^m is non local
    $M_b(w_i/2) = M_b + w_i/2$; // Attach weight to M_b

$w_i^b = w_i^b/2$;
Locate MH_k's base station process P_ℓ;
Send $M_b(w_i/2)$ to P_ℓ;
}

Rule B3: On receiving $M_{ack}(k)$ from $P_j^m \in MP_i$, P_i executes:

$out_i[j] = out_i[j] - k$; // Decrement unacknowledge message count
if ($out_i[k] == 0$ for all $P_k^m \in MP_i$) {
// P_i turn idle as it does not have any mobile process.
Sends $M_{wr}(w_i)$ to P_c; // Send residual weight to P_c
$w_i = 0$; // Reset local weight
}

Before we consider handoff and rejoin protocols, let us examine two important properties of the protocol specified by actions of three entities we have described so far.

Property 1 *If $out_i[j] = 0$ then P_j^m is idle and there is no in-transit basic messages between P_i and P_j^m.*

Proof: When $out_i[j] = 0$, all basic messages sent by P_i to P_j^m have been acknowledged. So P_j^m must be idle. As every channel is assumed to be FIFO, if message M is sent before M' over a channel, then M must be received before M' at the other end. When P_j^m turns idle it sends acknowledgement for all messages it has received from P_i. So when this acknowledgement (which the last among the sequence of messages between P_i and P_j^m) is received by P_i there can be no message in-transit message on the channel because the acknowledgement would flush all other messages before it.

Property 2 *If $w_i = 0$ then all mobile processes $P_j^m \in MP_i$ are idle and there is no in-transit basic message within P_i*

Proof: If $w_i = 0$, then the weights held by P_i on behalf of all mobile process $P_j^m \in MP_i$ have been sent back to P_c. This can happen only if $out_i[j] = 0$, for all $P_j^m \in MP_i$. By Property 1, it implies P_j^m is idle and there is no in-transit basic message between P_i and P_j^m. Hence the property holds.

5.7.7 Handoff

When a mobile process P_j^m moves from its current cell under a base station BS_i to a new cell under another base station BS_k then handoff is executed. As a part of the handoff process, the relevant data structure and the procedure for managing termination detection protocol concerning P_j^m should be passed on to BS_k.

P_j^m makes a request for handoff to BS_k and waits for the acknowledgement from BS_k. If the acknowledgement is received then P_j^m starts interacting with BS_k for initiating transfer of the data structure related P_j^m from BS_i to BS_k. If handoff acknowledgement is not received within a timeout period, P_j^m retries handoff. The handoff procedure is a slight variation from layer-2 handoff. In a cellular based wireless network, when a handoff is initiated, old base station BS_{old} sends handoff request to mobile switching center (MSC). MSC then forwards the request to new base station BS_{new} (determined through signal measurements). BS_{old} then accepts the request and handoff command is executed through BS_{old}. So, for protocol level support from handoff, the mobile process must send $M_{HF.req}$ to BS_i identifying destination BS_k. BS_k accepts the request and sends $M_{HF.ack}$ to P_j^m. After this P_j^m starts communicating with BS_k.

However, to keep the protocol simple P_j^m has been shown to approach BS_k directly for execution of handoff. BS_k before sending $M_{HF.ack}$ to P_j^m, ensures that the transfers the needed data structure from BS_i is complete. The negotiation for transfer of link between BS_i and BS_k involves messages like $M_{HF.ind}$ and $M_{HF.rep}$ between BS_i and BS_k. The handoff process has been illustrated in Figure 5.3.

In response to handoff indication message from BS_k, process at BS_i sends a message to BS_k that includes:

- Count of the number of unacknowledged messages in respect of mobile process P_j^m. These are the basic messages for which BS_i was expecting acknowledgements at the time P_j^m sought a handoff.

- A weight $w = w_i/2$.

The base station process P_i at BS_i should check if P_k^m is the last mobile process under it. In that case P_i should itself turn idle. But before turning idle, it should send its existing weight credit to weight collector process.

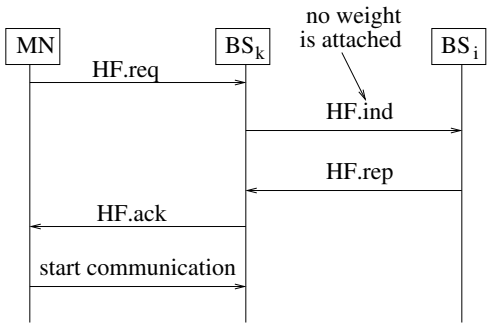

Figure 5.3: Illustration of handoff process

Once $M(w_i/2)$ has been received at BS_k, it should update its own data structure for keeping track of mobile processes. Thus the summary of the protocol rules for handoff execution are as indicated below.

Rule H1: On finding handoff condition P_j^m does the following:

Sends a $M_{HF.req}$ message to P_k and waits for $M_{HF.ack}$;
if ($M.HF.ack$ received from P_k)
    Starts communicating with P_k; // With base station BS_k
else // After time out
    retries handoff;

Rule H2: On receiving $M_{HF.req}$ from P_j^m, P_k:

Sends a $M_{HF.ind}(P_j^m)$ to P_i; // No weight attached.

Rule H3: On receiving $M_{HF.ind}(P_j^m)$ from P_k, P_i executes:

Sends a $M_{HF.rep}(P_j^m, out_i[j], w_i/2)$ to P_k;
$MP_i = MP_i - \{P_j^m\}$; // P_j^m no longer under P_i
$w_i = w_i/2$;
if ($out_i[l] = 0$ for all $l \in MP_i$) { // P_j^m is the last mobile process under P_i
    Sends $M_{wr}(w_i)$ to P_c;
    $w_i = 0$;
}

Rule H4: On receiving $M_{HF.rep}(P_j^m, out_i[j], w_i/2)$ from P_i, P_k does the following:

$MP_k = MP_k \cup \{P_j^m\}$;
$out_k[j] = out_i[j]$;
$w_k = w_k + w_i/2$;
Sends a $M_{HF.ack}$ to P_j^m;

5.7.8 Disconnection and Rejoining

It is assumed that all disconnections are planned. However, this is not a limitation of the protocol. Unplanned disconnection can be detected by timeouts and hello beacons. When a mobile process P_j^j is planning a disconnection it sends $M_{DISC.req}$ message to its current base station BS_i. Then BS_i suspends all communication with P_j^m and sends $M_{DISC.ind}(P_j^m, out_i[j], w_i/2)$ to P_c. It will allow P_c to detect a weak termination condition if one arises.

The rules for handling disconnection are, thus, as follows:

Rule D1: Before P_j^m enters a disconnected mode it

Sends $M_{DISC.req}$ message to P_i;
Suspends all basic computation;

Rule D2: On receiving $M_{DISC.req}$ from P_j^m, P_i executes:

Suspends all communication with P_j^m;
Sends a $M_{DISC.ind}(P_j^m, out_i[j], w_i/2)$ to P_c;
$MP_i = MP_i - \{P_j^m\}$;
$w_i = w_i/2$;
if ($out_i[k] == 0$ for all $P_k^m \in MP_i$) {
    Sends $M_{wr}(w_i)$ to P_c;
    $w_i = 0$;
}

Rule D3: On receiving $M_{DISC.ind}(P_j^m, out_i[j], w_i/2)$ from P_i, P_c executes:

$DP = DP - \{P_j^m\}$;
$out_c[j] = out_i[j]$;
$w_{DIS} = w_{DIS} + w_i/2$;

Final part of the protocol is for handling rejoining. If a mobile process can hear the radio of signals from a base station it can rejoin that base station. For performing a rejoin P_j^m has to execute the instructions given below.

Rule R1: A disconnected mobile process P_j^m on entering a new cell under BS_i does the following

Sends $M_{JOIN.req}$ message to P_i and waits;
if ($M_{JOIN.ack}$ is received from P_i)
    Starts basic computation;
else // After timeout
Retries JOIN;

Rule R2: On receiving $M_{JOIN.req}$ from P_j^m, P_i executes:

Sends a $M_{JOIN.ind}(P_j^m)$ to P_c; // No weight attached

Rule R3: On receiving $M_{JOIN.ind}(P_j^m)$ from P_i, P_c executes:

if ($P_j^m \in DP$) {
    Sends a $M_{JOIN.rep}(P_j^m, out_c[j], w_{DIS}/2)$ to P_i;
    $DP = DP - \{P_j^m\}$;
    if ($DP = \Phi$);
        $w_c = w_c + w_{DIS}$;
}

Rule R4: On receiving $M_{JOIN.rep}(P_j^m, out_c[j], w_{DIS}/2)$ from P_i, P_c does the following:

$MP_i^b = MP_i^b + \{P_j^m\}$;
$out_i[j] = out_c[j]$;
Sends a $M_{JOIN.ack}$ to P_j^m;
Restarts communication with P_j^m;

5.7.9 Dangling Messages

There may be messages in system which can not be delivered due to disconnection or handoff. Such dangling messages should be delivered if the mobile process reconnects at a later point of time. So, care must be taken to handle these messages. When a mobile process is involved in a handoff it can not deliver any message to static host or base station. So, the mobile process hold such undelivered messages with it until handoff is complete. This way message exchange history is correctly recorded. Dangling messages at base

stations are those destined for mobile processes involved in handoff or disconnection. These messages are associated with weights which is ensured by weight throwing part of the algorithm. So, handling dangling messages becomes easy. All dangling messages from base station processes are sent to weight collector. The weight collector process holds the messages for future processing.

Rule D1: P_c on receiving a dangling message carrying a weight x:

$w_{DIS} = w_{DIS} + x$;

Rule D2: On P_c sending a dangling message M_b to reconnected P_j^m:

appends $w_{DIS}/2$ to M_b;
$w_{DIS} = w_{DIS}/2$;
send $M_b(w_{DIS}/2)$ to P_i;// $P_j^m \in MP_i$
if (dangling message list == Φ) {// No disconnected mobiles exist
    $w_c = w_c + w_{DIS}$; // Reclaim residual weight
    $w_{DIS} = 0$;
}

5.7.10 Announcing Termination

The termination is detected by P_c when it finds $w_c + w_{DIS} = 1$. If $w_c = 1$ then it is a case of strong termination, otherwise it is case of weak termination. In the case of weak termination $w_{DIS} > 0$.

Bibliography

[1] Badrinath, B. R., Acharya, A., and Imielinski, T. Structuring distributed algorithms for mobile hosts. In *Proceedings of International Conference on Distributed Computing Systems* (1994), pp. 21–28.

[2] Dijkstra, E. W. Self-stabilizing systems in spite of distributed control. *Communications of the ACM 17*, 11 (1974), 643–644.

[3] Huang, S. T. Detecting termination of distributed computation by external agents. In *Proceedings of International Conference on Distributed Computing Systems* (1989), pp. 79–84.

[4] Lamport, L. A new approach to proving the correctness of multiprocess programs. *ACM Transaction on Programming Languages and Systems 1* (1979), 84–97.

[5] Mattern, F. Global quiescence detection based on credit distribution and recovery. *Information Processing Letter 30* (1989), 195–200.

[6] Tseng, Y.-C. Detecting termination by weight throwing in a faulty distributed sytem. *Journal of Parallel an Distributed Computing 25* (1995), 7–15.

[7] Tseng, Y.-C., and Tan, C.-C. Termination detection protocols for mobile distributed systems. *IEEE Transaction on Parallel Distributed System 12*, 6 (2001), 558–566.