Chapter 11
Distributed Algorithms for Mobile Environment

11.1 Introduction

From the prospectives of the application developers, a mobile computing system is a distributed systems consisting of thousands of mobile computers and a set of static computers connected by wireless networks [1]. A major part of the research in mobile computing system is directed towards establishing and maintaining connectivity between two types of computers through bridges between wireless with wired networks [2]. Over the years, however, mobile computing has emerged as a distinct paradigm for problem solving which is characteristically different from conventional distributed computing.

From an abstract point of view, a mobile computing system can be seen as a graph that consists of a fixed core of static nodes and a dynamic set of mobile leaf nodes [3]. Structurally, the organization is similar to a cellular mobile telephone network. The mobile leaf nodes can be viewed as a set of persistent messages moving through graph. With this underlying graph model, traditional distributed algorithms can be implemented directly on mobile system. Unfortunately, a direct mapping of distributed algorithms to mobile environment is not practical due to limited bandwidth, fragility of wireless links and many other resource specific constraints associated with mobile nodes.

A commonsense driven approach to design a distributed algorithm for mobile system will be to assign computing tasks to the fixed part of the system as much as possible. It intrinsically formulates a logical two-tier approach for computing in a mobile distributed environment. By offloading compute intensive tasks to the static computers [4, 5], mobile devices can save critical resources including batteries. Before we expand on the idea of two-tier approach, let us examine the differences between a traditional distributed system and a mobile computing system a bit more in details.

The mobility of a computing node brings up two important new issues in data delivery [6]:

R.K. Ghosh, *Wireless Networking and Mobile Data Management*,
DOI 10.1007/978-981-10-3941-6_11

1. Locating a node for delivery of message, and
2. Transparent semantic routing of the message to the node.

Consequently, any attempt to map existing distributed algorithms for execution in mobile computing environment in a simple way is unlikely to meet much success. Nevertheless, observing the differences between mobile computing and distributed systems will help in recognizing the issues that may arise in design of distributed algorithms or restructuring existing distributed algorithms for execution on mobile computing systems.

11.2 Distributed Systems and Algorithms

A distributed system consists of a set of autonomous computers (nodes) which communicate through a wired network. A program which runs on distributed system is typically organized as a collection of processes distributed over different nodes. A distributed computation consists of four iterative steps:

1. Broadcasting,
2. Gathering information,
3. Executing joint computation, and
4. Agreeing on coordinated actions.

The last three steps are closely related. Information gathering in a distributed setup requires the participants to share local information amongst themselves. Similarly, the progress of a joint computation requires exchange of partial results amongst the participants. In fact, any coordinated action requires information sharing. Since the processes are distributed over a set of autonomous computers, all such synchronization requirements can be met either through a shared memory or through exchange of messages over the network among the nodes. A shared memory in a distributed system is implemented at the software level either transparently by extending the underlying virtual memory architecture, or explicitly through a set of library functions. In other words, message passing is the basic interface for sharing and exchanging of information between computers in a distributed system.

All distributed algorithms are designed with following basic assumptions about the capabilities of the participating nodes:

1. The nodes are static and their locations (IP/MAC addresses) are known in advance. No cost is incurred for locating a host.
2. The participating nodes are resource rich, having enough computation power, memory.
3. The nodes are powered by continuous supply of power, and remain active during the execution of programs.
4. The inability to receive a message by a node, due to a power failure, is treated as a failure of the algorithm.

5. The message setup cost is fixed, and same for all the messages. The latency due to message transmission dominates communication cost.
6. The size of a message is limited by size of MTU supported by network, and the transmission cost of a message between two fixed nodes is fixed.
7. Sufficient bandwidth is available for transfer of messages.
8. Transmission of a large amount of data between two nodes is accomplished by fragmenting it into several messages, and transmitting each of these messages separately.

11.3 Mobile Systems and Algorithms

Before dealing with the design of distributed algorithms for mobile environment, there is need to understand how the efficiencies of such algorithms can be evaluated. The evaluation criteria influence the design of efficient algorithms. The efficiency requirements of a distributed algorithm for mobile distributed environment should focus on:

- Minimization of communication cost,
- Minimization of bandwidth requirement,
- Meeting all the synchronization requirements, and
- Overcoming the 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 a key issue for the correct execution of any distributed algorithm. Unlike static clients, mobile clients can appear and disappear in any cell of a service area at any time. The synchronization techniques have to be adjusted to handle dynamically changing locations of the peers. Thus, there is a need to evolve of a new model for evaluating the cost of distributed algorithms in mobile environments. Some of the easily identifiable cost criteria are:

- Computation on a mobile node versus that on a static node,
- Relocating computation to static host, and
- Communication on wireless channels,

There is also a number of other characteristics of a mobile distributed system which influence the cost computation. In particular, network *disconnection* and *reconnection* introduce complications in evaluation of the cost. Furthermore, the cost model applicable to mobile infrastructured network cannot directly be extended to infrastructureless mobile ad hoc networks. Therefore, separate cost models have to be evolved for different mobile distributed environments.

Finally, the cost model is of little help unless, algorithm designer adopt appropriate strategies in design of algorithms. In this connection, two major issues which an algorithm designer must appropriately address are:

- How a computation in a mobile environment can be modeled?
- How synchronization and contention problems arising thereof can be resolved?

11.3.1 Placing Computation

Whenever an operation is executed on a remote object, at first a message is sent to the node that hosts the object. The desired operation is then performed by the remote node on behalf of the initiating host. It is convenient to assume that the mobile host *logically* executes the required set of operations directly on a remote node by sending a message. Sending a message to a mobile host is a two-step process:

1. The first step is to locate the mobile host.
2. The next step is to actually send the message.

If destination of the message is a fixed host, the above two steps can be carried out by the base station (BS) of the source mobile node within the fixed network. The BS being a part of fixed network would be able to forward the message to the destination node by using the IP forwarding protocol. It does not involve location search. This implies the first step is unnecessary for a destination which is a static node. Thus, sending a message to a fixed host is a lot cheaper than sending a message to a mobile host. Therefore, it is preferable to avoid sending messages to mobile hosts except for the case when both sender and the receiver are under the same BS. Since a mobile host should try to avoid sending messages to another mobile host, executing operations on objects resident in another mobile host should be avoided. So, the first design principle is:

Principle 1 *[7] To the extent possible, all remotely accessed objects should be resident on the fixed hosts.*

In other words, a node which hosts an object, requires both computing power and bandwidth. Therefore, frequently accessed objects should not be stored in mobile hosts. The role of a mobile host in a thread of execution is normally restricted to initiating operations on a remote object by sending message to the fixed node holding the object.

11.3.2 Synchronization and Contention

Whenever a particular resource is concurrently accessed from a number of remote agents, the competing agents should follow a well defined contention resolution protocol. We can treat each resource as an object. A sequence of operations being initiated from a specific place (a mobile host) can be called a *thread* of execution. Thus, an execution scenario is represented by many concurrently running threads trying to operate on an object. For the moment, let us not make any assumptions

about where this object is located (at the risk of violating Principle 1). It may be either be resident on a fixed host or on a mobile host. The concurrent threads compete to gain access to the object.

Concurrent operations on an object by the competing threads should not leave the object in an inconsistent state. In other words, any attempt to access a shared object should be synchronized by mutual exclusion of competing threads. Let us examine the issue of object consistency a bit more to understand why mutual exclusion is an important issue for synchronization in a distributed settings. The execution of distributed algorithms can be visualised as a repeated pattern of *communication* followed by *computation*. The computation is limited to the individual hosts and during the execution of a computation, a host may need to communicate with its neighbors or other nodes for exchanging the results of partial computations. So the progress of a computation also needs synchronization. The synchronization requirements, among other things, may involve initialization of parameters for the next phase of computation. Thus a distributed system of hosts exhibit repeated bursts of communication in between the periods of local computations.

When hosts become mobile, one additional cost parameter, namely, *cost of location lookup* is introduced. Furthermore, due to resource poorness in mobile hosts, the cost assignment criteria for computational resources become drastically different from that used for fixed host. The communication cost also needs to distinguish between messaging over the wired and the wireless links. A simple technique, to avoid the high resource cost at mobile hosts is to relocate compute intensive parts of an algorithm to the fixed hosts as much as possible. Badrinath, Acharya and Imielinski [7] proposed three different strategies, namely, the *search*, *inform* and *proxy* to handle the issue of location search cost in the context of restructuring distributed mutual exclusion algorithm on a logical ring network. The techniques proposed by them are generic in nature and, therefore, can be used in synchronization requirements of distributed algorithms such as executing critical section of a code. We examine these strategies in this section.

11.3.3 Messaging Cost

The most important complexity measures of any distributed algorithm is the communication cost. It is dependent on the number of messages exchanged during one execution of the algorithm. But when hosts are mobile, the communication complexity should also include the cost of location search. Location search includes the messages exchanged to locate a mobile host in the coverage area. We know that mobile hosts have severe power constraints. They can transmit messages only on wireless links which require substantial amount of power. Furthermore, wireless links offer low bandwidth. Therefore, the cost of communication over a wireless link is more expensive than the cost of communication over a wired link. Badrinath, Acharya and Imielinski [7] proposed three different measures of cost for counting the number of messages exchanged during the execution of a distributed algorithms

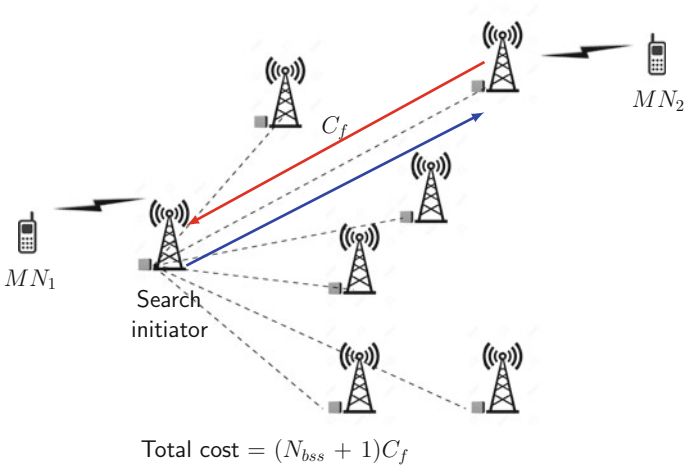

Fig. 11.1 Search cost

in a mobile computing environment. The cost model is specified by defining the units of cost as follows:

- C_w: cost of sending a message from MH to BS over wireless channel (and also identical to the cost in the reverse direction)
- C_f: cost of sending a message from a static node to another another static by the wired N/W.
- C_s: cost of searching/locating the current base station BS_{cur} of an MH and forwarding a message from a source base station BS_{src} to BS_{cur}.

C_w is assumed to represent a higher multiplicative cost compared to C_f. We may, therefore, assume C_w to be equivalent to k. C_f, where $k > 1$ is an appropriately chosen constant.

The simplest strategy to locate a mobile host is to let the searching base station query all the other base stations in the coverage area. The base station which responds to the query is the one which services 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, the messages exchanged for a location search as illustrated by Fig. 11.1 are:

1. In the first round, all the base stations, except one, receive message from the querying base station. It requires exchange of $(N_{BS} - 1) \times C_f$ messages.
2. The base station, servicing the searched mobile host, responds. It incurs a cost of C_f.
3. Finally the querying base station forwards a message (data packet) to the responding base station. This incurs a cost of C_f.

Adding all the three costs, the worst case cost for a search:

$$C_s = (N_{BS} + 1) \times C_f.$$

The cost of transmitting a message from a mobile host (MH) to another mobile host (MH′) is determined by overhead of search, and the cost of actual message transfer. The break down of the cost is given below.

1. The source MH sends the message to its own base station BS. The cost incurred for the same is: C_w
2. BS then initiates a search for the destination MH′ to locate its base station BS′ under whose cell area MH′ is currently active. BS delivers the message to BS′. The cost of the locating MH′ and delivering message to BS′, as explained, is C_s.
3. After receiving the message from BS, BS′ delivers it to MH′. This action incurs a cost of C_w.

Now adding all the costs together, the worst case cost of transmitting a message from a mobile host MH to another mobile host MH′ in the worst case is:

$$2C_w + C_s.$$

Figure 11.2 explains how a message from a source mobile can be delivered to a destination mobile.

The analysis of the cost structure which Badrinath, Acharya and Imielinski [7] have proposed, is captured by Fig. 11.3. It succinctly explains two aspects, namely,

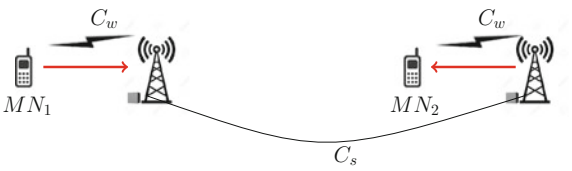

Fig. 11.2 Mobile to mobile communication

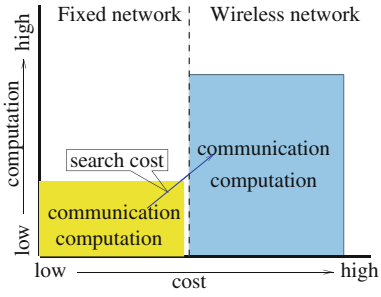

Fig. 11.3 Summary of cost model

- The major component of the cost in a mobile distributed environment is due to communication.
- However, the computation is also slow in mobile hosts. So, in the cost due to computation is relatively high compared to the cost in conventional distributed system.

Relocating computation on the fixed host makes sense. But, relocation may mean communication between a fixed host and a mobile host. After computation is over, the fixed host needs to report the result back to the mobile host. This would need a search for locating the mobile host.

11.4 Structuring Distributed Algorithms

Attempts to execute distributed algorithms directly without any restructuring for mobile environment may lead to design of inefficient algorithms. Inefficiencies, in design of algorithms for mobile distributed environment, as observed in the previous section arise out of synchronization, the asymmetry in model of the computation, and the imbalance in communication cost between wired and wireless interfaces. From the point of view of algorithm design, the problems are dependent on the abstract notions of coordination and control in distributed algorithms. As an analogy, consider the issue of quality in software design process. The efficiency of algorithms addresses the quality in the problem domain. But when algorithms are converted to software processes, the issue of quality goes beyond the problem domain. It becomes linked to the choice of technologies for the implementation such as computers and their capabilities, underlying network, storage, programming tools and languages, etc.

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

- Non-coordinator based systems:
- Coordinator based systems.

11.5 Non-coordinator Systems

In a non-coordinator based system, all the machines are equivalent. Therefor, no machine can exercise any control on another machine. A non-coordinator system is known more popularly as a peer-to-peer system. There are two different types of non-coordinator based systems. In the first type of non-coordinator based system, each peer execute the same code. In the second type, a few of the machines execute some specialized code, while the rest execute the same code.

11.5.1 All Machines are Equivalent

In such a system, each machine roughly shares the same amount of computational and communication load. Such systems can easily be modified to work in a mobile computing environment. We illustrate this with Lamport's Bakery algorithm [8] for mutual exclusion. In the Bakery algorithm, a process waiting to enter the critical section chooses a number. It allows all processes which have chosen smaller numbers to enter into the critical section before itself. The ties are resolved by process IDs, allowing the process with the lower ID to enter the critical section. Lamport's original algorithm makes use of two shared arrays, each consisting of n elements. One element is assigned for each process in each shared array. The values in a shared array can be examined by any process. In a peer to peer settings, no shared arrays can be used. So, local variables `choose` and `value` are maintained by each process. A process uses message passing mechanism when it needs to examine the values of local variables of another process. The pseudo code of the algorithm appears in Algorithm 13.

Algorithm 13: Lamport's bakery algorithm

```
boolean choosing_i = false;
int number_i = 0;
while (1) do
    choosing_i = true;
    set value_i = max {value_j | j ≠ i, j = 0 ... N_MH − 1} + 1;
    choosing_i = false;
    for (j = 0; j < N_MH, j != i; j++) do
        while (choosing_j) do
            {busy wait ...}
        end
        while (number_j != 0) && ((number_j, j) < (number_i, i)) do
        end
    end
    .
    .
    .
    { Critical section code}
    .
    .
    .
    number_i = 0;
end
```

Lamport's Bakery algorithm requires very little computation, and can be easily performed on a mobile device. So, ignoring computation we may just focus on the communication aspects. The execution of the algorithm in a mobile host MH can be analyzed in three distinct parts.

1. Choose own number.
2. Wait for the mobile hosts with lower numbers to avail their turns.
3. Execute the critical section.

In the first part of the execution, an MH fetches the numbers chosen by other $N_{MH} - 1$ mobile hosts in order to set its own number. So, an MH sends a query to all other mobile hosts for fetching their respective local numbers. As the end hosts are mobile, fetching each number involves location search for the other end host. It is assumed that the requester does not move when it is waiting for the replies to arrive. So, location search is not required for the delivery of the replies. Therefore, the message cost incurred for fetching the number chosen by another mobile host $(2C_w + C_s)$. The overall the message cost incurred by MH for choosing its own number is, therefore, equal to

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

In the second part of the execution, a requesting mobile host MH waits for all mobile hosts to choose their numbers. Then subsequently, allow those hosts to execute the critical section if their chosen numbers are smaller than the number chosen by MH. The waiting part consists of two steps. It requires communication with other mobile hosts to allow them choose their respective numbers and then allow the mobile host having a smaller number to execute critical section. At the worst, a MH has to wait till all other mobile hosts have finished choosing their respective numbers, and availed their respective turns to enter the critical section assuming each one of them has chosen a number smaller than the MH. The message cost involved in waiting for one mobile host is $2(2C_w + C_s)$. In the worst case, an MH may have to wait for $N_{MH} - 1$ other mobile hosts to take their respective turns before the MH can enter the critical section. This leads to an overall message cost of:

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

The execution of code for critical section may perhaps involve some common resources and possibly subsequent updates of those resources. It does not involve any communication with other mobile hosts. Adding the message cost of three parts, the overall communication cost for execution of Bakery algorithm only on mobile hosts is

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

Therefore, a straightforward way of mapping Lamport's Bakery algorithm to mobile peer to peer distributed system leads to a message cost of the order $6N_{MH} \times C_w$.

The correctness of the algorithm is heavily dependent on the fact that the messages are delivered in FIFO order. However, maintaining a logical FIFO channel between every pair of mobile hosts has to be supported by the underlay network.

11.5.2 With Exception Machines

This system similar to the previous category, where most of the machines execute the same code, except only a few of them, which execute a different code. The machines which execute a different code are known as *exception* machines. An example of this category is Dijkstra's self stabilizing algorithm [9]. It is a system consisting of a set of n finite state machines connected in the form of a ring, with a *token* or privilege circulate around the ring. The possession of the token enables a machine to change its state. Typically, for each machine, the privilege state is defined if the value of a predicate is true. The predicate is a boolean function of a machine's own state and the states of its neighbors.

The change of current state of a machine is viewed as a *move*. The system is defined to be *self-stabilizing* if and only if, regardless of the initial state and token selected each time, at least one token (privilege) is present and the system converges to a legal configuration after a finite number of steps. In the presence of multiple tokens in the system, the machine 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 token in the system.
- *Closure*: Every move from a legal state must place the system into a legal state. It means, once the system enters a legal state no future state can be illegal.
- *No starvation*: During an infinite execution, each machine should possess a token for an infinite number of times
- *Reachability*: Given any two legal states, there is a series of moves that change one legal state to the other.

Let us now look at Dijkstra's algorithm involving K states where $K > n$, and system consists of $n+1$ machines. Machine 0 is called the bottom machine, and the machine n is called the top machine. All the machines together form a logical ring, where the machine i has the machine $i+1 \mod (n+1)$ as its right hand neighbor, $i = 0, 1, \ldots, n$. The legitimate states are those in which exactly one privilege is present.

For any machine, let the symbols S, L, R respectively denote the machine's own state, the state of left neighbor, and the state of the right neighbor. The rules for change of states for this system are as follows.

- **Bottom machine**
  if $L = S$ then $S = (S+1) \mod K$.
- **Other machines**
  if $L \neq S$ then $S = L$.

An initial configuration C_0 may consists of at most $n+1$ different states. At least $K-(n+1)$ states do not occur in C_0. Machine 0 increments its state only after $n+1$ steps. Therefore, it reaches a state not in the initial configuration after at most $n+1$ steps. All other machines $i \neq 0$, copy states of their respective left neighbors. Hence, the first time Machine 0 computes its state, such a state becomes unique in the ring. Machine 0 does not get a chance to compute its state until the configuration reaches $S_1 = S_2, \ldots, S_n = S_0$.

The details of how the self-stabilization works is not important for structuring distributed algorithms to mobile environment. Let us look at the communication that occurs between two machines. In 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 does not make much of a difference to the communication costs involved. However, only the neighbor's values are needed for the change of state. As most of the neighboring hosts are expected to be under the same BS except for two at the edges, the overhead of wireless communication is expected to be low.

In Dijkstra's exception machine model, a single token or privilege circulates around the logical ring. In such a ring organization of mobile hosts, the communication is restricted between a host and its left or right neighbors. If we structure distributed mutual exclusion algorithm using a logical ring of cohorts, then it should be possible to reduce the communication cost. The algorithm becomes very simple, we just let the privileged mobile to access mutual exclusion. If the privileged machine is not interested, it just passes the privilege to the successor in the logical ring. This way every mobile host gets one chance to use a critical resource in a mutually exclusive manner during one full circulation of the token around the ring of mobile hosts.

The analysis of the message cost of the token ring algorithm outlined above (referred to as TR-MH) is provided below.

> Both the sender and the recipient are mobile hosts. The message is sent first from the sender to its local base station then from there to the base station under which the recipient is found. So the cost of messages exchanged on wireless links is $2C_w$. The cost of locating a mobile host and subsequently sending a message to its current base station is C_s.

Therefore the cost of token exchange between two successive MHs in the logical ring is $2C_w + C_s$. Assuming that the ring consists of N_{MH} mobile hosts, the cost of one full circulation token on the ring is $N_{MH}(2C_w + C_s)$. This cost does not include the cost for mutual exclusion requests met during the circulation of the token. Let K be the number of mutual exclusion requests satisfied during one complete circulation of token around the ring. The maximum number of mutual exclusion that can be met in one circulation of token is $\max\{K\} = N_{MH}$.

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

$$2N_{MH}C_w.$$

11.5.3 Coordinator Based Systems

Many distributed algorithms involve a coordinator. The coordinator bears substantially higher communication overhead compared to other participating nodes. It is basically responsible for resolving the coordination issues related to synchronization. A coordinator may or may not be fixed. In a fixed coordinator based system, one node is assigned the role of the coordinator for the entire duration of execution of the algorithm. However, in a moving coordinator based system the role of coordinator can be performed by different hosts at different times. Thus the coordinator is a function of time.

11.5.3.1 Fixed Coordinator Based System

Apart from the normal optimization for the mobile hosts, the communication pattern involving the coordinator has to be specifically optimized. Such a strategy yields better dividends in terms of reducing the communication cost, because most of the communication load in a coordinator based system is centered around the coordinator. Apart from increased communication load, it increases the probability of a failure, as the coordinator is a single point of failure.

An example of the fixed coordinator system is encountered in the case of total ordered atomic broadcast algorithms. The system consists of N hosts, each wishes to broadcast messages to the other hosts. After a broadcast message is received, a host time stamps the message and sends the same to a sequencer. On receiving the relayed broadcast message from all the nodes, the sequencer sets the time stamp of the message to the maximum of the received time stamps and then broadcast the same back to all the hosts. In this way, the coordinator ensures a total ordering of the broadcast messages. Figure 11.4 illustrates this process in a sequence diagram. Host 3 in Fig. 11.4 is the sequencer or the coordinator. The execution of the above atomic broadcast algorithm is mainly dependent on the coordinator. Therefore, in order to structure the algorithms for execution in a mobile environment, we must first turn our attention to the role of coordinator in a mobile environment. Obviously, if the coordinator is mobile then execution of any coordinated action will be expensive. It is, therefore, recommended that a static host should perform the coordinator's job. Since, the other hosts are mobile, search, inform or proxy strategies can be applied depending on the mobility characteristics of entities in 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. The cost of initial broadcast: $(N_{MH} - 1) \times (C_s + 2C_w)$,
2. The cost of unicasting received message from the participating nodes to the coordinator: $(N_{MH} - 1) \times (C_s + 2C_w)$,
3. The cost of sending time stamped messages back to participants: $(N_{MH} - 1) \times (C_s + 2C_w)$

Therefore, the overall messaging cost is

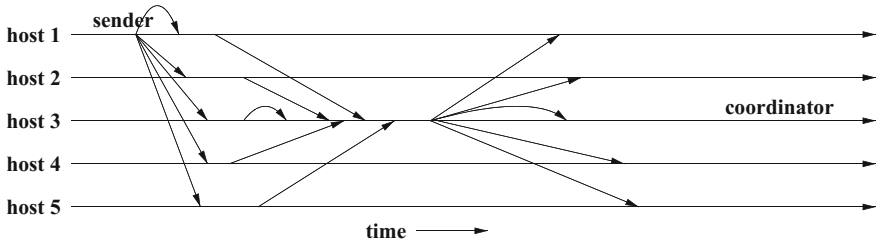

Fig. 11.4 Atomic broadcast using fixed/moving coordinator

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

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

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

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

1. Cost of initial broadcast: $C_w + (N_{BS} - 1) \times C_f$,
2. Cost of informing receipt timestamps to the coordinator: $(N_{BS} - 1) \times C_f + N_{MH} \times C_w$,
3. Cost of broadcasting coordinator's final timestamp to participants: $(N_{BS} - 1) \times C_f + N_{MH} \times C_w$.

This leads to a overall message cost of

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

Thus simple structuring of the atomic broadcast algorithm done by placing the coordinator on a base station leads to substantial savings in the cost of messaging.

11.5.3.2 Moving Coordinator Based System

As shown in Fig. 11.5 the coordinator of the algorithm changes over time. The sender, the coordinator and the receiver sets are the identical, but shown separately for the sake of clarity.

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. One of the three strategies, search, inform and proxy, can be used for all the hosts in the system.
2. As soon as a mobile host becomes a coordinator, the communication load on it rises drastically in a short space of time. Hence the MH should inform its BS

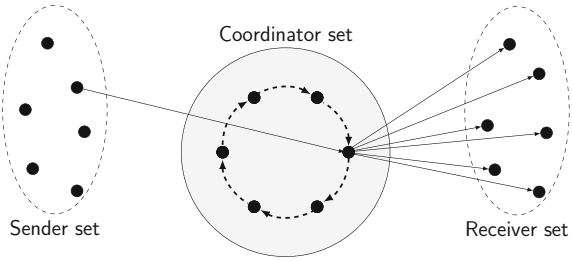

Fig. 11.5 Conceptual model of a moving coordinator system

about change of status to coordinator, which is then broadcast to all base stations. Also the MH during its tenure as the 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: $(N_{BS} - 1) \times C_f + N_{MH} \times C_w$,
3. Cost of broadcasting back the time stamped message: $(N_{BS} - 1) \times C_f + N_{MH} \times C_w$, and
4. Additional overhead associated with change of coordinator: $(N_{BS} - 1) \times C_f + N_{MH} \times C_w$.

Thus the total cost works out as:

$$(2N_{MH} + 1) \times C_w + 3(N_{BS} - 1)C_f + \alpha(N_{MH} \times C_w + (N_{BS} - 1) \times C_f),$$

where a change of the coordinator occurs every α broadcasts. The cost of executing the non-structured version of algorithm is: $3(N_{MH} - 1) \times (C_s + 2C_w)$. So, the saving in cost is significant by simple structuring.

11.6 Exploiting Asymmetry of Two-Tier Model

Most distributed algorithms can be structured suitably for execution on mobile environment by reducing communication costs. Still, if we consider the token ring algorithm described in Sect. 11.5.1, two of the key issues are not addressed, viz.,

1. Every mobile in logical ring has to maintain active network connectivity during the execution of the algorithm.
2. Relocation of computation to balance inherent asymmetry in mobile computing environment is not possible.

For example, none of the mobile hosts in the token ring algorithm can operate either in disconnected or in doze mode during the execution. This is because the token cannot be sent to a disconnected successor node, and also if the mobile, holding

the token decides to operate in disconnected mode then other mobiles may have to wait indefinitely to get their turns. The organization of distributed mobile system point towards a two-tier model with inherent asymmetry in node capabilities. The fixed nodes do not suffer from any of the resource related problem which the mobile nodes have. So, in order to balance the inherent asymmetry in the system, if the token circulation is carried out by fixed hosts, then it may be possible for the mobile hosts to operate in disconnected or doze mode. Furthermore, it may also be possible to relocate compute intensive tasks to the fixed hosts. This strategy not only removes the burden on resources of mobile nodes, but also enhances the performance of algorithms.

Using the above two-tier approach, Badrinath, Acharya and Imielinski [7] proposed three different variations for structuring of token ring algorithm. Their main strategy was based on exploiting the inherent asymmetry in computation model as indicated design Principle 1 of Sect. 11.3.1. The token is assumed to circulate on a ring in a previously determined sequence among the fixed hosts in infrastructured part of the network supporting the mobile computation. A mobile host MH wishing to access the token submits the request to its current BS. When the token becomes available, it is sent to MH at its current base station, BS′. After using the token MH returns it to BS′ which in turn returns the same back to BS. The cost of servicing token depends on the way location of a MH is maintained.

11.6.1 Search Strategy

Pure search strategy scans the entire area under coverage of service to find a MH. The algorithm consists of set of actions performed by the two component devices, namely, base stations and mobile hosts. Each base station assumed to maintain two separate queues: (i) a request queue Q_{req}, and (ii) a grant queue Q_{grant}. The token access requests by mobile hosts at a base station are queued up in Q_{req}. When the token is received by a BS from its predecessor in the logical ring, all pending requests are moved into Q_{grant}. Then all the requests are serviced from Q_{grant}, while new requests get added to Q_{req}. This implies all the requests made before the time token arrives are serviced and the requests which arrive subsequently are kept pending for the next round of servicing. After all requests are serviced, the token is passed on to the successor base station in the ring. So, the actions of BS are as provided in Algorithm 14 [7]: As far as a mobile MH is concerned, it can request for token servicing at any point of time to its base station. Once token is received from the base station, it uses the critical resource and returns the token after the use. So, the actions performed by a MH are as indicated in Algorithm 15. The correctness of algorithm relies on several implicit assumptions.

1. Firstly, all the message channels are assumed to be reliable and received in FIFO order. FIFO order means that the messages actually arrive and in the order they are sent. So, a mobile sends only one request at a time.

Algorithm 14: Search strategy: actions of BS

```
begin
    on receipt of (an ME request from MH) begin
        add MH's request to the rear of Q_req;
    end
    on receipt of (the token from the predecessor in the ring) begin
        move all pending requests from Q_grant;
        repeat
            remove request from the head of Q_grant;
            if MH which made the request is local to BS then
                deliver the token to MH over wireless link;
            end
            else
                search and deliver token to MH at its current cell;
            end
            await return of token from the MH;
        until (Q_grant == empty);
        forward token to BS's successor in the logical ring;
    end
end
```

Algorithm 15: Search strategy: actions of MH

```
on requirement for (an access of the token) begin
    submit request to current local BS;
end
on receipt of (token from local BS) begin
    hold the token and use the critical resource;
    return the token to local BS;
end
```

2. Secondly, a mobile cannot make a fresh request at a base station where its previous request is pending.
3. Thirdly, the algorithm does not handle the case of missing or captured token.
4. Fourthly, a mobile can hold token only for short finite duration of time.

With the above assumptions, it is clear that at any time only one MH holds the token. Therefore, mutual exclusion is trivially guaranteed. The token is returned to the same base station from which it was received. So, after the token is returned back, a base station can continue to use it until all the pending requests before the arrival of token have been satisfied. The maximum number of requests that can be serviced at any base station is bounded by the size of the Q_{grant} at that base station when the token arrives. No fresh request can go into Q_{grant}, as they are added only to Q_{req}. So, a token acquired by a base station will be returned after a finite time by servicing at most all the requests which were received before the arrival of the token. This number cannot exceed N_{MH}, the total number of mobile hosts in the system. It implies that the token will eventually reach each base station in the ring and, therefore, all the requests are eventually satisfied in finite time.

It is possible, however, for a mobile to get serviced multiple number of times during one circulation of token over the ring. It can happen in the following way. A mobile submits a request at one base station BS and after being serviced by the token, moves quickly to the successor of BS in the logical ring and submits a fresh request for the token at the successor. Though it does not lead to starving, a stationary or slow moving mobile may have to wait for a long time to get its turn. We will look into a solution to this problem in Sect. 11.6.3.1.

The communication cost of the above algorithm can be analyzed as follows:

1. Cost for one complete traversal of the logical ring 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 a BS receiving the token then the cost of servicing a request is C_w. But if the requesting MH has migrated to different base station BS$'$ before the token reaches BS where the request was initially made, then a location search will be required. So the worstcase cost of delivering token to the requesting MH will be $C_s + C_w$.
4. The worstcase cost of returning the token to BS delivering the token to MH is $C_w + C_f$. C_f component in cost comes from the fact that MH may have subsequently migrated from BS where it made request to the cell under a different base station BS$'$.

Adding all the cost components, the worstcase cost of submitting a single request and satisfying is equal to $3C_w + C_f + C_s$. If K requests are met in a single traversal of the ring then the cost will be

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

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

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.

- *Energy consumption*. The energy consumption in the present algorithm is proportional to $3K$ as only $3K$ messages are exchanged over wireless links. In TR-MH algorithm, it was $2N_{MH}$. As $K \ll N_{MH}$, 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.

11.6.2 Inform Strategy

Inform strategy reduces the search cost. It is based on simple idea that the search becomes faster if enough footprints of the search object is available before search begins. In other words, a search is less expensive if more information is available

about the possible locations of the mobile host being searched for. Essentially, the cost of search increases with the degree of imprecision in locations information. For example, if a MH reports about every change of its location to the BS, where it initially submitted a token request, then search is not needed. The difference between a pure search based algorithm to an inform based algorithm is that Q_{req} maintains the location area information along with the request made by a mobile host, and every mobile host with a pending request informs the change in location to the base station where the request is made. Algorithm 16 specifies the actions of a BS [7]. The actions to be performed by a mobile host *MH* are provided in Algorithm 17.

Algorithm 16: Inform strategy: actions of BS

```
begin
    on receipt of (a request from a local MH) begin
        add the request < MH, BS > to the rear of Q_req;
    end
    on receipt of (inform(MH, BS') message) begin
        replace < MH, BS > in Q_req by < MH, BS' >;
    end
    on receipt of (token from the predecessor of BS in ring) begin
        move Q_req entries to the Q_grant;
        repeat
            remove the request < MH, BS' > at the head of Q_grant;
            if (BS' == BS) then
                deliver the token to MH over the local wireless link;
            end
            else
                forward token to BS' for delivery to MH;
            end
            await return of the token from MH;
        until (Q_grant == empty);
        forward token to BS's successor in the ring;
    end
end
```

To compare the search and inform strategies, we observe the following facts about the search strategy:

- An MH makes *MOB* number of moves in the period between the submission of request and the receipt of token.
- After each of these moves, a *inform*() message is 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.
- 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 is currently found.

Algorithm 17: Inform strategy: actions of MH

```
on requirement of (token access) begin
    submit request to its current local BS;
    store the current local BS in the local variable req_locn;
end
on receipt of (token from BS req_locn) begin
    access the critical resource or region
    return token to BS (req_locn);
    set req_locn to ⊥; // ⊥ indicates null
end
on a move by MH begin
    send join(MH, req_locn) message to local BS
end
on entering (cell under a new BS) begin
    if (req_locn ≠ ⊥) then
        send a inform(MH, BS′) to BS (req_locn)
    end
end
```

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 more cost effective compared to the search strategy provided $MOB \times C_f < C_s$. In other words if after submitting a request, the frequency of movement of a MH becomes low then MH should inform BS about every change of location rather than BS searching for it.

11.6.3 Proxy Strategy

Proxy strategy incorporates the ideas from both search and inform strategies. It exploits the imbalance between the frequencies of the local and the global moves made by a mobile host. Usually, a mobile host moves more frequently between cells that are adjacent and clustered around a local area. Using this knowledge, the entire coverage area consisting of all base stations is partitioned into contiguous *regions*, where each region consists of a cluster of neighboring BSes. BSes within a region are associated with a common proxy. A proxy is a static host, not necessarily a base station. The token now circulates in a logical ring comprising of these proxies. Each proxy, on receiving the token, becomes responsible for servicing the requests pending in its request queue. So, the proxy strategy is just a minor variation of inform strategy that can capture locality of moves made by mobile hosts. Only, implicit assumption is that a mobile host must be aware of the the proxy assigned for handling token requests originating from its current cell. Since the movements of a mobile host MH at times can be unpredictable, it may not be possible for MH to pre-store the identity of its proxy. So, each BS may send out periodic beacons that include the identity of its

associated proxy. The actions executed by a proxy P are provided by Algorithm 18. Similarly, the actions executed by mobile host MH are provided in Algorithm 19 For convenience in analysis of proxy strategy, we use following notations [7]:

Algorithm 18: Proxy strategy: actions of proxy

```
begin
    on receipt of (an ME request from MH) begin
        // Request is forwarded by a BS within P's local area
        add request < MH, P > to the rear of Q_req;
    end
    on receipt of (a inform(MH, P') message) begin
        replace request (MH, P) in Q_req by (MH, P');
    end
    on receipt of (token from the predecessor in ring) begin
        move requests from Q_req to Q_grant;
        repeat
            delete request < MH, P' > from head of Q_req;
            if (P' == P) then
                // MH located within P's area
                deliver the token to MH after local search;
            end
            else
                // MH is in a different proxy area
                forward the token to P' which delivers it to MH;
            end
            await return of the token from MH;
        until (Q_grant == empty);
        forward token to P's successor in ring;
    end
end
```

1. N_{proxy}: denotes the total number of proxies forming the ring.
2. N_{BS}: denotes the number of BSes in the coverage area, which means there are N_{BS}/N_{proxy} base stations under a region.
3. MOB_{wide}: denotes the number of inter regional moves made by a MH in the period between submitting a token request and receiving the token.
4. MOB_{local}: denotes 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 BSes within its region. This search is referred to as local search with an associated cost C_{ls}. With the above notations and assumptions, according to Badrinath, Acharya and Imielinsk [7] the communication costs can be analyzed as follows:

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

Algorithm 19: Proxy strategy: actions of *MH*

```
on requirement (for access of token) begin
    submit request < MH, P) > to local BS;
    store the identity of local proxy in init_proxy;
end
on the receipt of (token from init_proxy) begin
    use the token;
    return the token to init_proxy;
    set init_proxy to ⊥;
end
on a inter regional move begin
    send a join(MH, init_proxy) message to new BS;
    if (init_proxy ∉ {P, ⊥}) then
        // P′ is the proxy of new BS
        new BS sends a inform(MH, P′) to init_proxy;
    end
end
```

3. The cost of delivering the token to the MH: $C_f + C_{ls} + C_w$
   C_f term can be dropped from the cost expression above, if MH receives the token in the same region where it submitted its request.
4. The cost of returning the token from the MH to the proxy: $C_w + C_f$

The above costs together 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

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

If the token gets circulated on the set of proxies instead of the set of BSes, then the cost of circulation is reduced by a factor of N_{proxy}/N_{BS}. However, the workload is comparatively higher on each proxy than the workload on a BS. Assuming all three schemes service identical number of mutual exclusion requests in one full circulation of ring,

- N_{BS} static hosts share the load in the search strategy,
- N_{proxy} static hosts share the load under the proxy method.

The efficiency in handling mobility by each strategy can be compared by estimating the communication cost in satisfying one token request.

search strategy: $3C_w + C_f + C_s$
inform strategy: $3C_w + C_f + (MOB \times C_f)$
proxy strategy: $3C_w + (3 + MOB_{wide}) \times C_f + C_{ls}$

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

$$\begin{aligned} &(3 + MOB_{wide}) \times C_f + C_{ls} < C_f + C_s \\ \equiv\ &MOB_{wide} + 2 < (C_s - C_{ls})/C_f \end{aligned} \tag{11.1}$$

Search strategy requires a BS to query all the other BSes within a search region to determine if a MH is active in a cell. The BS which currently hosts the MH responds. Then the BS where MH originally submitted the request forwards the token to the responding BS. The search cost is then equal to $(N_{region} + 1) \times C_f$, where N_{region} denotes the number of BSes within a search area. Replacing C_{ls} in Eq. 11.1 by above expression for search, we find:

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

From the above expression, we may conclude that the proxy scheme performs better than the pure search scheme if the number of inter regional moves is two less than the total number of BSes outside a given region.

Now let us compare proxy with inform strategy. Proxy strategy is expected to incur a lower cost provided:

$$\begin{aligned} &(3 + MOB_{wide}) \times C_f + C_{ls} < (MOB + 1) \times C_f \\ \equiv\ &C_{ls} < (MOB - MOB_{wide} - 2) \times C_f \\ \equiv\ &C_{ls} < (MOB_{local} - 2) \times C_f \end{aligned} \tag{11.2}$$

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 11.2 above reduces to:

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

From the above expression, we conclude that if the number of local area moves performed by a mobile host exceeds the average number of BSes under each proxy by just 2, then the proxy strategy outperforms the inform strategy.

11.6.3.1 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 represented by the following sequence of events:

1. A mobile host MH submits a request to its current local base station BS.
2. It gets the token from the same BS and uses it,

3. It then moves to the base station BS′ which is the next recipient of the token,
4. The same MH submits a request for the token access at BS′.

The situation leads a fast moving mobile host to gain multiple accesses to the token during one circulation of the token through the fixed hosts. It violates the *fairness* property of the token access among the 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 or proxy may cause starvation. Because a stationary mobile host is guaranteed to get its request granted when the token arrives in its local base station. We have already observed that the length of 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's state is represented by loop count (`token_val`). It represents the number of complete circulations performed by token around the logical ring.
2. A local count `access_count` is attached to each MH. It stores the number of successful token accesses made by the MH.
3. When making an access request each MH provides the current value of access count.
4. When a BS (or the proxy) receives the token, only requests with `access_count` less than the `token_val` are moved from the request queue to the grant queue.
5. MH after using the token, copies the value of `token_val` to its local `access_count`.

A MH reset its `access_count` to `token_val` after each access of the token. Therefore, a MH may access token only once in one full circulation of the token around the logical ring, even if the MH has a lower `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 BSes), 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 in full circular traversal of the token.

Of course one can devise an alternative definition of "fairness" as in Definition 11.1.

Definition 11.1 (*Fairness* [7]) A mobile host, MH, having a low access count may be allowed multiple accesses to the token during one traversal of the ring, with the

limitation that the total number of accesses made by the MH does not exceed the current `token_val`.

The above definition of fairness implies that if a mobile host MH has not availed its share of token access for a number of token circulations, then MH can access token multiple number of times bounded above by `token_val - access_count`. The above fairness criterion can be easily implemented by simply incrementing `access_count` of a MH on every access.

11.7 Termination Detection

There is a clear evidence that the generic design principle of exploiting asymmetry in 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 the global states of the computation. Recording a global state of a distributed systems is known to be difficult [10]. Therefore, it is difficult to record the termination state of a distributed computation. However, the termination, being one of the stable properties of a distributed computation, can be observed. When a computation terminates, there can be no messages in transit. Therefore, it is possible to design an 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 a mobile distributed system, termination detection is more complex. The complexity is due to the fact that the detection algorithm should also handle the issues 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.

11.7.1 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 [11]. 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 $n2$. Every node in a distributed computation starts with a a *neutral state*. Dijkstra and Scholten [11] 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 successor. 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 [12–14], 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.

11.7.2 Approach for Mobile Distributed Systems

Termination detection in mobile distributed system follows a a hybrid approach [15]. 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 cannot carry out any basic computation as long as it remains disconnected.

11.7.3 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 [15] 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 types of messages. These message and intended use of these messages can be understood in the context of the protocol discussed later.

11.7.4 Entities and Overview of Their Actions

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

- Mobile nodes,
- 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 $DP \cup \{\cup_i MP_i\}$.

11.7.5 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. So, 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* number of messages to P_i. After the acknowledgement has been sent, *in* is set to 0. The summary of the actions executed by a mobile node in the termination detection protocol is specified in Algorithm 20.

Algorithm 20: Actions of a mobile process P_j^m

```
begin
    // For sending a basic message to another process no
       additional operation is needed.
    on receipt of (a basic message from BS) begin
        // Increment number of messages with pending acks.
        in = in + 1;
    end
    on turning (idle from active) begin
        // Send all the pending acks.
        send M_ack(in) to current base station;
        in = 0;
    end
end
```

11.7.6 Base Stations

Every message to and from a mobile process P_j^m is routed always through P_i, its current base station process. P_i attaches a weight x from its available weights to every message M_b from P_j^m before sending to the base station process P_l for the destination. P_l receives $M_b(x) = M_b + \{x\}$. It extracts the weight x from $M_b(x)$ and adds x to its the current weight. After this, the message $M_b = M_b(x) - \{x\}$ is forwarded to P_k^m. This way any base station process can control the weight throwing part of the protocol on behalf of all the mobile processes it interacts with.

In order to control the diffusion part, a mobile process P_i keeps track of the messages it has sent to every mobile process P_j^m under it. P_i expects P_j^m to send acknowledgement for each and every message it has received. The completion of computation is signaled by sending the acknowledgements for all the messages of P_i. When all the mobile processes P_j^ms involved in a computation have sent their final acknowledgements to P_i, the termination in diffusion part is detected.

The rules for termination detection executed by a base station process P_i are as specified by Algorithm 21.

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.

Algorithm 21: Actions of a BS process P_i

```
begin
    on receipt of (M_b(x) for P_j^m from fixed N/W) begin
        w_i = w_i + x; // Update local weight
        M_b = M_b(x) − {x}; // Remove weight from M_b(x)
        out_i[j] = out_i[j] + 1; // Record number of messages sent to P_j^m.
        forward M_b (without appending x) to P_j^m;
    end
    on receipt of (M_b from P_j^m for P_k^m) begin
        if (P_k^m ∈ MP_i) then
            // P_k^m is local mobile process under P_i
            out_i[k] = out_i[k] + 1;
            forward M_b to P_k^m;
        end
        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; // Reduce current weight held.
            locate MH_k's base station BS_ℓ; // BS process P_ℓ
            send M_b(w_i/2) to P_ℓ;
        end
    end
    on receipt of (M_ack(k) from P_j^m ∈ MP_i) begin
        out_i[j] = out_i[j] − k; // Decrease number of pending acks by k
        if (out_i[k] == 0 for all P_k^m ∈ MP_i) then
            sends M_wr(w_i) to P_c;
            w_i = 0;
        end
    end
end
```

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 the 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.

11.7.7 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 should be 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 the 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 the protocol level support for handoff execution, the mobile process should send $M_{HF.req}$ to BS_i identifying BS_k. BS_i accepts the request and also sends $M_{HF.ack}$ to P_j^m. After this P_j^m starts communicating with BS_k. BS_i, before sending $M_{HF.ack}$ to P_j^m, also transfers the needed data structure to BS_k. The negotiation for transfer of link between BS_i and BS_k may involve messages like $M_{HF.ind}$ and $M_{HF.rep}$ between BS_i and BS_k. The handoff process has been illustrated in Fig. 11.6. To keep the protocol simple, P_j^m has been shown to directly approach BS_k for executing handoff. 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 $w = w_i/2$.

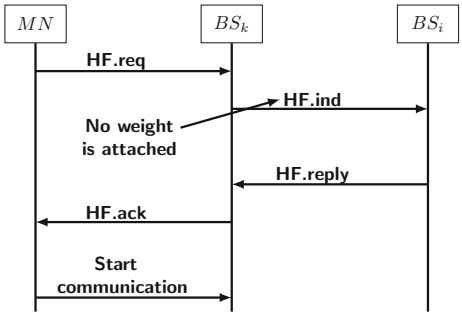

Fig. 11.6 Illustration of handoff process

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 the weight collector 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 mentioned in Algorithm 22.

Algorithm 22: Handoff protocol

begin
  // Actions of P_j^m
  on detection of *(Handofff conditions)* **begin**
    // BS_k is the new BS for P_j^m
    // Assume P_k in BS_k handles the request
    send $M_{HF.req}$ message to P_k and wait for $M_{HF.ack}$;
    if *($M.HF.ack$ received from P_k)* **then**
      start communicating with P_k; // process P_k in BS_k
    end
    else
      after time out retry handoff;
    end
  end
  // Actions of P_i of BS_i
  on receipt of *($M_{HF.ind}(P_j^m)$ from P_k)* **begin**
    send 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 $l \in MP_i$)* **then**
      // P_j^m is the last mobile process under P_i
      send $M_{wr}(w_i)$ to P_c;
      $w_i = 0$;
    end
  end
  // Actions of P_k of BS_k
  on receipt of *($M_{HF.req}$ from P_j^m)* **begin**
    sends a $M_{HF.ind}(P_j^m)$ to P_i; // No weight attached.
  end
  on receipt of *($M_{HF.rep}(P_j^m, out_i[j], w_i/2)$ from P_i)* **begin**
    $MP_k = MP_k \cup \{P_j^m\}$;
    $out_k[j] = out_i[j]$;
    $w_k = w_k + w_i/2$;
    send a $M_{HF.ack}$ to P_j^m;
  end
end

11.7.8 Disconnection and Rejoining

It is assumed that all the 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, summarized in Algorithm 23.

Algorithm 23: Disconnection protocol

```
begin
    // Actions of P_j^m
    before entering (disconnected mode) begin
        send M_DISC.req message to P_i;
        suspend all basic computation;
    end
    // Actions of P^i of BS_i
    on receipt of (M_DISC.req from P_j^m) begin
        suspend all communication with P_j^m;
        send 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 ∈ MP_i) then
            send M_wr(w_i) to P_c;
            w_i = 0;
        end
    end
    // Actions of weight collector P_c
    on receipt of (M_DISC.ind(P_j^m, out_i[j], w_i/2) from P_i) begin
        DP = DP ∪ {P_j^m};
        out_c[j] = out_i[j];
        w_DIS = w_DIS + w_i/2;
    end
end
```

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 Algorithm 24.

11.7.9 Dangling Messages

There may be messages in system which cannot be delivered due to disconnection or handoff. Such dangling messages should be delivered if the mobile process recon-

Algorithm 24: Rejoining protocol

```
begin
    // Actions of P_j^m
    on rejoining (a cell in BS_i) begin
        send M_JOIN.req message to P_i and wait for reply;
        if (M_JOIN.ack is received from P_i) then
            starts basic computation;
        end
        else
            after timeout retry JOIN;
        end
    end
    // Actions of P_i in BS_i
    on receipt of (M_JOIN.req from P_j^m) begin
        send a M_JOIN.ind(P_j^m) to P_c; // No weight attached.
    end
    on receipt of (M_JOIN.rep(P_j^m, out_c[j], w_DIS/2) from P_c) begin
        MP_i^b = MP_i^b + {P_j^m};
        out_i[j] = out_c[j];
        send a M_JOIN.ack to P_j^m;
        restart communication with P_j^m;
    end
    // Actions of weight collector P_c
    on receipt of (M_JOIN.ind(P_j^m) from P_i) begin
        if (P_j^m ∈ DP) then
            sends a M_JOIN.rep(P_j^m, out_c[j], w_DIS/2) to P_i;
            DP = DP − {P_j^m};
            if (DP = Φ) then
                w_c = w_c + w_DIS;
            end
        end
    end
end
```

nects 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 cannot 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. This protocol is specified by Algorithm 25

Algorithm 25: Handling dangling messages

```
begin
    // Actions of weight collector P_c
    on receipt of (Dangling message with weight x) begin
        w_DIS = w_DIS + x;
    end
    on completion of (Reconnection of P_j^m) begin
        // Let M_b be a dangling message for P_j^m
        M_b(w_DIS/2) = M_b + w_DIS/2;
        w_DIS = w_DIS/2;
        // Assume P_j^m reconects under BS_i
        send M_b(w_DIS/2) to P_i;
        if (dangling message list == Φ) then
            // No disconnected mobile exists
            w_c = w_c + w_DIS; // Reclaim residual weight
            w_DIS = 0;
        end
    end
end
```

11.7.10 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 $w_{DIS} > 0$.

References

1. M.H. Dunham, A. Helal, Mobile computing and databases: anything new? SIGMOD Rec. **24**(4), 5–9 (1995). December
2. G. Liu, G. Maguire Jr., A class of mobile motion prediction algorithms for wireless mobile computing and communication. Mob. Networks Appl. **1**(2), 113–121 (1996)
3. A.L. Murphy, G.C. Roman, G. Varghese, An algorithm for message delivery to mobile units, in *The 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, pp. 292–292, 1997
4. K. Kumar, Lu Yung-Hsiang, Cloud computing for mobile users: can offloading computation save energy? Computer **43**(4), 51–56 (2010)
5. K. Yang, S. Ou, H.H. Chen, On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. IEEE Commun. Mag. **46**(1), 56–63 (2008)
6. S. Acharya, M. Franklin, S. Zdonik, Dissemination-based data delivery using broadcast disks. IEEE Pers. Commun. **2**(6), 50–60 (2001)
7. B.R. Badrinath, A. Acharya, T. Imielinski, Designing distributed algorithms for mobile computing networks. Comput. Commun. **19**(4), 309–320 (1996)
8. L. Lamport, A new solution of dijkstra's concurrent programming problem. Commun. ACM **17**, 453–455 (1974)
9. E.W. Dijkstra, Self-stabilizing systems in spite of distributed control. Commun. ACM **17**, 643–644 (1974)

10. K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. (TOCS), **3**(1), 63–75 (1985)
11. E.W. Dijkstra, C.S. Scholten, Termination detection for diffusing computations. Inf. Process. Lett. **11**, 1–4 (1980)
12. S. Huang, Detecting termination of distributed computations by external agents, in *The IEEE Nineth International Conference on Distributed Computer Systems*, pp. 79–84, 1989
13. F. Mattern, Golbal quiescence detection based on credit distribution and recovery. Inf. Proc. Lett. **30**, 95–200 (1989)
14. Y.C. Tseng, Detecting termination by weight-throwing in a faulty distributed system. J. Parallel Distrib. Comput. **25**, 7–15 (1995)
15. Y.C. Tseng, C.C. Tan, Termination detection protocols for mobile distributed systems. IEEE Trans. Parallel Distrib. Syst. **12**(6), 558–566 (2001)