Chapter 10
Location Management

10.1 Introduction

The most important problem arising out mobility support is tracking of mobile objects. A mobile object may represent an automobile, a cellular phone, a PDA, a laptop or even a piece of software. The location management is concerned with the ability to track or locate a moving object with an intention to communicate.

At an abstract level, a location management scheme involves two operations:

- Look up or search which locates a mobile object.
- Update which records the new location, each time a mobile object makes a move.

The search and update operations are also basic to a conventional database system. But there are three significant differences in operation of a database and a location management scheme:

- A database records only precise data, whereas a location management scheme has to deal with imprecisions of various kind in location data.
- The update requirements in a database is directly linked with data consistency problem. An update must be applied to a database as soon as it is received. The update requirements in maintaining location data, depends on the ability of a location management scheme to tolerate the degree of imprecision in location data.
- The number of updates handled by a location management scheme is several times more than the updates any moderate sized database can handle.

The reason behind large number of updates that any location management is expected to handle can be explained as follows. There is absolutely no control in proliferation of mobile devices. In a cell size of about half a kilometer radius, roughly about 1000 mobile devices may be active at any point of time. Assuming every such mobile object makes just about five moves in a day (in 12-h period), the total number of updates exceeds 10,000 calculated at the rate of one insertion and one deletion per move

R.K. Ghosh, *Wireless Networking and Mobile Data Management*,
DOI 10.1007/978-981-10-3941-6_10

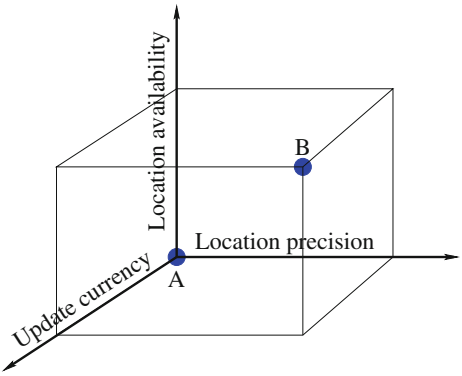


Fig. 10.1 Location space of all objects

per mobile object. Suppose a small city has a cellular coverage area of 1000 cells of the above kind. The active mobile devices operating in this city may potentially generate up to a minimum of a million location updates in 12-h period. Clearly, no conventional database can match the scale of updates expected to be handled by a location management scheme.

The three dimension of the approaches to the management of location information and its use can thus be identified as

- *Availability*: Represents the availability of location information at all the network sites or at a few selected places in the network.
- *Imprecision*: Represents the exactness of the location information.
- *Currency*: Represents the regularity of the location updates.

An abstract view of the space of location management problem is depicted in Fig. 10.1. The position *B* in the location space refers to the situation when exact information is maintained for each and every network sites. The location updates is required for every move and should be disseminated to all the sites. Under this scenario look up becomes immediate. The other extreme case represented by position *A*. It refers to the situation where no information about the location of any object is maintained anywhere in the network. The search for a mobile object under this situation becomes an exhaustive search of all the network sites. In between the above two extremities, several combination of approaches regarding location updates and look ups are possible.

We implicitly assume that location management is supported by a cellular network architecture acting as the backbone. Furthermore, location management is handled either in the data link or the network layer. Though cellular architecture is not the only possibility, it is a good candidate to understand the issues that arise in the location management. In absence of a infrastructures backbone architecture, an alternative approach such as global positioning system has to be used to locate mobile objects.

10.1.1 Registration and Paging

There are two distinct mobility types involving mobile devices [2].

1. *Terminal mobility*: Allows a terminal identified by an identity independent of its point of attachment in the network. It allows a terminal to be attached to different points of a fixed network at different time. Therefore, allows mobility with respect to the point of attachment with the fixed network.
2. *Personal mobility*: It is associated with a user. A user gets a distinct identity independent of terminal s/he uses. Personal mobility allows the users to receive or make calls any where through any terminals.

In practice there is no distinction between terminal and personal mobilities in a mobile network. Since, terminals are portable and carried by the users, a personal mobility always accompanied by a movement of the terminal carried by the user. In other words, personal and terminal mobilities occur concurrently.

In order to facilitate location tracking, a mobile user has to explicitly register, and notify about the location of the terminal to one or more location servers belonging to the fixed network [2]. The granularity of location data may vary from a single cell to a group of cells. The paging process is a system initiated polling by sending signals to the likely locations to track the users. When the exact location of a user is known, a single paging operation suffice. By changing the size of registration area, flexibility in combination of paging and registration can be attained.

10.2 Two Tier Structure

In two tier scheme [13], there are two location registers per mobile hosts:

1. *home location register* (HLR), and
2. *visitors location register* (VLR).

Every mobile user is associated with a HLR which is located at a network location prespecified for each user. It is essentially a database for the home location of a user. Home location is a group of cells in which a user is normally expected to move around. Initially, a user registers his/her mobile handset in home location to avail the cellular service. A HLR maintains the current location of an user. The HLR entry corresponding to a user gets updated every time the user makes a move.

Each zone also maintains a VLR database, each entry in VLR is associated with a user currently active in that zone. In HLR-VLR scheme, the call setup is quite straightforward. When a user A in a location area LAI_i wants to communicate with another user B in a cell belonging to LAI_j, then the VLR in LAI_i is searched first. If there is no entry for B, the search is then directed to the HLR maintained at LAI_b which is the home location area of B. Thus, the location update process for tracking movement of an user is as follows:

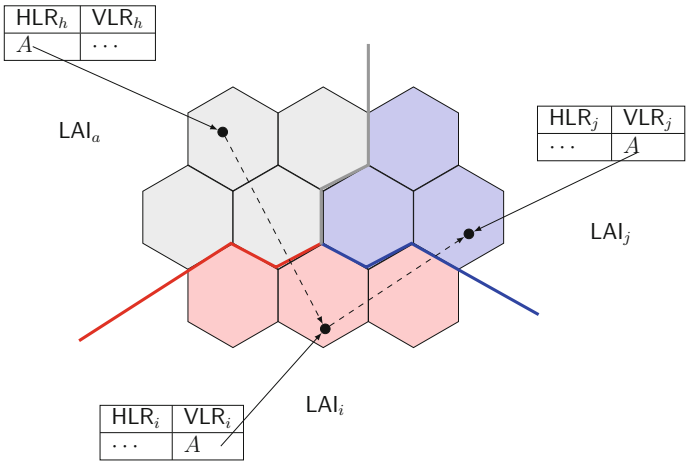


Fig. 10.2 Move in two-tier scheme

- When a user moves from a cell in LAI_i to a cell under LAI_j then the HLR of the user in its home location area gets updated.
- The VLR entry for the user is deleted from VLR database for area LAI_i, and an entry for the user is created in VLR database maintained for area LAI_j,

Figure 10.2 illustrates the movement of user A from a cell under LAI_i to a cell under LAI_j. LAI_a denotes the home location of A.

10.2.1 Drawbacks of Fixed Home Addresses

The home location is permanent. Thus long-lived mobile objects cannot change their homes. In other words, porting a mobile phone from one neighborhood to a different neighbourhood is not possible without explicit intervention by the service operators. The two-tier approach does not scale up too well as a highly distributed system would. Often the distant home locations may have to be contacted for look up even in the case of moderately mobile objects. In other words, the locality of the moves is not captured very well in a two-tier system.

10.3 Hierarchical Scheme

Hierarchical schemes [1, 8] are designed to take advantage of locality of movements. The location database at an internal node contains location information of users in the set of zones under the subtree rooted at the node. The information stored may simply

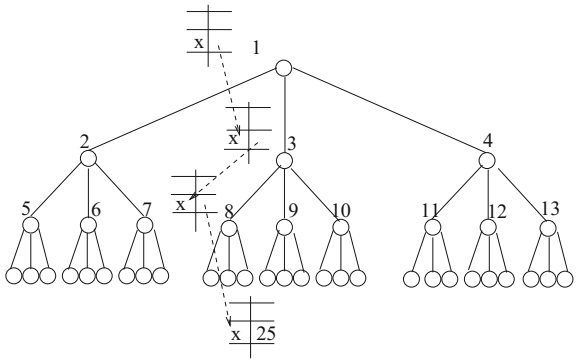


Fig. 10.3 Tree-based organization of location servers

be a pointer to the lower-level location server or the current location. A leaf node serves a single zone. The organization of location servers in a tree-based hierarchical scheme is illustrated in Fig. 10.3. A zone can be considered as a location area or a single cell served by a mobile support station.

The performance of any location management scheme is clearly dependent on two factors [5, 6], namely,

1. The frequency of the moves made by the users.
2. The frequency of the calls received by each user.

Based on these factors we can define two performance metrics:

(i) *local call to mobility ratio*, and
(ii) *global call to mobility ratio*.

Let C_i be the expected number of calls received, and let U_i be the number of location area crossings made by a mobile user over a duration of time T. The fraction C_i/U_i is the global call to mobility ratio for time T. If C_{ij} represents expected number of calls from location LAI_j to the user in time T, then the local call to mobility ratio (LCMR) is the ratio C_{ij}/U_i. Local call to mobility ratio $LCMR_{ij}$ at any internal node j of tree hierarchy can be computed as $\sum_k LCMR_{ik}$, where k is a child of j.

10.3.1 Update Requirements

Typically, location information at internal nodes of in the hierarchy are stored as pointers. As an MH A moves from a cell in LAI_i under location server LS_i to a new location, say to the cell LAI_j under location server LS_j, the tree paths:

- from $LCA(LS_i, LS_j)$ down to LS_i and
- from $LCA(LS_i, LS_j)$ down to LS_j

should be updated. For example in the Fig. 10.3 each node represents a location server. Each leaf server store location information of mobile users in one location

area. When a mobile user A moves from a cell in the location area under the server 25 to a cell in location area under the server 26, then updates will be needed at the nodes 3, 8, 9, 25, 26.

- At 8, 25 entries for A should be deleted.
- At 3, the entry for A should be updated.
- At 9, 26 new entries should be created for A.

If actual information on cell ID is stored at each node from the root to a leaf, then path from the root to $LCA(LS_i, LS_j)$ also has to be updated. However, typically pointers are stored. For example, if A is in a cell under 25, nodes 1, 3, 8 should have pointers leading to node 25 which indicates that A is in a cell under 25, and the cell ID information for A would be available in node 25. After a time t, suppose A moves to cell under 26, then all the nodes 1, 3, 9, 26 must be updated to indicate that A is now in a cell under 26. The information about location of A from node 25 should also be deleted.

10.3.2 Lookup in Hierarchical Scheme

The look up operation in hierarchical scheme can start with search for a mobile object A getting initiated at the neighbourhood, i.e., bottom up in the tree hierarchy. If A is not found in the current level of the hierarchy then the search process is just pushed one level up the hierarchy. Since the root maintains information about all the objects, the look up is bound to succeed at a higher level of the hierarchy. For example, suppose a search is initiated at a cell under LAI_i for the mobile object O_m which is currently located in a cell under LAI_j. The look up for O_m begins at location server LS_i, until it reaches $LCA(LS_i, LS_j)$ where an entry for O_m will be found. After that following the pointers to O_m, the search traverses down the hierarchy along the path from $LCA(LS_i, LS_j)$ to LS_j.

10.3.3 Advantages and Drawbacks

The tree-based hierarchical scheme does not require HLR and VLR type databases to be maintained. It supports locality of call setup process. However, it requires updates to be done at a number of location servers on different levels of hierarchy. The location information for each mobile object has to be replicated at each node on the tree path from the root to the leaf node corresponding to the location area in which the mobile object is located. The load for updates increases monotonically at the internal nodes higher up the hierarchy. The storage requirement also increases at higher levels.

10.4 Caching

Caching of locations [1, 6] is used primarily to reduce the lookup cost. It also helps to reduce the delay in establishing links. The idea has its root on the conventional use of caching.

In the two-tier architecture, when a large number of calls originate from the coverage area under a specific MSC to a particular mobile belonging to a different MSC, then the ID of the mobile and the address of its serving VLR can be stored at the calling MSC. This helps not only to reduce the signaling cost but also to reduce the delay in establishing the connection. Each time a call setup is attempted, the overhead associated with caching is as follows:

- First, the cached information of VLR is checked at calling MSC. If a cache hit occurs, the VLR of the callee is contacted directly.
- When the called mobile moves out of cached VLR, a cache miss occurs. Then the HLR of the called mobile is contacted to establish the call.

Checking cache at calling MSC adds little to overhead. On the other hand, if cache hit occurs, then the saving on signaling cost and the call setup latency improves. However, when called mobile crosses over to a new location area, old cache entry becomes invalid. So a cache revalidation is made by updating the old entry.

From the point of view of pure location update (no cache), new VLR address should be registered with HLR [9], and old VLR address should be purged from HLR. So, a location update cost is:

$$\text{update}_{nocache} = \text{cost}(\text{VLR}_{new} \leftrightarrow \text{HLR}) + \text{cost}(\text{VLR}_{old} \leftrightarrow \text{HLR})$$

For establishing a call in absence of a cache, HLR of the called mobile should be queried to obtain the address of VLR of the new region where the mobile is active. So, the cost of a lookup in a system without cache is:

$$\text{search}_{nocache} = \text{cost}(\text{VLR}_{caller} \leftrightarrow \text{HLR}) + \text{cost}(\text{HLR} \leftrightarrow \text{VLR}_{callee})$$

If the expected number of calls from a particular MSC to a particular mobile is ρ, then the cost for establishing calls will be

$$\text{update}_{nocache} + \rho \times \text{search}_{nocache}$$

Estimation of ρ requires an analysis of the call traces. A study [11] reveals that a user typically receives 90% of calls from his/her top 5 callers. A location area can be ranked according to the number of calls a user receives from that area.

Computation of local call to mobility ratio (LCMR) allows us to obtain a theoretical estimate of cache placement policy. Let us first model the calls received by a mobile and its mobility. Calls to a mobile and its movements are unrelated. Let the call arrival time at a mobile in a service area is exponentially distributed with mean

arrival rate λ. Then probability distribution function is:

$$\lambda e^{-\lambda t}.$$

Similarly, let the residence time of a mobile in a location area be exponentially distributed with mean residence time $1/\mu$, i.e., the probability distribution function for the residence time is:

$$\mu e^{-\mu t}.$$

Let p_{cache} represent the probability that the cached information for a mobile at a location area is correct. In other words, p_{cache} defines the probability that mobile has not moved from its current location since it received the last call from the same location area. So,

$$p_{cache} = \text{Prob}[t < t_1] = \int_0^\infty \lambda e^{-\lambda t} \int_t^\infty \mu e^{-\mu t_1} dt_1 dt = \frac{\lambda}{\lambda + \mu}$$

In order to know if caching could beneficial, we need to find the relationship between calls received by a mobile and the number of moves it makes in between receiving calls. Let,

1. C_B represent the lookup cost of connection setup in basic scheme, i.e., if caching is not used, and
2. C_H represent the cost when caching is used.

In other words, C_H is the cost of retrieving the address of VLR associated with the MSC where the called mobile is active directly from the VLR associated with MSC from where the call originated. Estimation of C_B, on the other hand, involves two things: (i) the cost of accessing HLR entry of the called mobile from VLR of calling MSC, and (ii) the cost of retrieving VLR address of called mobile and caching the same at the caller MSC. Hence,

$$\begin{aligned} C_H &= \text{cost}(VLR_{caller} \leftrightarrow VLR_{callee}) \\ C_B &= \text{cost}(VLR_{caller} \leftrightarrow HLR) + \text{cost}(HLR \leftrightarrow VLR_{callee}) \end{aligned}$$

A cost saving is possible in caching scheme, only if

$$p_{cache} C_H + (1 - p_{cache})(C_B + C_H) \leq C_B,$$

where p_{cache} is the probability of a cache hit. From the above inequality, we find that $p_T = \min\{p_{cache}\} = C_H/C_B$. This implies that caching would be useful in saving cost, if

$$p_{cache} > p_T \geq C_H/C_B,$$

where p_T denotes threshold for the cache hit.

Calling MSC typically uses Local Call to Mobility Ratio (LCMR) for determining if caching could be cost effective. LCMR is equal to λ/μ. Relating LCMR to threshold for cache hit, we have

$$\text{LCMR}_T \geq p_T/(1 - p_T)$$

10.4.1 Caching in Hierarchical Scheme

In a hierarchical architecture, caching can be deployed to reduce the lookup time. Combined with *forward* and *reverse* bypass pointers, caches can be placed at a higher level of the hierarchy to service the calls originating from a group of cells rather than a single cell. The tradeoff of placing cache at a higher level is that the calls have to traverse a longer path. The idea is illustrated by Fig. 10.4. When a connection setup is requested from a mobile phone in a location area LAI_i to a mobile phone in location area LAI_j, the control message traverses up the tree from LS_i to $LCA(LS_i, LS_j)$ and then downwards to LS_j. The call setup requires an acknowledgement (ack) to be sent from LS_j back to LS_i along the same path. All the ancestors of LS_i in the hierarchy overhear this ack message, and one of them, say LS_i^a, can create a forward bypass pointer to LS_j. Likewise, a reverse bypass pointer can be created from an ancestor of LS_j to LS_i. A cache can be deployed at LS_i^a for the callees located in the cells belonging to LAI_j. Then subsequent calls from the callers active under coverage area LAI_i may be able to reach the location database LS_j via a shorter route through the forward bypass pointer at LS_i^a. Similarly, the acknowledgement messages originating

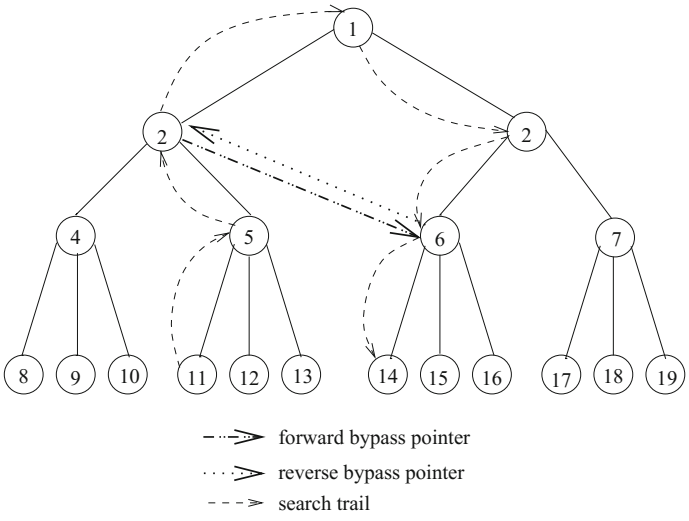


Fig. 10.4 Caching in hierarchical location scheme

from the callees under LAI_j traverse a shorter path to LS_i using the reverse bypass pointer. If the level of LS_i^a is high then all the callers from its subtree can use the forward bypass pointers for the callees under the subtree of LS_j. But each call have to traverse a longer path up to LS_i, and may incur longer average lookup time, because the latency for locating the callee in the subtree of LS_j^a would depend on its size.

10.5 Forwarding Pointers

For a mobile user receiving relatively less calls compared to the moves, it is expensive to update all the location servers holding the location of the user on every move. It may be cheaper to leave a forwarding pointer to the new location at the previous location of the user. Then any call arriving at the old location can be re-routed to the current location by the forward pointer, and the update to the database entries holding a user's location can be made less frequently.

In two-tier architecture, if a user is frequently away from home, and the user's moves are mostly localized around the neighborhood of its current locations, then the location updates to the user's HLR would require long latencies. In order to avoid such updates requiring long latencies, forward pointers may be used. It works as follows. Whenever the user makes a move in the neighborhood of its current location, a forward pointer for the new location is placed in the VLR of the serving location. Therefore, when a call to the user reaches its home location server, the call can be diverted to the present location of the user. The call diversion is possible because HLR can access the VLR associated with the user's first location away from home. The other locations of the user can now be reached by following a chain of forward pointers starting from the first VLR. The chain of forward pointers are allowed to grow up to a predetermined length of K. When the user revisits a location, the potential loop condition in forward pointers chain is avoided by an implicit compression due to a location update in current VLR on a fresh move by the user. The approach of forwarding is applied on a per user basis.

Forwarding pointer strategy can also be applied to the hierarchical architecture. In a *simple forwarding* strategy, the pointers are placed in the leaf level. No update is performed up the tree path as a user moves from one location to another. But a forwarding pointer is left at the previous location to the current location.

In a *level forwarding* strategy, the forwarding pointer is placed at a higher level location server in the hierarchy. In this case, more updates are required to delete the database entries of lower level location servers of the previous location, and also to insert the current location into lower level ancestor location servers of the current location. Figure 10.5 illustrate the two schemes for placing the forward pointers. The dashed pointers illustrate the level forwarding while the ordinary pointers illustrate simple forwarding. Assuming mobile user *mu* to be initially located in cell 11 decides to move to a the cell 14, then simple forwarding places a forward pointer at the old leaf level location server 11 as indicated by the ordinary pointers. Whereas in the case of level forwarding, a forward pointer is placed at the ancestor location server 2 at

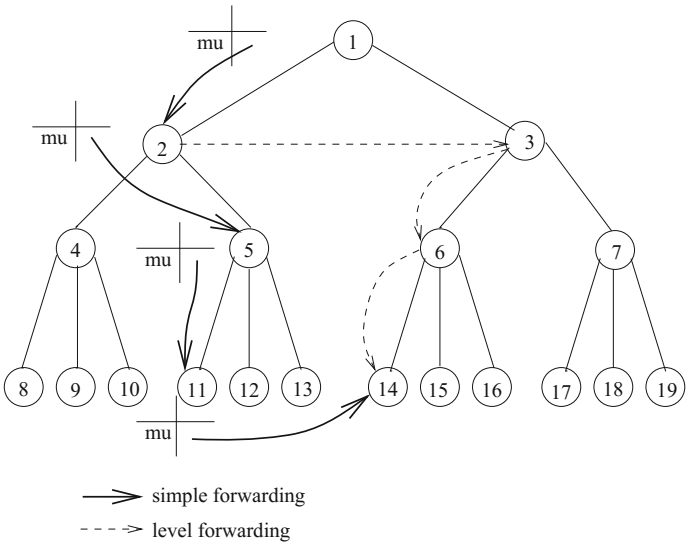


Fig. 10.5 Forwarding pointers in hierarchical scheme

level 3 which points to the location server 3 which is an ancestor of the new location. In this case, the updates are more than that required in simple forwarding. This is because, the entries concerning new location of *mu* should be made at nodes 3, 6, and 14 and database entries for *mu* at nodes 2, 5, and 11 are to be deleted.

10.6 Replication

Replication [5, 8] is another mechanism to reduce the cost for lookup. A high CMR (call to mobility ratio) value is the guiding principle for replication of the location of a user at selected nodes in the hierarchy.

In a two-tier architecture, replication may be employed, if the cost of replication is not more than the cost for non-replication. When many calls originate from a location area LAI_j to a mobile user, it may be cost effective to replicate the location of the user at the location server LS_j. However, if the user moves frequently then the maintenance of the replica incurs a heavy cost. A replica does not merely have location information. It also includes service information parameters such as call blocking, call forwarding, QoS requirements like the minimum channel quality and the acceptable bandwidth. Thus replicas should not be considered merely as extension of cache entries. Replicas may maintained both at the caller and the callee. If C_{ij} is the expected number of calls made from cells in LAI_j to a user over a period of time T, α is saving on replication at LS_j, β is the cost per update, and U_i is the number of updates then

$$\alpha.C_{ij} \geq \beta.U_i \tag{10.1}$$

should hold in order that the replication of the user at LS_j to be cost effective. The replicas of the user are kept at all the frequent callers areas which satisfy inequality 10.1. The set of the caller locations where the replica of a user is maintained is called a *working set* for that user. Every time a call is made to the user:

- From a member of the user's working set, no update is needed,
- From a non-member then if inequality 10.1 is found to be true then that cell is added to the working set of the user.

On the other hand, when the user makes a move:

- Inequality 10.1 is evaluated for every member of the working set,
- If it fails to hold for location area LAI_k, then it is removed from the working set.

The storage requirement for a single user's profile of size F in basic multi-tier location database having L levels is $F + ptr \times (L - 1)$, where ptr is the size of a pointer (user ID + database ID). If the profiles are replicated then the cumulative storage requirements should not exceed the storage space available in the database [7].

Apart from storage constraint, the decision to place a replica should also be based on minimization of network communication cost. In two-tier model, LCMR (local call to mobility ratio) is used for this purpose. A user i's profile is replicated a database j, only if LCMR_{ij} exceeds a minimum threshold, say R_{min}. In hierarchical database, it is impossible to arrive at a single parameter for databases at different levels of the hierarchy. However, by using an additional parameter R_{max}, it is possible to arrive at a decision. The computation of $LCMR_{ij}$ for hierarchical database is done by a simple bottom up summing the *LCMR* values of its children. Clearly, if high *LCMR* value is the criterion for the selection of replication then when a node is selected for placing a replica, all its ancestor nodes also should be selected for replica placement. Therefore, this selection process results in excessive updates at higher levels of databases. This calls for setting a number of constraints including high (R_{max}) and low (R_{min}) marks for *LCMR* values to determine the nodes in the hierarchy which may be selected for placing replicas. The rules for selecting replication site are as follows [7]:

1. If $\text{LCMR}_{ij} < R_{min}$, replica of i's profile is not placed at site j.
2. If $\text{LCMR}_{ij} \geq R_{max}$, then always place replica of i's profile at site j if the constraints on L and N are satisfied, where L represents hierarchy level, N is the bound on the number of maximum number of replicas for a user.
3. If $R_{min} \leq \text{LCMR}_{ij} < R_{max}$, then the decision to place i's profile at site j will depend on database topology.

The other constraints are the level of location server in the hierarchy, and the maximum number of replicas to be placed. The reader is referred to [7] for details of analysis and the algorithm for placing replicas.

10.7 Personal Mobility

In the context of location management, there is a trade off between search and update. Most of the location management schemes are based on the approach to balance between search and update. To what extent the trade off can swing between the two will depend on the bound on signaling requirements. But the question is how to define a bound on signaling requirements? Furthermore, even if a bound can be defined, is it possible to reach the bound? Due to technological limitations, it is difficult to find satisfactory answers to above questions. However, analyzing problem of location management from a different track, we notice that terminal mobility and personal mobility are tightly coupled in a cellular communication network. In reality a mobile equipment is a portable device, and cannot move on its own. The movement of a mobile terminal is caused by the movements of the user carrying it. The movement of a person or the user of a mobile terminal can be considered as a path in some random process [3, 12].

10.7.1 Random Process, Information and Entropy

Let us explore a bit about the randomness of a process. If the volume of information content in a random process is high then the unpredictability is low. The probability of occurrence of an event contains the amount of information about the event. For example, if the probability of occurrence of an event A is known to be more than the probability of occurrence of another event B, then the amount of information available about A is more than that available for B. In other words, A's occurrence is more predictable than B's occurrence. This qualitative measure of information can be interpreted in terms of *surprisal*. Shannon [10] formulated surprisal as measure of information content in a system. It captures the following two important aspects:

1. The extent of randomness is determined by the size of entropy.
2. If randomness of a process is more, its unpredictability is higher.

For example, if an event E is known to happen always, then there is no surprisal in its occurrence. Equivalently, E's occurrence carries no information at all. In contrast, if E is a rare event, the fact that E has occurred is a surprise. So, the information it provides is high. The relationship between probability $p(E)$ of an event E and the size of expected information $H(E)$ of E's occurrence can be expressed as follows:

$$p(E) \to 0 \text{ implies } H(E) \to \infty$$
$$p(E) \to 1 \text{ implies } H(E) \to 0,$$

where

$p(E)$: proability of event E
$H(E)$: expected information conent in occurrence of E

In other words, the richness in information varies as the inverse of the probability. The above connection between $p(E)$ and $H(E)$ is captured by Shannon as follows:

$$H(E) = p(E) \times \frac{1}{\log p(E)}$$

The reason for using logarithmic function instead of simple inverse is that it makes entropy an extensive property. In other words, if there are two systems A and B, then the total entropy should be additive. Any base greater than 1 should work. Typically, the base of logarithm is taken as 2, since $\log_2 q$ bits are needed to represent a quantity q.

The amount of information is measured in number of bits. For example, 3000 bits are needed in order to transmit the results of 1000 rollings of an unbiased hypothetical eight sided dice. If the dice is known to be biased, and the probability distribution is known, then a variable length encoding can be used. Since, the information bits are transmitted together, the encoding should be such that it is possible to disambiguate the block of bits representing the results of different rollings of the dice. This implies that the encoding must have *prefix property* which ensures that no code is a prefix of any code.

For example, let the probability distribution for a biased dice be:

$$p(i) = \begin{cases} 1/2^i, & \text{for } i \leq 7 \\ 1/2^{i-1}, & \text{for } i = 8, \end{cases}$$

Since, half the number of rollings result in 1, the shortest code should be used for 1. On the other hand, the code for the rarest event (a rolling that results in 8) could be the longest. A possible encoding scheme with the above mentioned properties would be as illustrated in Table 10.1. The above encoding satisfies the prefix property. The average number of bits needed for encoding the result of 1000 rollings is

$$\frac{1}{2} + 2 \times \frac{1}{4} + 3 \times \frac{1}{8} + \ldots + 7 \times \frac{1}{128} + 8 \times \frac{1}{128} = 1.984$$

So, with more information, the average number of bits required for transmitting the result 1000 rolling of the biased 8 sides hypothetical dice is reduced from 3000 to 1984 bits.

To reinforce our understanding, consider flipping of a biased coin. Suppose head shows up just once in 1000 flips. Let a 0 represent the fact that the result of a toss

Table 10.1 An encoding scheme with prefix property

Results of rolling	1	2	3	4	5	6	7	8
Code	0	10	110	1110	11110	111110	1111110	1111111

is a head. Similarly, let 1 represent a tail. Suppose the coin is tossed one million times. Without using any clever encoding, 1 bit will be required for the result of each toss. So, transmitting the result of one million tosses requires a million bits. Since a head shows up only once in 1000 flips of the biased coin, we may just record the sequence number of tosses that resulted in a head. The missing sequence numbers will then represent tails. Any number between 1 and 10^6 can be represented by at most $\log 10^6 = 20$ bits. Therefore, the information transfer for the result of 1000 flippings of biased coin will need just 20 bits. A total of 20000 bits will be needed for transmitting the results of one million tosses.

From the above examples, let us try to abstract out the answer to the general case of information coding for a random event. Consider a conventional dice with six faces to find an answer to the above question. If the dice is unbiased, the probability of occurrence of any value is $p = 1/6$, the number of bits required = $\log 6 = -\log(1/6) = -\log 6$. The result of one throw requires $\log 6 = 2.58$ bits. It is not immediately apparent how the result any particular throw of a dice can be encoded by less than 3 bits. However, if we group g successive throws, the results can coded by less than 6^g bits. For example, the number of possible outcomes for a group of three throws = $6^3 = 216 < 255$, and 0–255 can be coded using 8 bits. Thus, for a biased probability distribution, the number of bits required for the optimal code is determined by

$$-\sum_x p(x) \times \log p(x).$$

The examples discussed in this section, point to the fact that a rare event (probability of occurrence is low) has a high information content. In the coin toss example, the information content in occurrence of a head is $-\log(1/1000) = \log 1000 = 9.9658$. So, 10 bits will be needed to represent the occurrence of a head. As against this information contents in appearance of a tail is $-\log(999/1000) = 0.0044$ bit. The average information content is given by:

$$(1/1000) \times 10 + (999/1000) \times 0.0044 = 0.0114 \text{ bit}.$$

Let us try to generalize the method of determining information content as outlined in the above example. Suppose, the probability of an outcome A_i of a random event A is $p(A_i)$. Then the expected value of self information in A

$$\begin{aligned} H(A) &= \sum_{i=1}^{n} p(A_i) \times \log\left(\frac{1}{p(A_i)}\right) \\ &= -\sum_{i=1}^{n} p(A_i) \times \log p(A_i) \end{aligned}$$

According to Shannon [10], $H(A)$ represents the entropy of A.

Suppose X and Y is a pair of discrete random variables with joint probability distribution $p(x, y)$, where $x \in \mathscr{X}$, $y \in \mathscr{Y}$. The joint entropy of X and Y is:

$$H(X, Y) = -\sum_{x \in \mathscr{X}} \sum_{y \in \mathscr{Y}} p(x, y) \log p(x, y)$$

Assume that the conditional entropy $H(Y|X)$ represents the information content of a random event Y given that some event X has occurred. $H(Y|X)$ is given by the formula:

$$\begin{aligned} H(Y|X) &= \sum_{x \in \mathscr{X}} p(x) H(Y|X = x) = -\sum_{x \in \mathscr{X}} p(x) \sum_{y \in \mathscr{Y}} p(y|x) \log p(y|x) \\ &= -\sum_{x \in \mathscr{X}} \sum_{y \in \mathscr{Y}} p(x, y) \log p(y|x) \end{aligned}$$

Joint entropy and conditional entropy are closely related. Joint entropy is the sum of entropy of the first random variable and the conditional entropy of the second random variable given the first.

$$\begin{aligned} H(X, Y) &= -\sum_{x \in \mathscr{X}} \sum_{y \in \mathscr{Y}} p(x, y) \log p(x, y) \\ &= -\sum_{x \in \mathscr{X}} \sum_{y \in \mathscr{Y}} p(x, y) \log(p(x) p(y|x)) \\ &= -\sum_{x \in \mathscr{X}} \sum_{y \in \mathscr{Y}} p(x, y) \log p(x) - \sum_{x \in \mathscr{X}} \sum_{y \in \mathscr{Y}} p(x, y) \log p(y|x) \\ &= -\sum_{x \in \mathscr{X}} \sum_{y \in \mathscr{Y}} p(x, y) \log p(x) + H(Y|X) \\ &= -\sum_{x \in \mathscr{X}} p(x) \log p(x) + H(Y|X) \\ &= H(X) + H(Y|X) = H(Y) + H(X|Y). \end{aligned}$$

The generalization of the above result, known as chain rule, says:

$$H(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} H(X_i | X_{i-1}, \ldots, X_1).$$

If the random variables are known to be independent then according to the chain rule:

$$H(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} H(X_i).$$

10.7.2 Mobility Pattern as a Stochastic Process

In order to capture the personal mobility, a user's movement is considered as a random process. In a cellular infrastructure, a user's mobility can be represented as a sequence of cells that the user has visited. In a GSM type network, each symbol represents a Location Area Identity (LAI) consisting of several cells. There is a possibility that certain substrings of LAIs have repeated occurrences in a string of LAIs representing a user's movement. Such repetitions essentially represent locality of a user's movements. By characterizing mobility as probabilistic sequence, mobility can be interpreted as a stochastic process.

At first we need to be clear on two issues:

1. How a user may move in a service area?
2. How the movements may be recorded?

Figure 10.6 illustrates an example for a service area consisting of eight LAIs `a, b, c, d, e, f, g, h`. The shapes of actual cell areas are not hexagonal, but irregular geometric contours. The cell contours are determined by actual field measurement of signal strengths. Multiple cells are grouped into an LAI. The topology of LAIs in coverage area can be abstracted in form of a graph shown along side. Each node in the graph represents an LAI. Two nodes are connected with edge if and only if the corresponding LAIs are adjacent to each other. With the graph abstraction as explained above, the mobility pattern of a user is represented as a walk in the graph. A walk is formed by the updates of the user's locations as a user moves and enters new LAIs.

The mobiles send their location updates to network subsystem. The frequency of update is controlled by one of following three possible ways.

- *Distance based updates*: In the distance based updates, mobile terminals keep track of the Euclidean distance from the time of the last update. If distance travelled from the last update crosses a threshold D, the mobile should send an update.
- *Movement based*: A mobile sends an update if it has performed n cell crossings since the last update.
- *Time based*: a mobile sends periodic update.

It is also possible to combine three update schemes in several possible ways. For example, distance based updates can be combined with movement based updates or

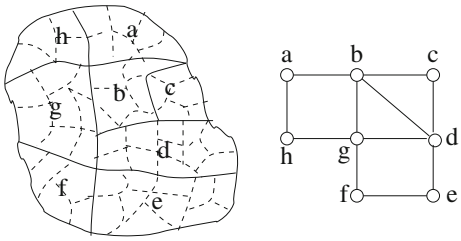


Fig. 10.6 GSM type location area map (Source [3])

Table 10.2 An example for LAI crossing by a mobile

Crossings in morning				
Time	11:04	11:32	11:57	
Crossing	$a \rightarrow b$	$b \rightarrow a$	$a \rightarrow b$	
Crossings in afternoon				
Time	3:18	4:12	4:52	
Crossing	$b \rightarrow a$	$a \rightarrow b$	$b \rightarrow c$	
Crossings in evening				
Time	5:13	6:11	6:33	6:54
Crossing	$c \rightarrow d$	$d \rightarrow c$	$c \rightarrow b$	$b \rightarrow a$

Table 10.3 Sequence of cells representing movement history

Update scheme	Movement history
$T = 1$ h	aaabbbbacdaaa...
$T = 1/2$ h	aaaaabbbbbbbbaabcddcaaaa...
$M = 1$	abababcdcba...
$M = 2$	aaacca...
$T = 1$ h, $M = 1$	aaababbbbbaabccddcbaaaa...

time based updates. Similarly, movement based updates can be combined with time based updates, and so on.

Let us consider an example to understand how different updates schemes would generate the updates. Suppose the service is started at 9.00 AM. An example of LAI crossings is provided in Table 10.2. With the movement history, shown earlier in Table 10.2, the LAI sequences reported by different update schemes may be as shown in Table 10.3.

In summary, the movement history of a user is represented by a string $v_1, v_2, v_3, \ldots$, where each symbol v_i, $i = 1, 2, \ldots$, denotes the LAI reported by the mobile in ith update. The symbols are drawn from an alphabet set $\mathcal{V}$ representing the set of LAIs covering the entire service area. The mobility of a user is characterized as a stationary stochastic process $\{V_i\}$, where V_i's form a sequence of random variables, and each V_i takes a value v_i from set $\mathcal{V}$.

Before proceeding further, let us define a stochastic process.

Definition 10.1 (*Stochastic process*) A stochastic process is stationary if the joint probability distribution does not change when shifted in time or space.

If observed over time, a normal mobile user is most likely to exhibit the preference for visiting known sequence of LAIs in a time invariant manner. Equivalently, the correlation between the adjacent LAIs in the string of visited locations remains unchanged over all periods of time. Consequently, personal mobility pattern can be treated as a stationary stochastic process. Thus,

$$\Pr[V_1 = v_1, V_2 = v_2, \ldots, V_n = v_n] = \Pr[V_{l+1} = v_1, V_{l+2} = v_2, \ldots, V_{l+n} = v_n].$$

The above general model could aid in learning, if a universal predictor can be constructed. Let us explore the possibility of evolving a statistical model for a universal predictor. The common models used for interpreting the movement history are:

- Ignorant Model (IM)
- Identically Independent Distribution (IID)
- Markov Model (MM)

IM disbelieves and disregards past history. So the probability of any LAI residence is the same, i.e., 1/8 for each of the 8 LAIs for the chosen example.

IID model assumes that the values random variables defining a stochastic process are Identically and Independently Distributed. It uses relative frequencies of symbols for estimating the residence probabilities of the LAIs. Assuming time and movement based scheme the probabilities for the string `aaababbbbbaabccddcbaaaa`, are:

$$p(\texttt{a}) = 10/23, p(\texttt{b}) = 8/23, p(\texttt{c}) = 3/23, p(\texttt{d}) = 2/23,$$
$$p(\texttt{e}) = p(\texttt{f}) = p(\texttt{g}) = p(\texttt{h}) = 0.$$

The string consists of 23 symbols. Symbols `e, f, g, h` do not occur at all. While the probability of occurrence of any of the remaining symbols is determined by its relative frequency.

The simplest Markov model is a Markov chain where distribution of a random variable depends only on distribution of the previous state. So, this model assumes the stochastic process to be a stationary (time-invariant) Markov chain defined by:

$$\begin{aligned} &Pr[V_k = v_k | V_1 = v_1, \ldots, V_{k-1} = v_{k-1}] \\ &\quad = Pr[V_k = v_k | V_{k-1} = v_{k-1}] \\ &\quad = Pr[V_i = v_i | V_{i-1} = v_{i-1}] \end{aligned}$$

for arbitrary choices of k and i. Notice that the LAIs `e, f, g, h` are not visited at all. This implies each of these four LAIs have zero residence probability or equivalently, the effective state space is {`a, b, c, d`}. One step transition probabilities are:

$$P_{i,j} = Pr[V_k = v_j | V_{k-1} = v_i],$$

where $v_i, v_j \in \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}$, are estimated by relative counts. So, the movement profile can be represented by the corresponding transition probability matrix:

$$P = \begin{bmatrix} 2/3 & 1/3 & 0 & 0 \\ 3/8 & 1/2 & 1/8 & 0 \\ 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}$$

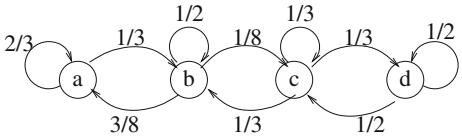


Fig. 10.7 One step state transition diagram for personal mobility

Table 10.4 Frequencies of symbols corresponding to three contexts

Order-0	Order-1	Order-2
a(10)|Λ	a(6)|a b(1)|c	a(3)|aa a(2)|ba a(1)|cb
b(8)|Λ	b(3)|a c(1)|c	b(2)|aa b(1)|ba d(1)|cc
c(3)|Λ	a(3)|b d(1)|c	a(1)|ab a(1)|bb d(1)|cd
d(2)|Λ	b(4)|b c(1)|d	b(1)|ab b(3)|bb b(1)|dc
	c(1)|b d(1)|d	c(1)|ab c(1)|bc c(1)|dd

Note that a occurs 6 out of 9 times in context of a and 3 times out of 8 times in context of b. The values of other transition probabilities can be found likewise. Thus, the state transitions with the respective probabilities can be viewed as shown in Fig. 10.7. Let $\Pi = [p(\mathtt{a})\ p(\mathtt{b})\ p(\mathtt{c})\ p(\mathtt{d})]^T$ be in steady state probability vector. Solving $\Pi = \Pi \times P$ with $p(\mathtt{a}) + p(\mathtt{b}) + p(\mathtt{c}) + p(\mathtt{d}) = 1$, we obtain $p(\mathtt{a}) = 9/22$, $p(\mathtt{b}) = 4/11$, $p(\mathtt{c}) = 3/22$ and $p(\mathtt{d}) = 1/11$.

To summarize the above discussion, though IID is the first step toward adaptive modeling, it can never be adaptive. The optimal paging strategy dependents on the independent probabilities of symbols {a, b, c, d, e, f, g, h}. But, if we already know that the user has reported the last update as d then neither a nor b should be paged. So, IID ignore the knowledge of the previous update.

Order-1 Markov model carries information to the extent of one symbol context. For uniformity, IM is referred to as order-(-1) Markov model, and IID as order-0 Markov model. Order-2 Markov model can be constructed by counting the frequencies of symbols appearing in order-2 contexts for the sequence aaababbbbbaabccddcbaaaa. Table 10.4 provides frequencies of different symbols for all three contexts.

According to order-1 model the probability of taking the route a $\rightarrow$ b $\rightarrow$ c $\rightarrow$ b $\rightarrow$ c $\rightarrow$ d is:

$$\begin{aligned} &= (9/22) \times (1/3) \times (1/8) \times (1/3) \times (1/8) \times (1/3) \\ &= 1/4224 = 2.37 \times 10^{-4} \end{aligned}$$

It is unlikely that any user will ever take such a zig-zag route. Though the probability is very low; still it is not zero. However, if order-2 model is used then the proposed route will be impossible. So, the richness of the information helps.

The question is how much of the past history should be stored so that it could lead to a good prediction? Storing the movement history for every movement of a user is not practical. The conditional entropy is known to be a decreasing function of the

number of symbols in a stationary process [4], implying that the advantage of higher order contexts die out after a finite value. To appreciate this fact, we need compare per symbol entropy rates $H(\mathscr{V})$ for a stochastic process $\mathscr{V} = \{V_i\}$. This quantity is defined as

Definition 10.2

$$H(\mathscr{V}) = \lim_{n\to\infty} \frac{1}{n} H(V_1, V_2, \ldots, V_n).$$

if the limit exists. The conditional entropy rate $H'(\mathscr{V})$ for the same process is defined by

$$H'(\mathscr{V}) = \lim_{n\to\infty} \frac{1}{n} H(V_n|V_1, \ldots, V_{n-1}),$$

if the limit exists.

Using the above definition, let us compute $H'(\mathscr{V})$ for the three different models for the example for personal mobility we have used in the text.

- Order-(-1) model:
  V_is are independently and uniformly distributed, so $p(v_i) = 1/8$ for all $v_i \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$ in the chosen running example. Due to independence of events, $p(v_n|v_1, \ldots v_{n-1}) = p(v_n)$. Therefore,

$$\begin{aligned} H(\mathscr{V}) = H'(\mathscr{V}) &= -\sum_{v_i} p(v_i) \log p(v_i) \\ &= \sum_{i=1}^{8} (1/8) \log 8 = \log 8 = 3. \end{aligned}$$

  The above equation implies that the per symbol entropy rate is 3 bits for order-(-1) model.
- Order-0 model:
  In this case, V_is are Independently and Identically Distributed. Due to independence $p(v_n|v_1, \ldots v_{n-1}) = p(v_n)$. Therefore,

$$\begin{aligned} H(\mathscr{V}) = H'(\mathscr{V}) &= -\sum_{v_i} p(v_i) \log p(v_i) \\ &= (10/23) \times \log(23/10) + (8/23) \times \log(23/8) \\ &\quad + (3/23) \times \log(23/3) + (2/23) \times \log(23/2) \\ &\approx 1.742. \end{aligned}$$

  Therefore, the per symbol entropy rate for order-0 model is 1.742 bits which is much better than order-(-1) model.

- Order-1 model:
  In this case, V_is form Markov chains. So $p(v_n|v_1 \ldots v_{n-1}) = p(v_n|v_{n-1}) = P_{v_{n-1},v_n}$. Substituting steady state probabilities $p(\mathtt{a}) = 9/22$, $p(\mathtt{b}) = 4/11$, $p(\mathtt{c}) = 3/22$ and $p(\mathtt{d}) = 1/11$, we find

$$\begin{aligned} H'(\mathscr{V}) &= -\sum_{v_i} p(v_i) \left(\sum_j P_{i,j} \log P_{i,j} \right) \\ &= \frac{9}{22}\left(\frac{2}{3}\log\frac{3}{2} + \frac{1}{3}\log\frac{3}{1}\right) + \frac{4}{11}\left(\frac{3}{8}\log\frac{8}{3} + \frac{1}{2}\log\frac{2}{1} + \frac{1}{8}\log\frac{8}{1}\right) \\ &\quad + \frac{3}{22}\left(3 \times \frac{1}{3}\log\frac{3}{1}\right) + \frac{1}{11}\left(2 \times \frac{1}{2}\log\frac{2}{1}\right) \approx 1.194. \end{aligned}$$

Note that 3 bits are sufficient to represent any symbol from a space of eight symbols $\{a, b, c, d, e, f, g\}$. So order-(-1) model cannot resolve uncertainties in any of the three bits. But both order-0 and order-1 models exhibit richness in information and gradual decrease in entropy rates. The entropy rates $H(\mathscr{V})$ and $H'(\mathscr{V})$ are same in both order-(-1) and order-0 MM due to independence of the events related to symbols. However, when order-1 MM is used, the per symbol entropy rate is 1.194 bits. It improves the entropy rate substantially over order-(-1) and order-0 models.

A mobile terminal's location is unknown for the interval between two successive updates. The approach in LeZi update is to delay the updates, if the path traversed by mobile is familiar. The information lag does not impact paging, because system uses prefix matching to predict location with high probability.

10.7.3 Lempel-Ziv Algorithm

LeZi update is based on Lempel-Ziv's text compression algorithm [14]. The algorithm provides a universal model for variable-to-fixed length coding scheme. The algorithm consists of an encoder and decoder. The encoder incrementally parses the text into distinct phrases or words (which have not been observed so far). A dictionary is gradually built as phrases keep coming. LeZi update's encoder is identical to the encoder of Lempel-Ziv's algorithm (see Algorithm 3). It runs at mobile terminal. The encoding process can be best explained by executing it on a string of symbols. Let the example string be:

```
aaababbbbbaabccddcbaaaa
```

Table 10.5 illustrates how encoding of each phrase is realized. The first incoming phrase is a, Λ or the null phrase is its prefix. The null phrase is assumed to be a prefix of any phrase having one symbol, and the index of Λ is set to 0. So, a is coded

Algorithm 3: Lempel-Ziv encoder

```
begin
    // Encoder at mobile or compressing algorithm.
    dictionary = null;
    phrase w = null;
    while (true) do
        wait for next symbol v;
        if (w.v in dictionary) then
            w = w.v;
        end
        else
            encode < index(w), v >;
            add w.v to dictionary;
            w = null;
        end
    end
end
```

Table 10.5 Encoding of different phrases

Index	Prefix	Last symbol	Input phrase	Output
1	Λ	a	a	(0, a)
2	a	a	aa	(1, a)
3	Λ	b	b	(0, b)
4	a	b	ab	(1, b)
5	b	b	bb	(3, b)
6	bb	a	bba	(5, a)
7	ab	c	abc	(4, c)
8	Λ	c	c	(0, c)
9	Λ	d	d	(0, d)
10	d	c	dc	(9, c)
11	b	a	ba	(3, a)
12	aa	a	aaa	(2, a)

as 0a. The index of an incoming phrase is determined by the position of the phrase in the dictionary which is largest proper prefix of the current phrase. A proper prefix excludes the last symbol in a phrase. For example, let us see how the next incoming phrase aa is encoded. The largest proper prefix of aa is a. Since the position of a is 1 in the dictionary, the index of the incoming phrase is 1. Therefore, appending *a* to 1, we get the encoding of aa as 1a. Therefore using encoding algorithm the string aaababbbbbaabccddaaaa is encoded as: 0a, 1a, 0b, 1b, 3b, 5a, 4c, 0c, 0d, 9c, 3a, 2a.

A code word consists of two parts (i) an index, and (ii) a symbol. The index represents the dictionary entry of the phrase which is the prefix of the code word. The prefix completely matches with the code word symbol-wise except for the last

symbol. Therefore, the decoding consists of finding the prefix and appending the last symbol to it. The major distinction between LeZi update and Lempel-Ziv's algorithm is in the decoder part. In the case of LeZi update, the decoder executes at the network side. The decoder algorithm is provided in Algorithm 4.

Algorithm 4: Decoder for LeZi update

```
begin
    // Decoder at the system side. It basically decompresses
       the string.
    while (true) do
        wait for the next code word < i, s >;
        decode phrase = dictionary[i].s;
        add phrase to dictinary;
        increment frequency of every prefix of the phrase;
    end
end
```

To see how the decoding process works, we consider code word specified in terms of tuples <index, last_symbol> and illustrate the decoding with the help of Table 10.6. When the code word tuple is received, the index part is extracted first. For example index part in the input tuple `(0,a)` is 0. The index is then used to retrieve the phrase from the dictionary. Since, index of Λ is 0, the phrase retrieved is Λ. Then symbol of input codeword is concatenated with retrieved phrase, i.e. `a` is appended to λ. Since Λ+a = a, the decoding of `(0, a)` outputs the phrase `a`. Similarly, when the code word `(5, a)` is received, the phrase having index 5 is extracted from the dictionary, and the symbol `a` is appended to it producing the output `bba`. Along with

Table 10.6 Decoding of phrases

Input tuple	Prefix phrase	Last symbol	Output phrase
(0, a)	Λ	a	a
(1, a)	a	a	aa
(0, b)	Λ	b	b
(1, b)	a	b	ab
(3, b)	b	b	bb
(5, a)	bb	a	bba
(4, c)	ab	c	abc
(0, c)	Λ	c	c
(0, d)	Λ	d	d
(9, c)	d	c	dc
(3, a)	b	a	ba
(2, a)	aa	a	aaa

the decoding, frequency count of all the prefixes are incremented. For example, when phrases `(0, a)`, `(1, a)`, `(1, b)`, `(4, c)`, and `(2, a)` get decoded `a`'s frequency count is incremented. So, total frequency count for `a` is 5. Similarly, frequency count `aa` is incremented during decoding of `(1, a)` and `(2, a)`, and total frequency count of phrase `aa` is computed as 2.

The decoding process helps to build conditional probabilities of larger contexts as larger and larger phrases are inserted into the dictionary. So, dictionaries are maintained at system as well as at each mobile terminal. A mobile terminal sends the updates only in coded form. It delays sending of update until a pre-determined interval of time has elapsed. The updates are processed in chunks and sent to the network as a sequence code words of the form $C(w_1)C(w_2)C(w_3)\ldots$, where each phrase w_i, for $i = 1, 2, \ldots$, is a non-overlapping segment of symbols from the string $v_1v_2v_3\ldots$ that represents the LAIs visited by the mobile since the time of sending the last update to the network. So, LeZi update can be seen as a path based update scheme instead of LAI based update.

10.7.4 Incremental Parsing

The coding process is closely inter-twined with the learning process. The learning process works efficiently by creating the dictionary and searching it for the existence of incoming phrases. An input string $v_1v_2\ldots v_n$ is parsed into k distinct phrases $w_1, w_2, \ldots, w_k$ such that the prefix (all symbols except the last one) of the current incoming phrase w_j is one of the previously occurring phrases w_i, $1 \leq i < j$. So, the context statistics related to all prefixes can be updated during the parsing of the current phrase itself. In addition, the prefix property also allows to store the history efficiently in a trie.

Figure 10.8 depicts the trie produced by classical Lempel-Ziv algorithm for the string in the example. The numbers alongside the symbols represent the frequency

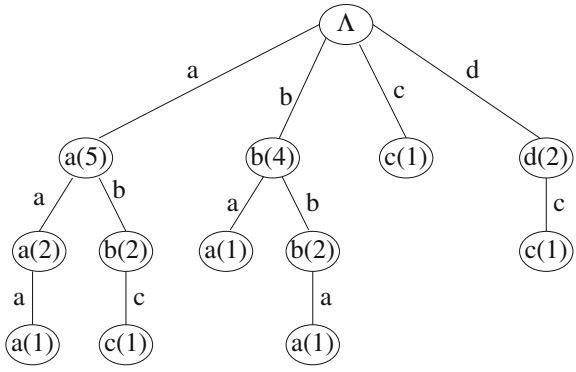


Fig. 10.8 Trie built by classical Lempel-Ziv algorithm

computed by Lempel-Ziv algorithm. The process of computing frequencies has been explained in the previous subsection.

A new dictionary entry can be created by appending one symbol to an already existing phrase in the dictionary. An existing phrase terminates at a node of the trie. An appended symbol to an existing phrase appears as a label of an edge leading from terminating node of the phrase to another. As the incremental parsing progresses, larger and larger phrases are stored in the dictionary. Consequently, conditional probabilities among the phrases starts to build up. Since there is limit to the richness of higher order Markov model, Lempel-Ziv's symbol-wise model eventually converge to a universal model.

As far as personal mobility is concerned, the location updates can be viewed as that of generating a new symbol for the sequence representing the movement history of the form $v_1 v_2 \ldots v_n$. So, the movement history can be parsed into distinct substrings and new update can be inserted into a trie which gradually builds the personal mobility pattern.

However, we cannot use Lempel-Ziv compression based model in a straightforward way. One serious problem with the above compression model is that it fails to capture conditional entropy early on due to following reasons:

- It works on one phrase at a time.
- Decoding algorithm counts only the frequencies of the prefixes of a decoded phrase.
- It is unaware about the contexts that straddle phrase boundaries.

All of the above slow down the rate of convergence. LeZi update uses an enhanced trie, where frequencies of all prefixes of all suffixes are updated. So instead of using the decoder of the previous section LeZi update uses a slightly altered decoder as provide by Algorithm 5. By doing so, it captures the straddling effect.

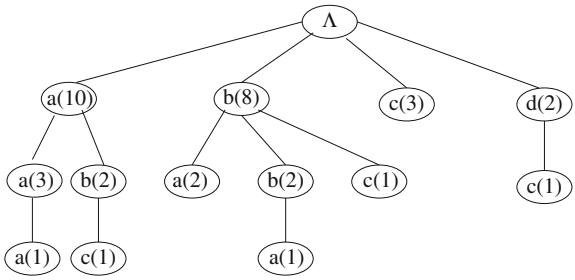
Algorithm 5: Enhanced decoder for LeZi update

```
begin
    // Works for symbols straddling phrase boundaries.
    while (true) do
        wait for the next code word < i, s >;
        decode phrase = dictionary[i].s;
        add phrase to dictinary;
        increment frequency of every prefix of every suffix the phrase;
    end
end
```

The revised frequency counting method for phrases in LeZi update is as follow:

- Find all the prefixes of all the suffixes of each incoming phrase.
- Increment the frequency each time a particular prefix is encountered starting from zero.

Fig. 10.9 Enhanced trie



Let us examine the effect of considering all prefixes of all the suffixes in generating frequency counts for symbols in the example string. For the phrase `aaa`:

- Suffixes are `aaa`, `aa` and `a`, so the frequency counts of all prefixes of `aaa`, `aa`, and `a` are incremented.
- The frequency counts of `aa` incremented by 2.
- The frequency counts of `a` incremented by 3.

The total count for `a` can be found by considering phrases `a`, `aa`, `ab`, `ba`, `abc`, `bba`, and `aaa`. The enhanced trie obtained by LeZi update method is provided by Fig. 10.9.

In order to estimate the effectiveness of the new model for personal mobility against the model that is based on the classical Lempel-Ziv compression scheme, let us compute the entropies for each case. The conditional entropy without considering suffixes (see Fig. 10.8):

$$\begin{aligned} H(V_1) &= \frac{5}{12}\log\frac{12}{5} + \frac{1}{3}\log 3 + \frac{1}{12}\log 12 + \frac{1}{6}\log 6 \\ &\approx 1.784. \text{ bits, and} \\ H(V_2|V_1) &= \frac{5}{12}\left(2 \times \frac{1}{2}\log 2\right) + \frac{4}{12}\left(\frac{1}{3}\log 3 + \frac{2}{3}\log\frac{3}{2}\right) + \frac{1}{6}\log 6 \\ &\approx 0.723 \text{ bits.} \end{aligned}$$

Other two terms of $H(V_2|V_1)$ being 0 are not included in the expression. The estimate for $H(\mathscr{V}) = (1.784 + 0.723)/2 = 1.254$ bits. The conditional probabilities of all order-2 contexts are 0.

When all the prefixes of all the suffixes are considered, the frequency count would be as shown in trie of Fig. 10.9. With enhanced tries, we still have

$$H(V_1) = \frac{10}{23}\log\frac{23}{10} + \frac{8}{23}\log\frac{23}{8} + \frac{3}{23}\log\frac{23}{3} + \frac{2}{23}\log\frac{23}{2}$$
$$\approx 1.742. \text{ bits, and}$$
$$H(V_2|V_1) = \frac{10}{23}\left(\frac{3}{5}\log\frac{5}{3} + \frac{2}{5}\log\frac{5}{2}\right) + \frac{8}{23}\left(2 \times \frac{2}{5}\log\frac{5}{2} + \frac{1}{5}\log 5\right)$$
$$\approx 0.952 \text{ bits.}$$

Therefore, the estimate for $H(\mathscr{V}) = (1.742 + 0.952)/2 = 1.347$ bits when all the prefixes of all the suffixes are considered, implying that the suggested enhancements carry more information. Hence, location update based on the enhancements is expected to perform better than the simple Lempel-Ziv compression method.

10.7.5 Probability Assignment

The prediction of a location for mobile terminal is guided by the probability estimates of its possible locations. The underlying principle behind the probability computation is PPM (prediction by partial matching). PPM uses the longest match between the previously seen strings and the current context. Our interest is in estimating the probability of occurrence of the next symbol (LAI) on a path segment that may be reported by the next update. A path segment is an LAI-sequence generated when traversing from the root to a leaf of the sub-tries representing the current context. The conditional probability distribution is obtained by the estimates of the conditional probabilities for all LAIs given the current context.

Suppose no LeZi type update is received after receiving `aaa` and we want to find the probability of predicting the next symbol as `a`. The contexts that can be used are all suffixes of `aaa` except for itself, namely, `aa` and `a` and Λ (null context). PPM tells that to determine the probabilities only the previously seen phrases should be considered. Therefore, the possible paths that can be predicted with the contexts `aa`, `a` and Λ are as shown in Table 10.7. Start from the highest order context, i.e., `aa`. The probability `a` in this context is 1/3. The probability of a null prediction in the context `aa` is 2/3. It leads to order 1 (with null prediction) context, where the probability of a's occurrence is 2/10 = 1/5. Now fall back to order-0 (with null prediction), which has probability 5/10 = 1/2. The probability of a's occurrence in order-0 is 5/23. So the blended probability of the next symbol being `a` is

$$\frac{1}{3} + \frac{2}{3}\left(\frac{1}{5} + \frac{1}{2}\left(\frac{5}{23}\right)\right) = 0.5319$$

To examine a variation, consider the occurrence of phrase `bba` after receiving `aaa`. The phrase `bba` does not occur in any of the contexts 1 or 2. The probabilities of escape from the order-1 and the order-2 contexts with null prediction are:

Table 10.7 Conditional probabilities of movement prediction

aa (Order-2)	a (Order-1)	Λ (Order-0)
a(1)|aa Λ(2)|aa	a(2)|a aa(1)|a b(1)|a bc(1)|a Λ(5)|a	a(5)|Λ ba(2)|Λ d(1)|Λ aa(2)|Λ bb(1)|Λ dc(1)|Λ ab(1)|Λ bba(1)|Λ Λ(1)|Λ abc(1)|Λ bc(1)|Λ b(3)|Λ c(3)|Λ

- *Order 2 context*: (with phrase aa), the probability of null prediction is 2/3 (out of 3 occurrences of aa).
- *Order 1 context*: (with phrase a), the probability of null prediction is 1/2, because Λ occurs 5 times out of 10.

In summary, the idea of prediction works as follows. It starts with a chosen highest order of context, and then escape to the lower orders until order 0 is reached. This essentially means given a path segment we try to predict the next symbol with high probability.

Therefore, the blended probability of phrase bba is

$$0 + \frac{2}{3}\left(0 + \frac{1}{2}\left(\frac{1}{23}\right)\right) = 0.0145$$

Since a occurs once and b occurs twice in the phrase bba, the individual probabilities of symbols a and b respectively are: (1/3) $\times$ 0.0145 = 0.0048, and 0.0145$\times$ (2/3) = 0.0097.

10.8 Distributed Location Management

In earlier sections of this chapter, we discussed about various location management schemes. Each scheme essentially uses some form of distribution of location information across the network. The underlying ideas is that location information is organized in a way such that the cost and latency in determining the exact location of a mobile device are minimized. Yet, the effect of information distribution on the performances of lookups and updates could be limited for wide area roaming of the mobile users. This is due to the fact that the distant network messages have to be exchanged to support wide area roaming. In this section, we describe a distributed system for storing and accessing location information.

The scheme grew out of the idea of maintaining the location information of a user in a single database at a time instead of replicating in several levels of hierarchy as described in Sect. 10.3. The entire coverage area is divided into several location areas or regions and consists of three levels of hierarchy. Each region consists of a

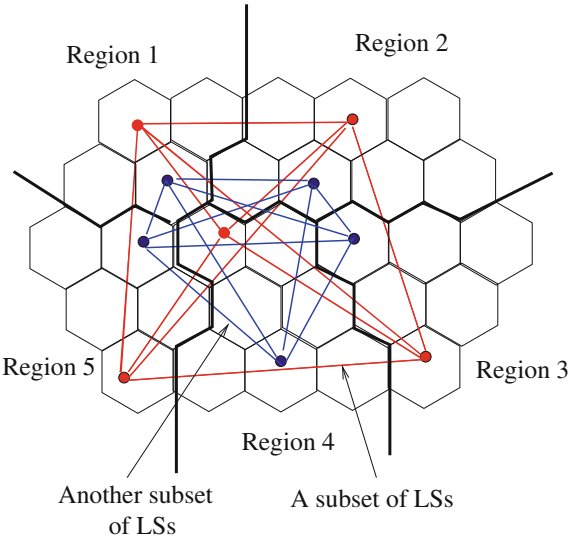


Fig. 10.10 The model

number of mobile switching centers (MSC), and the area under coverage of each MSC consists of a number of cells. Each cell is served by a base station (BS). Logically all mobile users which visit a region are partitioned into groups. There will be a Location Server (LS) in a region of coverage corresponding to each group of the mobile users. The location data of a mobile user is always stored in the LS that corresponds to its group in the region when the user visits a region. The grouping of mobile stations and the region division could be unrelated or related in some way. The matching LSs of the matching groups of mobile users in different regions are connected to each other. In other words, the model consists of

- LSs partitioned up into subsets.
- The LSs belonging to same subsets are interlinked.
- There is a LS of each subset in every region.
- Each mobile user can map to only one such subset.
- The location data of a mobile host MH visiting a region R is stored at a LS in R such that the LS is the member of subset of LSs to which the MH is mapped.

The model is depicted in Fig. 10.10. The regions in the figure are demarcated by heavy lines. Each cell is represented by a hexagonal area. There are two LSs per region, each belonging to a different subset as indicated by filled blue and red circles respectively. Each LS serves roughly for the volume of users in 2–3 cells. The LSs belonging to same subset are fully interconnected. Though the figure does not indicate, each MSC is connected to all the LSs in the same region. In other words, LSs are part of GSM's network subsystem.

10.8.1 The Call Setup Protocol

An MSC initiating a call setup on behalf of a caller sends a request for lookup to the local LS for the required (callee) mobile host (MH). The identity of the local LS can be determined by a simple hash function applied to the callee's ID. The local LS looks in its own registers. If the entry is found then LS returns the cell location of the callee. The requesting MSC can then carry out the call switching between the caller and the callee using the GSM call set up procedures as explained in Chap. 3. If the entry is not found, then the local LS multicasts a request message for the callee to the other LSs belonging to the same subset. If the callee is registered (it should be on) then one of LSs should reply. Following which the call switching is carried as in the previous case.

10.8.2 Update

When a MH moves from one location to the other its location information should be updated. It works as follows. The MH reports for registration to an BS under an MSC in the region where it moves. The MSC applies a hash function to determine the subset of LSs that maintain the location for MH. The process of update depends on the type of the move made by the MH. The moves are of the following types.

- Move within the region.
- Move between the regions.

If a MH moves from one cell to another cell under a same MSC then no update is required. However, if the MH moves from a cell to a cell under a different MSC in the same region, then an entry will be found in the local LS of the subset. The LS will update the information and send a *de-registration* to old MSC.

If the MH did not belong to the current region, i.e., it moved from a different region, the local LS multicast the information to the other LSs of its own set in the neighborhood. The new LS would know the possible regions from which the MH may have come. These regions are those which have common border with the current region. It depends on the location of the current region and its shape. One of the LSs of the neighbouring region must have an entry corresponding to MH. The LS having the entry then delete the same, and sends a delete entry message to the MSC which was serving the MH in the region. In case the registration message from the mobile device itself includes the ID of the previous LS or the region, the current LS can send unicast message to the previous LS.

10.8.3 Data Structures and System Specification

The important data structures for implementation of the lookup and update protocols are:

- `location[MH_id]`. This is the set of data registers available at an LS. The entry in each register is the `Cell_id` of the MSC which has the MH with `MH_id`. Additionally it may contain entries for all MHs whose hash function map to LS. This information would help in call setup between the mobiles belonging to the same LS set.
- `LSforRegion[Region]`. This is a table to lookup the LSs (other members of the same subset) which are associated with the LS holding the table. Each MH must be associated with one LS in each region which would contain the location of the MH.
- `neighborRegion[MSC]`. This is an array of neighboring regions for a particular MSC. If the coverage area of an MSC is completely within the interior of a region, the array will be empty. On the other hand, the border MSC (whose coverage area forms a part of the region's border) will have a number of neighboring regions depending on the shape of the regions.

As shown in Algorithm 6 a simple hash function can be used to determine the index of the subset of LSs. Hence, the specific LS in the current region associated with a MH can be found by applying the hash to ID of the MH. The variable `numLSperRegion` defines the number of LSs in a region. This helps in determining the load on each LS.

Algorithm 6: Method get_LSId(MH_id)

```
begin
    return MH_id mod numLSperRegion; // Apply hash to get LS ID.
end
```

A MH is expected to send updates about its locations from time to time. The frequency of update depends on the update protocol. On receiving a location update, at first a local update is attempted by the current MSC. If the conditions for a local update are not met, then a remote update is attempted by the MSC. Suppose an MH moves from a cell under MSC_{old} and joins a cell under another MSC, say, MSC_{new}, where both MSC_{old} and MSC_{new} belong to the same region, then a local update is needed. Algorithm 7 specifies local update method. However, no update is needed for the movements of an MH between cells under the same MSC, because GSM underlay takes care of updating VLR.

The update method in Algorithm 8 is executed by a LS when it is informed about the movement of mobile host to the current region. As the update method indicates, in case an MH has moved from a region outside the current region then

Algorithm 7: Method localUpdate(MH_id, MSC_{old}, MSC_{new})

```
begin
    // Executed by the MSC on behalf of a MH.
    LS_id = get_LSId(MH_id);
    LS[LS_id].localUpdate(MH_id, MSC_old, MSC_new);
end
```

`remoteDelete` method should be executed by the new LS storing location update from the roaming MH. Algorithm 9 gets invoked to delete entry for MH from the LS in the neighborhood. Only one of the LSs in the neighborhood may have a entry for the MH.

Algorithm 8: Method update(MSC, MH_id)

```
begin
    // Executed by LS of the region.
    if (exists(location[MH_id]) then
        if (location[MH_id] != MSC) then
            // MH moved within same region but from one MSC to
               another.
            replaceEntry(MH_id, MSC);
        end
        // Do nothing if location is unchanged
        return;
    end
    // MH was not in region under the LS of the current region,
       it must have arrived to the current region from a
       neighboring region.
    for (i ∈ neighborRegions[MSC]) do
        // Issues a remoteDelete to delete MH entry from LSs of
           the neighborhood region. Only one LS have such an
           entry.
        remoteDelete(i, MH_id); // See the Algorithm 9.
    end
    insertEntry(MH_id, MSC); // Update needed in current region.
end
```

Algorithm 9: Method remoteDelete(region_Id, MH_id)

```
on receiving (remoteDelete) begin
    if (exists(location[MH_id] in LS[region_Id]) then
        deleteEntry(MH_id);
    end
end
```

The method `remoteDelete` is called by an LS when it determines that a MH has moved from a region outside its own region. It sends remote delete request to LSs of its own set in the neighboring regions. It is assumed that the methods for inserting, deleting, or replacing an entry have obvious implementations and the details of these specifications are not provided here.

A lookup is necessary for paging during call set up. A local lookup is executed by the MSC of the caller to determine if the callee belong to the same region. A region based calling mechanism can be useful for modeling tariffs by a GSM service provider. Algorithm 10 illustrates local lookup method.

Algorithm 10: Method lookup(MH_id, MSC)

```
begin
    LS_id = get_LSId(MH_id);
    return LSforRegion[LS_id].localLookup(MSC, MH_id);
end
```

A local lookup fails in the case the local associated LS does not have an entry for the callee. In this case, a remote lookup initiated by local LS by sending a multicast message to the other LSs belonging to the same subset as the local LS in the caller's region. Algorithm 11 combines both local and remote lookup into a single method. One final part of the lookup method is provided by Algorithm 12. It allows a LS to check the location entries when LS receives a lookup message. If an entry is found then it returns the entry. Otherwise, it returns NULL indicating a failure.

Algorithm 11: Method localLookup(MSC, MH_Id)

```
begin
    if (exists(location[MH_id]) then
        // Local lookup successful.
        return location[MH_id];
    end
    else
        // Perform remote Lookup for MH.
        for each (region i)
        remoteLoc = LSforRegion[i].remoteLookup(MH_id);
        if (remoteLoc != NULL) then
            return remoteLoc;
        end
        // All remote lookups fail. MH must be in power off
           mode.
        return NULL;
    end
end
```

Algorithm 12: Method remoteLookup(MH_id)

```
on receiving (a lookup message) begin
    if (exists(location[MH_id]) then
        return location[MH_id];
    end
    else
        return NULL;
    end
end
```

10.8.4 The Cost Model

The cost of each operation required to update the LS entries as MH moves and the find an MH can be modeled using a set of simple assumptions as listed in Table 10.8.

Local Update Cost.

Using the above cost model, the cost of a local update can be computed by analyzing the involved steps.

- First, new MSC must find out its associated location server LS. This requires a simple application hash function. The cost of this step is H.
- New MSC send a message to its local LS for update. This cost is t.
- The LS, on receiving the message, performs a search or register lookup for update which is RL.
- The new LS must send information of this update to the old LS. This enable old LS to purge the entry. The cost of sending this message is t.
- After receiving the information the old LS deletes its register entry. The cost for deletion is MD.

Adding all the cost together, the total update cost: $2t + H + RL + MD$

Table 10.8 Notations used for cost analysis

Notation	Description
t	Time unit for delivery of local network message
X	A multiple of cost of the delivery of remote network message as compared to the delivery of a local network message
RL	Register lookup time at an LS
H	Time to perform hash
MD	Time to perform delete MSC
LN	Lookup time in the neighbouring regions for the new MSC

Remote Update Cost.

The break-up for the cost computation of a remote update is as follows.

- New MSC hashes to determine LS subset which holds the entry. The cost of performing hash is H.
- After identifying the LS subset, New MSC must sends a message to LS belong to subset which is available in local region. The cost of sending this message is t.
- The LS of the local region then determines probable neighbouring regions for the new MSC. This cost is LN.
- The LS then sends delete MSC message to the all the neighbouring regions incurring a cost of Xt
- The next step is to search remote LS for MH entry which takes time RL
- The remote LS sends message to old MSC for deletion incurring a cost of t.
- Old MSC then deletes the MH entry with a cost of MD.
- Local LS now performs a register lookup for creating a new entry for MH. For this it incurs a cost of RL.

The overall cost is obtained by adding all the cost mentioned above. It, therefore, works out as $2t + Xt + H + 2RL + MD + LN$.

Similarly, the cost for local lookup time and the remote lookup time can also be found. These costs are:

1. Local lookup time = $2t + H + RL$.
2. Remote lookup time = $2t + Xt + H + 2RL$.

References

1. I.F. Akyildiz, J.S.M. Ho, On location management for personal communications networks. IEEE Commun. Magaz. **34**(9), 138–145 (1996)
2. I.F. Akyildiz, J. McNair, J. Ho, H. Uzunalioglu, W. Wang, Mobility management in current and future communications networks. IEEE Netw. **12**(4), 39–49 (1998)
3. A. Bhattacharya, S.K. Das, Lezi-update: an information-theoretic framework for personal mobility tracking in pcs networks. Wireless Netw. **8**, 121–135 (2002)
4. T.M. Cover, J.A. Thomas, *Elements of Information Theory* (Wiley, New York, 1991)
5. J.S.M. Ho, I.F. Akyildiz, Dynamic hierarchical database architecture for location management in pcs networks. IEEE/ACM Trans. Netw. **5**(5), 646–660 (1997)
6. R. Jain, Y.-B. Lin, An auxiliary user location strategy employing forwarding pointers to reduce network impacts of pcs. Wireless Netw. **1**(2), 197–210 (1995)
7. J. Jannink, D. Lam, N. Shivakumar, J. Widom, D.C. Cox, Efficient and Flexible Location Management Techniques for Wireless Communication Systems, *Mobicom '96* (1996), pp. 38–49
8. E. Pitoura, G. Samaras, Locating objects in mobile computing. IEEE Trans. Knowl. Data Eng. **13**(4), 571–592 (2001)
9. K. Ratnam, I. Matta, S. Rangarajan, Analysis of Caching-Based Location Management in Personal Communication Networks, *The Seventh Annual International Conference on Network Protocols (ICNP '99)*, Washington, DC, USA, 1999 (IEEE Computer Society, 1999), pp. 293–300

10. C.E. Shannon, The mathematical theory of communication. Bell Syst. Techn. J. **27**, 379–423 (1948)
11. N. Shivakumar, J. Jannink, J. Widom, Per-user profile replication in mobile environments: algorithms, analysis, and simulation results. Mobile Netw. Appl. **2**(2), 129–140 (1997)
12. C. Song, Q. Zehui, N. Blumm, A.-L. Barabási, Limits of predictability in human mobility. Science **327**(5968), 1018–1021 (2010)
13. J.I. Yu, Overview of EIA/TIA IS-41, *Third IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '92)* (1992), pp. 220–224
14. J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding. IEEE Trans. Inform. Theory **24**(5), 530–536 (1978)