Location Management and Regional Directories

R. K. Ghosh

Feb-March, 2012

5.1 Introduction

Consider a doctor moving around in a big hospital attending patients. S/he may have to be paged frequently by the monitoring staff attached to critical patients. A service personal carrying a pager often have to be tracked and located in order to attend service requirements of the clients. Ambulances are usually fitted with tracking devices to lead the paramedicos to attend the emergencies and the rescue operations, and ferrying the critical patients to the nearest medical assistance centers or hospitals. Similarly, police petrol vehicles are always equipped with tracking devices, so that the control room can locate the nearest petrol to rush to a place of crime, disturbances or a distress call as fast as possible. In commerce and business, a timely response could provide a competitive edge to a business establishment over its rivals. Therefore, more often then not such organizations equip the executives and the field representatives with mobile cellular phones, or PDAs. All these mobile devices need to be tracked and located to communicate with the users. In the context of computing mobility assumes further dimensions such as a laptop or PDA user continue to access shared file systems or data repositories while on the move in cars, aircrafts or ships. The ramifications of the suggested movements associated with computing can be witnessed in migration of software (called mobile agents), transaction models with mobility support, collaborative computing, etc. Some of the issues are discussed in other chapters of this text. But the key to the various issues arising out mobility support is the tracking of the mobile objects. The object can be 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 and/or locate a moving object with an intention to communicate.

At an abstract level location management involves two operations which are also found to be basic to the usage of a database.

- Look up or search, which must be invoked each time a mobile object is to be located.
- Update, which is invoked each time the object moves.

But there are significant differences in the operations with a database and the location management. The differences are

- A database has precise data. But the location management must have to deal with the imprecision of various forms in location data.

- The update requirements in maintaining location data will depend on the degree of tolerance in of imprecision in the location of a mobile object. Whereas the update requirements in a database is directly linked with the consistency problem and, therefore, have to be done when needed.

- The scale of updates in location management is huge. This basically relates to the fact that there is absolutely no control in proliferation of mobile devices. For example, in a cell size of about half a kilometer radius in an average a thousand to thousand and five hundred mobile devices may be active. Assuming every object makes about five moves in a day, the total number of updates exceeds 10,000 in just one cell calculated at the rate of one insertion and one deletion per move per mobile object.

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

- *Availability* – refers to availability of location information at all network sites or at some selected places in the network.

- *Imprecision* – refers to the exactness of the location information.

- *Currency* – refers to the regularity of location updates.

An abstract view of the space of location management problem is depicted in figure 5.1. If exact information is maintained each and every network site, the updates have to be tied with every move and disseminated to all the sites, whereas look up becomes immediate. The other extreme case could be no information about location of any object is maintained anywhere. This case is represented by the origin of the frame of reference shown in the above figure. The search for a mobile object in this situation becomes an exhaustive search at each network site. In between the above two extremities several combination of approaches regarding location update and look up can be possible.

It may be mentioned here that the implicitly assumption is location management is supported by a cellular network architecture acting as the backbone. Of course, cellular architecture is not the only possibility. But it is a good candidate to understand the issues that arise in the location management. In absence of a infrastructures backbone architecture, an alternative

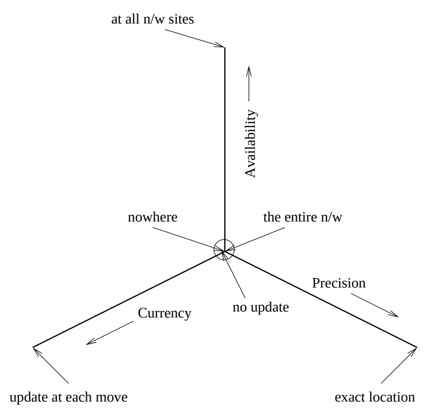

Figure 5.1: The space of location

approach such as global positioning system has to be used to track or locate mobile objects. In the discussion here we assume the mobility support is provided by cellular architecture. The location management is handled in the data link or network layer.

5.1.1 Registration and Paging

There are two distinct types of mobilities involving mobile devices. *Terminal mobility* allows a terminal to be identified by some identity independent of its point of attachment in the network. It allows the terminal to be attached to different points of the fixed network at different time. Therefore, allows mobility with respect to point of attachment with the fixed network.

Personal mobility is associated with a user. A user gets a distinct identity which is independent of terminal s/he uses as well as the point of attachment of that terminal in the network. Personal mobility allows the users to receive or make calls any where through any terminals. To facilitate the tracking, each mobile user has to explicitly register itself to notify the server or the sites on the network fixed about its location. The location registers stores these location data. The granularity of location can be a single cell to a group of cells. When the location is known a user can be tracked by paging. The paging process is a polling by the system by sending signals to likely locations to track the users. By changing the size of registration area, flexibility of combination of paging and registration can be attained.

5.2 Two Tier Structure

In two tier scheme [9] there are two location registers per mobile hosts, (i) *home location register* (HLR) and (ii) *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 normally a zone where the user was registered first, and may usually be dependent on the normal area of movement of the user. A HLR maintains the current location of an user. Hence, a HLR gets updated every time the user makes a move. Each zone also maintains a set of VLRs each of which is associated with a user currently active in that zone but is not a resident of the zone.

In HLR-VLR scheme the call setup is quite straightforward. A user A in location area LAI_i wanting to communicate with another user B in 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 update process coupled with movement and supporting the call setup procedure could be as follows:

- 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, and

- A VLR entry for the user is made in LAI_j, besides deleting the entry for the user from VLR of LAI_i.

Figure 5.2 illustrates the movement of user A from cell under LAI_i to a cell under LAI_j. The LAI_a denotes the home location of A.

5.2.1 Drawbacks of Fixed Home Addresses

The home location is permanent. Thus long-lived objects can not change home, i.e., resettlement at a different neighbourhood is not allowed. The two-tier approach does not scale up well with highly distributed system. Often the distant home locations may have to be contacted for for look up even in case of moderately mobile objects. Thus, the locality of the moves is not captured well.

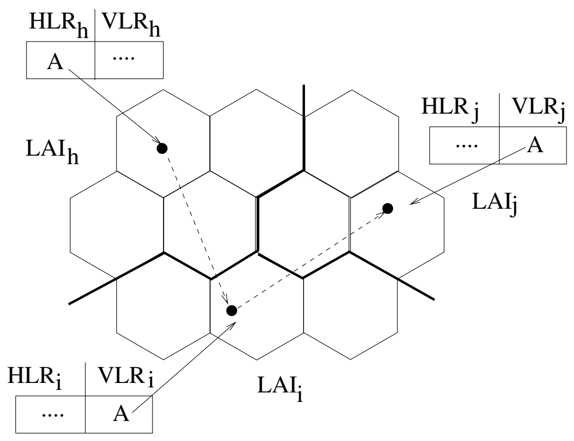

Figure 5.2: Move in two-tier scheme

5.3 Hierarchical Scheme

Hierarchical schemes [4] are designed to take advantage of locality. 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 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-base hierarchical scheme is illustrate in figure 5.3. 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, namely,

1. The relative frequency of moves of the users.
2. 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 expected number of calls received, and let U_i be location area crossings made by a mobile user mu_i over a period of time T. The fraction

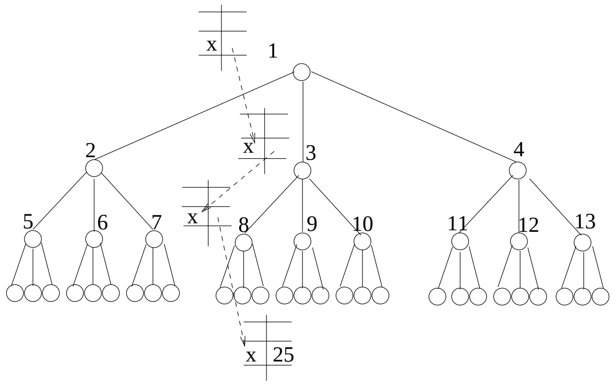

Figure 5.3: Tree-based organization of location servers

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 mu_i 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.

5.3.1 Update Requirement

When location information at internal nodes in the hierarchy are stored as pointers, and a 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 down:

- from $LCA(LAI_i, LAI_j)$ to LAI_i and
- from $LCA(LAI_i, LAI_j)$ to LAI_j

should be updated.

For example in the figure 5.3 each node represents a location server. Each leaf servers store location information of mobile users in one location area. When a mobile user mu moves from a cell in the location area under server 25 to a cell in location area under server 26, then updates will be needed at the nodes 3, 8, 9, 25, 27.

- At 8, 25 entries for mu should be deleted.

- At 3 entry for mu should be updated.
- At 9, 27 new entries for mu should be made.

If actual cell id information is stored at each node from the root to leaf, then path from the root to $LCA(LAI_i, LAI_j)$ also has to be updated. For example, at the time mu was in the cell under 25, nodes 1, 3, 8, 25 had entries pointing to cell 25 which indicates that mu was in cell under 25. So, when mu moves to cell under 27, all the nodes 1, 3, 9, 27 must be updated to indicated mu is now under cell 27.

5.3.2 Lookup in Hierarchical Scheme

The look up operation in hierarchical scheme can start with search initiated at the neighbourhood, i.e., bottom up in the tree hierarchy. If the mobile object being search for is not found in the current level of the hierarchy then the search process is just pushed up one level up the hierarchy. Since the root maintains information about all the objects, the look up is bound to succeed at a level of the hierarchy. In particular, suppose a search is initiated at a cell under LAI_i for the mobile object which is currently location in a cell in LAI_j. Then, look up starts at LAI_j itself and finally reaches $LCA(LAI_i, LAI_j)$ where an entry for the object will be found. After that following that pointer the search is guided down the hierarchy along the path $LCA(LAI_i, LAI_j)$ to LAI_i, and the object is thus found.

5.3.3 Advantages and Drawbacks

The tree-based hierarchical scheme does not require HLR and VLR type stores to be maintained at every cell. There is support for locality of call setup process. On the other hand, updates have to be done at a number of location servers on different levels of hierarchy. The location information for each mobile object has to 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 internal nodes up with the hierarchy. The storage requirement also increases at higher levels.

5.4 Caching

Caching is used primarily to reduce the lookup cost. However, it also helps to reduce the delay in establishment of link. The idea of caching similar to conventional use of caching. In a two-tier architecture, when a large number of calls originate from a particular MSC to a particular mobile belonging to a different MSC, then the mobile id and the serving VLR address can be stored at the calling MSC. It will not only reduce the signaling cost but also the delay in establishing the connection. Each time a call is attempted from the calling MSC overhead associated with caching is as follows:

- At first, the cached information of VLR is checked at calling MSC. If a cache hit occurs, the VLR of the callee is contacted directly. It helps to reduce the delay in establishing a connection with the callee and avoiding the unnecessary overhead for HLR lookup.

- When the callee mobile moves out, cache miss will occur. Then HLR of the callee is contacted to establish the call.

Checking cache at calling MSC adds little to overhead. On the other hand, the saving on signal cost and improvement on establishment of calls may be substantial. However, when callee's mobile crosses over to a new location area, an update cost is incurred. Old cache entry becomes invalid. So a cache revalidation will be needed by updating the old entry.

From point of view of pure location update (no cache), it involves new VLR address should registered with HLR [5] while old VLR must send cancellation of registration to HLR. So, the total cost is represented by:

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

In absence of cache, HLR is queried and data is provided by HLR from VLR of the new region where the callee mobile is found. So, the cost of lookup 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 total cost for establishing calls will be

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

Now we need to get an estimate of the expected number of calls ρ received by a mobile from the area under an MSC. This estimation requires analysis of call traces. A study [7] 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. Let f_k denote the proportion of calls the user receives from the location area ranked $k = 0, 1, \ldots, N_{lai}$. Let us assume that call arrival times to a mobile in a service area is exponentially distributed with mean arrival rate λ. Then probability distribution function is: $\lambda e^{-\lambda t}$ Also 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 cached information for a mobile at a location area is correct. In other words, p_{cache} represents the cache correctness ratio. At steady state, p_{cache} also defines the probability that mobile has not moved from its current location since it received the last call from same location area. So,

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

Let C_B represent the lookup cost of connection setup in basic scheme (no caching is used), and C_H be the cost when caching is used. Note that C_H is the cost of cache hit in VLR of calling MSC and retrieving information from callee's VLR. C_B, on the other hand, involves (i) the cost of invalidating VLR entry of calling MSC when mobile crosses to a new location, and (ii) the cost of updating mobile HLR entry.

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

A cost saving is achieved 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 if caching to be useful in saving cost, the threshold p_T for cache hit should be:

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

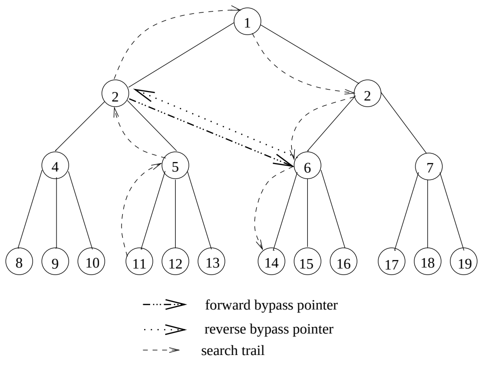

Figure 5.4: Caching in hierarchical location scheme

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

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

5.4.1 Caching in Hierarchical Scheme

In a hierarchical architecture, caching can be deployed to reduce the lookup time. Combined with *forward bypass pointers* and *reverse bypass pointers* it is place cache at a higher level of the hierarchy to serving the calls originating from a group of cells rather than a single cell. The tradeoff is the calls have to traverse a longer path if cache is placed at a higher level. The idea is illustrated by figure 5.4. When a call is requested from LAI_i to LAI_j, the call traverses up the tree from LAI_i to $LCA(LAI_i, LAI_j)$ and then downwards to LAI_j. The call setup requires an acknowledgement to be sent from LAI_j back to LAI_i along the same path. All the ancestors of LAI_i in the hierarchy overhear this ack message, and one of them, say LS_i can create a forward bypass pointer to an ancestor, say LS_j, of LAI_j. Likewise, LS_j can create a reverse bypass pointer to LS_i. The cache can be deployed at LS_i. Then subsequent calls from LAI_i reach the location database LS_j for a user in LAI_j via the shorter route using the forward bypass pointer at LS_i. Similarly, the acknowledgement message from LAI_j can traverse a shorter path using the reverse bypass pointer from LS_j to LS_i. If the level of LS_i is high then all the calls originating from its subtree to the cells in the subtree of LS_j can use the bypass pointers. But each call may have to traverse a longer path up to LS_i and spend longer lookup time since time to locate the callee in the subtree of LS_j would depend on its size.

5.5 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 old location can be re-routed to the current location by the forward pointer. While the update of database entries holding user location can be updated selectively.

In two-tier architecture, there may be excessive update traffic between the location of current VLR of the user to the HLR location of a user when it is far away from home frequently, and moves in the neighbourhood of the current locations. In order to avoid that, forward pointers can be left at VLR of the serving location. When a call to the user reaches its home location sever, the call is diverted to the present location from its home location following a chain of forward pointers. The chain of forward pointers are allowed to grow up to a fixed predetermined length of K. When the user revisits a location, the potential loop condition in forward pointers chain is avoided by an implicit compression. 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 done 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 5.5 illustrate the two schemes for placing the forward pointers. The solid pointers indicate the simple forwarding and dashed pointers illustrate the level 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

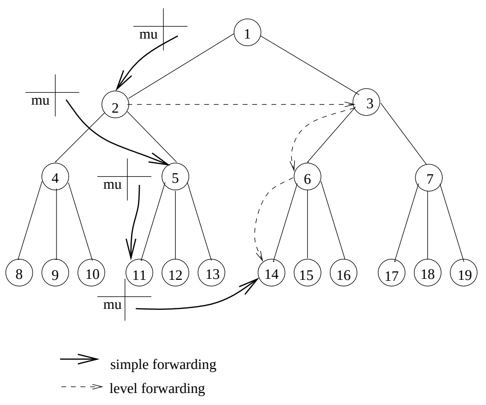

Figure 5.5: Forwarding pointers in hierarchical scheme

level location server 11 as indicated by the solid pointers. Whereas in the case of level forwarding, a forward pointer is placed at the ancestor location server 2 at level 3 which points to the location server 3 which is an ancestor of the new location. In this case, updates are more than that required in simple forwarding. This is because, 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.

5.6 Replication

Replication is another mechanism to reduce the cost for lookup. The basic criterion for replication of location of a user at selected nodes in the hierarchy is the large call to mobility ratio.

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

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

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

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

On the other hand if mu_i makes a move:

- inequality is evaluated for every member of the working set
- if it fails to hold for any member 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 [3].

Apart from storage constraint, the decision to place a replica should also be based on minimization of network communication cost. In two-tier model, 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 hierarchy. However, 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 [3]:

1. If $\text{LCMR}_{ij} < R_{min}$, replica of i's profile is not placed 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 replicas.
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. Unconstrained number of replications can put severe pressure in maintenance of the replication even for a moderately mobile user. The reader is referred to [3] for details of analysis and the algorithm for placing replication.

5.7 Personal Mobility

In the context of location management, there is trade off between search and update. Most location management schemes are based on the approach to achieve balance between the two. To what extent the the trade off can swing between update and search will depend on the bound signaling requirements for location management. But the question is how to define a bound on signaling requirement? Furthermore, even if a bound can be defined, is it possible to reach the bound? In order to seek answers to these questions, it is important to understand that terminal mobility and personal mobility are tightly coupled. In reality a mobile equipment at best is a portable device, it cannot move on its own. The movement of a mobile is caused by the movements of the user. The movement of a person or the user of a mobile can be considered as a path in some random process.

5.7.1 Random Process, Information and Entropy

Let us explore a bit about randomness of a process. If the volume of information content of 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* [8]. Because it provides a measure of an element of surprise in occurrence of one event compared to another. If the information content is high then the surprisal is low. Similarly, if the information content is low then surprisal is high. For example, if an event has a probability 1, then it occurs always or its surprisal is zero. On the other hand, an surprisal an event with zero probability is infinity. This implies that surprisal of an event A defined by the inverse of the probability of occurrence of A.

5.7.2 Shannon's Entropy Formulation

Shannon [6] formulated the surprisal in terms of entropy. Shannon's formulation capture the following two important aspects:

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

From a practical prospective, it implies that a large information base is needed for predicting any future events in a system with large entropy. Since a terminal movement is tied to the personal mobility of its user, for predicting future locations of a mobile terminal we need more information about the movements of its user. So, Shannon's entropy formulation can be used the calculating the optimal bounds.

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 die. If the die 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 possible to disambiguate the block of bits representing the results of different rollings of the die. 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 die 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 showing up, the shortest code should be used for 1. Likewise, 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:

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

The above encoding satisfies the prefix property. The average number of bits needed for encoding the result of 1000 rollings is

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

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

As another example, consider flipping of a biased coin. Let head show up just once in 1000 flips. Suppose the coin is tossed one million times. Let a 0 represent the fact that the result of a toss is a head, and similarly let 1 represent a tail. 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 to 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 die with six faces to find an answer to the above question. If the die is unbiased, the probability of occurrence of any value is $p = 1/6$, the number of bits required $= \log 6 = -\log(1/6) = -\log p$. The result of one throw requires $\log 6 = 2.58$ bits. It is not immediately apparent how the result any particular throw of a die 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 to 255 can be coded using 8 bits. Thus, for a biased probability distribution, the number of bits required optimal code is determined by

$$-\sum_{x} p(x) \times \log p(x).$$

The examples we have discussed so far, point to the fact that a rare event has a very low probability. Hence, the information content related to occurrence of a rare event is high. 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 occurrence of a head. As against this information contents in the probability of appearance of a tail is $-\log(999/1000) = 0.0044$ bit. So the average information content is

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

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 [6], $H(A)$ represents entropy of A.

Now consider X and Y, a pair of discrete random variables with joint probability distribution $p(x, y)$, where $x \in \mathcal{X}$, $y \in \mathcal{Y}$ then joint entropy of X and Y is given by the expression:

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

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

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

Joint entropy and conditional entropy are closely related. Joint entropy can be expressed as the sum of entropies of the first random variable and the conditional entropy of second random variable given the first.

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

The generalization of the above result is known as chain rule. It 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:

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

5.7.3 Mobility Pattern as a Stochastic Process

In order to capture the personal mobility, we may view a user's movement as a random process. With a cellular infrastructure, a user's movement can be captured as a sequence of cells. In a GSM type network, each symbol will represent a location area identity (LAI) consisting of several cells. There is a possibility that certain substrings of LAIs are repeated in the sequence of LAIs representing a user's movement. The repetitions represent locality of movement. By characterizing mobility as probabilistic sequence, mobility can interpreted as a stochastic process.

At first we need to be clear on two issues:

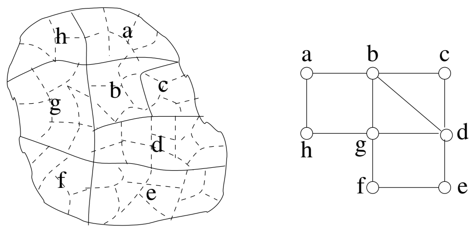

Figure 5.6: GSM type location area map (source [1]).

1. How a user would move in a service area?
2. How movement can be recorded?

Figure 5.6 illustrates an example for a service area consisting of eight LAIs a, b, c, d, e, f, g, h. The shapes of actual cells are not quite hexagonal, but have irregular geometric contour. The contour of a cell is 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 update of location as a user moves and enters new LAIs.

How location update can be recorded? The mobiles send their updates to network subsystem in periodic intervals or on crossing the boundary of an LAI. Thus, location updates could be distance based, time based, movement based or combinations thereof.

- Distance based update: in the distance based updates, mobiles keep track of the Euclidean distance from the time last update was sent. Then if distance travelled from the last update crosses a threshold D, the mobile sends 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.

Let us consider an example to understand how different updates schemes would generate the updates. Suppose the service is started at 9.00AM. An example of LAI crossings is provided in table below.

Crossings in morning			
Time	11:04	11:32	11:57
Crossing	$a \to b$	$b \to a$	$a \to b$

Crossings in afternoon			
Time	3:18	4:12	4:52
Crossing	$b \to a$	$a \to b$	$b \to c$

Crossings in evening				
Time	5:13	6:11	6:33	6:54
Crossing	$c \to d$	$d \to c$	$c \to b$	$b \to a$

With the movement history shown in previous slide, the LAI sequences reported by various update schemes will be as shown below.

Update scheme	:	Movement history
$T = 1$hr	:	$aaabbbbacdaaa\ldots$
$T = 1/2$hr	:	$aaaaabbbbbbbaabcddcaaaa\ldots$
$M = 1$	:	$abababcdcba\ldots$
$M = 2$	:	$aaacca\ldots$
$T = 1$hr, $M = 1$	:	$aaababbbbbaabccddcbaaaa\ldots$

The movement history of a user can be 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 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 random variables, and each V_i is take a value v_i from set $\mathcal{V}$. 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 user of a mobile device is most likely to exhibit the preference for visiting any known sequence of LAIs in a time invariant manner. The correlation between visiting adjacent LAIs of the sequence remain unchanged across 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 examine the possible models for evolving 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 probabilities of all LAI residences are the same, i.e., 1/8 for each of the 8 LAIs for the example.

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

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

The string consists of 23 symbols, the probability of a symbol is determined by its relative frequency, since it is independent of that for other symbols.

The simplest Markov model is a Markov chain where distribution of the random variable depends only on distribution of the previous state. So, this model assumes the stochastic process to be a 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 for k and i. Since, the LAIs e, f, g, h are not visited at all, each acquire zero probability. 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 $v_i, v_j \in \{a, b, c, d\}$, are estimated by relative counts. So, movement profile can be represented by the corresponding transition probability matrix:

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

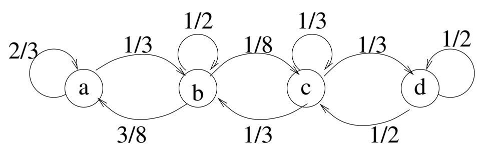

Figure 5.7: One step state transition diagram for personal mobility.

The state transitions with respective probabilities are shown in Figure 5.7. Let $\Pi = [p(a)\ p(b)\ p(c)\ p(d)]^T$ be steady state probability vector. Solving $\Pi = \Pi \times P$ with $p(a) + p(b) + p(c) + p(d) = 1$, we obtain $p(a) = 9/22$, $p(b) = 4/11$, $p(c) = 3/22$ and $p(d) = 1/11$.

To summarize the above discussion, we notice that IM can never be adaptive. IID is the first step toward adaptive modeling. The optimal paging strategy dependents on the independent probabilities of symbols $\{a, b, c, d, e, f, g, h\}$. Order-1 Markov model carries information to the extent of one symbol context. For uniformity, we may refer to IM as order-(-1) Markov model, and IID as order-0 Markov model respectively. Order-2 Markov model can be constructed by counting the frequencies of symbols appearing in order-2 contexts for the sequence *aaababbbbbaabccddcbaaaa*. The table below provides frequencies of different symbols for all three contexts.

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

According to order-1 model the probability of the route *abcbcd* being taken:

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

It is unlikely that any user will ever take such a zig-zag route. Though according order-1 MM the probability is very low, it is not zero. However, if order-2 model is applied then the proposed route will be impossible. So, the richness of 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 [2], implying that the advantage of higher order contexts dies out after a finite value. To appreciate this fact, let us compute the entropy 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 $v_i \in \{a, b, c, d, e, f, g, h\}$ in the chosen running example. Due to independence of events, $p(v_n|v_1, \ldots v_{n-1}) = p(v_n)$. Therefore, the joint probability $p(v_1, v_2, \ldots, v_n) = \{p(v_1)\}^n$, and both per symbol entropy $H(\mathcal{V})$, and the conditional per symbol entropy $H'(\mathcal{V})$ are same. Using the definition

  $$H(\mathcal{V}) = \lim_{n\to\infty} \frac{1}{n} H(V_1, V_2, \ldots, V_n) = \log 8 = 3.$$

  So, we get the per symbol entropy rate as 3bits 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,

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

  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 $p(a) = 9/22$, $p(b) =$

4/11, $p(c) = 3/22$ and $p(d) = 1/11$, we find

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

Thus using order-1 MM, we get a per symbol entropy rate of 1.194 bits. It improves the entropy rate substantially over order-(-1) and order-0 models.

Note that 3bits are sufficient to represent any symbol from a space of eight symbols $\{a, b, c, d, e, f, g\}$. So order-(-1) model can not resolve uncertainities 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 $H(\mathcal{V})$ and $H'(\mathcal{V})$ are same in both order-(-1) and order-0 MM due to independence of the events related to symbols.

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

5.7.4 Lempel-Ziv algorithm

LeZi update is based on Lempel-Ziv's text compression algorithm [10]. 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 builds as phrases keep entering. LeZi update's encoder is identical to the encoder of Lempel-Ziv's algorithm. It runs at mobile terminal.

```
//  Encoder at mobile or compressing algorithm.

dictionary = null;
phrase w = null;
```

```
while (true) {
     wait for next symbol v;
     if (w.v in dictionary) w = w.v;
     else {
        encode < index(w), v>;
        add w.v to dictionary; w = null;
     }
}
```

The encoding process can be best explained by executing it on an string of symbols. Let the example string be:

$$aaababbbbbaabccddcbaaaa$$

The table below illustrates how encoding each phrase is realized.

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)

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 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 1*a*. Therefore using encoding algorithm the string *aaababbbbbaabccddaaaa* is encoded as: 0*a*, 1*a*, 0*b*, 1*b*, 3*b*, 5*a*, 4*c*, 0*c*, 0*d*, 9*c*, 3*a*, 2*a*.

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 appears below.

```
// Decoder at the system side.
// It basically decompresses the string

while (true) {
   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;
}
```

To see how the decoding process works, we consider code word specified in terms of tuples <index, last_symbol> and illustrate the decoding with help of the table below.

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

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 contatenated with retrieved phrase, i.e. a is appended to λ. Since $\Lambda + a = a$, 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*.

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 both system as well as each mobile terminal. The 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.

5.7.5 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 symbos 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 5.8 depicts the trie produced by classical Lempel-Ziv algorithm for the string in the example. The numbers alongside the symbols represent the frequency computed by Lempel-Ziv algorithm. To see how frequencies are obtained, consider the parsing of *aaa*. It leads to frequency counts of a, *aa* being incremented by 1 each, as both are prefixes of the phrase *aaa*. Of course, the frequency count of *aaa* is set to 1. Total count of a's occurrences is obtained from phrases a, *aa*, *ab*, *abc* and *aaa*. Each contributes 1 to

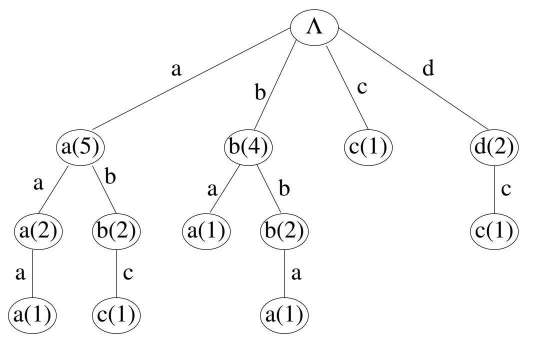

Figure 5.8: Trie built by classical Lempel-Ziv algorithm.

frequency count of a. Therefore, a total frequency count is 5 as shown in the figure. To provide a variation, consider how the frequency count of b is determined. The symbols b appears as a prefix in phrases b, ba, bb and bba, each of which contributes 1 to b's frequency. Thus, b frequency is 4.

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 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 $v_1v_2 \ldots v_n$. 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 can not use Lempel-Ziv compression based model in a straight-forward 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 prefixes of a decoded phrase.
- It remains 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 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.

```
// Decoder at the system side.
// It basically decompresses the string

while (true) {
   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;
}
```

By doing so, it captures the straddling effect.

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

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

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 Figure 5.9.

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

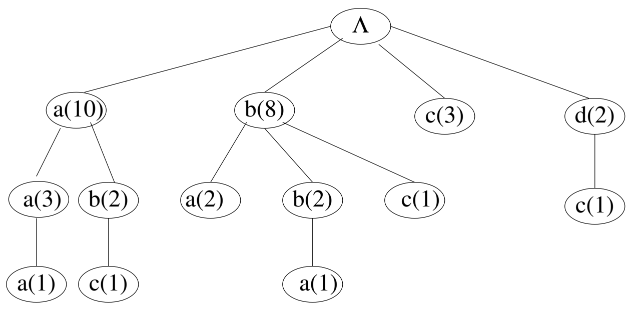

Figure 5.9: Enhanced trie.

scheme, let us compute the entropies for each case. The conditional entropy without considering suffixes (see figure 5.8):

$$\begin{aligned} H(V_1) &= \frac{5}{12}\log\frac{12}{5} + \frac{1}{3}\log 3 + \frac{1}{12}\log 12 + \frac{1}{6}\log 6 \\ &\approx 1.784. \text{ bits, and} \end{aligned}$$

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

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

When all prefixes of all suffuxes are considered, the frequency count will be as shown in trie of Figure 5.9. With enhanced tries, we still have

$$\begin{aligned} H(V_1) &= \frac{10}{23}\log\frac{23}{10} + \frac{8}{23}\log\frac{23}{8} + \frac{3}{23}\log\frac{23}{3} + \frac{2}{23}\log\frac{23}{2} \\ &\approx 1.742. \text{ bits, and} \end{aligned}$$

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

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

5.7.6 Probability assignment

The prediction of a location for mobile terminal is guided by probability estimates of its possible locations. The underlying principle behind probability computation is PPM (prediction by partial matching). Our interest is in estimate of probability of occurrence of next symbol (LAI) on path segment to be reported by next update. The segments are LAI-sequence generated when traversing from the root to the leaves of the sub-tries representing the current context. The estimated conditional probabilities for all LAIs at the current context gives the conditional probability distribution.

Suppose no LeZi type update received after receiving *aaa* and we want to find probability of predicting next symbol as *a*. The contexts that can be used are: *aa* and *a* and Λ. Possible paths that can be predicted with these contexts:

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

Start form the highest order, i.e. context *aa*, the probability *a* is 1/3. Then fall back with probability 2/3 to order 1 (with null prediction), 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 blended probability of 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:

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

The idea is to predict starting with a pre-determined highest order of context, and escape to the lower orders until order 0 is reached.

Therefore, the blended probability of phrase *bba* is

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

Now individual probabilities of symbols are: for *a* $(1/3)\times$ 0.0145 = .0048 (since *a* occurs 1 once in the phrase *bba*), for *b* it is 0.0145$\times$ (2/3) = .0097.

5.8 Distributed Location Management

In the earlier sections of this chapter, we discussed about various location management schemes which essentially distribute the location information across the network. The underlying ideas is that location information will be 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 fact that the distant network messages have to be exchanged to support wide area roaming. In this section, we deal with 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 a centralized HLR and several VLRs. The entire coverage area is divided into several location areas or regions. Each region would consists of a number of mobile switching centers (MSC), and the coverage area of each MSC consists of a number of cells. Each cell is served 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 should always be stored in the LS that corresponds to its group in the region when the user visits a region. The

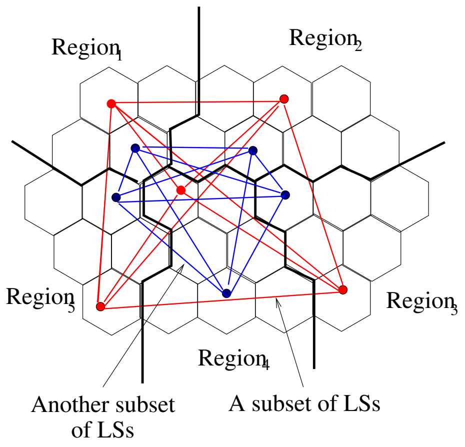

Figure 5.10: The model.

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

The model is depicted in figure 5.10. 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.

5.8.1 The Call Setup Protocol

The MSC initiating a call setup of 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 the call switching between the caller and the callee. 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.

5.8.2 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 for the 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.

Suppose the 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 different 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 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. Hence, the location of the current region and its shape. One of the LSs of the neighbouring region will have an entry, and it will then delete the entry and it would send a delete entry message to the MSC which was serving the MH in the region. Of course, if the registration message from the mobile device may include the id of the previous LS or the region. In this case, the current LS can send unicast message to the previous LS.

5.8.3 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 nieghbouring regions for a particular MSC. If the coverage area of an MSC completely within the interior of a region, this array will be empty. On the other hand, the border MSC (whose coverage area forms a part of the region's border) will have number of regions depending on the shape of the regions.

A simple hash function is used to determine the index of the subset of LSs. Hence, the specific LS in the current region which is 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.

```
get_LSId(MH_id) {
    return MH_id mod numLSperRegion;
}
```

MH sends an update about its location from time to time. The frequency of update depends on the update protocol. When an update is received, 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.

When an MH joins a cell under a new MSC from a cell under a different MSC within the same region a local update is needed. No update is needed for the movements of an MH between cells under the same MSC.

```
locationUpdate(MH_id, MSC) {
    // Executed by the MSC on behalf of a MH

    LS_id = get_LSId(MH_id);
    LS[LS_id].localUpdate(MSC, MH_id);
}
```

The update method is executed by an LS. If LS is informed about the movement of mobile terminal to the current region, then update is made. In case it has moved from a region outside the current region then `remoteDelete` method is invoked to delete entry for MH from LSs in the neighborhood. Only one the LS in the neighborhood can have a entry for MH. If the MH move is reported within the same region but from the coverage area of different MSC then, the movement is also recorded by the LS. However, this update does not require `remoteDelete`.

```
update(MSC, MH_id) {
    // Executed by LS of the region.
    if (exists(location[MH_id])) {
        if (location[MH_id] != MSC) {
            // MH moved within same region but from
            // one MSC to another.

            replaceEntry(MH_id, MSC);
        }
        return;
    }

    // MH was not in region under the LS of the current.
    // region, it must have come from a neighboring region.

    for (i in neighborRegions[MSC]) {
        // remoteDelete delets MH entry from LSs of the
        // neighborhood region. Only one LS have such an entry.
        LSforRegion[i].remoteDelete(MH_id);
    }
    insertEntry(MH_id, MSC);
}
```

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.

```
remoteDelete(region_id,  MH_id)  {
    if (exists(location[MH_id])
        deleteEntry(MH_id) ;
}
```

We have assumed that the methods for inserting, deleting, or replacing an entry have obvious implementations and do not provide details of these specifications.

Local lookup fails in the case the local associated LS does not have an entry for the callee. If local lookup fails 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.

```
lookup(MH_id, MSC) {
   LS_id = get_LSId(MH_id);
   return  LSforRegion[LS_id].localLookup(MSC, MH_id);
}
```

The specification of local look is provided below. It invokes remote lookup if local lookup fails.

```
localLookup(MSC, MH_id) {
      if (exists(location[MH_id])) {
          // local lookup successful
          return location[MH_id];
      }
      else {
          // Perform remote Lookup for MH.
          for each (region i) {
              remoteLoc = LSforRegion[i].remoteLookup(MH_id);
              if (remoteLoc != NULL)
```

```
                return remoteLoc;
        }
        // All remote lookups fail. MH must be
        // in powered off mode.
        return NULL;
    }
}
```

The remote lookup just checks the location entry if one exists, and returns the entry. Otherwise, it returns NULL indicating a failure.

```
remoteLookup(MH_id) {
    if (exists(location[MH_id]))
        return location[MH_id];
    else
        return NULL;
}
```

5.8.4 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 the following simple assumptions.

- 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.

Local Update Cost. Using the above cost model, the cost of a local update can be computed as follows.

1. New MSC does a hashing to find LS subset – H
2. New MSC sends a message to the local LS – t
3. The LS does register lookup for MH entry – RL
4. The LS sends message to old MSC – t
5. The old MSC deletes the MH entry – MD
6. Total: $2t + H + RL + MD$

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

1. New MSC hashes to determine LS subset – H
2. New MSC sends message to local LS – t
3. Local LS determines probable neighbouring regions new MSC – LN
4. Sends delete MSC message to the all the neighbouring regions – Xt
5. Lookup in remote LS for MH entry – RL
6. Remote LS sends message to old MSC – t
7. Old MSC deletes the MH entry – MD
8. Local LS register lookup for creating MH entry – RL
9. Total: $2t + Xt + H + 2RL + MD + LN$

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 $= 2t + Xt + H + 2RL$.

Bibliography

[1] Bhattacharya, A., and Das, S. K. LeZi-Update: An information-theoretic framework for personal mobility tracking in PCS networks. *Wireless Networks 8* (2002), 121–135.

[2] Cover, T. M., and Thomas, J. A. *Elements of information theory.* John Wiley, New York, 1991.

[3] Jannink, J., Lam, D., Shivakumar, N., Widom, J., and Cox, D. C. Efficient and flexible location management techniques for wireless communication systems. In *Mobicom '96* (1996), ACM/IEEE, pp. 38–49.

[4] Pitoura, E., and Samaras, G. Locating objects in mobile computing. *IEEE Transaction on Knowledge and Data Engineering 13*, 4 (2001), 571–592.

[5] Ratnam, K., Matta, I., and Rangarajan, S. Analysis of caching-based location management in personal communication networks. In *Proceedings of the Seventh Annual International Conference on Network Protocols* (Washington, DC, USA, 1999), ICNP '99, IEEE Computer Society, pp. 293–300.

[6] Shannon, C. E. The mathematical theory of communication. *Bell System Technical Journal 27* (1948), 379–423.

[7] Shivakumar, N., Jannink, J., and Widom, J. Per-user profile replication in mobile environments: algorithms, analysis, and simulation results. *Mob. Netw. Appl. 2*, 2 (Oct. 1997), 129–140.

[8] Tribus, M. *Thermostatistics and thermodynamics.* D. van Nostrand Company, Inc., Princeton, NJ, 1961.

[9] Yu, J. I. Overview of EIA/TIA IS-41. In *Third IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '92)* (1992), pp. 220–224.

[10] Ziv, J., and Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Transaction on Information Theory 24*, 5 (1978), 530–536.