

addition to the caching of its local traffic data items, it also caches the data items of the control points in other cells and the roads connections of the whole region covered by the system. The reason for maintaining the traffic conditions of the control points at other cells is to improve the speed in performing the path-finding function (since a client request may involve roads located in other cells). The information servers at different cells are connected by a high-speed wired network through the switch office.

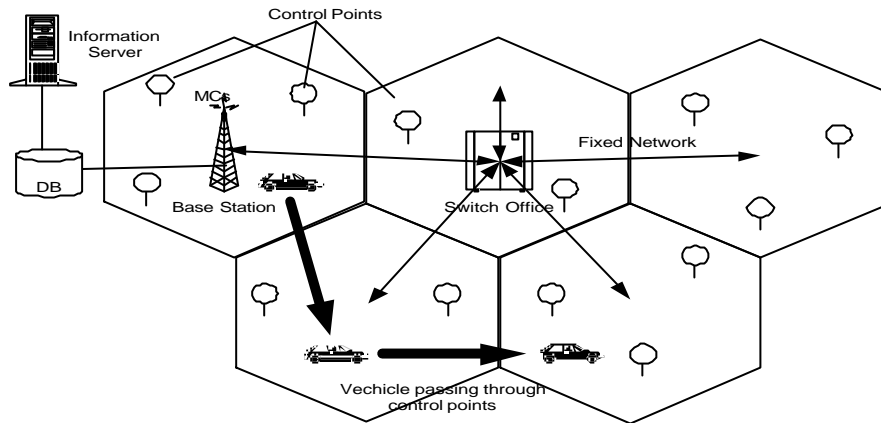


Figure 1: System Model

Mobile clients communicate with the traffic information servers of their current cells through a low-bandwidth mobile network. Each mobile client is preloaded with a set of road maps and is connected to a location detection device, e.g., a global positioning sensing (GPS) system. Depending on its current location, a road map will be loaded and displayed. Each map is labeled with a set of control points whose values are obtained from the information server at its current cell. A mobile client periodically generates a location-dependent query [5], which contains the current location of the client, to its current server. The server responds by sending the traffic data to the client based on the current location of the moving client, i.e., the traffic of the surrounding control points of the client. At the same time, the information server periodically broadcasts the traffic status of some important/critical control points of the whole area, e.g., the main roads in the region or a specific control point whose traffic condition is very worse, to all of the clients. Due to the bandwidth limitation, the number of traffic control points under broadcasting is limited.

If a mobile client is in the navigation mode, it may send a location-dependent continuous query to its traffic information server to invoke the path-finding function by providing its destination. Note that the destination of a mobile client may be at another cell. The path-finding function calculates the best-connected path for going from the current position to the destination based on the connections of the roads as well as the current and future traffic conditions of the roads. While the mobile client is following the defined shortest path (as suggested from the server), the traffic server may re-calculate the path if the current traffic of the roads is “significantly” different from that at the time when the previous calculation is made. The continuous evaluation will stop once the client reaches its destination.

3 Design Issues

In the design of the system, there are four important issues, which must to be addressed to meet the real-time requirements of the system. They are:

- (1) The processing of traffic updates and the management of real-time road traffic data;
- (2) The distribution and the management of cached traffic data;
- (3) The derivation of the shortest path; and
- (4) The dissemination of data and the processing of queries.

3.1 Processing of Traffic Updates and Management of Real-time Road Traffic Data

Since the total update workload at an information server can be very heavy, we design a buffering scheme with data time-stamps to reduce the total update overhead. Each update is associated with a time-stamp when it is generated. The value of the time-stamp is its creation time. Each information server maintains a cache buffer for the real-time data of the road traffic, including the data items for the control points in its cell and other cells. Whenever the server receives an update, it will save the new value into the corresponding entry for the data item in the cache buffer, instead of the database immediately. The purpose of maintaining the cache buffer and delaying the database update is to reduce the update overhead and to reduce the delay in retrieving the “real-time” data since disk data access delay is much longer and highly unpredictable. Whenever the server receives a request from a mobile client, it will get the data from the buffer, instead of retrieving it from the database. Although

the conventional database system may also support with a buffering mechanism, this buffering mechanism will make the system performance unpredictable since it is difficult to control the number of cache hit.

For each data item x in the cache buffer, three copies are maintained: current value (x_c), previous value (x_p) and last database updated value (x_d). Current value, x_c , is the value from the last update. Previous value, x_p , is the value before the current data value. Last database update value, x_d , is the current value of the data item in the database. Whenever a new update arrives, the current value will be copied into the previous value, and the new value from the update will be copied into the current value. Associated with each value is a time-stamp, $ts(x)$. In addition to the three data versions, a prediction function $pf(x)$ is maintained for each data item x . $pf(x)$ consists of two attributes: trend of data item x and avi of x . Trend of a data item x is used to define how the value of x changes, compared to the last value. It will be used to predict the future value of x . avi of x is the absolute validity interval of x which indicates the time duration of validity of the last current value of x , e.g., $ts(x_c) - ts(x_p)$ [8]. The value for the trend of a data item is calculated from the difference between previous value and current value. Since the two successive values only represent the change of values between two successive updates (an update period), it may not be a very good indicator to describe the actual trend of the data values. Therefore, the other previous values are considered, such as that using an exponential weighed mean average approach: $f_{x,n} = (f_{x,n} + \alpha f_{x,n-1} + \alpha^2 f_{x,n-2} + \dots + \alpha^{n-1} f_{x,1}) / S$

where S is the sum of the weights $(1 + \alpha + \alpha^2 + \dots + \alpha^{n-1})$

$f_{x,n} = (x_{p,n} - x_{p,n-1}) / (ts(x_{p,n}) - ts(x_{p,n-1}))$, where α is a tuning parameter with a value smaller than 1.

A linear exploitation is assumed and the future predicted value for data item x at time t is then calculated as:

$$x_c(t) = f_{x,n} \times t + x_c$$

Periodically, an update transaction is executed to save the current values of the data items, x_c , at the buffer (into the database). The update period is dynamically defined and is based on the difference of the data values. For example, in every fixed period, a process will go to examine the cache data, and the current value is compared with the last database updated value. If the difference is very small, e.g., smaller than a pre-defined threshold value, it will assumed to be the same as the previous value. Therefore, no update will be performed on the database, and the current value at the database will be considered being similar to the existing status of the corresponding control point.

3.2 Distribution of Traffic Data and Cached Data Management

Caching the same data item at several servers can facilitate the dissemination of traffic data to mobile clients and can also greatly improve the efficiency in calculating the best path in the path-finding function. It is especially important for the case when the path crosses more than one cell. Note that in the searching of the best path, it may require many backtrackings. If the graph of the connection of the roads and traffic conditions is partitioned into sets at several servers, heavy communication delays will be introduced to the searching algorithm. On the other hand, a replicated graph approach, where the same graph is replicated at different servers, will create the coherency problem of the cached traffic data items amongst graphs at different servers. The overhead will be very heavy if a tight coherency is required. That is, after the completion of each update, a number of updates needed to be created to update the versions at all other servers. If a very loose coherency scheme is adopted, the correctness of the data cannot be maintained, and the effectiveness of the path-finding function will be affected. Of course, a centralized approach may be used to resolve the heavy update problem. A central database server contains all the traffic data items for searching the paths across different cells. However, the problem of the centralized approach is that the central server will become the bottleneck resource, and the whole system performance will be highly affected by its performance. Consequently, the scalability of the system will be seriously affected.

In this prototype, we have designed a *similarity-based data forwarding (SBDF) scheme* to minimize the cache data management overhead and, at the same time, to maintain the coherency of the cached data items at different servers within some pre-defined values. The SBDF scheme is a controlled data replication method in which each server maintains a graph which contains the traffic information and road connections of its cell and other cells. Once a server has realized a traffic update into its buffer, it may generate an update and send the update to another server to refresh the corresponding version at that server if the value difference between the versions at the two servers is greater than the similarity distance, defined for them. By adjusting the similarity bounds for the data items at different servers, we can control the total update processing workload in the system and, at the same time, maintain a balance between the value accuracy of the data items at different servers and the total update workload. In the SBDF scheme, we define a larger similarity distance for the servers with a greater physical distance apart. This scheme fits perfectly into the requirement and characteristics of RETINA. The importance and accuracy of the traffic conditions of the roads to a driver (mobile client) depends on the current location of the driver and the physical distance between the driver and the roads. The driver would like to have a high accuracy for the traffic condition of the roads that are close to him. For the roads, which are further away from his current position, he may only need a rough idea about the road conditions. Even if a very accurate status is reported for a road that is far away, the actual status of the road may change to another status at the time when he reaches the road.

3.3 Derivation of the Shortest Path

A graph of connections among the roads and traffic conditions is built using the information maintained in the cache buffer of each information server. The nodes of the graph are the start points and end points of the control points. The “length” of each edge is derived from the traffic status of the control point and the length of the control point. Although there are many efficient shortest-path searching algorithms, most of them are for static information. The technical problem here is that the “length” of each edge is dynamic, due to dynamic road traffic conditions. Therefore, a re-calculation has to be performed once the value has changed significantly, e.g., greater than a pre-defined threshold. Also, for the determination of the length for an edge, instead of using the current value, the predicted value in the future will be used. For each edge, we calculate the predicted future traffic value as a function of its $pf(x)$ and the accumulated time from the starting node to the node. In addition to the reporting of the shortest path, other paths and their difference in values are also reported.

3.4 Dissemination of Data and Processing of Queries

In RETINA, we use both the data broadcast approach [7] and the on-demand approach [10] for disseminating the traffic information to the mobile clients. We use data broadcasting to disseminate the traffic information of the control points that are interested to all the mobile clients (e.g., the control points for the main roads or the points where the traffic conditions are very bad). The on-demand approach is used to provide the road traffic information to a mobile client that sends requests to its current server to ask for the traffic information of its surrounding control points and the best path to its destination. The technical question for the on-demand requests is what should be the period for generating the requests. If the period is too large, the coherence between the data values at the client cache and the information server can be very weak. On the other hand, if the period is too short, the information server and the mobile network may be overloaded. Since the design of RETINA aims at minimizing the bandwidth consumption of traffic data between the mobile clients and the information server, we again adopt the concept of *data similarity* to define the periods of request generation [4]. We define a *similarity period* function which dynamically calculates the new generation time for the next request based on the change rate of the values of the set of traffic data. If the traffic data do not change very quickly, we use a longer period, and vice versa. The assumption is that if the rate is lower, the validity duration of the traffic data at the client cache will be longer.

4 Implementation

Figure 2 depicts the main components of RETINA. The server executes on the Windows NT environment. Although many special-purpose real-time operating systems, e.g., Lynx OS and QNX, can provide better real-time supports, they are often not used in most government agencies and the private sector. On the other hand, the RETINA is merely a soft real-time system and does not need to operate in a perfectly predictable operating environment. The database for RETINA can be some common simple database systems, such as Sybase. In the implementation, we choose Sybase because of its availability and simplicity. The server and client programs are implemented in the C++ programming language. The mobile clients are thin clients, i.e., handheld PC running the Win CE.

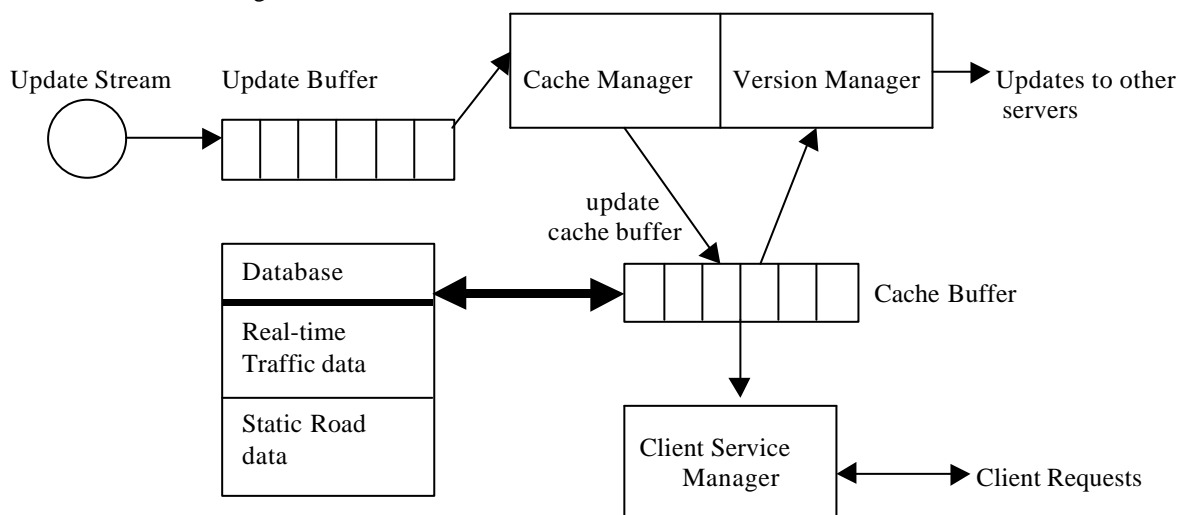


Figure 2: The Information Server

The information server consists of six main components: the update stream, the update buffer, the database, the data manager, the cache buffer, and the client service manager, as shown in Figure 2. The database consists of two types of data items: real-time and static data items. The real-time data items are state data and are used to record the traffic conditions of the roads. In order to reduce the update overheads, RETINA uses a single update stream for all the updates, instead of one

transaction per update [6]. The update stream puts the new updates into the update buffer. The data manager consists of two components: cache manager and version manager. The cache manager saves the new values from the update buffer into the cache buffer. The version manager is responsible to maintaining the coherency amongst the versions in different servers based on the SBDF scheme. The client service manager services the traffic requests and navigation requests from mobile clients.

When the client service manager accepts a client request that is basically a coordinate for the location of the client, it retrieves the traffic data of all the control points in the map where the mobile client is. Then, a new period of transaction request will be calculated and sent with the traffic data to the mobile client. In the calculation of transaction period, the minimum and maximum period bounds are defined. The new period is calculated as the mean value of the minimum bound and maximum bound, and then multiplied by the mean traffic values of all the control points in the map. In the implementation of the path finding function, a shortest path algorithm with the traffic information is used. In the derivation of the shortest path, the "real-time" traffic will also be considered such that the "distance" between two points is a function of the geographical distance and the road traffic conditions.

5 Demonstration

A simulation testbed is developed to demonstrate the functionality of RETINA and the capability of the designed methods for managing the real-time traffic data. A database containing the road connections is defined in the database. (It is defined based on the real road connections in Hong Kong.) Two servers are setup, and each server is assumed to be responsible for a region in the system. Traffic workload is generated by a process to simulate the new traffic updates from the control points. The location of a mobile client is generated by another process at the mobile machine which is a handheld PC. The process generates continuous requests for navigation and requests for asking the real-time traffic of the surrounding area of its current location. The process also simulates how it moves while it is following the path suggested by the system.

6 Conclusions

An important issue in many mobile computing systems is how to disseminate a large amount of information, which is often real-time and highly dynamic, to a large number of mobile clients. The objective of RETINA is to develop a real-time traffic navigation and information system prototype to show how real-time techniques can help in a practical mobile application. Since RETINA is not a hard real-time system, it does not need to be highly predictable. Instead, we have to explore the balance between the "real-time" properties and the cost/overheads in the implementation of the system. In the design and development of RETINA, we found the usefulness and effectiveness of soft real-time techniques in the implementation of a mobile computing system with soft real-time requirements. An important future work of the RETINA is to provide the real-time traffic information and path finding function to mobile phone users through the wireless application protocol (WAP). Another on-going work is to support location-dependent real-time video playback while a client moving.

References

- [1] R. Srinivasan, C. Liang and K. Ramamritham, "Maintaining Temporal Coherency of Virtual Data Warehouses", in *Proceedings of Real-time Systems Symposium*, Spain, 1998.
- [2] Alpine Car Navigation System, <http://www.carnav.com/>.
- [3] S.J. Ho, T.W. Kuo, A.K. Mok, "Similarity-Based Load Adjustment for Real-Time Data-Intensive Applications," in *Proceedings of IEEE 18th Real-Time Systems Symposium*, 1997.
- [4] O. Ulusoy, "Real-Time Data Management for Mobile Computing", *Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98)*, Berlin, Germany, 1998.
- [5] M.H. Dunham and V. Kumar, "Location dependent data and its management in mobile databases," in *Proceedings of International Workshop of Database and Expert Systems Applications*, pp. 414-419, 1998.
- [6] Hüseyin Gökmen Gök and Özgür Ulusoy, "Transmission of Continuous Query Results in Mobile Computing Systems", *Information Sciences*, vol. 125, no. 1-4, pages 37-63, 2000.
- [7] S. Acharya, M. Franklin, S. Zdonik, "Balancing Push and Pull for Data Broadcast", in *Proceedings of ACM SIGMOD*, Tucson, 1997.
- [8] Ramamritham, K., "Real-time Databases", *International Journal of Distributed and Parallel Databases*, vol. 1, no. 2, 1993.
- [9] Kam-yiu Lam, Tei-Wei Kuo, Wai-Hung Tsang and Gary C.K Law, "Concurrency Control in Mobile Distributed Real-time Database Systems", *Information Systems*, vol. 25, no. 4, June 2000, pp. 261-322.
- [10] P. Xuan, O. Gonzalez, J. Fernandez & K. Ramamritham, "Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments", *Proceedings of 3rd IEEE Real-Time Technology Application Symposium*, 1997.
- [11] A.P.Sistla, O.Wolfson, S.Chamberlain, S. Dao, "Modeling and querying moving objects", *Proceedings of the 13th International Conference on Data Engineering*, pages 422-432, Birmingham, UK, April 1997.