# A Scalable Technique for Large Scale, Real-Time Range Monitoring of Heterogeneous Clients

Erin J. Hastings, Jaruwan Mesit, Ratan K. Guha
School of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL 32816
(hastings, jmesit, guha)@cs.ucf.edu

*Abstract* - **Range monitoring is the continuous query on location data of mobile, real-world objects in real-time. Such real world objects are typically wireless, low capability clients. Therefore, tracking techniques must limit client computation and memory overhead, allow for client/server heterogeneity, and most importantly, minimize wireless transmissions. This paper presents a technique for range monitoring based on multi-level spatial hashing. The technique addresses: (1) real-time queries on mobile object locations, (2) real-time query on the proximity of mobile objects in relation to each other, (3) user defined special query areas, and (4) allows for variable levels of mobile client capability (heterogeneity). The spatial hashing-based method presented here provides a level of scalability similar to the best existing methods for client processing requirements, transmission size, and transmission frequency. Additionally, it provides the flexibility of multiple tracking modes, proximity queries, and support for multiple server base stations which other methods may not. The results of a simulation that computes total transmission overhead and data server requirements based on mobile object characteristics are presented.**

***Keywords-range monitoring; multi-level spatial hashing; mobile object database; continuous query***

## I. Introduction

*Range monitoring* is the continuous query on real-world mobile object position data. Typically, mobile objects will report their locations via wireless medium to a central server, which can be queried as desired by users interested in mobile object location data. Examples of some settings where range-monitoring is used include: fleet vehicle tracking, military battlefield personnel or vehicle tracking, automated tour guides and vehicles at theme parks, wild animal location tracking, and others. When designing an efficient range monitoring scheme, three major issues must be considered: (1) limited mobile device capability, (2) minimization of transmissions, (3) heterogeneous client capability [1] [2].

First there is the problem of limited mobile device resources. Since they are necessarily small and have a limited battery life, constraints on mobile devices usually include limited processing power, limited persistent storage, and limited memory [3] [4] [5]. Therefore storage requirements and processing complexity for these limited clients must be minimized. Second, due to the inherent nature of a wireless environment connection issues must be considered such as drain on battery power by transmission, and bandwidth constraints [6] [2]. Since transmission is expensive with regard to battery power it must be done as infrequently as possible and the transferred data must be small in size. Limiting transmission frequency/size also reduces collisions between client messages, which usually results in re-transmissions. Finally, heterogeneity among clients is possible, where not all clients have the same hardware or capabilities. Since not all clients are necessarily minimal devices, a system should allow more capable, or smart clients [6] [2].

Fast, real-time tracking of mobile objects happens to be a common requirement for simulations and games as well. Thus, the basis the range-monitoring system presented here is *T-Collide* [7], a technique for optimizing collision detection between mobile objects in graphics, simulations, and games. The T-Collide algorithm is based mainly on *spatial hashing* where the world is divided into evenly spaced grid cells (*uniform spatial subdivision*). Based upon their positions in the world, objects are then hashed to grid cells using a hash function. A modified version of this hashing-based method can be applied to range-monitoring in order to: (1) reduce mobile transmissions, (2) quickly determine which mobile objects are in special query areas, and (3) allow fast detection of object proximity in relation to each other. The technique presented here has similar or better scalability, transmission overhead, and memory requirements compared to existing range-monitoring methods. In addition, this hashing-based method provides the flexibility of different tracking modes, proximity queries, and multiple base station support which other published methods may not. The remainder of the paper will first discuss the details of spatial hashing, then outline the range monitoring technique, and finally memory and transmission requirements will be examined in detail.

## II. Related Work

The range-monitoring technique presented is a conglomeration of methods from several areas of computing including: databases (queries, hashing, indexing), wireless networks (range-monitoring), and graphics, simulation, and visualization (collision detection and spatial subdivision). Excellent sources on general mobile computing databases and transaction processing can be found in [3] [4] [5] and [8]. A system designed specifically to conserve power in mobile devices is outlined in [6]. The real-time collision system (*T-Collide*) upon which this range-monitoring technique is based, is outlined in [7]. T-Collide is, in turn, based heavily upon another collision system utilizing spatial hashing presented in [9].

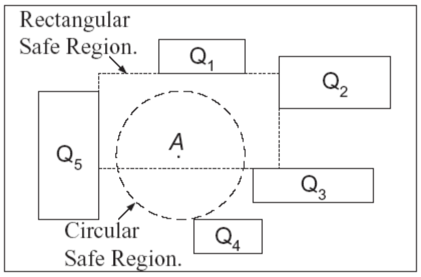


Figure 1 – Query areas and Safe Regions in the Q-Index method.

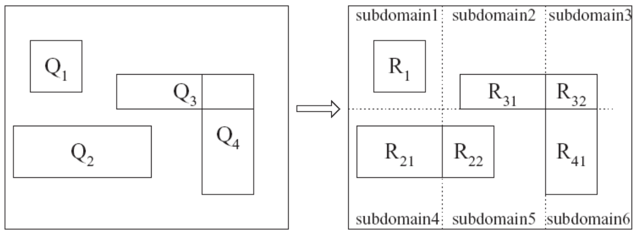


Figure 2 – Domain and query decomposition in the MQM method.

*Q-index* is a range monitoring system presented in [11] and [12]. In Q-index, queries are defined as explicit rectangular or circular regions of space. Every mobile client must store query regions on board. *Safe regions* are defined as areas that do not overlap any existing query area (see Figure 1). Reduction of transmission occurs where objects only transmit their position if they leave their current safe region. The main drawback of this system is – whenever a new query is added, safe regions must be recomputed by the server and subsequently transmitted to clients. This becomes a bottleneck, likely preventing scalability to a system of many thousands of mobile objects [2].

*Monitoring Query Management* (MQM) [2] addresses some of the deficiencies of Q-Index. Again queries are defined as rectangular areas of space. The domain space is separated into *sub-domains* (see figure 2). A *monitoring region* (designated $R_x$ in the figure) is where a query overlaps a sub-domain. Every mobile object is associated with a *resident domain* – a rectangular space surrounding it. A resident domain can be one or more sub-domains, along with related queries. The number of sub-domains within an object's resident domain is dictated by the object's capability (i.e. memory, cpu, bandwidth, etc…). Under MQM, each mobile object is aware of the monitoring regions (queries) within its resident domain (they are transmitted by the server when the object is assigned a resident domain). As in Q-Index a mobile object only transmits location data when it enters or leaves a query area (reduction of transmissions). When an object leaves its resident domain, it informs the server, and then the server computes a new resident domain along with accompanying query areas and transmits the data to the mobile object. MQM's superior scalability is a result of this mechanic – since resident domain size (along with associated monitoring regions) can be scaled to mobile object capability.

The hash-based technique presented in this paper provides the scalability of MQM with similar memory and processing requirements and similar transmission size, but with the addiction of proximity queries and an optional tracking modes which neither Q-index nor MQM provide. The improvements come with the trade off of possible reduction of query area precision. As with MQM, motion estimation or dead reckoning techniques are avoided, since they inhibit scalability. Additionally, in many situations accurate client motion estimation is impossible [2].

## III. Spatial Hashing

*Spatial hashing* is a process by which a 3D or 2D domain space is projected into a 1D hash table [7]. To implement spatial hashing at least three things are required, (1) a 2D or 3D grid, (2) a hash function, and (3) a hash table.

First, the entire domain space is subdivided by a grid (uniform spatial subdivision) which in our case is 2D. This naturally lends itself to a system where real-world objects are tracked by GPS latitude and longitude data. The grid can be defined by three variables. *Cell size*, that defines the size of each cell, and *min* and *max*, two points that anchor the grid in domain space. The hash function takes any given 2D or 3D positional data and returns a unique grid cell that corresponds to a 1D bucket in the hash table. Objects are hashed periodically and their locations can then be quickly queried in the hash table.

## IV. Range Monitoring Process Overview

Briefly, range monitoring with this hashing-based method proceeds in the following manner. Mobile clients wander the world, going about their designated routines. Periodically the processors on board the mobile clients will hash their current position in the grid based on GPS location data (or some similar method). The hash function maybe loaded onto clients before deployment or at anytime updated by a message from the central server. Only when a client changes grid cell does it transmit information to the server. Servers listen for location data transmissions from the clients and users may interactively query the servers. Some example queries may include: "Who is in grid cell (x,y)?", "Which grid cell is object A in?", "Which objects are within X distance of object A?", "What objects are in user-defined query area J?", or "What course has object B followed over the past X hours?".

The detailed aspects of the range-monitoring process will be presented in this order: (1) *clients*, or the mobile objects being tracked, (2) *servers*, or the server(s) collecting location data, (3) *hash functions*, simple functions periodically calculated by all clients that determines what unique grid cell the client occupies, (4) *queries*, which are performed by users on the server location data, (5) *modes*, or different modes of tracking which provide increased detail at the expense of greater transmission bandwidth or client memory requirement, and finally (6) s*mart clients*, or extra capable clients that are location aware of all other clients.

## V. Mobile Clients

Each mobile client has a unique CLIENT_ID. While roaming, clients will periodically hash their location using the hash function. Minimally the hash function is based on GPS location and CELL_SIZE. Conceivably, clients could transmit their hash data every time it is computed. Clearly this would result in: (1) a non-scalable system due to transmission collisions, and (2) clients quickly draining battery power. Therefore, clients will only transmit location data when it changes. For example when the previous hash resulted in cell 3 and the new hash results in cell 4, the client informs the server that it has moved to a new cell. In this way transmissions are greatly reduced while still retaining a good estimate of where the client is.

If we are interested in constantly tracking all clients the above method is sufficient. However, suppose we are only interested in tracking clients in key locations or query areas. In this case, the server transmits local query cells to the client. Clients then only transmit location data to the server if they enter or leave a query area. This mechanic results in a much more scalable system due to location awareness as described by MQM above. In this hash-based method, however, management of queries and distribution to clients differs considerably. Clients may run in two "tracking modes". In Constant-Tracking mode, clients broadcast their position on every cell change. In query-tracking mode, clients only transmit their location if they enter or leave a query area.

A summary of the basic client routine is as follows. On startup, (1) broadcast new client message to server, (2) receive hash data from server, (3) hash initial position and send to serve. Then, loop continuously, (1) hash position, (2) if hashed cell has changed, send new cell data to server.

## VI. Servers

A server constantly listens for incoming client messages, and maintains a client list and a hash table with location data for all clients that may be queried by users. When the server comes online, it must broadcast the relevant hash data to all clients. The details of hashing in a range-monitoring setting are covered in the next section. A brief summary of the basic server routine is as follows. On startup, (1) broadcast hash data to all clients, then (2) continuously listen for incoming messages. On receipt of new client message, (1) add new client to the client list. On receipt of location data, (1) find client's old data in client and remove it from the hash table, (2) add the new data to the hash table.

## VII. Hash Functions

Given a mobile object's location data, the hash function returns the unique grid cell that the object occupies. The grid and hash function are the key to scalability of the range-monitoring system. Mobile objects only transmit their location data when changing grid cells. In this manner transmissions are greatly reduced, and the server is always aware of the approximate location of the object.

The hash function translates the object location into a single integer representing a grid cell. An object periodically hashes its position and only sends this information to the server if it has changed grid cells. If grid cell size is fixed, the hash function may be loaded onto the mobile objects. If the cell size is not fixed, the mobile objects may have to obtain cell size, or other hash function data, from the server.

## VIII. Queries

Queries are performed on the server. The following types of queries are supported: *Cell Query*, or "which objects are in cell X", *Object Query*, or "in which cell is mobile object A located", *Proximity Query*, or "which objects are near object A", *Special Query*, or "which objects are in a range of cells selected by the user", and finally a *History Query*, or "what cells has object A traversed over the past N units of time?".

For a Cell Query, simply return the contents of hash bucket X. An Object Query returns the object's cell reference in the object index. A Special Query is a range of cells selected by a user. A Cell Query on each of the selected cells is performed. For Proximity Queries, an Object Query is first performed on object A – returning cell X. Then a range of cells surrounding object A is selected. All objects within cell X and the surrounding cells are safely assumed to be "near" object A. Figure 3 shows an example Proximity Query. Suppose the CELL_SIZE is 10, object A hashes to cell (x,y) in the grid, and all objects within approximately 20 units of object A are desired. All objects in the range of cells (x-2,y-2) to (x+2,y+2) are within approximately 20 units of cell (x,y) since the CELL_SIZE is 20. Therefore, all objects within the shaded area are returned by the query. A History Query can only be made when in a certain range-monitoring mode (modes are covered in the next section). A History Query returns the cells an object has traversed over a certain amount of time. To implement such queries, a finite list of the last X cells traversed by each object (along with a time stamp of each cell transition) is kept on the server.

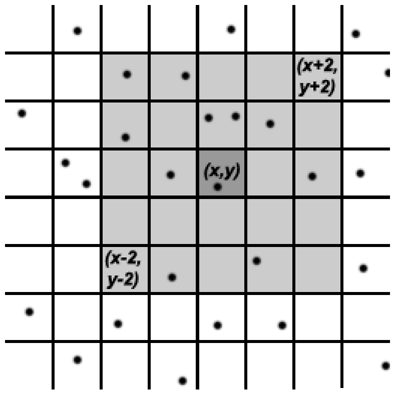


Figure 3 – A Proximity Query for all objects within 2 grid cells of a client.

## IX. Modes

Different modes provide varying levels of tracking detail at the expense of transmission bandwidth, processing, or memory requirements. The proposed modes are: (1) *Single-grid Constant-Tracking* mode, (2) *Single-grid Query-Area* mode, (3) *Multi-grid Constant-Tracking* mode, (4) *Multi-grid Query-Area* mode.

*Single-grid* denotes that the entire domain space is covered by a single grid, whereas *multi-grid* denotes that more than one grid overlays the domain space. Single-Grid mode assumes a single wireless base station server supporting all mobile clients, whereas, multi-grid mode assumes multiple wireless base station servers for different sections of the grid. *Constant-Tracking* means that mobile objects report every cell change to the server; whereas *Query-Area* means that objects only transmit location data on data on entering or leaving a query area (mobile clients must be aware of query areas). Note that Constant-Tracking is required for proximity queries. In general, the following apply. Single-Grid is less scalable, and single server. Multi-grid is more scalable, and supports multiple servers or base stations. Constant-Tracking is required for proximity queries, but results in more location transmissions and is therefore less scalable. Query-Area results in fewer location transmissions, is more scalable, and supports location aware clients.

In Single-Grid, Constant-Tracking mode the entire domain space is covered by a single grid and all clients are served by a single wireless base station. Clients hash their position at some specified interval. Whenever a cell change is detected, the change is transmitted to the server. This mode has a high number of transmissions but the lowest client-side memory requirements (the client need only know the hash function).

Single-Grid, Query-Area mode proceeds in the following manner. Again a single grid divides the entire domain-space; however clients are made “location aware” of grid query areas. When clients come on line, they inform the server they are active. The server then transmits a list of specific grid cells that are query areas (essentially a list of integers). Clients then periodically hash their locations. Whenever a mobile client enters or leaves a query cell, it transmits location data to the server. In this manner transmissions are greatly minimized. This mode has few transmissions but higher client-side memory requirements – each client must be aware of all query cells in the domain space.

Multi-grid, Constant-Tracking mode is for applications that require proximity detection and more than one server or wireless base station. The domain space may be divided by multiple grids which overlay each other (same MAX and MIN but different CELL_SIZE) or lay side by side (same CELL_SIZE but different min and max). Generally each cell of the largest grid will be served by a different wireless server. Mobile clients will do a separate hash for each grid (since CELL_SIZE, MIN, and MAX differ). Clients will transmit location data for any grid cell change. When clients change cell in the for the larger grid they are handed off between wireless base stations.

Multi-Grid, Query-Area mode is the most scalable mode, requires the fewest transmissions, and supports the largest number clients. First, the domain space is divided by at least 2 overlaying grids. One grid will divide the domain space at a coarse-level and another at a fine-level (see figure 4). As usual, mobile objects will have a different hash function for each grid and periodically hash their locations. Since this is “Query-Area” mode objects are made aware of query cells, but only those in the local area. Thus when a client comes online, the local server informs it of only those queries within the coarse-level grid cell it occupies. Queries are composed of fine level grid cells (a list of integers). When a mobile enters or exits a query area it transmits location data to the server. When a mobile object enters a new coarse-level grid cell, it gets a new list of queries from the server – those queries with the client’s local coarse-level cell. The scalability of this mode lies in the fact that course level cells can be scaled to insure clients only see a certain number of queries and need not be aware of every query within the entire domain space. Thus transmissions are greatly reduced.

As an example of multi-grid hashing consider Figure 6. The inner grid (gray) finely divides the domain space into 16 cells. The outer grid (black) coarsely divides the domain space into 4 cells. The mobile object in the figure lies in cell 10 of the inner grid, and cell 3 of the outer grid. There is no significant client-side overhead associated with multiple grids since the client simply hashes twice using a different CELL_SIZE. Each cell of the coarse grid may be served by a dedicated wireless base station.

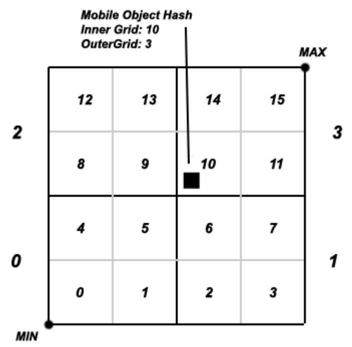


Figure 4 – Multi-grid hashing provides maximum scalability. Each area of the coarse grid (outer numbers) is served by a different wireless base station and special queries are defined by areas of the fine grid (inner numbers).

## X. Smart Clients

Some applications may require smart clients, or those clients that are aware of their surroundings. In this case, every client will listen as a server to broadcast transmissions from other clients. Essentially smart clients will function almost as servers, storing data about other clients and query areas with the same data structures as a server. For applications that require it, smart clients could do proximity queries on their data to very quickly determine queries such as: “who is near me?” or “who is in the nearby critical monitoring areas?”

## XI. Client Path History

History-mode may be applied to any Constant-Tracking mode. A list of previous cells for each object is maintained on the server along with a time stamp. Thus the path over time for any object maybe queried as outlined previously in the query section.

## XII. Performance Analysis

First, server-side query performance is analyzed. Cell Queries are O(1) by direct access of hash bucket X. Object Queries are O(1) by direct access of the object-index. Proximity_Queries have the range of buckets is computed, then an O(1) cell query performed on each bucket. Special Queries are an O(1) cell query on each selected cell. History Queries, if stored as an index of lists, are O(N) where the correct list from N objects lists is found, then searched back to time T. Next server memory requirements are considered. The proposed method requires at a minimum requires (1) a hash table with an integer per mobile object, and (2) an object index with one integer object-ID, and one integer to index the object’s current cell.

Mobile clients need only perform integer and floating point multiplication, division, addition, and subtraction. Client-side memory requirements depend upon the range-monitoring mode but overall are quite low. In Single-Grid Constant-Tracking mode_clients only store the hash function. In Single-Grid Query-Area mode, clients store the hash function and 1 integer for every query cell in the domain space. In Multi-grid Constant-Tracking mode, clients the hash function and different values of MIN, MAX, and CELL_SIZE. And in Multi-grid Query-Area mode, clients store the hash function, different values of MIN, MAX, and cell size for each grid, and 1 integer for every query cell in the local area (coarse grid cell).

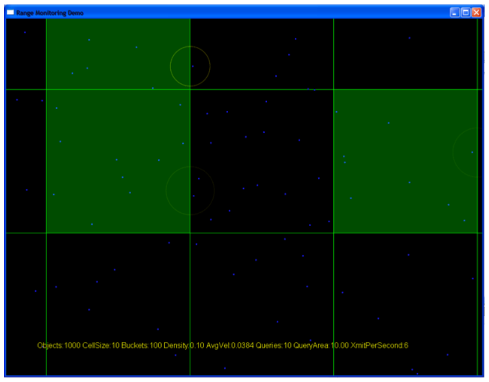
Most importantly, with regard to number of transmissions Single-Grid, Query-Area mode performs exactly as MQM since each object only transmits when it enters or leaves a grid cell. Multi-Grid ,Query-Area mode performs exactly as MQM as well, but in separate cells of the coarse grid. Constant-Tracking modes however will result in significantly more transmissions since objects transmit every cell change.

## XIII. Simulation and Results

A simulation/visualization for the presented technique was implemented using C++ and OpenGL, which is shown in Figure 5. Evaluation of a range monitoring technique boils down to one main question: “how many transmissions per second can we expect given X number of clients in a given space?” The number of transmissions per second in a range monitoring environment is a function of several variables: (1) *number of clients* – more clients result in more transmissions, (2) s*ize of the domain space* – client domain space will affect the size of grid cells and how quickly clients change cells, (3) *client average velocity* – fast clients will change grid cells more often, resulting in more frequent transmissions, (4) *grid and cell size* – smaller cells mean more frequent cell changes, (5) *percentage of the domain space queried* – as the number of queries rises the number of transmissions will increase.

The above attributes will vary widely between systems, since there is no standard range monitoring environment. Considering these issues, for experimental analysis a theoretical worst case scenario is tested where: (1) a large number of mobile clients, (2) the clients constantly move at high velocity, and (3) the domain space is a fairly contained area. Objects in the simulation have a random distribution over the domain space and all clients have a constant velocity which traverses any given cell in approximately 10 seconds. This worst case scenario is simply designed to test the limits of the algorithm.

Figure 6 shows the results in average number of location data transmissions per second for a number of mobile clients and a given query area. As shown in the data, number of transmissions for a substantial number of clients is scaled down to almost trivial levels in many cases. For example, suppose a single server handles 1000 clients and on average 75% of the domain space is under query. In that case we can expect only around 108 location data transmissions per second on average.



Objects:1000 CellSize:10 Buckets:100 Density:0.10 AvgVel:0.0384 Queries:10 QueryArea:10.00 XmitPerSecond:6

Figure 5 – Range Monitoring simulation used to estimate bandwidth requirements based on number of objects, domain size, and average object velocity. The shaded cells represent query areas and the circles represent client transmissions.

| | | Number of Mobile Clients | | | | | |
|---|---|---|---|---|---|---|---|
| Query Area | | 100 | 1000 | 2000 | 4000 | 8000 | 16000 |
| | 10% | 2 | 23 | 45 | 88 | 164 | 388 |
| | 25% | 6 | 37 | 81 | 180 | 365 | 742 |
| | 50% | 7 | 84 | 158 | 272 | 590 | 1268 |
| | 75% | 11 | 108 | 218 | 447 | 893 | 1750 |
| | 100% | 12 | 125 | 243 | 465 | 958 | 1873 |
| | | | | | | | |

Figure 6 – Results of experimentation in average number of transmissions of location data per second. The number of clients denotes the number of mobile clients in the domain space. Query area denotes the percentage of the domain space covered by queries. 100% query area denotes constant tracking where clients transmit on every cell change.

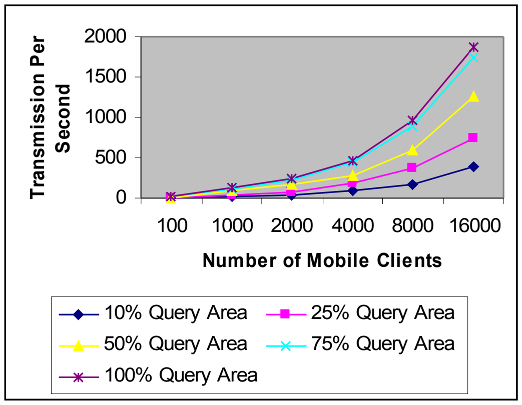


Figure 7 – A graphical trend of the data presented in figure 8. As number of clients and query area grows, the number of transmissions increases. Mobile clients and transmissions per second are for the entire system.

## XIV. Conclusions and Future Work

As with MQM, clients under this hash-based scheme only transmit when entering or leaving a query cell. Essentially, there cannot be fewer transmissions while range monitoring in real time. The area where improvement may be made is that of scalability. This hash-based scheme offers the performance of MQM and the support for a heterogeneous client base, but in addition adds: (1) option of constant tracking modes, proximity queries, and history mode, (2) capability to tune network performance on the fly by adjusting grid sizes, (3) the grid-based scheme naturally lends itself to a multiple base station setup.

Since number of transmissions cannot be reduced further the only true obstacle to scalability in range monitoring schemes is the limits on server hardware and the network protocols themselves. Specifically, as the number of transmissions increases, there comes a point where the medium is flooded and there is constant collision. As an example, again refer figure 8. Suppose a range monitoring system is expected to serve about 2000 clients per wireless base station with around 75% or the domain space queried. In that case we must select hardware and protocol that will gracefully handle around 218 transmissions per second on average and the slight additional overhead from bookkeeping transmissions and the protocol itself.

There are at least two promising possibilities for future work. The first is an algorithm for automated monitoring and adjustment of the grids for optimal network performance. Some work has been done on dynamically adjusting grids in other areas of computing that could certainly be incorporated into range monitoring. The second is the exploration of multi-channel schemes and multiplexing.

## References

[1] Y. Cai and K. Hua. “Processing Range-Monitoring Queries on Heterogeneous Mobile Objects”. 2002.

[2] Y. Cai and K. Hua. “An Adaptive Management Technique for Real-Time Monitoring of Spatial Regions in Mobile Database Systems”. IEEE Performance, Computing, and Communications Conference 2002.

[3] S. Avancha, F. Perich, A. Joshi, Y. Yesha Y. and K. Joshi. “Query Routing and Processing in Mobile Ad-Hoc Environments”. Tech Report, University of Maryland. 2001.

[4] S. Balakrishnan, M. Dunham, and A. Helal. “A Mobile Transaction Model that Captures Both Movement and Behavior”. Mobile Networks and Applications. Volume 2. Number 2. 1997.

[5] B. Bhargava and S. Madria. “A Transaction Model for Mobile Computing”. International Database Engineering and Applications Symposium. 1998.

[6] S. Banik and L. Gruenwald. “A Power Aware Technique to Manage Real-Time Database Transactions in Mobile Ad-Hoc Networks”. 12th International Workshop on Database and Expert System Applications. 2001.

[7] J. Mesit, E. Hastings, and R. Guha. “Optimized Collision Detection for Flexible Objects in a Large Environment”. Computer Games: Artificial Intelligence, Design, and Education. 2004.

[8] D. Chakraborty, O. Ratsimor, S. Tolia, D. Khushraj, A. Kunjithapatham, A. Joshi, T. Finin, and Y. Yesha. “Allia: Alliance Based Service Discovery for Ad-Hoc Environments”. ACM Mobile Commerce Workshop. 2002.

[9] M. Gross, M. Teschner, B. Heidelberger, M. Mueller, and D. Pomeranets. “Optimized Spatial Hashing for Collision Detection of Deformable Models”. Vision, Modeling, and Visualization. 2003.

[10] E. Hastings, J. Mesit, and R. Guha. “T-Collide: Temporal, Real-Time Collision Detection for Mobile Objects”. Computer Games:Artificial Intelligence, Design, and Education. 2004.

[11] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. “Queries as Data and Expanding Indexes: Techniques for Continuous Queries on Moving Objects”. Purdue University. 2000.

[12] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. “Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects”. IEEE Transactions on Computers. 2002.

[13] K. Lam, L. Ulusoy, T. Lee, E. Chan, and G. Li. “An Efficient Method for Generating Location Updates for Processing of Location-Dependent Continuous Queries”, 7th International Conference on Database Systems for Advanced Applications. 2001.