01 Oct 2005

Location Management in Mobile Ad Hoc Wireless Networks Using Quorums and Clusters

Maggie Xiaoyan Cheng
Missouri University of Science and Technology, chengm@mst.edu

David H.-C. Du

Ding-Zhu Du

Recommended Citation

M. X. Cheng et al., "Location Management in Mobile Ad Hoc Wireless Networks Using Quorums and Clusters," *Wireless Communications and Mobile Computing*, John Wiley & Sons, Oct 2005.
The definitive version is available at https://doi.org/10.1002/wcm.342

WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
Wirel. Commun. Mob. Comput. 2005; **5**:793–803
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/wcm.342

Location management in mobile ad hoc wireless networks using quorums and clusters

Maggie X. Cheng[1*,†], David H.-C. Du[2] and Ding-Zhu Du[2]

[1]*Computer Science Department, University of Missouri, Rolla, MO 65401, U.S.A.*
[2]*Computer Science and Engineering Department, University of Minnesota, Minneapolis, MN 55455, U.S.A.*

Summary

Position-based reactive routing is a scalable solution for routing in mobile ad hoc networks. The route discovery algorithm in position-based routing can be efficiently implemented only if the source knows the current address of the destination. In this paper, a quorum-based location management scheme is proposed. Location servers are selected using the minimum dominating set (MDS) approach, and are further organized into quorums for location update and location query. When a mobile node moves, it updates its location servers in the update quorum; when a node requests the location information of another node, it will send a query message to the location servers in the query quorum. We propose to use the position-based quorum system, which is easy to construct and guarantees that the update quorums always intersect with the query quorums so that at least one location server in the query quorum is aware of the most recent location of the mobile node. Clusters are introduced for large scale ad hoc networks for scalability. Experiment results show that the proposed scheme provides good scalability when network size increases. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: quorum; virtual backbone; minimum dominating set; location service; clustering

1. Introduction

Location management is an important part of wireless communication in both cellular networks and ad hoc wireless networks. In cellular networks, the communication between the mobile host and the base station is just a single hop, and the communication across the network is achieved through base stations that are connected by a wired network. A mobile user's location information is stored in its designated base station called Location Register and is updated every time the mobile user moves around [1]. The location information is retrieved when a location query is sent to the location register.

In ad hoc wireless networks, there is no fixed infrastructure, and mobile hosts move around frequently. In the worst case, a point to point connection between two mobile hosts may need network wide flooding. Position-based routing schemes take advantage of the locality of the destination and effectively restrict the flooding to the area where the destination node actually presents [9]. A prerequisite for position-based routing is that the sender must know the current position of the destination. A location service is used by the sender to find out the position of the destination in order to determine the best route. The reliability and efficiency of the location service has direct effects on the efficiency of the communication.

*Correspondence to: Maggie X. Cheng, Computer Science Department, University of Missouri, Rolla, MO 65401, U.S.A.
†E-mail: chengm@umr.edu

The location service is provided by location servers in ad hoc networks, which store the location information for mobile nodes, and provide location information whenever it is requested. These procedures are called location update and location query, respectively. The deployment of location servers is critical on the quality of the location service. There is always a trade-off between the efficiency of location update and location query. Using more nodes as location servers can provide fast location queries at high success rates, but it introduces more overhead in location updates.

In this paper, we present a dominating set based location service scheme for ad hoc networks. We use the dominating nodes as location servers and further organize the servers into quorums. A dominating set is a set of wireless nodes such that every node in the network is either in the set or directly connected to a node in the set. The role of the dominating nodes are to store the location information of mobile nodes and respond to location queries from the network. In addition, the same set of dominating nodes can also route flows and packages, compute and maintain routes, etc. The advantage of using dominating nodes over a flat structure without dominating nodes is that not every node is involved in location service or routing. Only the nodes that have the most recent information about the topology of the network compute the routes. The topology change caused by node mobility is only propagated to these dominating nodes rather than all nodes, therefore the update overhead is reduced.

Quorum systems have been widely used in distributed file system for mutual exclusion of conflicting actions. In a distributed file system, file servers are divided into subsets, such that any two subsets intersect, and no subset is included in another. These subsets are called quorums. An important property of the quorum system is that any two quorums intersect. A read/write action needs to get permissions from all the servers in a quorum, and a file server can only issue a permission to one request at a time. If two conflicting actions request to access the same resource, there must be at least one server that receives two conflicting requests, but it can only send the permission to one of them. Thus, mutual exclusion is guaranteed.

The property that any two quorums intersect can also be used in the location service in mobile computing. We can divide wireless nodes into quorums and make each quorum a set of location servers. When a mobile node moves, it only needs to update a quorum of nodes; when a location server receives a location query, it will send the query to all the nodes in the same quorum. Since an update set and a query set always intersect, at least one node in the query quorum knows the most recent location information of the mobile node.

The quorum design in wired networks is more concerned with the other properties rather than the location of the nodes in each quorum. Reference [7] is an example of using unified quorum system for location service, in which traditional quorum design techniques are used. In wired/cellular networks, location servers are connected via wired networks, therefore the number of hops between location servers is not a concern. However, in multihop ad hoc networks, it is a serious concern. The locations of server nodes, the total span of each quorum, and the distance measured in hops between the nearest nodes within each quorum are very important, but have never been an issue in wired networks. In previous works such as References [6,7], quorums are formed among location servers without considering the connectivity of the quorum itself. Therefore, the location servers in a quorum are interleaved with non-server nodes and might be many hops away from each other. Thus the quorum span can be spatially very large. By sending a packet through a multiple-hop path, the message is broadcasted to all nodes within one-hop distance of the path due to its omni-directional transmission. The longer the path is, the more energy and bandwidth are wasted. With a unbounded quorum span, this quorum system can be very inefficient in terms of energy and bandwidth.

We propose to construct a minimum dominating set first, which will be used as location servers. Then we form quorums among dominating nodes based on their locations. We further distinguish them as query quorums and update quorums so that the intersection is only guaranteed between query quorums and update quorums. For large networks, we first divide dominating nodes into clusters that each have a complete quorum system. Location updates are only propagated in the same cluster, therefore, the network-wide flooding is avoided. Location queries may involve communication across different clusters, which are routed through cluster heads.

The major contributions of this work are the following:

(1) We presented an efficient distributed algorithm to find the location servers, which has the same performance ratio as the centralized greedy

algorithm and the worst case running time of which is bounded.

(2) We designed a position-based quorum construction scheme, which is more suitable for wireless communication in ad hoc networks than traditional quorum systems. The span of each quorum is optimized based on the location distribution of wireless nodes.

(3) To provide scalable location service in large networks, we proposed to use a clustering scheme and then build quorum systems within clusters. The clustering scheme makes it possible to have a bounded quorum size when the network size increases.

We assume the mobility pattern is pause and move, i.e., nodes move and then pause for a longer time and then move again, so we can model the node movement by a join and a leave. We further assume the pause time period is longer than the location update time. If the network is highly mobile such that it changes faster than the information is updated and retrieved, then a network wide flooding is unavoidable.

The rest of the paper is organized as follows: in Section 2, we review some of the recent works related to location management. In Section 3 and 4, we introduce our location management scheme in detail and explain how location server nodes are selected and organized into quorums. In Section 5, we present how to preserve scalability by use of clusters. In Section 6, we briefly discuss how to maintain the quorum systems and clusters in response to the node mobility and topology changes. Section 7 concluded the paper with summaries and future work plan.

2. Related Work

2.1. Cellular Networks

Quorums have been used in cellular mobile systems for locating the cell in which a mobile node is currently located [10–12]. In cellular networks, the cellular coverage area is divided into cells, and a cluster of neighboring cells constitute a registration area (RA), and one or more RAs constitute a zone. In each zone, there is only one location server associated with it. Since using only the mobile node's identity to determine its location servers fails to exploit the locality of reference characteristics, and using solely the mobile node's location will lead to uneven distribution of responsibility, a mobile node's location is stored in several location servers scattered across the network. Quorums are used to guarantee that the update set and the query set intersect, and dynamic hashing is used to balance the service load among location servers. This is different from mobile ad hoc networks, where the network is infrastructure-less, and each mobile node is potentially a candidate location server that also moves around.

2.2. Ad Hoc Networks

In Reference [8], a virtual backbone is used to store the location databases, and the databases are dynamically organized into quorums of the same size, such that every two quorums intersect. The set of backbone nodes are determined using a network wide flooding such that the backbone nodes are uniformly distributed within the network, and each non-backbone node is connected to at least one backbone node.

How to divide the backbone nodes into quorums is a design choice, and it is also based on the traffic pattern. In large networks, the actual number of location databases is large. As a result the quorum size has to be large in order to ensure the intersection between any two quorums. Thus the cost of location update increases. In this case, network nodes can be divided into p groups by calculating the modulo p of the node ID number. Each resulting group contains the virtual backbone nodes as well as non-virtual backbone nodes, and the databases are organized into quorums among the virtual backbone nodes within each group. The problem with this approach is that the path between a pair of backbone nodes is possibly interleaved by nodes in other groups. So the communication involved in a location update or location query may be routed through a multihop path consisting of both backbone nodes and non-backbone nodes of different groups, of which the number of hops is unbounded.

The proposed location management system also uses quorums, but it is different from References [6–8] in that it limits the path length of location update and location query by having quorums with bounded spans. The location server selection and quorum construction are detailed in Sections 3 and 4 separately.

3. Minimum Dominating Set Construction

The proposed location management scheme involves two phases: location server selection and query/update set construction. We first build the minimum dominating set (MDS) among all wireless nodes, and use the dominating nodes as location servers. Thus, each wireless node is at most one hop away from a location server. In the following sections, we use black nodes to represent the dominating nodes (or dominators), gray nodes the neighbors of dominators. Initially, every node is a white node; upon the termination of the algorithm, every node is either black or gray.

3.1. A Polynomial Time Algorithm for MDS

We define the white degree as the number of white nodes within its one-hop neighborhood including itself. In every step, we color a node to black, so the number of white nodes in its one-hop neighborhood is reduced to zero. We define the effective reduction as (white degree, color), where color is the color of the node itself. Let white $= 0$, and gray $= 1$, so the effective reduction can be compared in a lexicographical order. For example, (white degree $= 3$, color $=$ gray) $>$ (white degree $= 3$, color $=$ white).

Greedy Algorithm: Initially, all nodes are white nodes. At each step, choose the node with the maximum effective reduction, color this node to black, and color its white neighbors to gray. Repeat until there is no white node left.

In case of a tie, it is broken in the following order:

(1) In favor of the node with a larger degree
(2) In favor of the node with a lower node ID

Remark: At the end of this algorithm, for each black node, there must be another black node within its three-hop distance unless there is only one black node in the same connected domain, and three kinds of link topologies are possible: black–black, black–gray–black, and black–gray–gray–black.

To compute a minimum dominating set is an NP-complete problem [4]. The performance ratio of this greedy algorithm is $\ln \Delta + 1$, where Δ is the maximum number of nodes in a one-hop neighborhood. The proof of this performance ratio is presented in the appendix.

3.2. Distributed Implementation of MDS

We propose a distributed implementation of the above greedy algorithm, which produces the same set of dominators as the centralized algorithm does, but does not need any global effort. In addition, the time complexity under the worst case is bounded because it is localized and independent of the network topology.

```
distributedMDS
  initialize color = white
  while color ≠ BLACK do
    exchange state info within one-hop neighborhood, and compute effective reduction
    if there is no white node within one-hop neighborhood then
      break
    else
      send and receive state packet within two-hop neighborhood
      if it has the maximum effective reduction then
        color itself to black
        set dominator to itself
        send out BLACK message to one-hop neighbors
      else if receives BLACK message from one-hop neighbor then
        if color = = WHITE then
          color itself to gray
          set dominator to the sender
        end if
      end if
    end if
  end while
  while color = = BLACK do
    send and receive state packet within two-hop neighborhood
    if there is no white node within two-hop neighborhood then
      break
    end if
  end while
END of distributedMDS
```

A gray node terminates the MDS algorithm and stops exchanging information with its neighbors once it finds out that it cannot be selected as a dominator. A black node terminates the MDS algorithm once it finds out that there is no white node in its two-hop neighborhood. Because the gray nodes it dominates cannot become black later, therefore, the dominator–dominatee relationship will not be changed. The advantage of this algorithm is that it is localized, and it does not need global effort to decide when to stop. Once the MDS construction is finished, the selected black nodes can go on to start quorum construction by sending out a SCAN message. The time lines of this localized algorithm and the centralized algorithm are shown in Figure 1.

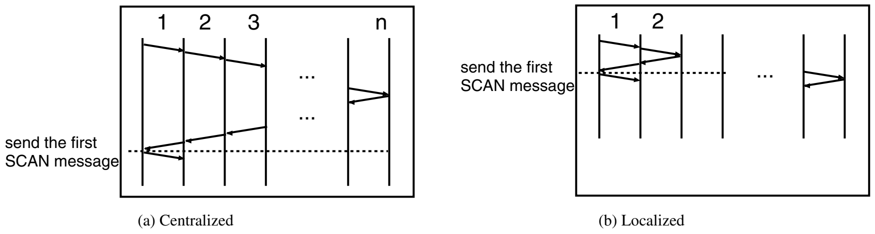


Fig. 1. Time-lines in the centralized implementation and the localized implementation of the minimum dominating set.

The distributedMDS produces the same set of dominators as the centralized algorithm does. Recall that if a node v has the maximum effective reduction within its two-hop neighborhood, it will be selected as a dominator according to distributedMDS. If another node u three (or more) hops away from v is selected as a dominator, it would not change the effective reduction of v for all possible timings. Therefore, v is selected as a dominator in both the centralized and the distributed implementations.

3.3. Performance Analysis

Minimum connected dominating sets (MCDS) have been used in Reference [8] for location servers, in which a set of dominating nodes are selected first using a different scheme, then non-dominating nodes are used to connect them to form a connected dominating set, called virtual backbone in Reference [8]. Quorums are then built from this virtual backbone.

We propose to use the MDS instead of the MCDS as the first step to select location servers based on the following reasons:

(1) MCDS requires more time and message complexity for selecting server nodes. The MCDS construction usually consists of two phases: to select dominating nodes in the first phase and then connect the dominating nodes to form a MCDS in the second phase. The second phase in a distributed environment needs a lot of global collaboration, which increases the time and message complexity.
(2) MCDS uses more server nodes. The number of nodes in a MDS and a MCDS are illustrated in Table I. Figure 2 shows that increasing the network size will increase the number of nodes in a MDS. However, the number of nodes in a MCDS increases much faster than that of a MDS, as shown in the table. When more server nodes are involved, the quorum size will increase, yet the server nodes in a quorum may still be interleaved by the non-server nodes. Thus increase the location update cost, and possibly increase the location query cost too. So a MCDS approach would not fit in a large network.

Table I. Performance comparison of MDS versus MCDS [2,5] experiments were done for upto 1000×1000 square with node transmission range 200 (Dimensionless), the average size of each was taken oven 500 instances of networks.

Performance measurements	MDS	MCDS
Centralized algorithm		
Performance ratio	$\ln \Delta + 1$	$\ln \Delta + 3$
Distributed algorithm		
Performance ratio	$\ln \Delta + 1$	7
Normalized average size from experiments: increasing density at constant area		
For 50 nodes	19	27
For 100 nodes	22	36
For 200 nodes	23	61
For 400 nodes	24	127
For 600 nodes	24	187
For 800 nodes	26	254
For 1000 nodes	28	306
Normalized average size from experiments: increasing area at constant density 10 node per 100×100		
For 40 nodes, 400×100	1.4	1.4
For 40 nodes, 200×200	1	1
For 80 nodes, 400×200	2.5	2.5
For 80 nodes, 300×267	1.2	1.2
For 160 nodes, 800×200	4.45	37
For 160 nodes, 400×400	4.27	4.27
For 250 nodes, 1250×200	5.54	70.9
For 250 nodes, 500×500	6.24	6.98
For 300 nodes, 1500×200	7.10	80
For 300 nodes, 500×600	8	39.6
For 400 nodes, 2000×200	9.61	112
For 400 nodes, 600×667	12	86
For 500 nodes, 2500×200	11	153
For 500 nodes, 700×714	12.28	101.56
For 600 nodes, 3000×200	13	180
For 600 nodes, 800×750	14	148
For 700 nodes, 3500×200	15.86	217.56
For 700 nodes, 875×800	16.65	199.3
For 800 nodes, 4000×200	18	248
For 800 nodes, 889×900	19	242
For 900 nodes, 4500×200	19.85	283.47
For 900 nodes, 1000×900	23.78	273.4
For 1000 nodes, 5000×200	22.24	317.53
For 1000 nodes, 1000×1000	28	306

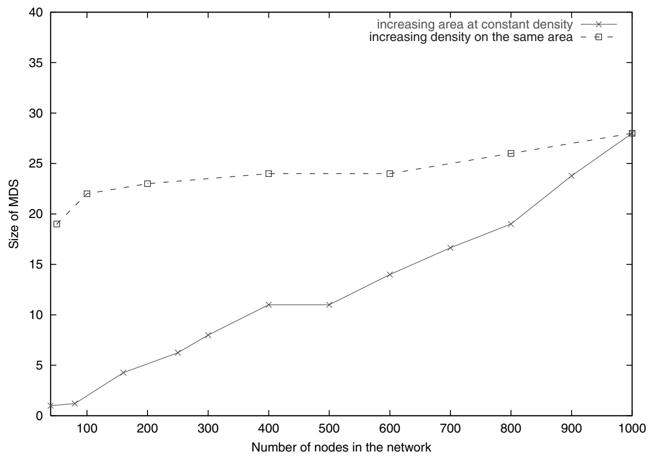


Fig. 2. The size of the minimum dominating set (MDS) increases as the network size increases.

4. Position-Based Quorum System Construction

Input: n server nodes.
Output: A grid of $M \times N$ or $N \times M$ so that there is no empty cell.

M and N can be selected based on the distribution of server nodes. For example, when nodes are uniformly distributed on a square region, let $N = \lfloor\sqrt{n}\rfloor$, $M = \lfloor\frac{n}{N}\rfloor$. Since $M \geq N$, intuitively if the n points are more sparsely spanned in y-coordinates, then we want a grid of $M \times N$, otherwise, we want a grid of $N \times M$. We allow some cell to contain more than one point such that each cell has at least one point. In the following, we will show how to construct a grid of $M \times N$, we call it row-first construction. An alternative way is called column-first construction, which can be done similarly. Instead of allocating the dominating nodes to the logic grid using node IDs, we use their physical locations to map points into cells. The advantage of using a location-based approach is that the number of hops between a pair of quorum nodes is decreased significantly.

Procedure:
(1) Order the n points according to their y-coordinates and x-coordinates, put a point in cell (i, j) if its x-coordinate has the ith rank, and y-coordinate has the jth rank. Finally, n points are put in a grid of $n \times n$ and there is exactly one point in each row, and one point in each column. Figure 3(a) shows the result from this step.
(2) Combine every N rows into one single row, and then we spread the N points in each row into N columns in the increasing order of x-coordinates. Some cell can have more than one point in it. Figure 3(b) and (c) show the procedure in this step.

Next, we build a quorum system from this grid. For intersection $= 1$, each query quorum consists of one row, and each update quorum consists of one column (or vice versa for the column-first construction). There are M query quorums and N update quorums, and

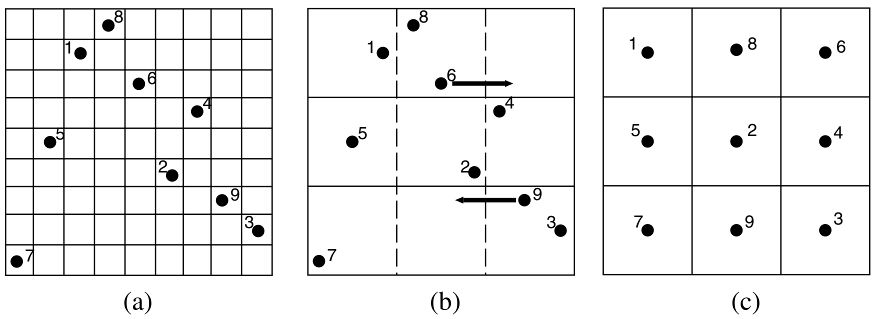


Fig. 3. Construct quorums from server nodes.

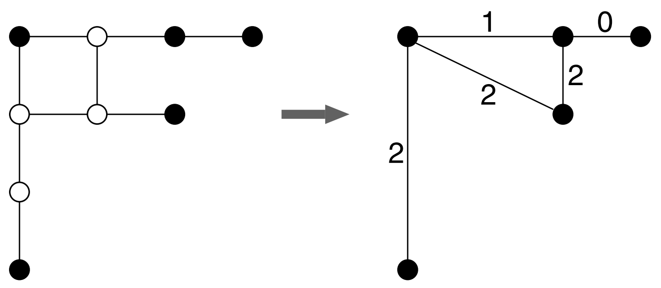


Fig. 4. The virtual graph on the dominating nodes.

each query quorum intersects with exactly one update quorum. For intersection = 2, each quorum consists of an entire row and column, and there are a total of *NM* quorums. For better performance, we can calculate the average span of all quorums for both row-first and column-first methods, then choose the one with a smaller average span.

As *n* increases, to construct quorums that have pairwise intersection requires large quorum size, which is not desired in location service. Ideally, the number of nodes that store the location information for a mobile node should not increase significantly as the network size increases. In the following section, we describe a hierarchical clustering method that maintains a desired quorum size that does not increase with the network size.

5. Location Management for Large Networks

For large networks, to maintain one largely-spanned quorum system that guarantees the pairwise intersection of quorums is expensive. The solution for scalability is to add levels of hierarchy. We propose to group mobile nodes into clusters, and in each cluster, quorums are formed that intersect with each other. Each cluster has a single cluster head, which can be selected during the clustering procedure. Once a cluster head is selected, it computes the quorum system within its own cluster, communicates with other cluster heads, and dynamically maintains the routes to other cluster heads.

Location updates are only limited to quorums in the same cluster, so the location update cost does not increase with the network size. However, it is possible to have query failures, i.e., the read (query) set does not have intersection with the write (update) set. In case of a query failure, the cluster head is responsible for communicating with other cluster heads. Since each cluster head must belong to the quorum system of its own cluster, it can query any read set, which always has an intersection with the write set in the same cluster.

Next, we describe how the clusters are formed.

5.1. Forest Algorithm

Input: Maximum cluster size N_1, and minimum cluster size N_2.
Output: Clusters of size n_1 with $(1 - \epsilon 2)N_2 \leq n_1 \leq (1 + \epsilon 1)N_1$.

After the MDS algorithm is terminated, each black node must be able to find another black node within three hops as long as there are more than one black nodes in the connected network.[‡] All black nodes thus form a virtual graph: an edge exists between two black nodes if and only if they are within three hops of each other, and the weight of the edge is the number of gray nodes in between (Figure 4). The clustering algorithm will run on top of this virtual graph. The construction of this virtual graph is presented in Subsection 5.2.

Before the clustering begins, each black node has the complete list of one-hop neighbors. During the clustering procedure, each node also maintains a cache of the list of nodes that are already in the same cluster. This cache may not contain all the nodes in the same cluster, but it helps to reduce the unnecessary traffic. The cluster head, on the other hand, always maintains the complete list of nodes in its cluster.

Our clustering scheme is a modification of the distributed minimum spanning tree algorithm ([3]). Initially, all black nodes are in a cluster of itself, and the level of the cluster is zero. At each step, the nodes in each cluster collaboratively select an outgoing edge and merge with the cluster at the other end of the edge. This procedure is repeated until there is only one cluster left or the size of the cluster has reached the

[‡]In a connected network, every node has a path to reach every other node in the network.

limit. It is essentially a distributed implementation of the Kruskal algorithm.

We first describe the simple case in which two nomadic nodes combine into a cluster. Initially, every black node is a nomadic node in a cluster of itself. The cluster ID is its own node ID, and the cluster level is zero. It sends out a CONNECT message directly to the nearest neighbor (in case of a tie, a node with a lower node ID is favored). If a node sends and receives CONNECT message over the same edge, two nomadic nodes merge into one cluster of level one, and the cluster ID is the ordered (i.e., ascending order) combination of the two node IDs, the cluster head is one of the ends of the merging edge with a lower node ID.

Next, we describe how two clusters merge into one bigger cluster. First, each black node send a REPORT message to its cluster head including a list of possible outgoing edges and their weights. If a node has no outgoing edge, it will not report. The cluster head first eliminates the non-outgoing edges, decides a best outgoing edge to merge over, and then sends the reporting node the permission to merge. The decision is based on the weight of the edges. In our virtual graph, the edge weight $\in \{0, 1, 2\}$. The edge with the lowest weight is selected, and ties are broken in favor of lower node IDs. The black node then sends out a CONNECT message including its cluster ID, cluster size, and level over this edge. Upon receiving the CONNECT message, the node at the other side will respond differently based on the level and size of its cluster. The other cluster will not agree to merge if its level is lower than the requesting cluster. Our strategy is never to let low level clusters wait.

Let's say node a in cluster A sends a CONNECT message over link (a, b) to node b in cluster B. Next, five different scenarios are possible.

(1) If Level$_B$ is higher, b will send an UPDATE message directly to node a and the message is forwarded to the other nodes of cluster A. In this case, cluster B absorbs cluster A. All nodes in cluster A will update its cluster head, cluster ID, cluster size, and level upon receiving the UPDATE message.
(2) If Level$_B$ is the same as Level$_A$, and edge (b, a) is also the best outgoing edge selected by cluster B, then b will send a CONNECT message to node a (b may have already done so). In this case, two clusters of level L merge into one cluster of level $L + 1$. Both a and b send UPDATE message to its original cluster member to update the level, cluster head, cluster size, and cluster ID. Link (a, b) becomes the root edge. Node a or b, whichever has a lower node ID, will become the new cluster head, and the new cluster ID is the ordered (i.e., ascending order) combination of the two node IDs.
(3) If Level$_B$ is the same as Level$_A$, and edge (b, a) is not selected as the best outgoing edge, node b will not respond.
(4) If *Level*$_B$ is lower, node b will not respond to the CONNECT message. So the connection is delayed until cluster B has a sufficient level.
(5) If cluster B has stopped clustering due to its size, node b will send a REJECT message back to a, so cluster A can eliminate this outgoing edge and try the second best edge.

In all possible cases, if a merge happens, the previous cluster head(s) will update the new cluster head with the list of members so the cluster head always has the most recent information.

This implementation is different from Reference [3]. First, we have less number of message exchanges. Reference [3] uses message exchanges with the other cluster to find out if an edge is outgoing. In *Forest* approach, a cluster can find the best outgoing edge without the participation of the other cluster. Second, we do not require distinct edge weights, but we do require distinct node IDs.

Finally, we discuss when the clustering algorithm should be terminated. If all nodes in a cluster have no outgoing edge, the algorithm is terminated. Another signal to terminate clustering is the cluster size. Once the cluster size n_1 has hit the thresholds such that $(1 - \epsilon 2)N_2 \leq n_1 \leq (1 + \epsilon 1)N_1$, then the next merge operation is to be examined. If further merge generates an oversized cluster, then stop.

Upon the termination of the clustering algorithm, it is possible that a small number of clusters are undersized: i.e., $n_1 \leq (1 - \epsilon 2)N_2$. One remedy for this is to split the cluster from the most recent merging edge and re-cluster each fragment with other clusters in the neighborhood. Re-clustering can be done recursively until there is no fragment left.

5.2. Virtual Graph

So far, we have assumed that the links in the virtual graph are already established before clustering. Next we describe how these virtual links are build to form a virtual graph. Each black node remains inactive initially when it is colored to black. It keeps exchanging state information with the nodes within its two-hop neighborhood periodically. Once it finds out that there is no white node in its two-hop neighborhood, it becomes active and broadcasts a SCAN message. Since there is no white node in its two-hop neighborhood, there must be some other black node within three-hop distance. The SCAN message contains the node ID and TTL. The initial TTL is set to 3. Any non-black node will forward the message with $TTL > 0$ to all its one-hop neighbors except the one it receives from; at the same time, it decreases the TTL by one, and appends its ID to the head of the message. So when a black node wants to reply this message, it will just reverse the path for its REPLY message. After the SCAN period, each node in the neighborhood including the gray node learned the paths between these black nodes. Note a black node does not forward the SCAN message. An REPLY message contains the receiver's node ID only. No TTL is necessary in REPLY messages.

Once a black node receives a SCAN message originated from another black node, it assumes there is a direct link between itself and the sender. It computes the weight of this link as the number of non-black node in between and replies to the sender. All the edge weights are in the set {0,1,2}. After a certain time period, all black nodes have a complete view of its neighboring black nodes.

5.3. Location Service with Clusters

With the clustering scheme, location queries and location updates are implemented differently: A location update is just the same as in a flat structure without clusters. Each moving node contacts the nearest black node, and the black node randomly chooses a quorum that it associates with to store the location information. Location updates are only limited to the cluster in which the node presents. A location query can be more complicated than that in the flat structure. It first queries a randomly chosen quorum within its cluster, if the query fails, the cluster head is responsible for broadcasting to all the other cluster heads, then each cluster head queries a quorum in its cluster. The result is guaranteed to be found and forwarded back to the cluster head sending out the request, and forwarded to the querying node.

6. Dynamic Dominating Set and Quorum Maintenance in Mobile Environment

We assume the mobility pattern is pause and move, and the node pauses for longer time than it moves. A movement can be simplified as disappearing from one site and appearing at another site, so we only deal with the join and leave activities.

Clusters do not change as nodes leave and join unless the cluster size hit the thresholds, or the remaining nodes in a cluster is disconnected. If the cluster size drops below the lower threshold, it will resume clustering as described before; if it is oversized, it will split into two clusters; if it is disconnected, each disconnected part will form a separate cluster and then resume clustering with neighboring clusters if necessary.

Next we describe how to maintain the dominating set and quorum systems when a black node drops out and joins in later.

(1) When a black node drops out of a quorum, some other node in the same quorum first finds out its absence, and then it recruits another node (black or gray) to take its place. If a gray node first finds out that the black node it associated with is out, it will try to connect itself to another black node; if it fails, it will elect itself as a black node and announce its join to the nodes in the same quorum.
(2) If the black node happens to be a cluster head, another black node in the same cluster bearing the same cluster ID is randomly chosen to take its place.
(3) When a node joins in, it simply connects to a nearest black node as a gray node. Packet loss on the fly is possible.

7. Conclusion and Future Work

Routing in a mobile ad hoc network is a challenging task. The position-based routing takes advantage of the location information of mobile nodes and effectively reduce the routing discover overhead and reduce the network-wide flooding. However to maintain the location information of mobile nodes is another challenging task that needs to be implemented efficiently in order to be useful in practical systems.

In this paper, a quorum-based location management scheme is introduced. The location servers are selected from mobile nodes first, and are further organized into quorums that intersect with each other. For large ad hoc networks, clusters are used to prevent the quorum size from growing with network size. Preliminary experiments showed the performance gain of using the proposed system.

Future work along this line would be to implement the position-based routing in a distributed simulation environment using this location management scheme as part of it, and to observe the performance gains and overheads in the routing process. Another work that needs to be done is to provide the optimal quorum size for a given network through analytical study and simulation, especially when it is pertinent to different mobility patterns.

References

1. Bar-Noy A, Kessler I. Tracking mobile users in wireless communication networks. *INFOCOM (3)*, 1993; pp. 1232–1239.
2. Das B, Bharghavan V. Routing in ad-hoc networks using minimum connected dominating sets. *ICC (1)*, 1997; pp. 376–380.
3. Gallager RG, Humblet PA, Spira PM. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 1983; **5**(1): 66–77.
4. Garey MR, Johnson DS. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company: New York, 1979.
5. Guha S, Khuller S. Approximation algorithms for connected dominating sets. In *European Symposium on Algorithms* 1996; pp. 179–193.
6. Haas Z, Liang B. Ad hoc mobility management with randomized database groups. In *Proceedings of IEEE International Conference on Communications (ICC'99)*, Vol. 3, 1999; pp. 1756–1762.
7. Haas ZJ, Liang B. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Transactions on Networking* 1999; **7**(2): 228–240.
8. Liang B, Haas ZJ. Virtual backbone generation and maintenance in ad hoc network mobility management. *INFOCOM (3)*, 2000; pp. 1293–1302.
9. Mauve M, Widmer J, Hartenstein H. A survey on position-based routing in mobile ad hoc networks. *IEEE Network Magazine* 2001; **15**(6): 30–39.
10. Prakash R, Haas ZJ, Singhal M. Load-balanced location management for cellular mobile systems using quorums and dynamic hashing. *Wireless Networks* 2001; **7**(5): 497–512.
11. Prakash R, Singhal M. A dynamic approach towards location management in mobile computing systems. In *SEKE*, 1996; pp. 488–495.
12. Prakash R, Singhal M. Dynamic hashing + quorum = efficient location management for mobile computing systems. In *Proceedings of ACM Symposium on Principles of Distributed Computing*. Santa Barbara, CA, August 1997; p. 291.

Appendix: Performance Ratio of the MDS Greedy Algorithm

In this appendix, we will show that the MDS greedy algorithm has a performance ratio of $\ln \Delta + 1$. Recall the centralized greedy algorithm works as follows.

At each iteration, pick the non-black node with the maximum number of white nodes in its one-hop neighborhood (including itself) and color it to black, and color the white neighbors to gray. Repeat until there is no white node left.

This algorithm has a performance ratio of $\ln \Delta + 1$, where Δ is the maximum number of nodes in a one-hop neighborhood.

Proof: Assume there are n nodes in the network. Let a_i be the number of remaining white nodes after the ith iteration, and b_i be the number of white nodes to be covered in the $(i+1)$th iteration. Therefore, $a_0 = n$, and $b_0 = \Delta$. Let K be the size of the dominating set by the greedy algorithm, so $b_i \geq (a_i/K - i)$ according to the greedy algorithm.

We have $|OPT| \times \Delta \geq a_0$, so $|OPT| \geq \frac{a_0}{\Delta}$

$$\begin{aligned} a_{i+1} &= a_i - b_i \\ &\leq a_i - \frac{a_i}{K-i} \\ &\leq a_i\left(1 - \frac{1}{K-i}\right) \\ &\leq a_{i-1}\left(1 - \frac{1}{K-i+1}\right)\left(1 - \frac{1}{K-i}\right) \\ &\leq \ldots \\ &\leq a_0\left(1 - \frac{1}{K}\right)\ldots\left(1 - \frac{1}{K-i}\right) \\ &\leq a_0\left(1 - \frac{1}{K}\right)^{i+1} \end{aligned}$$

Let $m = i + 1 = |OPT| \times \ln\frac{a_0}{|\mathrm{OPT}|}$. Since $K \geq |\mathrm{OPT}|$, thus

$$\begin{aligned} a_{i+1} &\leq a_0\left(1 - \frac{1}{K}\right)^{|\mathrm{OPT}|\times\ln\frac{a_0}{|\mathrm{OPT}|}} \\ &\leq a_0(1 - \frac{1}{K})^{K\times\ln\frac{a_0}{|\mathrm{OPT}|}} \\ &\leq a_0 \times e^{-\ln\frac{a_0}{|\mathrm{OPT}|}} \\ &\leq a_0 \times \frac{|\mathrm{OPT}|}{a_0} \\ &= |\mathrm{OPT}| \end{aligned}$$

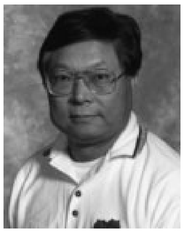
So after m steps, there are at most $|\mathrm{OPT}|$ white nodes uncovered, it takes at most $|\mathrm{OPT}|$ more steps to cover them. The total steps to cover all nodes are therefore

$$\begin{aligned} K \leq m + |\mathrm{OPT}| &= |\mathrm{OPT}|\left(\ln\frac{a_0}{|\mathrm{OPT}|} + 1\right) \\ &\leq |\mathrm{OPT}|(\ln \Delta + 1) \end{aligned}$$

At each step, we choose one node into the dominating set, so the performance ratio is $\ln \Delta + 1$.

Authors' Biographies

Maggie X. Cheng received her Ph.D. in Computer Science from the University of Minnesota at the Twin Cities in 2003. She is currently an assistant professor at the Computer Science Department in the University of Missouri, Rolla. Her current research interests include wireless networking and mobile computing, sensor networks, network security, and combinatorial optimization. She is a member of the IEEE.



David H.-C. Du received the B.S. degree in Mathematics from National Tsing-Hua University, Taiwan in 1974, and the M.S. degree and his Ph.D. in Computer Science from the University of Washington, Seattle, in 1980 and 1981, respectively. He is currently a professor at the Computer Science and Engineering Department, University of Minnesota, Minneapolis. His research interests include multimedia computing, storage systems, high-speed networking, high-performance computing over clusters of workstations, database design, and CAD for VLSI circuits. He has authored and co-authored more than 170 technical papers, including 80 referred journal publications in his research areas. Dr Du is an IEEE fellow and a fellow of Minnesota Supercomputer Institute. He is currently served on the editorial board of *Journal of Cluster Computing*, *Journal of Parallel and Distributed Computing Practices, and Journal of Information Sciences and Engineering*. He has also served as guest editors for a number of journals including *IEEE Computer, IEEE and Communications of ACM*. He has also served as conference chair and Program Committee Chair to several conferences in multimedia and database areas.

Ding-Zhu Du is a professor in Department of Computer Science and Engineering, University of Minnesota. He received his Ph.D. from University of California at Santa Barbara in 1985 and his M.S. from Chinese Academy of Sciences in 1982. Before settled in Minnesota. He worked in Mathematical Sciences Research Institute at Berkeley, MIT, Chinese Academy of Sciences, and Princeton University for totally six years. Currently, he has published more than 140 journal papers and several books. He is the editor-in-chief for *Journal of Combinatorial Optimization* and also in editorial board for several other journals. There are 25 Ph.D.s graduated under his supervision; 15 of them are university professors now. His current research interests include design and analysis of approximation algorithms for combinatorial optimization problems with various applications in computer networking, telecommunication, VLSI designs, etc., especially in wireless networking and mobile computing.