Location Tracking Using Quorums in Mobile Ad Hoc Networks

Hyunyoung Lee[†] Jennifer L. Welch[‡] Nitin H. Vaidya[§]

August 5, 2003

Abstract

We explore applications of quorum systems to the problem of tracking locations of mobile users in mobile ad hoc networks (MANETs). The location tracking system uses biquorum systems, a generalization of traditional quorum systems. We performed extensive simulations of the location tracking system. The simulation results show that our strict biquorum implementation has better performance than the traditional strict quorum implementations. Moreover, our results show that randomized dynamic quorum implementations have better overall performance than strict (bi)quorum implementations.

Index Terms: MANET, location tracking, quorum system, biquorum, randomization

[†]**Corresponding author**. `hlee@cs.du.edu`. Department of Computer Science, University of Denver, 2360 S. Gaylord St. Denver, CO 80208, U.S.A. Phone 1-303-871-7732, fax 1-303-871-3010.

[‡]Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, U.S.A.

[§]Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 60801, U.S.A.

1 Introduction

A mobile ad hoc network (**MANET**) consists of mobile computing entities that communicate with each other through wireless links, without relying on a static infrastructure. Thus, a mobile ad hoc communication environment is highly unpredictable (e.g., unstable links) and resource-poor (e.g., limited battery power). We show that randomization is a promising approach to develop protocols in such unpredictable and resource-poor system environments. As discussed in [4], MANETs differ from mobile cellular telephone networks as follows. Mobile cellular telephone networks consist of two types of communication components: Firstly, there are base stations which serve to maintain location table registers and store location databases. Base stations are not mobile and communicate among each other to forward the information regarding call requests. The second type of components are cell phones, which are mobile and communicate with the base stations to make and receive calls.

Typically, MANETs also consist of two types of functional components [4]: Firstly, there are special participants, which perform administrative functions similar to those performed by the base stations in cellular telephone networks (e.g., they might maintain the location database). The difference of this special participant from the base station is that the participant itself is a mobile entity, i.e., it does not have a fixed location. The other class of functional components consists of mobile entities, which correspond to the cell phones in the cellular telephone network, and which communicate with the special participants to get the needed information. These classes are 'functional' in the sense that a single physical device may participate in the network in both roles.

Thus the difference between the MANET and the mobile cellular telephone network is that in the latter, the location databases are stored in fixed (i.e., non-mobile) locations, whereas in the former, no fixed infrastructure exists. Therefore, in mobile ad hoc communication environments, managing the mobility (i.e., keeping track of the current location of mobile nodes) is an important problem. In [4], an ad-hoc mobility management scheme is proposed, which routes most packets through arbitrary participants. This reduces the danger that the special participants may become a bottleneck. The role of the special participants is limited to storing location tables and computing routes through the general network. As described in [6], the *information dissemination* problem in ad hoc wireless networks is to track the location of each mobile node, and to gather information on the state of each mobile node.

Those problems described above can be classified as a location tracking problem using distributed shared information servers. The mobile nodes communicate with each other using shared information servers. Since MANETs have no fixed infrastructure, every mobile node must be capable of serving as a distributed location information server. One extreme way of constructing the distributed location information server is to replicate the information database in every mobile node. Another way is to replicate the database over a subset of mobile nodes. In such replicated server systems, in order to obtain a correct location information, we need to keep the replicas consistent. One way of implementing replica consistency is to use quorum systems. When the communication environment is unpredictable and resource-poor (as described above), using quorum systems has the following advantages: By selecting a subset (quorum) of replica servers and performing an operation on the subset, we can avoid flooding the network, thus avoid wasting mobile nodes' battery power. Furthermore, the load of each replica server can be reduced by evenly distributing the load among the servers. Finally, a higher fault-tolerance can be achieved by choosing a quorum dynamically out of the reachable nodes. This is particularly desirable in mobile environments with unstable communication links.

Motivated by these issues, we design a distributed location tracking protocol using a biquorum system that is a generalization of the traditional quorum system. We describe the definitions of various quorum systems that we use in this paper as follows. Consider a universe Ω of n replica servers. A *strict biquorum system* B is a pair (U, Q), where each of U and Q is a subset of 2^{Ω}, such that for all $u \in U$ and $q \in Q$, we have $u \cap q \neq \emptyset$. Each element of U is called an *update quorum*, and each element of Q is called a *query quorum*. A *probabilistic biquorum system* relaxes this intersection property, such that $u \in U$ and $q \in Q$ intersect only with high probability. A *strict quorum system* is a strict biquorum system $B = (U, Q)$, where $U = Q$. Similarly, a *probabilistic quorum system* is a probabilistic biquorum system $B = (U, Q)$, where $U = Q$. Our definition of a probabilistic quorum system is equivalent to the definition given in [10].

In this paper, we propose a dynamic quorum (DQ) construction scheme that is a variation of the probabilistic quorum system: In a probabilistic quorum system, a quorum is chosen uniformly at random. A DQ system uses a heuristic that keeps track of the list of unreachable nodes when constructing a quorum. As a consequence, the probability of a replica server being chosen as a member of a quorum no longer has a uniform distribution. This yields some phenomena that are not explained in the probabilistic quorum system, which we describe in the simulation section.

We compare the traditional strict quorum implementation of [6] with our strict biquorum implementation and with the DQ implementation. We perform simulations to answer the following questions:

- How does the strict biquorum model perform in MANETs in comparison with traditional strict quorum models?
- How does the randomization in constructing quorums affect the performance in mobile ad hoc communication environments?

We give an overview of related work in Section 2. We describe the location tracking protocol, and its various implementations with the quorum based replica systems in Section 3. To compare the performance of different quorum implementations, we define several performance measures in Section 4. In Section 5, we explain the simulation setup, and discuss the simulation results and our observations. We conclude in Section 6 with discussions of future work.

2 Related Work

Traditional strict quorum based replica systems have been proposed in [4, 6] for the ad-hoc mobility management problem. The scheme in [4] dynamically constructs a distributed location information database.

Reference [6] also proposes a quorum based solution for the information dissemination problem in partitionable MANETs. To alleviate the problem of query failures, a set of heuristics is used in selecting servers for updates and queries, by maintaining a list of servers that are believed to be unreachable [6].

The quorum systems employed in the two papers [4, 6] are traditional strict quorum systems (as defined above). A traditional strict quorum system with optimal load can be constructed with quorums of size $\Theta(\sqrt{n})$, where n is the number of replica servers. Even though it may provide a good complexity measure (relatively small cost to keep the needed information on a quorum of size $\Theta(\sqrt{n})$), such quorum systems may suffer from low availability[*] in the face of node failures or unreachable nodes.

The probabilistic quorum system, which has been proposed in [10], has the advantages that the load on the busiest replica server is limited and the availability in the face of server crashes is high. However, probabilistic quorums may return outdated information with some small probability. This appears to be a tolerable problem with respect to location information, since traditional strict quorum systems can also return outdated information in mobile ad hoc communication environments, as shown in [6].

The term bicoterie is introduced in [3], for the construction of a quorum system that has two separate sets of quorums (read quorums and write quorums). Our biquorums are different from bicoteries in that a biquorum system does not have the nonredundancy property of a bicoterie system. The nonredundancy property of a set of subsets means that no member (subset) contains any other member (subset). We conjecture that biquorum systems are simpler and easier to build than bicoterie systems due to the absence of the non-redundancy property.

An abstraction of probabilistic quorum systems to randomized distributed shared data structures was shown in [7]. There, it was shown that a monotone variation of the probabilistic quorum systems yielded better performance than the traditional strict quorum systems in terms of the message complexity in some important situations. This observation motivated the work in this paper.

3 The Protocol

We define the quorum-based location tracking protocol for MANETs as follows. We have m mobile nodes in the system. As described above, every mobile node should be able to serve as a location information server. However, w.l.o.g., let a subset of size n ($n \leq m$) mobile nodes serve as the servers. Thus, the location database X is replicated over n mobile nodes. Each replica consists of m tuples, $X_1, \ldots, X_m$. The tuple X_i ($1 \leq i \leq m$) represents the location information of node i, and each tuple is in the form of $\langle$node-id; location information; timestamp$\rangle$. The operations performed on X_i are *update* and *query*. Each operation invocation has its corresponding response: *ack* for update, and *reply* for query. Tuple X_i is updated by node i (single-writer), and queried by other nodes (multiple-reader).

[*]The term *availability* of quorum systems is defined in [12].

Message Format. Each operation invocation and its response is transformed into a message. For example, the format of a *reply* message to a query is illustrated in Figure 1. The *source id* and the *dest id* fields

source id	dest id	message type (reply)	subject/object node id	location info	timestamp

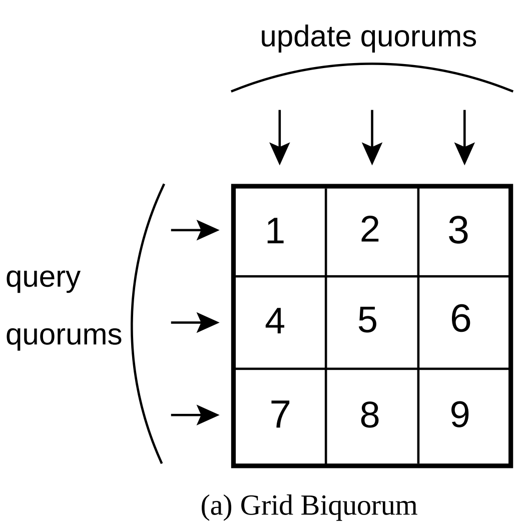
Figure 1: Format of a reply message in location tracking protocol.

indicate the source and destination nodes of the message, and those fields are needed for every operation. The *message type* field indicates one of the four types of messages: query, reply, update, and ack. For a query operation, the *subject/object node id* field is interpreted as the object node whose location is being queried. The *location info* and *timestamp* fields are not needed for a query message. For a reply message, as shown in the above figure, all the fields are needed. For an update operation, the *subject/object node id* field is interpreted as the subject node who is invoking the update operation. And the *location info* and *timestamp* fields contain the corresponding information of the subject node. For an ack message, the *location info* and *timestamp* fields are not needed.

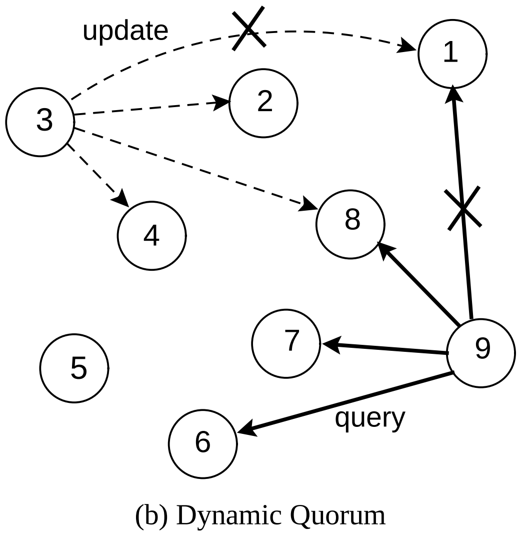
Server and Client Algorithms. Each mobile node runs the following distributed algorithm: As a server, the mobile node h keeps a replica X^h of X. When h receives an update(j, v, t) message from the node j, h updates its replica X^h_j with the new location value v and the new timestamp t. Then h sends j an ack message to acknowledge the update. When h receives a query(j) message from g, h sends g a response message with the location information and timestamp of X^h_j.

As a client, to perform an update, the mobile node h increases its timestamp t by one, chooses an update quorum Q_{update}, and sends out update(h, *new-location*, t) message to each mobile node $q \in Q_{\text{update}}$. When h has received all the acks from every $q \in Q_{\text{update}}$, the update operation is said to be *complete*. To perform a query for j's location, h chooses a query quorum Q_{query}, and sends out query(j) message to each $q \in Q_{\text{query}}$. When h receives all the reply(v, t) messages, the query operation is said to be *complete*. It then chooses the location information v associated with the largest (most recent) timestamp t, out of all the replies and its own local copy. By keeping track of the most recent timestamp (and its associated location information), we can achieve the monotonicity of the queried values.

When a node h tries to access a quorum, we count this as an *attempt*. When h has successfully accessed a quorum, we count this as a *success*. The ratio of successes per attempts will be used to represent the degree of fault-tolerance of the quorum construction scheme in the face of link failure due to mobility. We describe the measure of fault-tolerance in detail in Section 4. When h has failed to access a quorum, a recovery strategy is applied. We discuss different access and recovery strategies in constructing quorums in the following subsection.



(a) Grid Biquorum



(b) Dynamic Quorum

Figure 2: Examples of grid biquorum construction and dynamic, random construction.

3.1 Quorum Constructions

We describe in this section four different strategies to construct quorums: three strict biquorum (SQ) constructions and a dynamic quorum (DQ) construction.

SQ1: Grid Biquorum. Assume that the number of servers, n, is a perfect square. Then we can place the n servers in a $\sqrt{n}$ x $\sqrt{n}$ square grid. The strict biquorum system is defined by taking update quorums to be the $\sqrt{n}$ columns and query quorums to be the $\sqrt{n}$ rows.

We denote this square grid construction by **SQ1**. An example of the square grid construction is shown in Figure 2 (a). Consider a grid biquorum system of nine server nodes. Nodes are logically placed in a square grid (3 x 3). Every node knows the a priori membership information: who belongs to which quorum, including itself. There is always one member in the intersection of an update quorum and a query quorum. The advantage of such quorum system is simplicity. However, such static membership information may result in poor scalability. Furthermore, this system may suffer from the low fault-tolerance of quorums, because whenever a single node in a quorum is currently not accessible, the system fails to access a full quorum, which may fail to guarantee some of fundamental properties of the quorum system, such as the intersection property.

SQ2: Strict Quorum Construction with Increased Intersection. The construction SQ1 guarantees that there is always one member in the intersection of a pair of query and update quorums. Karumanchi et al. [6] union one row and one column to construct a quorum. We will denote this quorum construction by **SQ2**. In the SQ2 construction, there are n a priori quorums formed, at least two members are in the intersection of any pair of quorums, and the quorum size is $2 \cdot \sqrt{n} - 1$.

SQ3: Using Unreachable Nodes List. Karumanchi et al. [6] keep a list of unreachable nodes (*unreachable nodes list: UNL*) to get better performance in case of network partitions, server crash, or link failures. Three kinds of heuristics are used when choosing quorums: (1) Eliminate the quorums that have any of those nodes in the UNL, and then uniformly randomly select one from the remaining quorums (Eliminate-Then-Select: ETS). (2) First select a quorum uniformly at random, and then eliminate those nodes in the UNL from the chosen quorum (Select-Then-Eliminate: STE). (3) Use ETS for updates and STE for queries (Hybrid). We dub the hybrid strategy **SQ3**. After choosing a quorum, timeouts can still occur while accessing the members in the quorum. The SQ3 method does not try to recover from a timed-out access, and returns the best information (i.e., with the most recent timestamp) to the application.

Note that the SQ1 and SQ2 methods do not use the UNL. They first choose a quorum randomly out of the a priori constructed set of quorums and send out the corresponding messages. Then if timeout occurs while accessing a quorum, it is assumed that those timed-out members are currently unreachable, and the best possible information is returned to the application (as in the SQ3 method). The only difference between SQ2 and SQ3 is the use of the UNL: they have the same a priori set of quorums, and their recovery strategies are the same.

DQ: Dynamic Quorum. For dynamic quorum systems, we first eliminate the nodes in the UNL, and then choose uniformly at random k servers to dynamically form a quorum of size k, where k is an input parameter to the quorum system. Quorums are chosen independently. When one or more servers in the dynamically constructed quorum are not responding (even after eliminating the nodes in the unreachable list), the quorum system tries other random choices until k responses are obtained, or until the threshold of the number of additional trials is reached, whichever comes first. When the threshold is reached without obtaining a full quorum of size k, the best possible information gotten so far is returned to the application (as in the SQs). Such a dynamic construction of random quorums may yield high scalability, and high fault-tolerance in the face of link failures due to mobility. Furthermore, the recovery strategy of the DQ tries to access a full quorum, which will be likely to provide a better result in terms of correctness (discussed below) of the location information returned by the tracking system.

Figure 2 (b) illustrates this situation. Let us consider a DQ system of nine mobile nodes. Assume the quorum size is three. Node 3 tries to update its location, and chooses an update quorum as $\{1, 2, 4\}$. By timeout, node 3 learns that node 1 is not reachable, thus chooses another node uniformly randomly, and successfully completes the update operation on the quorum of $\{2, 4, 8\}$. Shortly after the update, node 9 queries the location of node 3, by dynamically choosing a random quorum of $\{1, 7, 8\}$. Node 9 then learns that node 1 is not reachable, thus selects node 6 as the third node of the random quorum, and can complete its query operation. There is node 8 in the intersection of the update quorum and the query quorum, thus node 9 can get the most recently updated location information of node 3.

4 Performance Measures

We define three kinds of measures in order to compare the performance of quorum based location tracking systems in MANETs:

Correctness Rate. The correctness rate indicates the correctness or recency of the location information that the location tracking system returns. We count the number of outdated locations returned by query operations. We define outdatedness of location information as follows.

Definition 1 *The location information returned by a query for node x's location is* outdated *if the returned timestamp is older than the timestamp of the most recent update by x.*

In the simulation, the simulator can compare the timestamps simultaneously at the time when the reply is given back to the application. Let $\mathcal{N}_q$ denote the number of query operations and $\mathcal{N}_o$ the number of *outdated* values returned by those queries. Then the correctness rate is defined to be $(\mathcal{N}_q - \mathcal{N}_o)/\mathcal{N}_q$.

Quorum Fault-Tolerance. The quorum fault-tolerance indicates how well the quorum system can deal with various failures of the communication environment, when constructing a quorum, where failures can reflect network link stability, network partition possibility, crash failures of the nodes, or simply very slow processes (causing timeouts). We say that an attempt to access a quorum has failed if one or more members in the quorum had not responded within the timeout threshold, so the corresponding operation has timed-out. Since different quorum construction schemes have different recovery strategies, we do not count the additional attempts that are involved while in a recovery strategy as failures. Let $\mathcal{N}_a$ be the number of attempts to access quorums and $\mathcal{N}_f$ the number of failures in accessing quorums. Then the fault-tolerance is measured by $1 - \frac{\mathcal{N}_f}{\mathcal{N}_a}$.

System Throughput. The location tracking system throughput indicates how efficiently the system performs. It also reflects the network load. The throughput is measured by the number of completed, successful location tracking operations per unit time (second).

5 Simulation

In our simulation, we use the ns-2 Network Simulator [14] with CMU/Monarch group's mobility extension [2]. We use TCP (Reno) for the transport layer and DSR (Dynamic Source Routing) [5] for the routing (network) layer, IEEE 802.11 MAC protocol for the link layer, and Two Ray Ground Radio Propagation model for the physical layer. The transmitter range is set as 200 meters.

We modified the telnet application so that it incorporates the location tracking subsystem (LTS) layer between the actual application that invokes update/query location tracking operations and the communication network layer. Each application process invokes update and query operations periodically. An application can have at most one pending query operation and one pending update operation simultaneously. The sequence of update operations is independent of the sequence of query operations at each process. As soon as each mobile node starts running, the LTS application process invokes the first update operation. The first query by the process is invoked β time after the process starts. For a query operation, the process selects uniformly at random (among others), the object node whose location is going to be queried.

Subsequently, the invocation of the next update (resp., query) is scheduled for δ time after the invocation of the current update (resp., query). The values for β and δ are given as parameters.

The essential role of the LTS is to interpret the update/query operation invoked by the application process, to generate a set of messages to a quorum, and to handle the response messages gotten from the quorum to generate the result of the operation for the application process. We implemented the LTS with the four algorithms (DQ, SQ1, SQ2, and SQ3) described in Section 3, ran each implementation with sets of parameters, and compared their performance based on the three measures defined in Section 4.

We let T be the timeout interval. An operation times out if some member(s) of the quorum does not respond within T seconds. Then the recovery strategies for different quorum construction schemes are applied in order to conclude the operation. If the recovery strategy requires retrials, the number of retrials is limited by γ. Each application keeps the unreachable nodes list (UNL) by trying to find if there exists a path to every other node in the system. The UNL is updated every τ seconds. The values for T, γ, and τ are given as parameters.

The simulation results shown in this section are with $m = 100$ mobile nodes in the system, placed randomly in a 2-D rectangular area. Out of those, $n = 25$ nodes role as location information servers for the LTS; each of those n nodes acts as both client and server of the LTS. For the movement pattern of the mobile nodes, we used the Random Waypoint mobility model from [1]. The maximum speed of each mobile node is set as 4 meters/sec. The area where the nodes may move around is varied from 300×300 m^2 to 1000×1000 m^2 with an increment of 100 meters in each direction. We ran each simulation for 3600 seconds. The parameter values for this particular simulations are as follows. The parameter β is set to be 20 seconds, δ 7 seconds, T 4 seconds, γ 5 times, and τ 10 seconds.

For the DQ implementation, we ran simulations varying the quorum size from 1 to 15. In the following subsection, we discuss the performance of the DQ implementation regarding various quorum sizes in various areas. And then, we compare dynamic and strict quorum constructions in the next subsection. Each plot in the figures is the computed average of seven runs on seven different movement scenarios.

5.1 Dynamic Quorum Implementations

In Figure 3, it is noted that for the quorum sizes smaller than 5, larger areas show better correctness rate than smaller areas. It is explained as follows. In a small area, almost every node is reachable to each other, thus every LTS client process can have many different choices of random quorums of small size. Therefore, when different clients choose random quorums of small sizes independently from each other, it is unlikely that the two (update and query) quorums overlap. In a large area, almost every node has a limited number

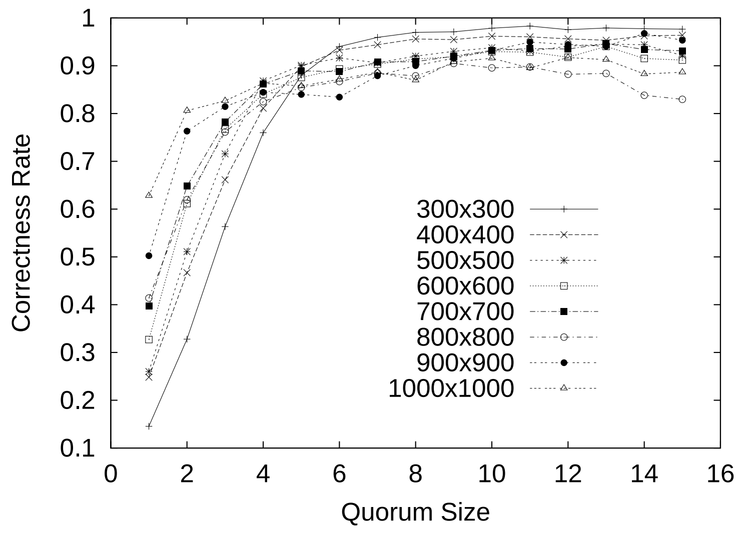


Figure 3: Dynamic quorums: quorum size vs. correctness rate.

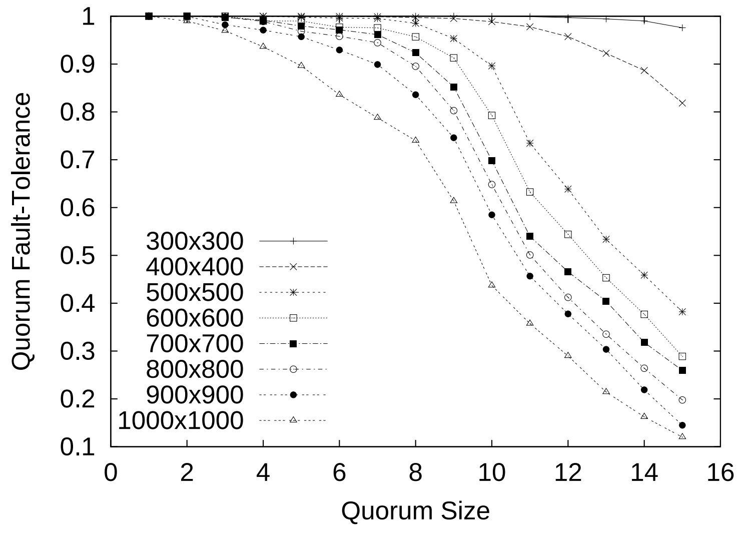


Figure 4: Dynamic quorums: quorum size vs. fault-tolerance.

of reachable nodes, having only a few (or very few) choices of quorums. Thus, when the updating node and the querying node are in a reachable region, the chance that their two (update and query) quorums overlap is high. That is, the intersection property is achieved locally. When the quorum size increases (larger than 5) in larger areas, the LTS clients start having difficulty in constructing quorums, possibly because that many servers are not currently reachable. In small areas with large quorum sizes, the intersection property is achieved with a high probability.

The correctness rate drops in some large area with large quorum sizes (e.g., in area 800×800 with

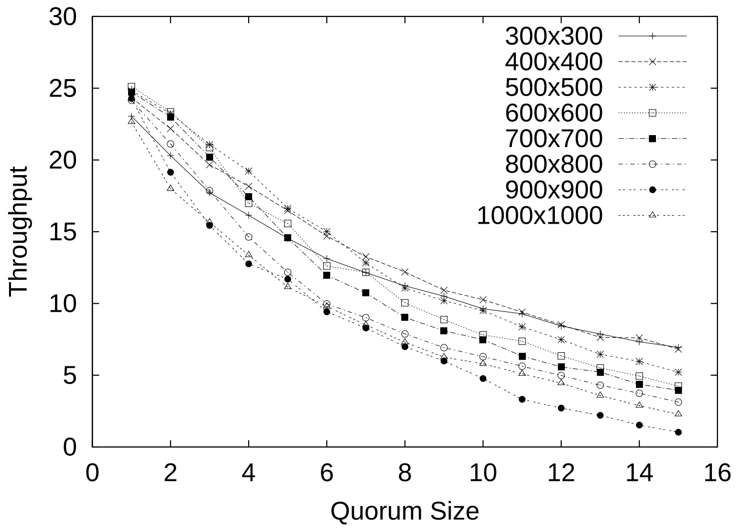


Figure 5: Dynamic quorums: quorum size vs. throughput.

quorum size 11 or more). One observation here that was not explained by the theoretical results shown for the probabilistic quorum system in [7] is that having a large quorum size does not always result in a better correctness rate of the LTS for MANETs with possible network partitions.

In Figure 4, the quorum fault-tolerance clearly displays how network partition affects the success in accessing random quorums dynamically. As the area gets larger, the quorum fault-tolerance significantly drops. Even in small areas (e.g., 400×400), it is often not possible to access large quorums. This is because the LTS first eliminates the nodes in the UNL from the list of servers, when a client tries to construct a quorum.

The system throughput is shown in Figure 5. With the parameter values described above, the maximum number of location tracking operations is approximately 28 ($\approx$ $(3600$ sec $/7$ sec interval $\times\ 2$ update and query operations $\times 100$ clients$)/3600$ sec). With quorum size of one, the smallest area (300×300) and the largest area (1000×1000) equally show the lowest throughput. This is because, in a very small area, there likely occur collisions and contentions on accessing resources (e.g., communication channels and LTS servers). In a large area, due to network partitions, some client may be isolated and may not have access to even a single server.

Since we are counting only the completed, successful operations, as the quorum size increases, no matter how small or large the area is, the throughput drops. This is because increasing the quorum size means increasing the possibility that the operation may not succeed or complete. When the quorum size becomes 10 or larger, the smaller areas (300×300 and 400×400) show better throughput compared to larger areas. This is because even with collisions and contentions, the operations in small areas will more likely succeed than in large areas where accessing large quorums is simply impossible due to the fact that the network is not dense enough to have so many reachable neighbors.

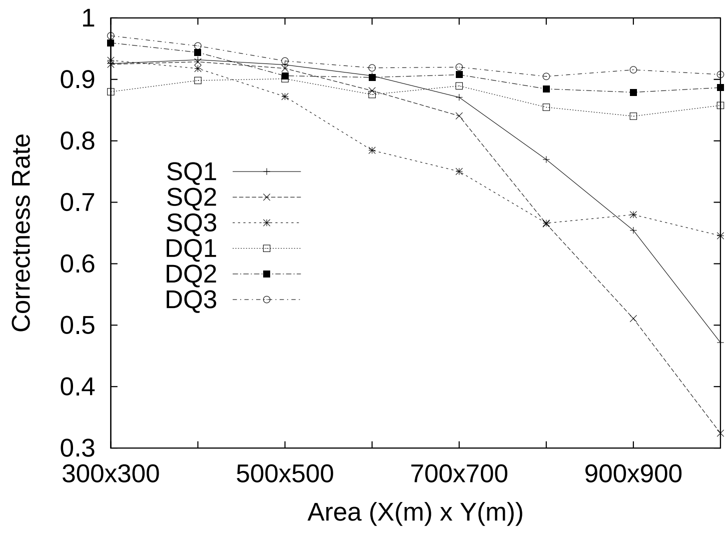


Figure 6: Strict and dynamic quorums: area vs. correctness rate.

5.2 Comparison of Strict and Dynamic Quorum Implementations

To compare the dynamic quorum implementation with the strict (bi)quorum implementations, we selected to plot three different quorum sizes of the DQ implementation as follows. In DQ1, we have chosen quorum size 5 that is the same size as in SQ1. In DQ2, we have chosen quorum size 7, where the intersection of two quorums is expected to have at least one element, by the birthday paradox (note that $7 \approx \sqrt{2n}$). Finally in DQ3, we have chosen quorum size 9 because it is the same size as the quorums used in SQ2 and SQ3.

Correctness Rate. In Figure 6, DQ2 and DQ3 display superiority to any others, even in very large area of 1000×1000. This is mainly because DQs' recovery strategy always tries to access a full quorum of size k, by dynamically including another members whenever timeout occurs at some of the members. The biquorum SQ1 has better correctness rate than that of DQ1 in areas smaller than 700×700, but as the area gets larger, SQ1's correctness rate quickly drops, whereas DQ1's stays almost constant. Similar phenomenon is shown with SQ2. This is because in small areas, the network is rarely partitioned, thus timeouts rarely occur, and the quorum intersection property between update quorums and query quorums of the SQ systems is well preserved. However, when the network becomes partitioned, there are not many quorums available in SQ systems, thus with SQ1 and SQ2, many timeouts can occur (cf. Figure 7). And then the best-effort recovery strategy is applied. This can yield two situations: One situation is that for an update operation, the client LTS may not be able to access the full quorum to update its location information. Thus, it may be that not enough servers have the correct information of the client. The other situation is that for a query operation, the client LTS may not be able to gather sufficient responses from the servers to obtain the recent location information. Consequently, the application receives the location information that is already outdated. Therefore the correctness degrades significantly.

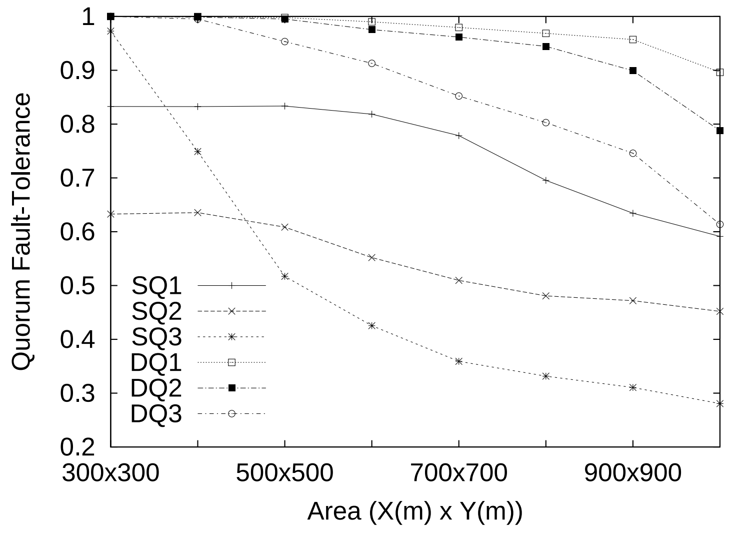


Figure 7: Strict and dynamic quorums: area vs. fault-tolerance.

Among SQs, the biquorum SQ1 shows quite good correctness rate in relatively small areas (say, up to 700×700). Surprisingly, the correctness rate of SQ1 is better than that of SQ2 in all areas, although SQ1 has smaller quorum size than that of SQ2. This can be explained as follows. When almost all server nodes are reachable, the intersection property of both SQ1 and SQ2 is preserved. Thus, having a quorum of larger size only incurs possibly higher overhead, which may slow down the query operations. This affects the correctness of queried values. Especially when the area becomes large, so not every server is in a reachable range, the overhead of a query induced by the timeout becomes significant. For example, a client c wants to query the location of a node d. If c needs to access a quorum in SQ2 that is almost twice as large as a quorum in SQ1, then it is more likely that timeouts occur. In fact, while the query operation is blocked for the timeout period, d may have done another update of its location. This may cause c to receive an already outdated value after the timeout. Therefore, as the area becomes large such as 900×900, SQ3 performs better than the others, while the correctness of SQ1 and SQ2 degrades quickly (recall that SQ3 uses the UNL information, but not SQ1 and SQ2). Even though SQ3 has the worst correctness rate in the mid-sized areas, it displays some lower bound, which indicates that perhaps it can guarantee some bounded deterioration of the correctness rate.

Quorum Fault-Tolerance. In terms of the fault-tolerance, DQs show very good overall performance, of which DQ1 is the best, as shown in Figure 7. DQs do not have a priori construction of quorums, and always dynamically construct the quorum of size k. Furthermore, by first eliminating the unreachable nodes, DQs have higher chances to succeed in accessing the quorums. However, as shown in Figures 6 and 7, DQs have tradeoffs between the correctness rate and the quorum fault-tolerance. This is because the smaller the quorum size, the easier it is to construct a quorum, but the less likely it is that two quorums (for update and

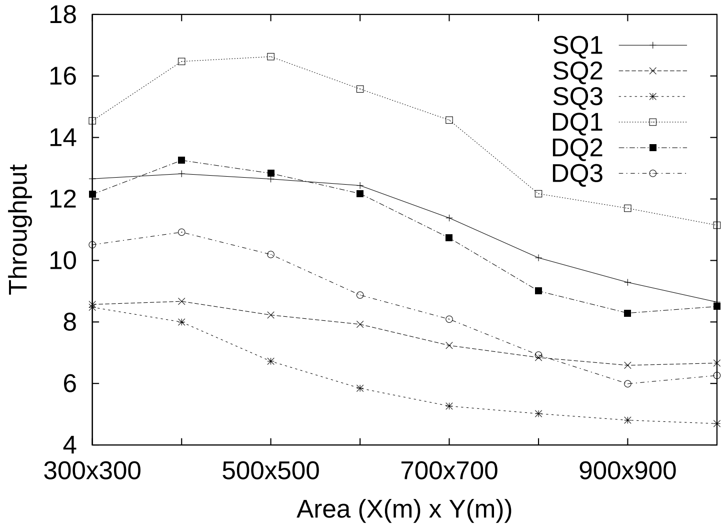


Figure 8: Strict and dynamic quorums: area vs. throughput.

query) overlap.

The quorum fault-tolerance of SQ3 drops significantly as the area increases, and the dropping rate is higher than those of SQ1 and SQ2. This is mainly because SQ3's use of the UNL. This also explains the tradeoff phenomenon between the correctness rate and the quorum fault-tolerance of SQ3 compared to SQ1 and SQ2. In large areas (starting from 900$\times$900), SQ3's fault-tolerance is lower than those of SQ1 and SQ2, but SQ3's correctness rate does not drop as much as those of SQ1 and SQ2 do. This indicates that SQ3's keeping track of nodes' reachability makes it harder to construct quorums, but once a quorum is constructed, then the chance of the returned result being correct is higher than that of SQ1 or SQ2.

We simulated with some other values for the timeout threshold T (2, 8, and 10 seconds) and the time interval τ to compute the UNL (5, 8, and 15 seconds). Varying the timeout threshold affected the dropping rate of the fault-tolerance of SQ1 and SQ2 in the areas larger than 700$\times$700 (e.g., shorter timeouts resulted in more significant dropping rates). Varying the time interval to compute the UNL did not affect much the fault-tolerance of SQ3. However, there were many more operations timeout, when a longer time interval was used, due to the outdated information of the UNL. This affected the query operations more than the update operations because of the hybrid heuristic we simulated (ETS for updates and STE for queries). This yielded worse correctness rates. On the other hand, longer time interval implies less overhead incurred in computing the UNL, which yielded a slightly increased throughput.

System Throughput. Figure 8 shows again that DQ1 is the best in regard to throughput. This is because constructing a smaller quorum implies needing to contact a smaller number of nodes, which takes less time than constructing a larger quorum, thus the system can complete more number of operations in unit time. Furthermore, the flexibility induced by the dynamic construction of quorums incurs fewer collisions than in

SQ1 (with the same size of quorums) in which only an a priori set of quorums are used. We again notice that there is a tradeoff between the throughput and the correctness rate. This gives us some idea about how to set the quorum size, as well as other parameters, to satisfy different requirements by different applications, when the LTS is designed with the dynamic quorum system. DQ2 shows almost identical performance to SQ1, even though the size of a quorum in DQ2 is slightly larger than in SQ1.

Among SQs, the biquorum SQ1 shows the best performance in terms of throughput. Here again the simple mechanism (SQ1) does not incur much overhead, compared to others (e.g. SQ2 and SQ3), thus the system can complete more operations overall.

Our overall observation is that when the network is not much partitioned, simpler algorithms work better. In a reasonable area (of size smaller than 700×700), the biquorum SQ1 shows quite good overall performance. SQ3 does not show, in our simulations, much of the effectiveness of adopting the heuristics. Furthermore it seems that SQ3 with the heuristics is more likely to be affected by the system environment such as the area (density) and system parameters (e.g., time interval to update the UNL). Overall, dynamic quorum implementations show improved performance in all measures.

6 Conclusions

We have explored a distributed location tracking protocol in MANETs, based on a novel, generalized notion of quorum systems, called biquorum systems. We showed that dynamic quorum systems help to improve performance in MANETs. We performed extensive simulations on different quorum implementations. We compared the traditional strict quorum implementations (SQ2 and SQ3) with our strict biquorum implementation (SQ1) and with the dynamic quorum implementations (DQs). In a reasonable-sized area, the strict grid biquorum implementation shows quite a good overall performance thanks to its simplicity. In general, the dynamic quorum implementation performs better than the strict (bi)quorums because the former can construct quorums dynamically, adapting to the changes in the communication environment, whereas the latter rely on the static set of quorums constructed a priori. In summary, we obtained better results by use of randomization, than the previously proposed schemes based on traditional strict quorum systems.

We have observed that the topology change due to the continuous movement of nodes has a significant impact on the performance of the location tracking system based on quorum systems. Thus, understanding the node movement pattern and the communication system environment of the MANET in consideration will perform a critical role in providing an appropriate implementation algorithm for the location tracking system in the MANET. Another direction of future research would be to investigate whether query/update data stores are the most suitable abstraction for the location information database in MANETs, or if different (new) data structures may provide more efficient construction of such location tracking system.

Acknowledgment. This work was supported in part by NSF grant CCR-9972235. We thank Andreas Klappenecker for numerous valuable comments.

References

[1] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva. A Performance Comparison of Multi-hop Wireless Ad Hoc Network Routing Protocols. In *Proc. ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom)*, pages 85–97, October 1998.

[2] CMU Monarch Project Team. The CMU Monarch Project's Wireless and Mobility Extensions to *ns*. August 1999. Available from `http://www.monarch.cs.cmu.edu/`.

[3] A. Fu. *Enhancing Concurrency and Availability for Database Systems.* PhD thesis, Simon Fraser University, Burnaby, B.C., Canada, 1990.

[4] Z. J. Haas and B. Liang. *Ad Hoc* Mobility Management With Uniform Quorum Systems. *IEEE/ACM Trans. on Networking*, Vol. 7, No. 2, pages 228–240, April 1999.

[5] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. *Ad Hoc Networking*, pages 139–172, 2001.

[6] G. Karumanchi, S. Muralidharan, and R. Prakash. Information Dissemination in Partitionable Mobile Ad Hoc Networks. In *Proc. IEEE Symp. on Reliable Distributed Systems*, pages 4–13, October 1999.

[7] H. Lee and J. L. Welch. Applications of Probabilistic Quorums to Iterative Algorithms. In *Proc. of 21st Int. Conf. on Distributed Computing Systems (ICDCS-21)*, pages 21–28, April 2001.

[8] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris. A Scalable Location Service for Geographic Ad-Hoc Routing. In *Proc. of the Sixth Annual ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom)*, pages 120–130, August 2000.

[9] D. Malkhi and M. Reiter. Byzantine Quorum Systems. *Distributed Computing*, Vol. 11, No. 4, pages 203–213, 1998.

[10] D. Malkhi, M. K. Reiter, A. Wool, and R. N. Wright. Probabilistic Quorum Systems. *Information and Computation*, Vol. 170, pages 184–206, 2001.

[11] MIT LCS's Parallel and Distributed Operating Systems Group. The Grid Ad Hoc Mobile Networking Protocol. Available from `http://www.pdos.lcs.mit.edu/grid/`.

[12] M. Naor and A. Wool. The Load, Capacity and Availability of Quorum Systems. *SIAM J. Computing*, Vol. 27, No. 2, pages 423–447, April 1998.

[13] R. RoyChoudhury, S. Bandyopadhyay, and K. Paul. A Distributed Mechanism for Topology Discovery in Ad Hoc Wireless Networks using Mobile Agents. In *Workshop on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2000.

[14] VINT Project Team. The *ns* Manual. February 2001.
Available from `http://www.isi.edu/nsnam/ns/`.