Database Caching in MANETs Based on Separation of Queries and Responses

Hassan Artail, Haidar Safa, and Samuel Pierre

Abstract—**This paper proposes a model for caching database data in mobile ad-hoc networks through caching the queries and their responses. The approach makes a distinction between the queries and their responses and caches them on separate mobile nodes. This method is driven by the fact that queries do not become invalid unless when the design of the database entities changes while data changes depending on the application that feeds information into the database. The architecture we use is hierarchical and employs three types of designated nodes: a Query Directory (QD), Service Manager (SM), and Caching Node (CN). The one or more QDs are responsible for caching the queries and are assigned and supervised by the SM that also oversees the mobility activities and the availability of nodes in the network and makes "managerial" decisions accordingly, including appointing backup nodes. With this model, any node that joins the ad hoc network will either contribute services to other nodes (willingness to become an SM, a QD, or a CN) or consume services offered by other nodes. This model attempts to coordinate the query executing and query caching mechanisms in a seamless manner while maintaining minimal communication among nodes. We present preliminary results of a model that was simulated using the NS-2 software and show the viability of the proposed approach.**

Index Terms—Ad Hoc Networks, Database Caching, Mobile networks, MANETs.

I. Introduction

Mobile ad hoc networks (MANETs) are making the focus of current research. In this type of networks, each mobile node's transmitter has a limited range and mobile nodes communicate using multi-hop wireless links as shown in Fig. 1. Nodes are capable of moving actively and can be connected dynamically as each node can act as a router [1]. Most previous researches have been focusing on the development of dynamic routing protocols that make efficient use of bandwidth and computational overhead [2]. Although routing is an important issue, the ultimate goal of a MANET is to provide mobile nodes with access to services, but for any service to be successful it needs to be accessible from most mobile devices.

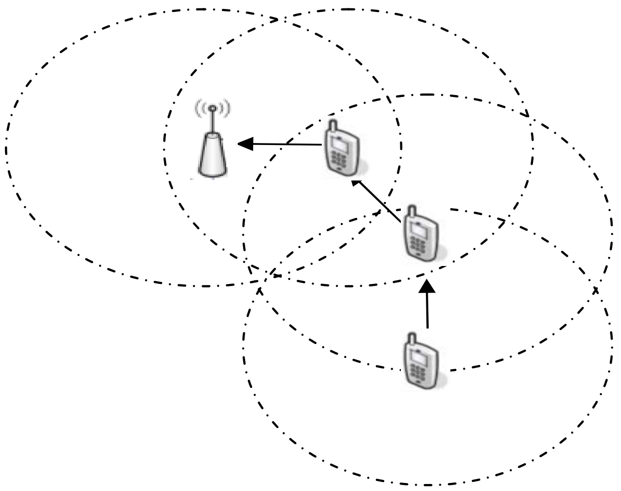
Fig. 1. Wireless nodes routing a request to the base station.

In MANETs accessing services and data over the network can be very slow and hence, caching frequently accessed data is an effective technique for improving performance. While caching data on the device is useful for the device itself, this mechanism does not provide a consistent framework for allowing all other devices in the network to benefit from this data. Therefore, devising an effective caching strategy for the whole MANET is of special importance since it allows for improving the performance of the network as a whole.

The objective of this paper is to propose a new model for caching database data in MANETs through caching the queries and their responses on separate nodes. This model attempts to allow the ad hoc network to function with minimal communications with the database so as to keep the network up and running even with database disconnections.

H. Artail is with the Electrical and Computer Engineering department, American university of Beirut, Beirut, Lebanon (email: ha27@aub.edu.lb)

H. Safa is with the Department of Computer Science, American University of Beirut, Beirut, Lebanon (e-mail: haidar.safa@ aub.edu.lb) and with the Mobile Computing and Networking Research Laboratory (LARIM), Ecole Polytechnique de Montreal, Montreal, Canada H3T1J4 (e-mail: haidar.safa@ polymtl.ca)

S. Pierre is with the Mobile Computing and Networking Research Laboratory (LARIM), Department of Computer Engineering, Ecole Polytechnique de Montreal, Montreal, Canada H3T1J4 (e-mail: samuel.pierre@polymtl.ca).

II. Related Work

Many papers have proposed or dealt with models that involve caching or replication as a way to make services more accessible to mobile devices [3],[5]-[15].

A. Cooperative cache-based data access in ad hoc network

A cache-based data access framework was proposed in [3], that describes how mobile nodes can work as request-forwarding routers. Three different caching techniques were proposed: CachePath, CacheData, and HybridCache.

In CachePath, a node need not record the path information of all passing data; rather, it only records the data path when it is closer to the caching node than the data source. An example is shown in Fig. 2. When node N11 forwards data *di* to the destination node N1 along the path N5 – N4 – N3, node N4 and node N5 will not cache *di*'s path information because they are closer to the data source than the caching node N1.

In CacheData, the router node caches the data instead of the path when it finds that the data is frequently accessed. In Fig. 2, if both node N6 and node N7 request data *di* through node N5, the latter might think that *di* is popular and cache it locally. N5 can then serve N4's future requests directly. However, if node N3 forwards several requests for *di* to node N11, the nodes along the path—N3, N4, and N5— might want to cache *di* as a frequently accessed item. Consequently, they'll waste a large amount of cache space if they all cache *di*. To avoid this, a node does not cache the data if all requests for the data are from the same node.

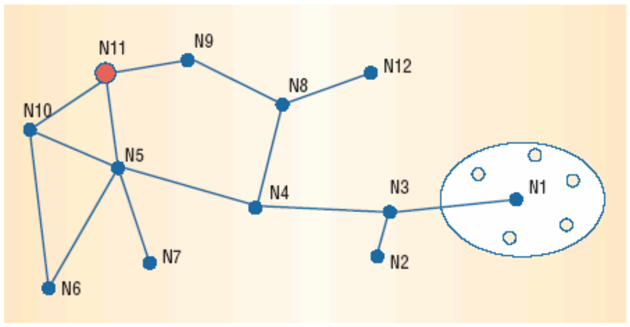

Fig. 2. An Ad Hoc network used to demonstrate the caching techniques.

In HybridCache, a node caches the data or path based on parameters that include the data size and data time-to-live (TTL). If data size is small, CacheData is optimal because the data item *di* only needs a small part of the available cache. Otherwise, CachePath is preferred because it saves cache space. If TTL is small, then CacheData is preferable since *di* might soon become invalid. For a large TTL value, CachePath is acceptable. Caching a data path only requires a small overhead and hence, in HybridCache, when a node caches *di* using CacheData, it also caches *di*'s path. Later, if the cached *di* becomes invalid, the validation algorithm can remove *di* but keeps the path.

With HybridCache, router nodes help other mobile nodes to get the requested data quickly. A mobile node doesn't know whether the data source or some other nodes serve its request. If multiple data sources exist, or if the mobile node doesn't know where the data source is, HybridCache might not be a good option. One possible solution may be the proactive cooperative caching, in which the requesting node actively searches for data from other nodes and the data search can go through multiple hops. Indeed, in proactive cooperative caching, the requesting node broadcasts a request to its neighbor nodes. If a node receiving the request has the data in its local cache, it sends an acknowledgment (ACK) to the requesting node; otherwise, it forwards the request to its neighbors. In this way, a request is flooded to other nodes and eventually acknowledged by the data source or a node with the cached copy. Flooding can create problems such as redundancy, contention, and collision—collectively referred to as the broadcast storm problem [4].

B. *Replica allocation methods in ad hoc network*

Three replica allocation methods for improving data accessibility in ad hoc networks were proposed in [5] and [6]. They all assume that each mobile host creates replicas of the data items, and maintains the replicas in its memory space. When a mobile host issues an access request for a data item *di*, the request is successful in either case: 1) the requesting host itself holds the original/replica of *di* or, 2) at least one mobile host which is connected to the requesting host with a one-hop/multi-hop link holds the original/replica. Thus, first, the requesting host checks whether it holds the original/replica of the target *di*. If it does, the request succeeds on the spot, otherwise it broadcasts the request for the target *di*. Then, if it receives a reply from another host which holds the original/replica of the target *di*, the request is also successful. Otherwise, the request fails.

The three replica allocation methods differ in the emphasis put on access frequency and network topology. They are: Static Access Frequency method (SAF), Dynamic Access Frequency and Neighborhood method (DAFN), and Dynamic Connectivity based Grouping method (DCG). In the SAF method, only the access frequency to each data item is taken into account while in the DAFN method, the access frequency to each *di* and the neighborhood among mobile hosts are taken into account. In the DCG method, the access frequency to each data item and the whole network topology are taken into account i.e. stable groups of mobile hosts are created, and replicas are shared in each group. The technique in [5] assumes that data items are periodically updated, which is not the case of [6].

The replica allocation methods replicate the data item on the requesting node. Using SAF, the replication redundancy may become enormous if a considerable number of mobile nodes frequently access the same data. Using DAFN, the redundancy is eliminated among neighbors. However, this elimination is taking place after the occurrence of redundancy. Consequently, it will consume a considerable portion of the bandwidth.

C. *Cache management for mobile databases*

In [7] a mobile caching mechanism for a mobile environment is described. It investigates three levels of granularity of caching a database item namely, attribute caching, object caching, and hybrid caching. Intuitively, in attribute caching frequently accessed attributes of database objects are cached in a client's local storage. In object caching, the objects themselves are cached. Finally, in hybrid caching, only frequently accessed attributes of those frequently accessed database objects are cached. This ensures that the cached attributes of the cached objects will have a high likelihood to be accessed in the future. This mechanism was implemented using a cache table in each client, to identify if a database item (attribute or object) is cached in local storage. Also, if a client is connected to a server, the client is able to retrieve the cached items from the local storage and the un-cached items from the server. Otherwise the client retrieves only the cached items. The mechanism suffers from some drawbacks since it assumes that each mobile client only communicates with one server while in real applications mobile client might requests items from multiple servers.

D. Caching web services with load balancing

In [8] caching of web services is organized as a service itself. In order to reduce cellular communication, each cached service has a single proxy cache within each ad-hoc network in which it is being used. The node that caches the service is referred to as a local proxy. As stated in [9], the two main issues in any caching system are proxy placement and proxy lookup. The former deals with deciding which nodes should act as proxies for which services and the latter solves the issue of how to find a proxy and how to route to it. The assignment of proxies in the network is subject to fairness criteria that mostly concern distributing the load among the devices. When a device needs to access a service, it first tries to access the proxy it knows about from gossiping and then by referral.

Choosing the right proxy can have a dramatic impact on the performance of the system. Proxies need to be updated continuously to make up for topological changes and partitioning and merges of the MANET caused by mobility.

E. Other issues with caching

Caching mechanisms in conventional client-server environments are usually page-based, primarily because the overhead for transmitting one item or a page is almost the same. Page-based caching mechanisms require a high degree of locality among the items within a page to be effective. In practice, database items requested by different mobile clients via dedicated channels are different. A physical organization that favors the locality exhibited by one client might result in poor locality for another. Database items within a page at a database server thus barely exhibit any degree of locality. Furthermore, mobile clients are powered by short-life batteries and caching a page will result in wasting energy when the degree of locality is low. The overhead of transmitting a page over a low bandwidth wireless channel would be too expensive to be justified. It is, therefore, necessary to consider caching at a smaller granularity in this context.

III. PROPOSED ARCHITECTURE

Our aim is to increase the ability of mobile devices to have access to database data given their highly dynamic characteristics, such as mobility, power limitations, and intermittent availability. Our proposal calls for building a hierarchical architecture for handling the distribution of data, for providing services to the requesting nodes, and for managing the roles of the nodes.

Database access is normally handled using Structured Query Language (SQL) and as such, we can either cache the sent SQL queries (or statements) along with the corresponding return data, or replicate the database data among the mobile nodes. With the latter approach however, the job of interpreting and processing SQL would have to be delegated to one or more nodes, which may be taxing especially if we need to implement the full capability of a database server. Even if one resorts to a light version, replicating the raw data among the mobile nodes brings with it a lot of issues. Among those is the fact that in order to answer certain queries, data may need to be collected across nodes, which could add delays and consume additional power. Further, certain strategies would have to be adopted for deciding how to break the data among the nodes in order to increase the probability of answering most queries by contacting a single node.

A. Components of the Architecture

We base our caching model on caching the SQL statements that correspond to queries and their return data. In effect, our proposed architecture makes query discovery the underlying service that enables access to cached database data. The entities in the architecture are the Service Manager (SM) and its backup (BSM), the Query Directories (QDs) and their respective backups (BQDs), and the mobile nodes that cache the query responses, which we refer to as the Caching Nodes (CNs). The CNs only store the query responses, i.e., the database data, and associate them with the queries that caused them to be returned by the database server. Similarly, the QDs store the queries without the data and also associate these queries with the CNs that hold the corresponding data. Finally, the SM keeps track of which nodes have the role of Query Directories and backup Query Directories and makes assignments and reassignments based on availability and fairness criteria.

The SM is elected based on capabilities, availability and other factors whenever the network is formed or when its structure changes. The SM elects a backup SM (BSM), with which it will synchronize every interval of time. Upon demand, the SM will assign the QDs and CNs, and directly assigns the BQDs. First, the SM starts with assigning the first QD, which caches the queries and maintains corresponding pointers to the nodes that submitted these queries. These nodes become the CNs while the pointers form entries in hash tables that link the queries to the corresponding CNs. Each QD will synchronize its entries (cached queries and hash table entries) with its BQD every specified time interval.

To provide an overview of the roles of the basic components of the architecture, we provide a physical view of the architecture in Fig. 3 and a conceptual view in Fig. 4.

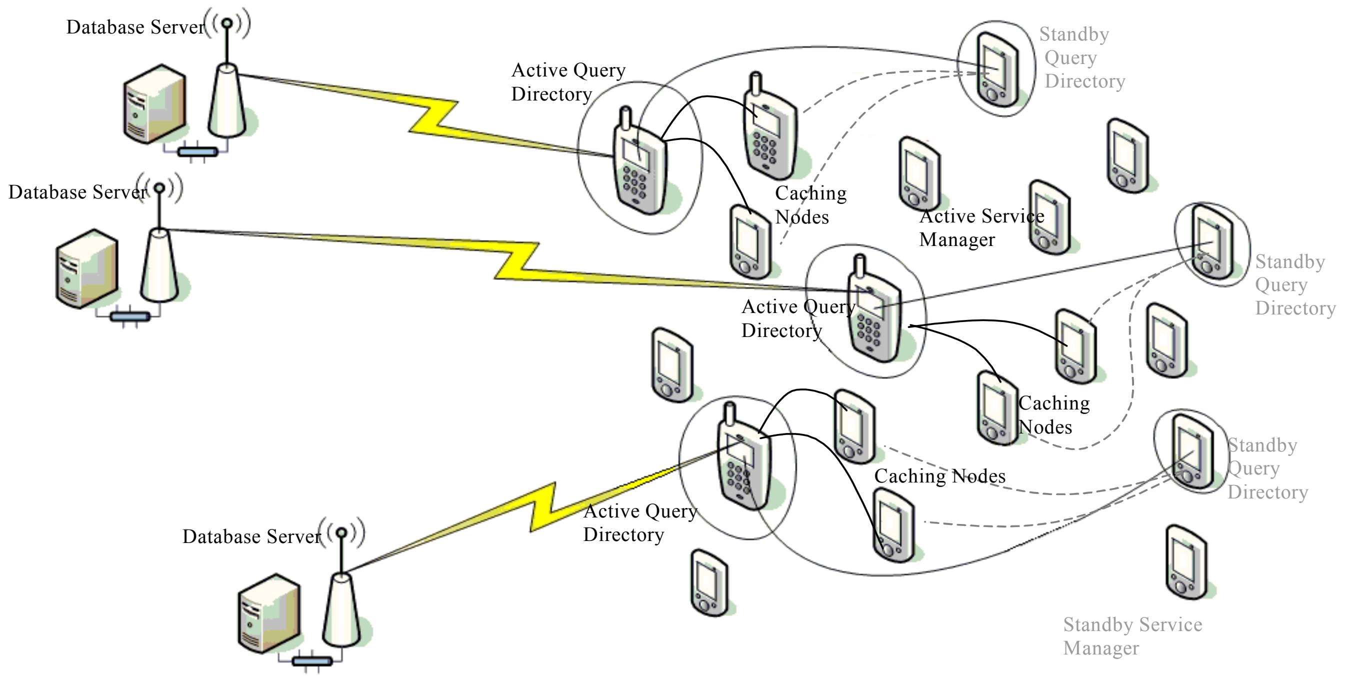

Fig. 3. Physical overview of architecture.

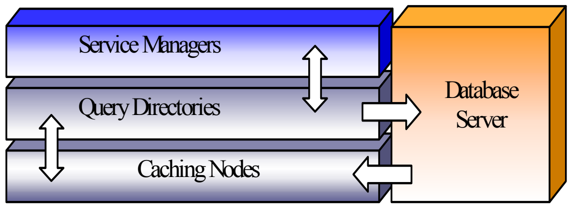

Fig. 4. Conceptual overview of architecture.

B. *Query and Result Caching Strategy*

Caching data (numerical, textual, or mixed data) on the Caching Nodes requires the reservation of storage space, which is limited on mobile devices and hence, we need to make every attempt to save space and be able to increase the collective amount of cached data in the network. Many strategies have been discussed in the literature that we can benefit from. This includes duplication management techniques (also related to node clustering) and compression algorithms.

Caching the queries themselves in the Query Directories requires special attention as these can be lengthy and storing them as normal text is costly. Instead we devise a simple algorithm that builds on the limited set of SQL keywords and on the usually small number of database tables and associated fields (columns). Actually, the only SQL statement that concerns us is SELECT because the others deal with data modification. Although this is relevant, we elect to treat this subject in another paper and focus instead on the design of the proposed architecture and on studying its performance.

C. *Updating the Mobile Network*

The Service Manager (SM) keeps track of QDs through periodic multicasts that it issues to the QDs, which it knows about (since it assigned them these roles). Information about the available QD nodes is then broadcasted across the mobile network so all mobile nodes know the number of QDs and where to send their queries to get the data (if such queries are cached). Forwarding pointers can be used to link QDs.

When submitting a query, the requesting Mobile Node (MN) will generate a random integer number between 1 and the number of QDs and then send its query to the corresponding QD in the list of QDs. If this QD is far, i.e. not in a one hop range, the query will be routed to it using the Ad hoc On-Demand Distance Vector Routing (AODV), used in mobile Ad hoc networks.

If the requested query is not available in the chosen QD, the latter will send the query with the MN's identifier (IP for example) to the next QD in the list. Also, if the query was not available, this next QD will send the query with the MN's identifier to the following QD, so on until the last QD in the circular list is reached. If the query was found on a QD, the latter will check its hashing table to know which CN holds the output of this query. Knowing the requester MN's identifier, the CN will send the previously cached data to the requester.

To cover for the case where no QD exists that holds the query (equivalent to saying that the sought data is not cached), the non-full QD in the list will save this query and send it to the database to get the results. As implied from above, each requesting node becomes a CN once it obtains the results of its query and will thus participate in providing caching services to the network. If its designated storage space ever becomes full of cached results, it may use one of the replacement algorithms (e.g., least recently used or least frequently used) to store the new result at the expense of an old one. This however necessitates informing the QD that holds the corresponding query to invalidate this query, hence the need for a hash table on each CN to link to the corresponding QDs. When serving other nodes, the CN forwards the data item to the requester using its identifier that it retrieves from the request packet it gets from the QD.

Initially, the system starts with a single QD that caches queries. When this QD is filled beyond a given threshold, it informs the SM, which will assign an additional QD. When this new QD becomes filled, another node will join the list of QDs that are linked in a circular manner (see Fig. 5). The number of QDs will increase until some kind of steady state level is reached (i.e., increasingly less queries are not found in the cache). As implied, at any point in time, there is one QD that is available to cache non-cached queries and we refer to it as the non-full QD (NF-QD). If the submitted query was not found in the QDs, it is the NF-QD that goes to the database to fetch the results, adds the query to its local cache, updates its hash table, and forwards the result to the requesting node.

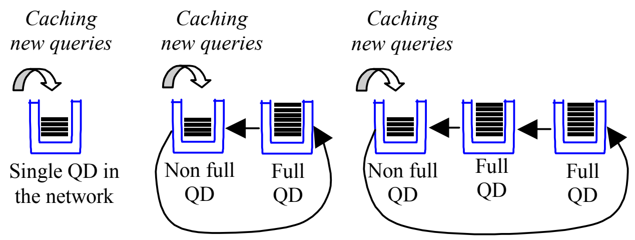

Fig. 5. Formation of Query Directories.

Note that with the above strategy, the Service Manager does not have to issue new updates to the network. That is, the nodes in the network do not need to be updated whenever the cache of the NF-QD is updated because they follow a chain of pointers between the QDs during the search procedure. When no QDs are known to hold the query, the last QD in the circular list informs the NF-QD about the need to cache this query and submit it to the database in order to obtain the results. A requesting mobile node will send its query to the one of the QDs, chosen randomly for the purpose of distributing the load among the QDs. If this QD has the query cached, it fetches the results from the corresponding CN and return them to the node. Otherwise it forwards the query to the next QD in the chain (the NF-QD could be anywhere in this chain). If the last QD in the chain is reached and the query is not cached, the last QD informs the NF-QD, which in turns gets the data from the database and updates its cache and hash table, and sends the results to the requesting node (as explained above).

As mentioned above, certain cached query responses may be replaced by new ones in the CNs, thus causing the corresponding queries to become invalid and consequently creating free space in possibly filled QDs. For simplicity however and because this situation is not expected to occur frequently (i.e., CNs becoming filled with cached responses), we do not take advantage of this freed space and instead keep on using the NF-QD to cache non-cached queries. Unlike this scenario, when updated data in the database causes cached responses in the CNs to become invalid, the cached queries in the QDs do not get affected.

D. Backups

Since the content of the QDs does not change under normal conditions, it would not be a burden bandwidth-wise to institute a cache replication strategy for the QDs. For each QD, we propose that the first faraway node (>H hops) making a request to use this QD is chosen as its BQD (a node can be both a CN and QD). We chose a faraway node to act as a backup node for the mere observation that if a node went down because of signal loss, a close node will likely experience the same problem and a faraway node will be a better choice to act as backup with a less probability to go offline simultaneously.

As previously mentioned, the QDs will synchronize with their backup nodes every specified amount of time. When a node, with a specified function, for example a QD, goes offline, its respective backup (the BQD) will take charge and become an active node (the QD). Then it will ask the SM to assign a backup node for it (a BQD), to maintain the redundancy and to protect against data loss.

E. Consistency of Query Responses

While the content of the cache in the QDs is not expected to change considerably within a given period of time, the data cached in the CNs may change depending on the nature of the data in the database and the underlying application. Dealing with issues related to cache data consistency is not new and has been discussed extensively in the literature (known as cache validation). For the proposed architecture, keeping the query responses consistent with the database is challenging due to their association with queries. We will not discuss this topic any further as it is the subject of another paper.

IV. SIMULATIONS AND RESULTS

To simulate the proposed architecture, we used the NS-2 software, which is a network simulator that is widely used for research [16]. It simulates various IP networks and implements many network protocols such as TCP, UDP, and FTP. It also implements multicasting and some MAC layer protocols to simulate LANs and can be used to simulate large networks. An important feature in NS-2 is the wireless model, which simplifies the simulation process for the proposed architecture. The simulations were built using Tcl scripts and C++ programs while the results that we present below were obtained from the log-trace files that the simulator generates. In this section, we will show the simulation results of our architecture and compare the results to no caching where all the queries are sent to the database.

A. *Simulation Setup*

For the simulations, we chose an 800m×800m terrain and placed in it twenty five nodes that use the AODV routing protocol. The nodes issue random requests at intervals of 2 seconds over a period of 1000 seconds. As for mobility, we used CMU's node movement generator *setdest* to create random waypoint [17] node movements for the 25 mobile nodes. The Random Waypoint model is mainly used for ad-hoc network simulation. Fig. 6 shows the locations of the nodes at a particular instant of time.

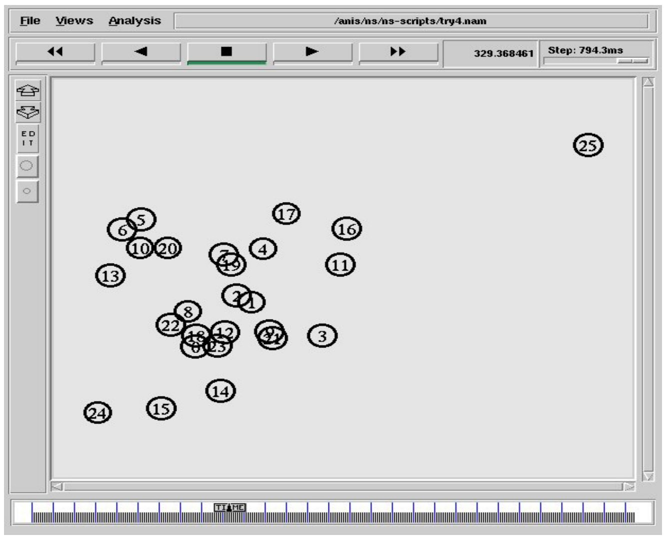

Fig. 6. Network Topology using NS-2 (after 329s).

For the simulations, we made some simplifications to get preliminary results that confirm the viability of the approach:

1) We manually selected three QDs to be near the center of gravity of the network and made the assumption that all mobile nodes (MNs) know about it.

2) The nodes that requested queries were selected randomly and uniformly. This directly implies that the CNs were uniformly distributed throughout the network.

3) No replacement algorithm was implemented to update the contents of the CNs as their storage capacity was assumed unlimited.

4) The database server was implemented as a fixed node far from the mobile nodes (node 25 in Fig. 6)

5) At this stage, the SM and the backup nodes (BSM and BQDs) were not implemented.

6) The network comprises equally capable nodes.

7) It was assumed that the DB server is always accessible by the network.

8) We simulated 40 total queries that represent all the possible queries that the MNs could request

B. *Simulation 1*

It is intuitive to examine the time to answer the queries using the proposed approach and compare it to the case of no caching. We had all the 24 mobile nodes (QDs and CNs included) submit queries randomly from the pool of 40 possible queries and then measured the time it took to get the response back. The database response was simulated with a packet that was processed by the database within a random delay value that is uniformly distributed to simulate the random nature of the response size. That is, the responses to different queries have varying lengths, depending on the type of the application. The delay value ranged between 10 and 40 milliseconds. The total number of queries that were submitted by the MNs is 100 and the MNs that submitted them along with their order were also randomized. The result of the simulation is shown in Fig. 7. To understand the graphs in the figure, we note that each value shown for the caching case involves multiple values as follows:

In the case of misses, the value is the summation of:

- Time it took to route the request from the requesting node (MN) to HQD.
- Depending on how full the QDs are, the request may be handled by QD1, QD2, or QD3. By handling we mean

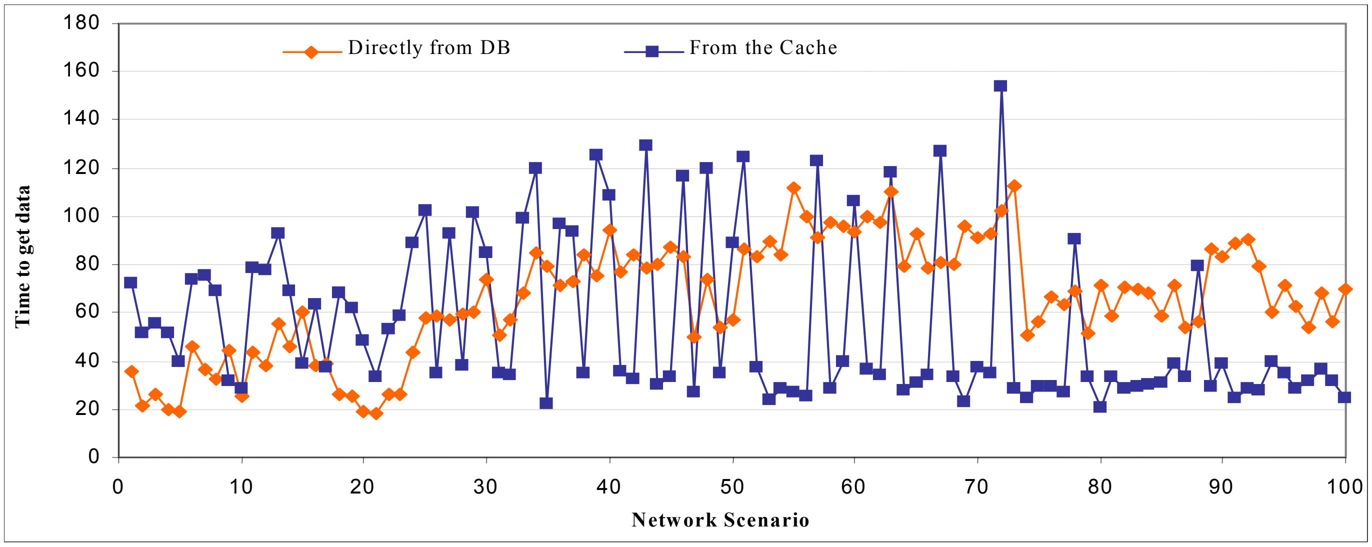

Fig. 7. Time consumed to get query responses.

checking for a match then routing it to one of the CNs. The worst case corresponds to the cumulative times to check and route the request from QD1, to QD2, to QD3, and then to one of the latter's CNs.
- Time it took the CN to send the request to the DB server.
- Time it takes the DB server to process the query, fetch the response, and then send it to the requesting CN.
- Time for the CN to return the response to MN.

In the case of hits, the value is the summation of:
- Time it took to route the request from the requesting node (MN) to the chosen QD.
- Depending on how full the QDs are, the request may be handled by QD1, QD2, or QD3. By handling we mean checking for a match then asking the CN that has the response to send it to MN.
- Time it takes CN to send the response to MN

Before discussing the results in Fig. 7, we present Fig. 8, which clearly shows the database accesses and the times it took to get the responses back. The large bars correspond to queries that were sent to the database for processing by a CN that belongs to one of the QDs. Their number is 40, which is the total number of possible queries used by the MNs. The other small bars represent individual delays as indicated in the figure's legend. The figure is busy but it helps to note that the bars are divided into groups of four individual bars, whose cumulative values are depicted in the curve with square symbols in Fig. 7.

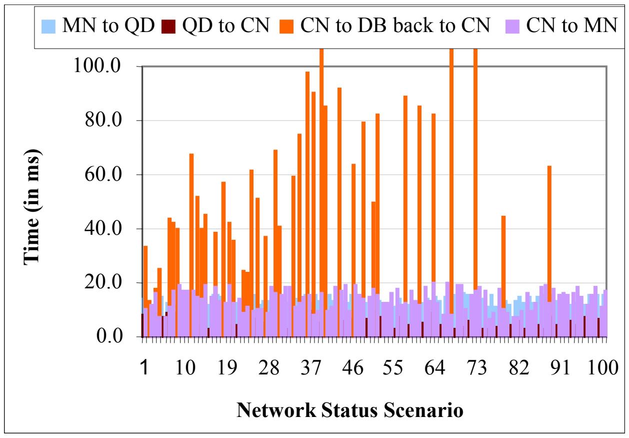

Fig. 8. Hop Count vs. Number of requests.

Back to Fig. 7, the cache hits and misses are easily identifiable from the associated curve. Misses are represented by the top peaks while hits correspond to bottom peaks that fall in the range of 20 to 40 milliseconds. The top peak values vary depending mostly on delays incurred by query processing on the database. The curve with the diamond symbols reflects direct requests issued to the database (node 25 in Fig. 6). This curve was generated by having each MN that sends a query to a QD, also send a query to the database immediately after, and then measuring the time it took to get the response.

We observe that initially most of the requests cause traffic to the database and fill the QDs and CNs with queries and responses, respectively. With more requests, increasingly more queries result in hits. In the case of misses, the caching approach results in longer delays before the MNs get their data but in the case of hits, the MNs get much faster responses. As is evident, the exact gains depend mainly on the delays to reach the database and have it process the queries. More importantly though, the entire MANET or parts of it may become disconnected from the database, in which case the cache will prove to being essential.

C. Simulation 2

For this simulation we studied how the cache size affects the average hop count. By average hop count, we mean the number of hops from the time when the request is issued until the reply is received. Here, we varied the cache size of the QDs between 2 and 15 queries and measured the hop counts in ten different simulation runs and plotted their averages in Fig. 9. For example, when the cache size was eleven, the average hop count (over the 10 simulations) was between five and six hops. As anticipated, the hop count decreases as we increase the cache size per QD. In reality the cache size is dependent on many factors, including the device model and its available storage space. It should be noted that such results can provide valuable information for the Service Manager to take into consideration when electing QDs.

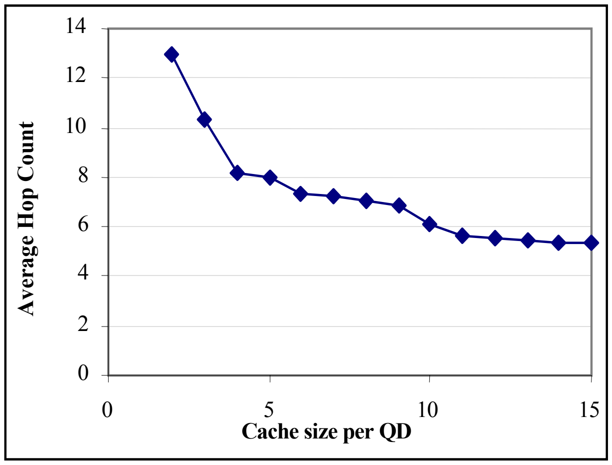

Fig. 9. Hop Count vs. cache size per QD.

D. Simulation 3

In this simulation, we measured the average hop count for the two cases: with caching and without caching. As shown in Fig. 10 below, comparable results for both cases were obtained, which implies that no major losses were incurred in the case of the proposed caching approach.

V. Conclusion And Future Work

We presented an architecture for database caching in mobile ad hoc networks and simulated the architecture using the NS-2 tool. Our preliminary results proved effectiveness of the architecture especially in the case where the nodes have moved away from the database server. The results also show an improvement in response time when compared to no caching. We note that this work is part of ongoing research the will study the different aspects of the proposed model and compare to other architectures that were proposed for MANETs.

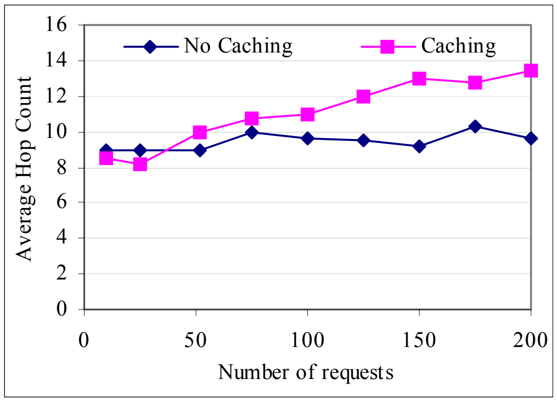

Fig. 10. Hop Count vs. Number of requests.

ACKNOWLEDGMENT

We would like to acknowledge the work of the following students on programming the simulations and acquiring the data: Ibrahim Abdel Nabi, Mark Akl, Fahim Al Halabi, Sami El-Arayssi, Gabriel Habre, Ahmad Husseini, Majdi Kiwan, Anis Nazer, Hamza Nizameddine, Ayman Tajeddine, and Rami Zakhour.

REFERENCES

[1] J. Schiller, *Mobile Communications*,2/E, Addison Wesley, ISBN:0-321-12381-6
[2] C. E. Perkins, E. M. Royer, and S. R. Das, "Ad Hoc On Demand Distance Vector (AODV) Routing," Internet Draft. Mobile Ad Hoc Networking Group. 14 July 2000.
[3] Cao, L. Yin, and C.R. Das "Cooperative cache-based data access in ad hoc networks", *IEEE Computer*, vol. 37, no. 2, 2004, pp 32 – 39.
[4] Y. C. Tseng, S.Y. Ni, and E. Y. Shih "Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network," *IEEE Transactions on computers*, vol 52, no. 5, 2003.
[5] T. Hara, "Replica allocation methods in ad hoc networks with data update," *Mobile Networks and Applications*, vol. 8, Issue 4, 2003.
[6] T. Hara, "Effective Replica allocation in ad hoc networks for improving data accessibility," *In proceedings of IEEE INFOCOM 2001*, pp. 1568-1576.
[7] B. y. Chan, A. Si, and H. V. Leong, H.V. "Cache management for mobile databases: design and evaluation," *In the Proceedings of 14th International Conference on Data Engineering*, February 1998, pp. 54 – 63.
[8] R. Friedman, "Caching web services in mobile ad-hoc networks: opportunities and challenges," *Proceedings of the second ACM international workshop on Principles of mobile computing*, Toulouse, France, 2002, pp. 90 – 96.
[9] R. Friedman, "Locating cache proxies in manets," *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*., May 2004.
[10] Barbará and T. Imieliński, "Sleepers and workaholics: caching strategies in mobile environments," *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, vol. 23, issue 2, 1994, pp. 1-12.
[11] Chi-Hung Chi, and Lau S.L., "Data Prefetching with Co-operative Caching," *5th International Conference on High Performance Computing*, HIPC '98, 1998, pp. 25 – 32.
[12] Valera, W.K.G Seah, S.V. Rao, "Cooperative Packet Caching and Shortest Multipath Routing in Mobile Ad Hoc Networks," *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE INFOCOM 2003, vol. 1, March 30th to April 3rd 2003. pp. 260 - 269
[13] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Trans. Knowledge and Data Eng*., vol. 15, 2003, pp. 1251-1265
[14] G. Cao, "Proactive power-aware cache management for mobile computing systems," IEEE *Transactions on computers*, Vol. 51, no. 6 , 2002, pp. 608 – 621.
[15] X. K. Shao and Y. S. Lu, "Maintain Cache Consistency of Mobile Database Using Dynamical Periodical Broadcast Strategy," *In the Proceedings of the Second International Conference on Machine Learning and Cybernetics*, Xi'an, November 2003.
[16] NS-2 simulator. http://www.insi.edu/nsnam/ns/ [April 2002].
[17] Christian Bettstetter, Giovanni Resta, and Paolo Santi, "The node distribution of the random waypoint mobility model for wireless ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 2, no. 3, pp. 257-269, July-September 2003.