A Two-layer Cache Replication Scheme for Dense Mobile Ad hoc Networks

Kassem Fawaz and Hassan Artail
Department of Electrical and Computer Engineering
American University of Beirut
Bliss Street, Beirut, 1107 2020, Lebanon
{kmf04, hartail}@aub.edu.lb

Abstract—**This paper proposes a data replication scheme implemented on top of a cooperative data caching architecture in MANETs that caches submitted queries in special nodes, called query directories (QDs), and uses them to locate data (responses) stored in the nodes that requested them, and called caching nodes (CNs). The QD entries are replicated according to a cost minimization model, and the actual data items are placed in nearby CNs. The proposed system is dynamic, as it adapts to topology changes and relocates replicas as necessary. The preliminary prototype of the proposed method is simulated using ns2 to assess its performance experimentally. Enhancements in performance in terms of lowered access delay and improved hit rates are reported, while maintaining a cap on overhead traffic.**

Keywords-replication; data accessibility; caching; MANETs

I. INTRODUCTION

In a mobile ad-hoc network (MANET) environment data caching is essential due to its role in reducing contention in the network, increasing the probability of mobile nodes to have access to desired data, and improving system performance, by essentially reducing access delay [13], [14]. Many caching paradigms have been proposed for MANETs over the past decade, including the COACS system introduced by the authors in [1]. However, most of these caching architectures lack a replication infrastructure, and suppose that data items are cached in only one place in the network. In a MANET, some nodes may join the network others may leave, while most nodes will change location. This makes some cached data unavailable or unaccessible for nodes in the network. Replicating cached data in the network will hence improve data accessibility and availability, and lowers access delays by reducing costly requests to servers behind the MANET.

Nevertheless, incorporating replication into a caching system is far from being an easy problem, as many challenges arise. The main one is deciding on the best allocation scheme that assigns data items to nodes, noting that in a MANET, there is no central entity that does the decision making. Moreover, the allocation strategy must adapt to the varying conditions of the network to provide acceptable quality of service, reflected by data availability and consequently low access delays. In this work, we propose a replication scheme on top of the COACS caching architecture [1], where data is cached in nodes and indexed by a small set of nodes that act as directories. The proposed work replicates the directory data and actual data items according to a cost minimization model.

II. RELATED WORK

Replication has its roots in distributed and parallel databases, where the environment is stable and allocation is done beforehand. It did not take researchers long to realize its benefits in wireless networks. To start with, the work in [3] proposes three different allocation strategies that consider memory limitations on mobile devices. Four parameters are considered for optimal allocation: item access frequency, probability of a node joining the network, probability of a link failure, and the probability of link formation. If a node can afford *C* items in its memory besides its original item, it stores the *C* highest access frequency items such that duplicate items between neighboring nodes are deleted. In each set of connected nodes, the node which has not been a coordinator with the lowest index is chosen to manage the replicas with its neighbors. This method relies on fixed relocation periods, which, as we explain later, cannot adapt fully to the dynamics of mobile networks. In [17] a fidelity aware replication scheme for personal devices is described. A middleware is proposed to replicate data between powerful and weaker devices where data is replicated with lower fidelity on weaker devices, like phones. In [16], on the other hand, replication allocation is tackled through a distributed solution to an optimization problem while adapting to network dynamics. However, the authors do not consider replica management or node disconnections.

In a second class of approaches, replication schemes try to predict network partitions and react proactively by replicating data to ensure data availability in all partitions. In this class, there are centralized and distributed approaches. The former include the works of [2], [10], and [11]. In [2], the authors propose to cluster nodes into groups, where nodes in a group move with a velocity that slightly varies from their mean velocity. Server nodes hold the data and are chosen from stable nodes at regular intervals. They gather information about the location and velocity of the nodes using GPS or signal strength (thus constituting major privacy concerns), and apply a clustering algorithm to predict network partitions and replicate the data accordingly. In [10], the authors define a reliability metric for the link depending on the number of disjoint paths between the node and the server. If this metric falls beneath a defined threshold, the data is replicated from the server to the node. The focus of this work is on the partition prediction, and little focus is given to the actual process of replication. Finally, the scheme in [11] constructs a directed acyclic graph of the nodes, rooted at a data server, and classifies the links from the nodes to the server as weak or strong. If a node does not have

978-1-4673-0921-9/12/$31.00 ©2012 IEEE

any strong path with the server, it decides that it is about to disconnect, and replicates the server data.

It is worth noting that there have been little attempts to incorporate replication into caching architectures, which is the focus of this paper. Caching architectures differ from most of the cited literature in that the data source is external to the network, and data is not updated from within the network. Architectures similar to the one in [12] used replication inside the network, but assumed one hop communication which is not applicable to MANETs. In most of the caching literatures related to multi-hop MANETs, multiple replicas would exist as a byproduct of caching, but not exploited in a fully pledged replication technique. In [15], the authors explore a game-theoretic approach to decide whether a node caches a replica or not, however, issues related to node disconnections are not addressed. One exception is the work in [12], where replication is implemented through a cache admission protocol. This protocol is controlled by a value that sets the minimum distance between the replicas. When this value is small, the access delay decreases but at the expense of hindering data availability.

III. Replication Scheme Design

The system consists of a MANET in which nodes have interest in certain data generated at an external data source (server) that is connected to the MANET via a gateway through a wired network. The data exchanged in the system is abstracted by data items representing answers to queries, and might be webpages, files, SQL query replies, etc. An overview of the proposed system is shown in Figure 1. It builds on top of COACS [1], and offers data replication services.

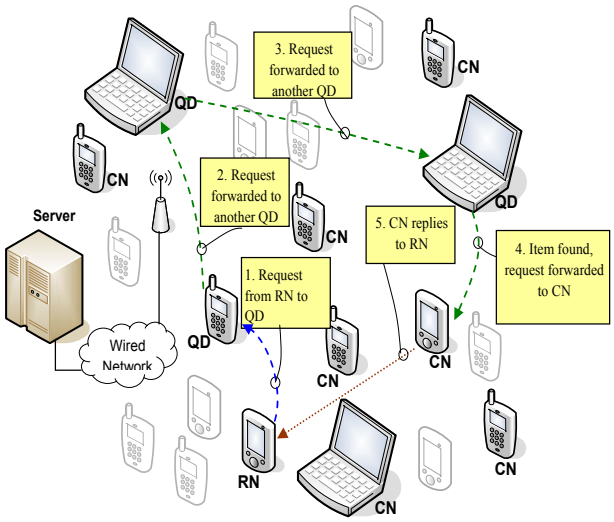

Figure 1. Overview of basic COACS operations

To facilitate nodes access to data items, requesting nodes (RNs) cache requested items if not found in the network, and subsequently become Caching nodes (CNs). A small set of nodes, called Query Directories (QDs) index the queries whose answers (i.e., data items) are hosted at the CNs, along with the addresses of these CNs. QDs are elected based on an algorithm that incorporates computing a score based on the capabilities of the node, basically, its memory, battery, bandwidth, and uptime. Any time a QD goes offline, or its score falls below a threshold, it gets reassigned. COACS is flat in the sense that there is no strict clustering, as it relies on distributed indexing of cached queries by the virtue of QDs, which act as distributed indices for the cached information, and are the core of the caching system. The major enhancement that QDs introduce is that the request of a node traverses the QDs sequentially from nearest to farthest rather than the whole set of CNs. The number of QDs in a network is much less than the number of CNs, and thus a request might hop several QDs before the query is found, upon which the request is forwarded to the corresponding CN, which in turn replies directly to the requesting node.

In in our two-layer approach, in the top layer the QD replicates its directory (its indexes) to other QDs in the system, while, in the second layer the actual items stored in the CNs are replicated. Below is a description of each layer.

A. Directory Allocation Strategy

The system caches data items from an external data source, and then attempts to place replicas optimally in the system. The replication is not performed for every cached item and surely is not performed directly upon caching. This is because placing replicas optimally requires access information that is not available when an item is first cached. A QD waits until there are sufficient requests for the item to decide on replicating this item. The replication strategy aims to decrease the access delay of the data items by placing and indexing the items closer to their requesting nodes. In fact, searching the QD system constitutes the major part of the access delay, since the query may traverse one or more QDs till it finds a match.

In our setting, we consider a set of N data items D = $\{d_1, d_2, \ldots, d_N\}$, and a QD system consisting of K nodes $Q = \{q_1, q_2, \ldots, q_K\}$. The allocation problem involves finding the "optimal" distribution of D to Q. We define optimal allocation, as the placement that results in the minimal cumulative cost of indexing each d_i at QD q_j, querying d_i, and data communication. We ignore the cost of updating in this stage, as we assume that updates occur from a data source outside the network, so that all QDs will experience the same update cost on average. We formulate the problem by considering a single data item, d_k and making some assumptions and definitions:

- Suppose d is accessed by a number of RNs, so we define the set of RNs that access this items as $R = \{r_1, r_2, \ldots, r_M\}$, and the associated access frequencies as $F = \{f_1, f_2, \ldots, f_M\}$.
- If the communication cost between two nodes is defined by the number of hops between them, we define the communication cost between an RN r_i and a QD q_j as c_{ij}.
- Let the cost of storing the index of a data item at a QD q_i be s_i. We can define S = $\{s_1, s_2, \ldots, s_K\}$ for the storage cost of the index of item d_k at the QDs. This cost incorporates the memory constraints of the QD, and may incorporate in the future a measure on how much the node is willing to cooperate, or how much trustworthy it is.

The allocation problem can then be specified as a cost minimization problem to determine the set of QDs where the copies (replicas) of the data items will be stored. In the following, x_j denotes the decision variable for the placement:

$$x_j = \begin{cases} 1 & \text{if the data item d}_K \text{ is assigned to QD } q_j \\ 0 & \text{otherwise} \end{cases}$$

The precise specification of the cost minimization is as follows:

$$\min\left[\sum_{i|r_i \in R} f_i \times \left(\min_{x_j \neq 0} c_{ij}\right) + \sum_{j|q_j \in Q} x_j s_j\right]$$

The above expression states that if each RN communicates with the QDs that store the data of interest (i.e. where x_j is not zero) we have the minimum cost for the data access (*one* of the copies), and the storage cost. The minimum cost of data access is basically the QD assignment that minimizes the number of hops from a requesting node to the closest QD that stores the index. This assignment takes into consideration the access frequencies of the item by each RN, which will give more weight to RNs requesting the item. The above formulation is NP-complete for large problems (i.e., large number of QDs). However, we consider only partial replication where we have a primary and secondary copy for each data item. The primary copy is the cached copy when the item was first requested, and so, the QD that has the index to the primary copy has the decision variable set to 1. The problem then reduces to finding another QD to store the index to the secondary copy. Having K QDs and M RNs, and x_j being not zero for only one value of j, the complexity of the problem reduces to $O(K \times M)$.

B. QD replica Allocation Mechanism

The mechanism that describes how replicas are first assigned is depicted in Figure 2. Initially, the QD system is built as described in [1]. Then, QDs start indexing items as they are requested, where each index consists of the query, and the address of the CN that stores the actual item in addition to other pertinent information. QDs monitor the requests for the items they index, and store the numbers of requests from each RN along with their IDs. The result is a table for each item indexed by the RN ids and has the counts of the requests.

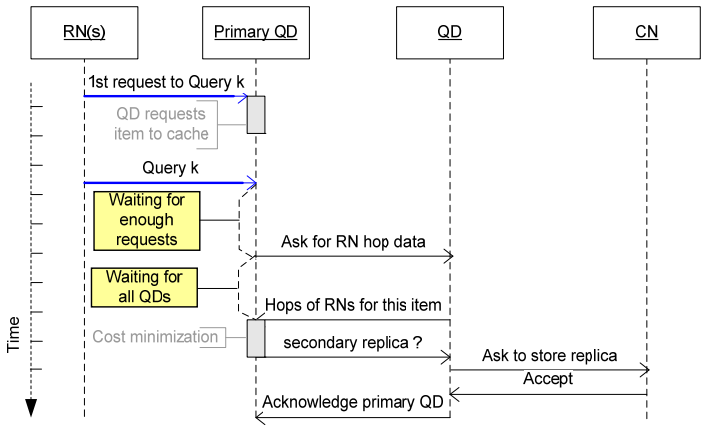

Figure 2. Sequence diagram describing secondary replica allocation

When the combined number of requests for a certain item is large enough, the QD replicates this item. At first, the QD marks the item as the primary copy, and asks the other QDs about their hop count to the RNs that have accessed the item. This enables the QD to compute the relevant c_{ij}. The QD next runs cost minimization and decides on another QD q_k to hold the secondary copy of the item's index. If q_k accept the assignment, it stores the index for the data item and marks it as secondary, and searches for a node to store the secondary replica for the actual data item.

C. Data Allocation Strategy

The second part of the replication scheme considers the replication of the actual data items. Similar to the QD replication, we have two replicas for each item. The primary copy is the one stored by the node that initially requested it. We address the allocation strategy of the replicas of the data items in a different manner than the index at the QDs. We do not apply cost minimization, as the number of caching nodes in the network is rather large, and the hop information from every RN to every CN must be known. Instead, we resort to a heuristic where the node chosen for the secondary replica is the closest CN to the secondary QD, meaning that when a QD is chosen to act as a secondary index for a data item, it chooses the nearest CN it knows about. Although the proposed method may not yield an optimal solution, we argue that it is suboptimal and more practical. To start with, it requires minimal computations and overhead traffic, as it does not entail information exchange. This issue is crucial when considering frequent relocations due to topology changes. Moreover, we claim that the solution is suboptimal, since the QD forwards all the requests through the CN to the RN. As a result, the majority of the cost is in the communication between the QD and the CN. This is why we choose the CN for holding the secondary copy of the data item to be the closest to the QD. Putting it all together, a request will have to traverse the QD system (partially or completely) to find a hit. Since there is another replica in the system, chosen using a minimal cost criterion, the RN is more likely to find a hit at a nearby QD. The QD then forwards the request to the closest CN, which in turn forwards the item to the RN.

D. Relocation strategy

A major part of any replication scheme is the relocation of the replicas. There are two main approaches to tackle this issue. In the first approach, allocation of replicas is examined each relocation period, whereas the other approach reacts to changes in the topology and relocates replicas accordingly. In our work, we use a different strategy, one that is based on the load of the secondary replica as compared to that of the primary. Moreover, we make a distinction between the relocation of the QD replicas (first stage) and CN replicas (second stage).

In the first stage, the QD checks if the allocation is achieving satisfactory results by computing the ratio between requests for each item at the primary and secondary QD since the last relocation. If this ratio is within a range, e.g., between 0.75 and 1.25, the QD deduces that the assignment is still satisfactory. In this scheme, if the mobility is low to medium, the network will tend to be stable over larger periods of time. When the ratio is outside the range, the QD can safely assume that a major topology change has taken place, and there is a need to relocate the data items. Periodically, QDs exchange small messages that consist of the IDs of the secondary data items they index along with the number of requests to each item since the last relocation. In particular, the QDs piggyback these messages to the item requests that traverse the QD system. Each

QD indexes a number of data items which are either primary or secondary. The QD sends only the secondary items it indexes as to reduce the size of the message. Consequently, each QD receiving these messages can determine which of the items in the message are in its memory. Since only secondary items are sent in the message, the QD that receives items it indexes but did not send is definitely the primary QD for that item. The QD next calculates the ratio for every item and decides on the items whose ratio is outside the range to be relocated. The relocation is performed in the same way that allocation is done, but with a small difference: if the ratio is below 0.5, the QD concludes that many more requests are being handled by the secondary QD. This prompts the primary QD to treat it as the primary while performing the cost minimization procedure. After this, the QD instructs the secondary QD to delete the associated indices, and sends them to the newly chosen QD.

In the second stage, the relocation of the actual items is done in reaction to the relocation of the indices. When this occurs, at least one QD must change with respect to the data item. In effect, each newly chosen QD checks the associated CN with the old QD and chooses a candidate CN according to the procedure described before. If the two CNs are no more than two hops away, the QD decides not to relocate the actual item. Else it relocates the item from the old CN to the new CN and updates the old QD about the newly chosen CN.

E. Node Disconnection management

As indicated before, the proposed system deals with node disconnections, by predicting them and appropriately reacting to maintain acceptable level of data availability. We assume that there are two main reasons for node disconnections: node isolation due to node mobility pattern, and battery depletion. In the context of the proposed system, QD and CN disconnections are of relevance, and in order to alleviate them, the system attempts to predict them and then react proactively. In fact, the system relies on the cooperative behavior of the QD and CN nodes to predict their own disconnections and react accordingly. That is, the CN and QD nodes are asked to communicate their anticipated disconnections. Nodes detect both types of disconnections as follows: a node can keep track of its battery level and energy consumption rate, and if it detects that the battery energy will be used up in the coming minutes, it determines that it is about to disconnect. For disconnection due to isolation, the node monitors its links with its neighbors by monitoring the received signal strength (RSS) from its neighbors, and if it detects that the RSS is decreasing, it decides that it may disconnect. The actions the nodes perform upon their anticipated disconnections depend on their in the system.

If the node that expects to disconnect is a CN, it chooses one of its first hop neighbors to take over. The disconnecting CN then moves its cache to the new CN and informs the QDs of this change. On the other hand, if the node is a QD, a first hop neighbor is elected to be a QD, using the algorithm in [1]. The QD then informs the QDs of this change, but not the associated CNs, simply because the CNs are not aware of which QDs index their cached items. In [1], the CNs have to be aware of the QDs that index their data so as to be able to reconstruct the QD cache after disconnections. This is no more needed as items and their indices are replicated in the network.

Predicting disconnections does not save the nodes from abrupt node disconnections that could not have been predicted beforehand. Here too, the behavior depends on the node's role. If it is a CN, reconstructing its cache is a costly task in terms of network traffic. When the disconnection could be predicted, transferring the contents between two neighboring nodes is not as costly as when it involves multi hop communication. So, in this case, the QD can update the new CN with items that were cached in the previous CN and have high request rates, and with other items that have their secondary index replica deleted from the QDs. On the other hand, if the node is a QD, the QD election algorithm is run on its first hop neighbors to choose a new QD for assuming the old QD's role. The new QD can reconstruct the old QD tables from the other QDs. Since all items are replicated, it issues a request to traverse the QDs that hold the old QD's ID. Each QD that receives this request sends the new QD the list of items that the old QD indexes along with relevant information (one of which if the item is secondary or primary). The elected QD uses the responses to reconstruct its table, and acknowledges all the QDs about its status.

At the end of this section, it is important to note that the QD system is used to construct the entries of the disconnected nodes since it is composed of a relatively small number of items, as opposed to the number of the CNs available. We also note that a first hop neighbor is always used to replace a disconnected node so as to have the resultant cost close to the original allocation cost, without resorting to a new allocation.

IV. PROVISIONS FOR CONSISTENCY CONTROL

This work focuses on the allocation of replicas along with their management. Nevertheless, we provide in this section provisions for replica consistency control. We address this problem as a two-layer problem. In the first layer, the system aims to insure consistency between the primary replica and the external data source by increasing the probability of serving from cached data items that are identical to those on the data source. In the second layer, we aim to propagate the updates to the secondary replicas in a bandwidth efficient manner.

The first layer problem can be viewed as a cache consistency problem. For this purpose, we use a TTL based algorithm on the QDs to ensure the consistency of the primary replica with the data source. In TTL based algorithms, a TTL value (e.g., T) is stored alongside each data item in the cache. The data item d is considered valid until T time units go by from the instant of cache update. Such algorithms are popular due to their simplicity, sufficiently good performance, and flexibility to assign TTL values for individual data items [4]. Also, they are attractive in mobile environments [5], and are considered suitable for MANETs because of device energy and network bandwidth limitations [6], and because of frequent device disconnections [4]. Moreover, TTL algorithms are completely client based and require little server functionality.

Hence, in the proposed system, each QD stores along with the primary replica of a data item a TTL value and monitors its expiry. When the item expires, it is marked, and a future request to this item will be forwarded to the data source, which in turn replies with an updated item or a "not-modified" message. The primary CN and the TTL values on the primary QD are next modified accordingly. This consistency control

mechanism between the cache in the MANET and the external source is referred to as weak consistency, as there is a probability that requests might be served with stale data items. On the other hand, we attempt to apply strong consistency control between the primary and secondary replicas by using a push based mechanism: whenever a QD detectes the expiration of a data item expired, it directly informs the QD holding the secondary replica. Of course, a QD might decide to wait a bit and piggyback all the expired items in one message to the QDs to save traffic, but this will decrease the consistency between the primary and secondary index replicas. Another option would be to have both QDs (primary and secondary) monitor the TTL values of the same data item, and invalidate it at the same time without a need for any communication. However, this requires clock synchronization between the nodes, which has its own overhead. As to the CN holding the secondary replica of the actual item, it is only updated when the request rate for the item is high. The primary CN is updated always from the data source when a request is made to the expired data item, while the secondary replica item is updated by the QD holding the secondary index if its request rate is high.

V. Experimental Results

A preliminary version of the proposed approach was implemented in ns2 [7] on top of the COACS implementation [1]. This version does not include the adaptation to the request rate, and hence, all items are replicated. Also, it includes only the replication of the QD data, not that of the CNs. The simulated topology consisted of a 750×750m^2 area, populated with 100 randomly distributed nodes. The propagation model is a two ray model, and the node's bitrate was set to 2 Mbps. A mobility pattern based on the random waypoint model was used, with a maximum speed of 2 m/s. The server node was connected to the MANET via a gateway and a wired link whose propagation delay was simulated at 40ms, thus resulting in a server access delay of 80ms. The server has 10,000 items. Each node issues a request every 10 seconds to a data item according to a Zipf access pattern, used frequently to model non-uniform distributions [8]. In Zipf law, an item ranked i ($1 \le i \le n_q$) is accessed with probability $1/\left(i^{\theta}\sum_{k=1}^{n_q} 1/k^{\theta}\right)$, where θ ranges between 0 (uniform distribution) and 1 (strict Zipf distribution). In the default scenario, there are 7 QDs, and the capacity for each of the CNs (Caching Nodes) is 200 Kb. The simulation parameters are summarized in Table 1.

Simulation Parameter	Default Value	Simulation Parameter	Default Value
Simulation time	2000 sec	Node pause time	30 sec
Network size	750×750 m^2	Total number of data items	10,000
Wireless bandwidth	2 Mb/s	Delay at the data source	40 ms
Node transmission range	100 m	Node request period	10 sec
Number of nodes	100	Node request pattern	Zipf distribution (θ=1)
Node mobility model	Random Way Point	Node caching capacity	200 KB
Node speed (v)	2 (m/s)	Cache Replacement	LRU

Table 1. Summary of the default simulation parameters

The reported results correspond to four experiments which involve varying the request rate, the zipf parameter, the maximum velocity, and the disconnection rate. The results for the first three experiments include only the cached data query delay (Figures 3, 4, and 5), and the reported the hit rate for the fourth experiment (Figure 6). In Figure 3, we plot the delay versus the request interval, which is varied between 6 and 120 seconds. It is obvious that the access delay is much less when the replication is applied, and this confirms that the proposed mechanism decreases the access delay by reducing the number of hops to find a match.

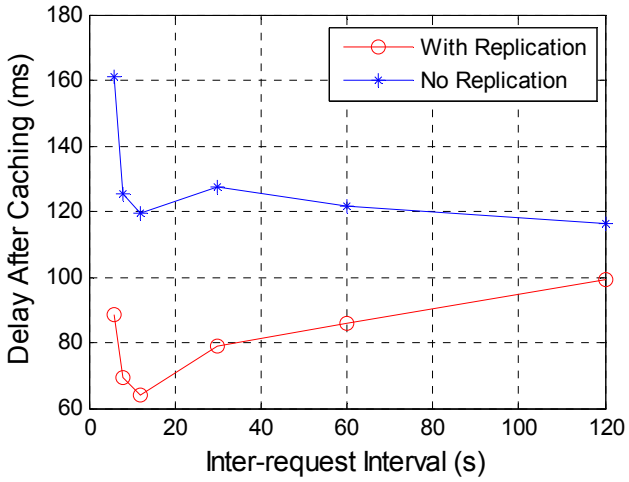

Figure 3. Access Delay versus the request interval

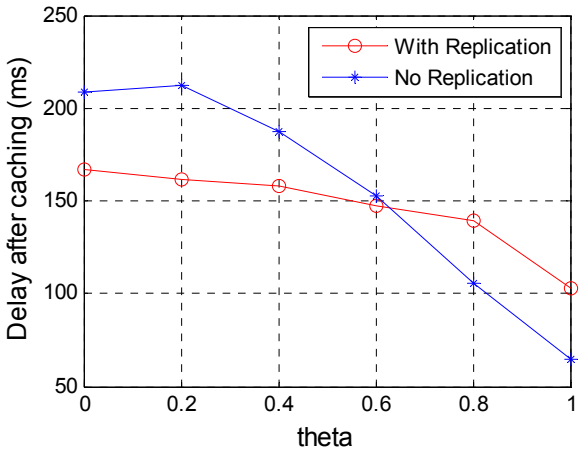

Figure 4. Access Delay versus the zipf parameter (θ)

Figure 4, on the other hand, shows the access delay versus the zipf parameter. For low values of theta, the access delay is higher in the case of replication. This is not a strange result given that the implementation does not adapt to the request rate. In fact, a lower value of theta signifies that there is a higher variety of the accessed elements. Given that the number of QDs is kept constant with and without replication, and since all items are replicated, this causes some of the items being evicted from the QD tables in accordance with the replacement method. As a result, maintaining two replicas is no more feasible, and the

delay increases. However, at high values of theta, the replication scheme's performance improves greatly and shows significant enhancement. It is worth noting that high values of theta are used in the literature to model web requests, as in the case of [9], where a value of 0.8 was selected. Next, Figure 5 shows that the proposed system reacts well to topology changes. The delay improvement is still significant for high values of node speeds under the RWP model.

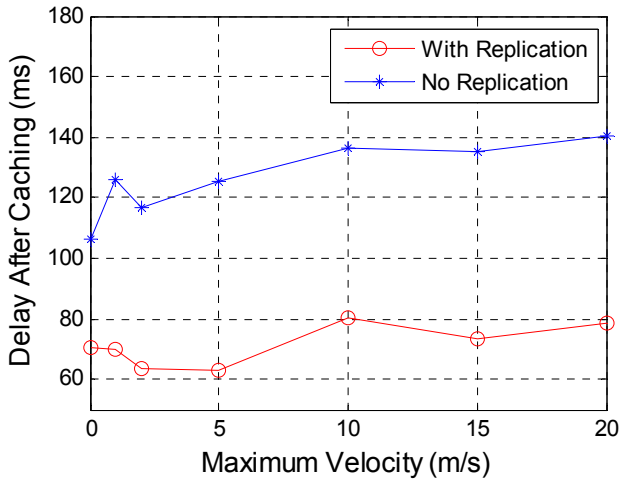

Figure 5. Access Delay versus the maximum velocity

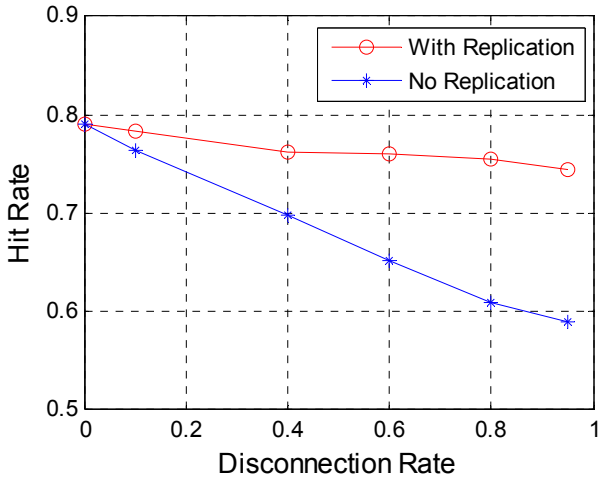

Figure 6. Hit rate versus the disconnection rate

Moreover, the hit rate as a function of the disconnection rate is shown in Figure 6. The disconnection rate ranges from 0 to 1, where a value of 0 signifies no disconnections, and the other values represent the portion of time the node is disconnected. In the simulations, disconnections are modeled through having the QDs go into cycles of sleep and wakeup each 100 seconds. The sleep time is 100 seconds multiplied by the disconnection rate. In these preliminary simulations, the CNs do not get disconnected and this is the reason why the hit rate is not that low (the lowest value is 0.59 without replication). However, the replication scheme improves the hit rate, and keeps it approximately constant.

VI. Conclusion and Future Work

In this work, we presented a data replication scheme that operates on top of the COACS caching architecture. The replication scheme allocates only two replicas of the actual data item and its index at the directory. The latter is chosen according to a cost minimization model. A preliminary prototype of the system was simulated in ns2, and enhancements in terms of access delay and hit rates are reported over the replication free version of the system. In the future, we will simulate a complete prototype and compare it to existing literature methods to show further improvements. Moreover, we will further develop the consistency control scheme, and compare it with existing methods.

References

[1] H. Artail, H. Safa, K. Mershad, Z. Abou-Atme, N. Sulieman, "COACS: A Cooperative and adaptive caching system for MANETS", *IEEE TMC*, v. 7, n. 8, pp. 961-977, 2008.

[2] K. Wang, B. Li. "Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks," *IEEE INFOCOM*, 2002.

[3] T. Hara, S. Madria: "Data replication for improving data accessibility in ad hoc networks," *IEEE TMC*, v.5, n.11, pp.1515-1532, 2006.

[4] X. Tang, J. Xu, W-C. Lee, "Analysis of TTL-based consistency in unstructured peer-to-peer networks," IEEE TPDS, v. 19, n. 12, pp.1683-1694, 2008.

[5] L. Bright, A. Gal, L. Raschid, "Adaptive pull-based policies for wide area data delivery," *ACM TDS*, v. 31, n. 2, pp. 631 – 671, 2006.

[6] L. Yin, G. Cao, "Supporting cooperative caching in ad hoc networks," *IEEE TMC*, v. 5, n. 1, pp. 77- 89, 2006.

[7] NS-2 Simulator, http://nsnam.isi.edu/nsnam/index.php/User_Information, Feb. 2010.

[8] G. Zipf, Human Behavior and the Principle of Least Effort. Addison-Wesley, 1949

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," *INFOCOM '99*, pp.126-134 v.1, Mar 1999

[10] M. Hauspie, D. Simplot, J. Carle. Replication decision algorithm based on link evaluation for services in manet. Technical Report 2002-05, IRCICA/LIFL, Univ. Lille 1, 2002.

[11] A. Derhab, N. Badache, A. Bouabdallah. "A partition prediction algorithm for service replication in mobile ad hoc networks," *WONS'2005*, pp, 236–245, January 2005.

[12] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies for Mobile Environments," *ACM SIGMOD*, pp. 1-12, May 1994.

[13] G. Cao; L. Yin; C. Das, "Cooperative cache-based data access in ad hoc networks," *Computer*, v. 37, n. 2, pp. 32-39, 2004

[14] M. Denko, J. Tian, "Cooperative Caching with Adaptive Prefetching in Mobile Ad Hoc Networks," *IEEE WiMob'2006*, pp.38-44, June 2006.

[15] Hirsch, D.; Madria, S.; , "A cooperative game theoretic approach for data replication in mobile ad-hoc networks," *CollaborateCom'2011*, pp.115-124, Oct. 2011

[16] La, C.-A.; Michiardi, P.; Casetti, C.; Chiasserini, C.-F.; Fiore, M.; , "A Lightweight Distributed Solution to Content Replication in Mobile Networks," *WCNC'2010*, pp.1-6, April 2010

[17] Ramasubramanian, V.; Veeraraghavan, K.; Puttaswamy, K.P.N.; Rodeheffer, T.L.; Terry, D.B.; Wobber, T.; , "Fidelity-Aware Replication for Mobile Devices," *IEEE TMC*, v.9, n.12, pp.1697-1712, Dec. 2010