IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 5, Oct-Nov, 2013
ISSN: 2320 - 8791
www.ijreat.org

FEASIBLE MECHANISM FOR BOOSTING THE DATA ACCESS BY PUSH BASED CONSISTENCY & EFFICIENT REPLICATION TECHNIQUE IN MANET

Nithya Preya.S[1] and Dr. V.L.Jyothi[2]

[1]M.E, C.S.E, Jeppiar Engineering College, Chennai, Tamil Nadu, India

[2]C.S.E, Jeppiar Engineering College, Chennai, Tamil Nadu, India

Abstract

Data caching is essential in a mobile ad hoc network (MANET) as it reduces contention in the network, increases the probability of nodes getting the desired data, and improves the system performance. The major issue that cache management faces is the maintenance of data consistency between the client cache and the server. All messages sent between the server and the cache are subject to network delays and download delays that are considerably noticeable and more severe in wireless mobile devices. In order to cope up with these cache inconsistencies, we propose a cache consistency scheme for caching database data in MANETs. In this method for data caching, the queries that are submitted by requesting nodes are stored in special nodes, called query directories and the response data is stored in the requested nodes, called caching nodes. The technique uses query directories to locate the data (responses) from caching nodes. In this server-based scheme, control mechanisms are implemented to adapt the process of caching a data item and updating the cache according to its popularity and its data update rate at the server. The proposed server based approach intends to solve many issues associated with traditional push-based cache consistency approaches and reduce wireless traffic by tuning the cache update rate to the request rate for the cached data. Moreover MANETs are more susceptible to attacks. The security and cache consistent scheme presented in this paper mitigates security attacks in query nodes of server control cache mode in mobile networks. It eliminates fake requests in query nodes. Furthermore to enhance the boost the data access over the network, the data replication is also implemented along with the consistency. The replication is done for the most requested data item for feasible data access results. The results are implemented using the NS2 simulator and evaluation is done based on the analysis.

Keywords— Data caching, cache consistency, invalidation, server-based approach, MANET.

1. INTRODUCTION

In a mobile ad hoc network (MANET), data caching is essential as it reduces contention in the network, increases the probability of nodes getting desired data, and improves system performance . The major issue that faces cache management is the maintenance of data consistency between the client cache and the server. As mobile ad hoc network is becoming increasingly widespread, the need for developing methods to improve their performance and reliability increases. One of the biggest challenges in mobile ad hoc network s lies in the creation of efficient routing techniques, but to be useful for applications that demand collaboration; effective algorithms are needed to handle the acquisition and management of data in the highly dynamic environments of mobile ad hoc networks.

In many scenarios, mobile devices or nodes may be spread over a large area in which access to external data is achieved through one or more access points. However, not all nodes have a direct link with these access points. Instead, they depend on other nodes that act as routers to reach them. In certain situations, the access points may be located at the extremities of the mobile ad hoc network, where reaching them could be costly in terms of delay, power consumption, and bandwidth utilization.

Additionally, the access points may connect to a costly resource for example a satellite link or an external network that is susceptible to intrusion. For such reasons and others dealing with data availability and response time, caching data in mobile ad hoc network s is a topic that deserves attention.

Mobile ad hoc networks are dynamic in nature, and therefore, a reliable caching scheme is more difficult to achieve. Links between nodes may constantly change as nodes move around, enter, or leave the network. This can make storing and retrieving cached data particularly difficult and unreliable.

The use of mobile devices adds even more complexity due to their relatively limited computing resources for example the processing power and storage capacity and limited battery life. It follows that an effective caching system for mobile ad hoc network s needs to provide a solution that takes all of these issues into consideration. Here the client nodes have sufficient resources to cache portions of the database as well as storing some data base management system query and processing modules. An important policy of such a solution is not to rely on a single node but to distribute cache data and decision points across the network.

In 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. Other issues with caching are that caching mechanisms are 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 favours 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.

This work describes a server-based scheme implemented on top of the COACS caching architecture we proposed in [1].In COACS, elected query directory (QD) nodes cache submitted queries and use them as indexes to data stored in the nodes that initially requested them (CN nodes). Since COACS did not implement a consistency strategy, the system described in this paper fills that void and adds several improvements: 1) enabling the server to be aware of the cache distribution in the MANET, 2) making the cached data items consistent with their version at the server, and 3) adapting the cache update process to the data update rate at the server relative to the request rate by the clients. With these changes, the overall design provides a complete caching system in which the server sends to the client's selective updates that adapt to their needs and reduces the average query response time.

2. IMPROVED SERVER UPDATE MECHANISM

It is a server-based approach that avoids many issues associated with push-based cache consistency approaches. Specifically, traditional server-based schemes are not usually aware of what data items are currently cached, as they might have been replaced or deleted from the network due to node disconnections. Also, if the server data update rate is high relative to the nodes request rate, unnecessary network traffic would be generated, which could increase packet dropout rate and cause longer delays in answering node queries. This mechanism reduces wireless traffic by tuning the cache update rate to the request rate for the cached data.

2.1 Basic Operations

Given that no consistency mechanism was implemented in COACS, it was necessary to introduce four additional messages. In this mechanism, the server autonomously sends data updates to the CNs,

meaning that it has to keep track of which CNs cache which data items. This can be done using a simple table in which an entry consists of the id of a data item (or query) and the address of the CN that caches the data. A node that desires a data item sends its request to its nearest QD. If this QD finds the query in its cache, it forwards the request to the CN caching the item, which, in turn, sends the item to the requesting node (RN). Otherwise, it forwards it to its nearest QD, which has not received the request yet. If the request traverses all QDs without being found, a miss occurs and it gets forwarded to the server which sends the data item to the RN. In the latter case, after the RN receives the confirmation from the last traversed QD that it has cached the query, it becomes a CN for this data item and associates the address of this QD with the item and then sends a Server Cache Update Packet (SCUP) to the server, which, in turn, adds the CN's address to the data item in its memory. This setup allows the server to send updates to the CNs directly whenever the data items are updated. Fig. 1 illustrates few data request and update scenarios that are described below.

In the figure, the requesting nodes (RNs) submit queries to their nearest QDs, as shown in the cases of RN1, RN2, and RN3. The query of RN1 was found in QD1, and so the latter forwarded the request to CN1, which returned the data directly to the RN. However, the query of RN2 was not found in any of the QDs, which prompted the last searched (QD1) to forward the request to the server, which, in turn, replied to RN2 that became a CN for this data afterward. The figure also shows data updates (key data pairs) sent from the server to some of the CNs.

2.2 Dealing with Query Replacements and Node Disconnections

A potential issue concerns the server sending the CN updates for data that have been deleted (replaced), or sending the data out to a CN that has gone offline. To avoid this and reduce network traffic, cache updates can be stopped by sending the server Remove Update Entry Packets (RUEPs). This could occur in several scenarios. For example, if a CN leaves the network, the QD, which first tries to forward it a request and fails, will set the addresses of all queries whose items are cached by this unreachable CN in its cache to -1, and sends an RUEP to the server containing the IDs of these queries. The server, in turn, changes the address of that CN in its cache to -1 and stops sending updates for these items. Later, if another node A requests and then caches one of these items, the server, upon receiving an SCUP from A, will associate A with this data item. Also, if a CN runs out of space when trying to cache a new item in, it applies a replacement mechanism to replace id with in and instructs the QD that caches the query associated with id to delete its entry. This causes the QD to send an RUEP to the server to stop sending updates for id in the future.

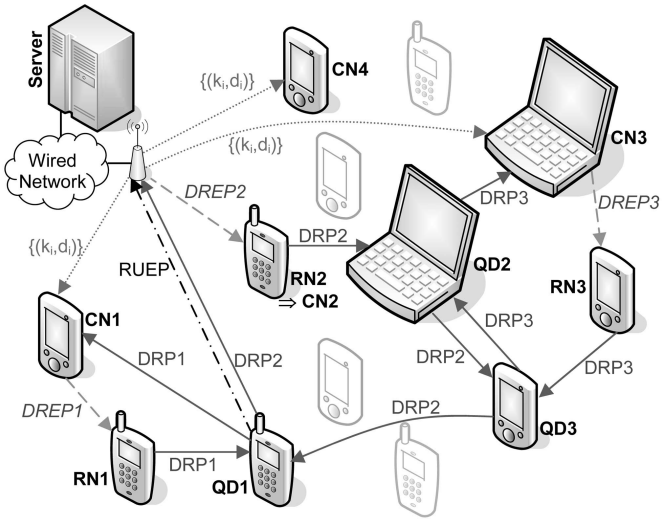

Fig. 1. Scenarios for requesting and getting data in the COACS architecture.

If a caching node CNd returns to the MANET after disconnecting, it sends a Cache Invalidation Check Packet (CICP) to each QD that caches queries associated with items held by this CN. A QD that receives a CICP checks for each item to see if it is cached by another node and then sends a Cache Invalidation Reply Packet (CIRP) to CNd containing all items not cached by other nodes. CNd then deletes from its cache those items whose IDs are not in the CIRP but were in the CICP. After receiving a CIRP from all QDs to which it sent a CICP and deleting nonessential data items from its cache, CNd sends a CICP containing the IDs of all queries with data remaining in its cache to the server along with their versions. In the meanwhile, if CNd receives a request from a QD for an item in its cache, it adds the request to a waiting list. The server then creates a CIRP and includes in it fresh copies of the outdated items and sends it to CNd, which, in turn, updates its cache and answers all pending requests.

Finally, QD disconnections and reconnections do not alter the cache of the CNs, and hence, the pointers that the server holds to the CNs remain valid.

2.3 Adapting to the Ratio of Update Rate and Request Rate

The mechanism suspends server updates when it deems that they are unnecessary. The mechanism requires the server to monitor the rate of local updates, Ru, and the rate of RN requests, Rr, for each data item di. Each CN also monitors these values for each data item that it caches. Whenever a CN receives an update from the server, it calculates Ru=Rr and compares it to a threshold Ґ. If this ratio is greater than or equal to Ґ, the CN will delete di and the associated information from its cache and will send an Entry Deletion Packet (EDP) to the QD (say, QDd) that caches query qi. The CN includes in the header of EDP a value for Ru, which tells QDd that di is being removed due to its high update-torequest ratio. Normally, when a QD gets an EDP, it removes the cached query from its cache, but here, the nonzero value of Ru in the EDP causes QDd to keep the query cached, but with no reference to a CN. Next, QDd will ask the server to stop sending updates for di. Afterward, when QDd receives a request from an RN node that includes qi, it forwards it to the server along with a DONT_CACHE flag in the header to be later passed in the reply, which includes the results, to the RN.

Under normal circumstances in COACS, when an RN receives a data item from the server in response to a query it had submitted, it assumes the role of a CN for this item and will ask the nearest QD to cache the query. The DONT_CACHE flag instructs the RN to treat the result as if it were coming from the cache and not become a CN for it. Now, at the server, each time an update for qi occurs and a new Ru=Rr is computed, if this ratio falls below a second threshold, the server will reply to the RN with a Data Reply Packet (DREP) that includes the CACHE_NEW flag in the header. Upon receiving the DREP, the RN sends a Query Caching Request packet (QCRP) with the CACHE_NEW flag to its nearest QD. If this QD caches the query of this item (with -1 as its CN address), it sets its address to its new CN, else it forwards the request to its own nearest QD. If the QCRP traverses all QDs without being processed (implying that the QD caching this item has gone offline), the last QD at which the QCRP arrives will cache the query with the CN address.

By appropriately selecting the values of Ґ and ƛ, the system can reduce unnecessary network traffic. The processing time of qi will suffer though when Ru=Rr is above ƛ after it had passed Ґ since QDd will be sending qi to the server each time it receives it. However, the two thresholds allow for favoring bandwidth consumption over response time, or vice versa. This makes the proposed mechanism suitable for a variety of mobile computing applications: a large Ґ may be used when disconnections are frequent and data availability is important, while a low Ґ could be used in congested environments where requests for data are infrequent or getting fresh data is not critical. Fig. 2 summarizes the interactions among the entities of the system.

2.4 Accounting for Latency in Receiving Server Updates

Given the different processes running at the server and since it sends the updates to the CNs via unicasts, there may be a time gap between when an update occurs and when the CN actually receives the updated data item d. Hence, if the CN gets a request for d during this time, it will deliver a stale copy of d to the RN. Our design uses the time stamp that the server sends with each update in an attempt to mitigate this issue. To explain this, suppose that the time stamp sent with d is ts and the time of receiving d by the CN is tc. Upon getting an update, the CN checks if it had served any RN a copy of d from its cache in the past ts-tc milliseconds. If it is the case, the CN sends a new DREP to the RN, but now it includes the fresh copy of data d.

The above solution assumes that the clocks of the nodes in the MANET and that of the server are synchronized. This assumption is realistic given that node clock synchronization is part of the MAC layer protocol, as specified by the IEEE 802.11 standards [8]. In particular, IEEE 802.11 specifies a Timing Synchronization Function (TSF) through which nodes synchronize their clocks by broadcasting their timing information using periodic beacons. Since the Access Point (AP) is considered a node in the MANET and it can synchronize its clock with that of the server asynchronously with respect to the MANET through the wired network, it will be possible to synchronize the clocks of the mobile nodes with that of the server at almost a zero cost to them (no protocol besides the MAC layer's TSF is needed).

3. SECURED CACHE NODES FOR CONSISTENT SERVER UPDATE MANET

The nodes in the mobile ad hoc have cooperative cache. Elected query directory (QD) nodes cache submitted queries. Cache nodes (CN nodes) store

indexed data initially requested. Cooperative server cache maintains table containing id of a data item (or query) and address of the CN that caches the data. Node needing a data item sends its request to its nearest QD. If QD finds in its cache, it forwards to CN, CN turn sends the item to requesting node (RN). Otherwise, it forwards it to its nearest QD, if the request traverses all QDs without being found, a miss occurs and it gets forwarded to the server, then server sends the data item to the RN. The cooperative cache adopts a widely accepted system model in which each data object is associated with a single node that can update the source data. This node is referred to as the data source node. Each data object can be cached by a collection of nodes called the caching nodes. The data copies held by the caching nodes are called the cache copies. There are two basic mechanisms for cache consistency maintenance i.e., push and pull. Using push, the data source node informs the caching nodes of data updates. Using pull, the caching node sends a request to the data source node to check the update. In designing cooperative cache the source data updates and the cache queries follow the Poisson Process. The routing protocol employed in the network layer provides the hop count between each pair of nodes, and the hop count of data transmission is used to measure the consistency maintenance.

3.1 Consistent Server Cache Updates

The consistent server cache updates provides data Consistency based on the Pull with TTR. In Pull with TTR, each cache copy is associated with a timeout value Time to Refresh (TTR). The initial value of TTR is set to d. When TTR is valid (TTR > 0), the caching node directly serve cache queries. When TTR expires, the caching node first pulls the data source node to update the cache copy and to renew TTR to d. Then the caching node can directly serve cache queries. By associating a TTR value with each cache copy, GWSP guarantees that deviation between the source data and the cache copy will not be over d, thus ensuring data Consistency. Although the Pull with TTR algorithm guarantees data Consistency, it is not cost effective, mainly due to the round–trip consistency maintenance cost imposed by the pull mechanism.

3.2 Selective Push

Using push, the data source node informs the caching nodes of data updates, which only imposes one way consistency maintenance cost (traffic overhead, query latency etc.). However, if the data source node is unaware of the cache status of each cache copy, there exist redundant data update propagations in the following two cases i.e., After the cache copy and the associated TTR are renewed via push, there comes no cache query before the TTR expires. Before the TTR expires, there are multiple data updates (only the last data update should make the data source node push the caching nodes); Thus, the following design principles should be followed in designing GWSP: use the push mechanism to save consistency maintenance cost, but Push only if the cache copy is expected to serve queries, Push if there are probably no other data updates before TTR expires, Upon the source data update at tu, consider the cache status on one aching node. The TTR of this cache copy was renewed (via push or pull) at time t0. According to the design principles, the data source node pushes one caching node only when the following two requirements are both satisfied. The first one is the probability that there is at least one query in period (t0+ä, tu+ä) is greater than threshold value Γ. The second one is the probability that there is at least one update in period (tu, t0+d) is less than threshold value Γ.

3.3 Data Update Propagation

After the data source node has decided the Push Set, i.e., the caching nodes which should receive the data update, it needs to decide how to propagate the data updates among the selected caching nodes. The mechanism employs a greedy but efficient strategy to disseminate the PUSH message (which contains the source data update and IDs of the push set nodes) to all caching nodes in the push set. The data source node sends the PUSH message to the nearest caching node in the push set via unicast.

The caching node which receives the PUSH message deletes itself from the push set. It acknowledges the PUSH message with a PUSH_ACK message, and then goes on relaying the push message to the nearest caching node in the push set (The path length between two nodes are provided by the routing protocol, according to our assumption). This process is repeated until the push set is empty. The last caching node will send a PUSH message to the data source node, which initiates the PUSH message propagation process.

3.4 Attack Vulnerabilities in Query Nodes

MANET with low-overhead made complex to monitor an environment, some of their attributes make them even more susceptible to security attack or damage. A device with scarce resources is at risk of resource consumption under normal circumstances. When an

adversary is actively attempting to improperly consume or destroy its resources, the situation is worse. Other attributes are remote location, i.e., networks that are distant or unmonitored have a greater response time if manual (physical) intervention is required.

Large scale, due to the large number of devices likely to be deployed, manual intervention on each device is not feasible within cost constraints. If vulnerability is discovered and exploited in the program code, it will be no small matter to physically collect, reprogram, and redeploy each device. Application specificity, resource constraints dictate that well-defined and uncoupled network layers are compressed or merged, reducing code modularity. Unforeseen interactions between network layers and services may give rise to new vulnerabilities. Attractive target, the systems monitored or controlled is safety-critical or highly visible, with significant consequences for failure. Depending on the motivation of the attacker, this may be precisely the profile of an attractive target. To achieve a better understanding of the risks faced by the network, explain taxonomy of security attacks against cache nodes of the MANET.

The taxonomy allows us to reason about attacks at a level higher than a simple list of vulnerabilities. It provides a classification system that ideally suggests ways to mitigate attacks by prevention, detection, and recovery. It can aid risk management by identifying vulnerabilities and making attacker characteristics explicit. The attacker has an identity and a motive, and is able to do certain things in or to the MANET. An attack targets some service or layer, exploiting vulnerability. An attack may be thwarted, or it may succeed with varying results. Each of these elements is necessary to understand the whole process of security attacks of cache nodes.

The two types of security attacks are passive and active. In passive attacks, selfish nodes use the network but do not cooperate, saving battery life for their own communications: they do not intend to directly damage other nodes. In Active attacks, malicious nodes damage other nodes by causing network outage by partitioning while saving battery life is not a priority.

Firewall Mechanism to Thwart Security Attack

In security attacks, the hacker's objective is to render target cache nodes inaccessible by legitimate users. MANET without sufficient protection from security attacks may not be deployable in many areas. Apart from special cases whereby an a priori trust exists in all nodes, the nodes of an ad hoc network cannot be trusted for the correct execution of critical network functions. Essential network operations that assure basic connectivity can be heavily jeopardized by nodes that do not properly execute their share of the network operations like routing, packet forwarding, name-to address mapping, and so on. Node misbehavior that affects these operations range from simple selfishness or lack of collaboration due to the need for power saving to active attacks aiming at security attacks and subversion of traffic are safeguarded by the firewall.

4. BASIC FEATURES OF THE SYSTEM

Follwing are the important features of the system,

4.1 Request Handling

The request from the client i.e from the request Node(RN) is handled by passing over the query to the Query directory, which processes further to the cache node or server. Hence this feature functionalities involved in Query Directory (QD), Caching Node (CN), Requesting Node (RN) and Server.

4.2 Network Monitoring

This feature plays a major functionality over the network nodes. Whenever a node goes offline or fails to respond, to avoid congestion and delay the network is monitored periodically. The monitoring also enables the cache CN for maintaining and replacing the stale data.

4.3 Traffic Maintenance

When there is a delay in the data response the request rate of the data becomes higher due to repeated queries, To avoid congestion of queries tuning of the update rate and request rate of each data item are made simultaneously between the CN and server.

4.4 Election of QD & CN

Over a large area network, based on the statistical analysis of successful data transmissions and of high battery power and powerful efficiency the Query directory QD is chosen initially as soon as the network is initiated by a node. Whenever the node gets the requested response from the server it become the

official owner of the data until updation of data, the requested node itself serves thereafter the caching Node CN for that data item. This data item information is sent to the QD, so when other RN request data the data is cached from the current CN.

4.5 Cache Replication

Let us consider a situation of if the request rate for the particular data item is high, it results in numerous hits in a particular QD & CN for that particular data item. This might cause network congestion. One optimal solution for the above problem is that by replicating the cache.

To make the replication feasible for the data item, to avoid unnecessary data storage over the cache, the replication is done only for the data item whose request rate higher than the threshold value, those data items are distributed to more than one caching node in the network. The threshold value is set on the basis of the network tuning condition of the request rate and update rate. When the congestion in the network is high the threshold value is set to minimum, and for low congestion the we tune the threshold to the lower value. Usually for the normal networks i.e when there is no congestion, the threshold value is set as 0.50 for normal networks, while 0.75 for congests networks. While replicating the data the we must consider the following factors for optimal storage over the cache nodes over the network,

- The same QD should not contain the replication cache, since the current QD stores the information about the present caching node for that data in order to avoid the network congestion in that QD.

- The ratio Ru/Rr is calculated for each and every requested data item, if the ratio falls within the replication's threshold value then the data is replicated in a different QD.

- Once the ratio fall below 0.75 for a data item the caching node send the request to the server for cache replication and get the permission for the replication. When a new RN node requests for the same data item, then the Query caching request packet (QCRP) is sent to the new RN, and then cached by that node.

4.6 Data Replication

This feature is responsible to replicate the most requested data to the new cache node and provide the corresponding cached information to the nearest QD. Also this information is sent to the server to maintain the cache consistency.The also the enables to reduce the network contention.

The above features are responsible for the network functionalities based on which the entire system is built.

5. LITERATURE SURVEY

Several cache consistency (invalidation) schemes have been proposed in the literature for MANETS. In general, these schemes fall into three categories: 1) pull- or client-based, where a caching node (CN) asks for updates from the server, 2) push- or server-based, where the server sends updates to the CN, and 3) cooperative, where both the CN and the server cooperate to keep the data up-to-date. In general, pull-based strategies achieve smaller query delay times at the cost of higher traffic load, whereas push-based strategies achieve lower traffic load at the cost of larger query delays. Cooperative-based strategies tend to be halfway between both ends. In this section, we restrict our literature survey to server-based strategies so as to provide a comparative study in relationship to the approach of our proposed system.

5.1 Invalidation Reports

Server-based approaches generally employ invalidation reports (IRs) that are periodically broadcasted by the server. An IR normally carries the IDs of the updated data items and the time stamps of the updates. When a query is generated, the node waits for the periodic IR to invalidate its cache (if connected) and answer the query if it has a valid item. If the requested item is invalid, it usually waits for the periodic IR, or as in some proposed schemes, like the Modified Time Stamp (MTS) mechanism of [12], it broadcasts a request packet that gets forwarded to the server without waiting for the periodic IR. Such schemes generally suffer from large average delays due to waiting for the periodic IR or from high traffic in case broadcasts are employed when misses occur and the request rate is high.

Most research in this area has focused on reducing the time intervals between updates or making the process of sending update reports less static. In the improved IR technique that was presented by Cao [5] and which we implemented to compare the proposed system to, the time between two consecutive IRs was divided into intervals. At the beginning of each interval, the

server broadcasts a UIR, which contains the IDs of data items updated since the last IR. This UIR reduces the query latency time since a node that needs to answer a query only waits until the next UIR to see whether the item has been updated instead of waiting until the next IR (the UIR interval may be adjusted dynamically according to the average request rate of the network). This approach can also decrease generated traffic by making the server save a list of the submitted requests to each item after which it broadcasts this list followed by the updated data items.

This approach may be best suited to high update rate situations because it does not overwhelm the network with updates immediately upon their occurrence. Another improvement over the basic IR approach was proposed by Li et al. [11]. The basic idea is for each node to cache the last K reports that are broadcasted by the server, and for the server to store the IRs for future use. Using the time stamps of the IRs it caches, when a node goes offline and comes back online, it determines the IRs that it missed and pulls them from the server to validate its cache. If the node misses more than K IRs, it drops the entire cache. Several approaches attempted to make the process of sending reports dynamic [6], [12], [17]. Yuen et al. [17] proposed a scheme by absolute validity interval (AVI) that is calculated dynamically by the server based on the update rate. A client considers an item invalid after its AVI expires and may drop or update it from the server. The server periodically broadcasts an IR that contains the changes to AVIs of items to make clients caching these items update the cached AVI with its new value in the IR. Thus, the size and interval of the IR could be improved. This approach, however, still exhibits access delays introduced by periodic broadcasts. Similarly, Kai and Lu [10] proposed to broadcast the reports dynamically based on the updating frequency of the data items. Chung and Hwang [12] took this concept further by adapting the broadcast of reports to the update pattern and data locality on the server. A serious issue, however, arises with the above approaches when the frequency of updates at the server is high, which could overload the clients with update reports.

5.2 Energy-Efficient Cache Invalidation

A different approach was taken by Cai and Tan [3] in proposing three energy-efficient selective cache invalidation schemes based on the group invalidation method in which a separate IR is broadcasted for each group of items. These strategies allow clients to selectively tune to the portions of the invalidation report that are of interest to them, while returning to sleep or idle modes during the rest of the IR. This can be done by including at the beginning of the IR an index of the rest of the remaining data in the report. This technique produces good results when the update rate is high, but makes the size of the IR quite large.

5.3 Cache Invalidation Using Bit Sequences

In order to reduce the size of the IR, Jing et al. [9] proposed using bit sequences (BSs), according to which the data server broadcasts a sequence of bits with a time stamp, where each bit represents a data item, and its position in the sequence represents an index to its corresponding item. The clients get the mapping of bits to the names of data items from the server. A value of 1 in the BS means that the corresponding item has been updated since the last IR, and a value of 0 indicates otherwise. The authors improved on the BS algorithm in [7] through a multidimensional bit sequence. The granularity of each bit in the BS was varied according to the size of the database at the server. For example, for an 8 GB database, each bit can represent an 8 MB data block, which allows for representing the database with a BS of 1,024 bits. The client retrieves data from the database as data pages and updates each cached item that contains at least one page from an updated block (with corresponding bit equal to 1 in the BS). Moreover, the granularity of each bit was dynamically adjusted according to the popularity of the corresponding block of the database that it represents (whether it is hot or cold). All the BS algorithms perform best when the update rate is low, but suffer from drawbacks when the update rate is high, as was stated by the authors.

5.4 Selective Data Push

Huang et al. present in [14] a Greedy Walk-Based Selective Push (GWSP) method that attempts to reduce the redundancy in conventional push mechanisms by pushing updates only if the CN is expected to serve queries and if there are no more data updates before the Time-to-Refresh (TTR) expires. This is done by maintaining the state of the cached data at the data server, which stores the TTR value and the query rate associated with cached data. Based on this state information, the server selects the CNs to send the updates to and creates a push set that contains the CNs that should receive updates. This method is compared with a dynamic pull-mechanism in which the CN maintains a TTR for the cached data and pulls updates accordingly. Reported results show that

GWSP exhibits lower generated traffic load and lower query latency at low query rates, but tends to have similar delays at larger rates.

5.5 Consistency Based on Location

The consistency of location-dependent data was studied in [16], which proposes methods to validate and update information that change their values depending on the location of the client, like traffic reports and parking information. In this context, not only the temporal consistency but also the location-dependent consistency (due to client mobility) is taken into consideration. A data item can have different values for different locations: when a mobile host caching an item moves from one of the cells to another, the value of the item may become obsolete and needs to be updated. One of the methods proposed by the authors is the Implicit Scope Information (ISI) scheme in which the client stores the valid scopes of each item and the server periodically broadcasts the changes of validity of each data item. Finally, we conclude that besides the latency issue that is associated with the push-based techniques, their basic problem is that the IR or UIR reports typically contain information that is not of interest to a large number of nodes, thus costing them wasted processing time and bandwidth.

Even cooperative-based mechanisms, which try to combine the advantages of push and pull strategies, also inherit their disadvantages halfway. For example, the simulation results of the scheme in [9] show that both the query delay and traffic load fall between the measures of pure push and pull strategies. To our knowledge, no existing scheme actually adapts the frequency of update reports to both the rate of updates at the server and that of data requests by the clients, to render a truly dynamic system. Our work addresses both of the above two issues, which is made in part possible by the architecture of COACS.

6. EXPERIMENTAL RESULTS

The described system is very expensive to implement in reality, hence the system is implemented using NS2 simulator and results are verified accordingly,
The comparisons of the parameters show the performance analysis of the system.

1. Delay Vs Threshold Rate:

The results below show that the query delay is reduced correspondingly at different threshold levels,

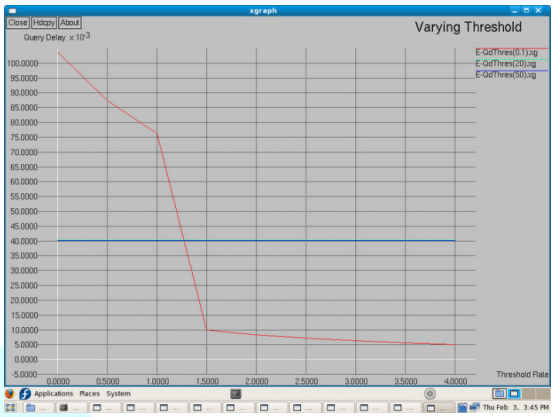

Fig 2

1. Update Delay Vs Threshold Rate

The results below show that the update delay is reduced correspondingly at different threshold levels,

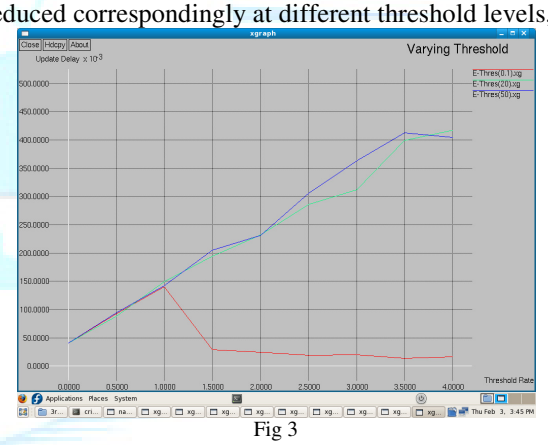

Fig 3

2. Update Delay Vs Number of Nodes

The results analysis results in efficient even if the number of nodes increase the update delay decreases in a constant manner.

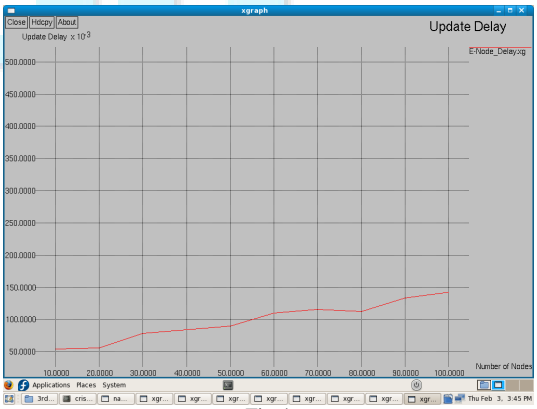

Fig 4

3. Update Delay Vs Query Request Rate

Update delay with respect to the query rates Ru/Rr is calculated, the system produces efficiency by reducing the update delay,

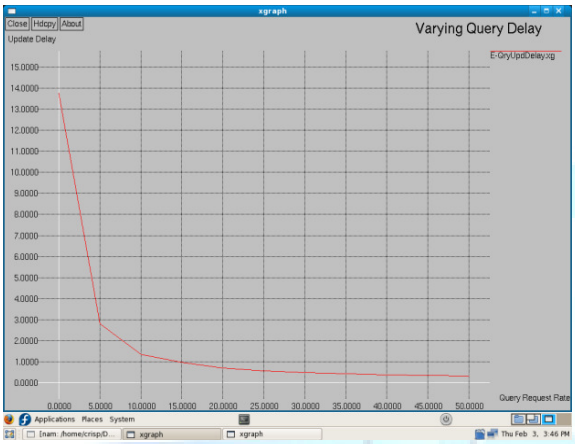

Fig 5

4. Query Delay Vs Data Update Rate:

The query delay for different data update rate in the cache nodes, are evaluated and resulted in update delay was reduced to a feasible extent, as shown in fig.5

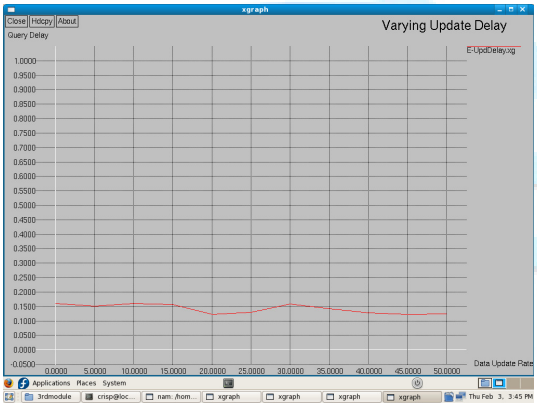

Fig 5

7. CONCLUSION AND FUTURE ENHANCEMENT

In this paper, a novel mechanism is presented for maintaining cache consistency in a Mobile Ad hoc Network. The evaluation results confirmed our analysis of the scalability of the system. That is, they indicate that even when the node density increases or the node request rate goes up, the query request delay in the system is either reduced or remains practically unaffected, while the cache update delay experiences a moderate rise. Yet, even if a higher update. The delay means that the probability of nodes getting stale data increases, the proposed system includes a provision for keeping track of such nodes and supplying them with fresh data within a delta time limit. Hence, it can be concluded that the system can scale to a moderately large network even when nodes are requesting data frequently. Moreover, the increase in the node speed and disconnection rate also only affected the cache update delay, and to a lesser extent, the traffic in the network. This reflects on the robustness of the architecture and its ability to cope with dynamic environments. This has been achieved by the simulation of nodes and the data is send from request node through query directory, the request node got the response from the cache node. The cache node is automatically updated by the server.

The approach in [14], on the other hand, ensures data confidentiality, integrity, and availability by establishing a security association (SA) between each two end-to-end communicating nodes. Both approaches were discussed and evaluated in [13] from which we can deduce that [2] is better suited since it satisfies many security requirements, while [14] introduces much overhead as it requires data redundancy. For our future work, we will do a thorough investigation of the possible threats that could disrupt the operations of the proposed mechanism and devise security mechanisms that safeguard it by either building on existing approaches, like [2] and [15], and/or developing new ones.

REFERENCES

[1] H. Artail, H. Safa, K. Mershad, Z. Abou-Atme, and N. Sulieman,"COACS: A Cooperative and Adaptive Caching System for MANETS," IEEE Trans. Mobile Computing, vol. 7, no. 8, pp. 961-977, Aug. 2008.

[2] N.A. Boudriga and M.S. Obaidat, "Fault and Intrusion Tolerance in Wireless Ad Hoc Networks," Proc. IEEE Wireless Comm. and Networking Conf. (WCNC), vol. 4, pp. 2281-2286, 2005.

[3] J. Cai and K. Tan, "Energy-Efficient Selective Cache Invalidation," Wireless Networks J., vol. 5, no. 6, pp. 489-502, Dec. 1999.

[4] J. Cao, Y. Zhang, L. Xie, and G. Cao, "Consistency of Cooperative Caching in Mobile Peer-to-Peer Systems over MANETs," Proc. Third Int'l Workshop Mobile Distributed Computing, vol. 6, pp. 573-579, 2005.

[5] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," IEEE Trans. Knowledge and Data Eng.,vol. 15, no. 5, pp. 1251-1265, Sept. 2003.

[6] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World-Wide Web," IEEE Trans. Computers, vol. 47, no. 4, pp. 445-457, Apr. 1998.

[7] A. Elmagarmid, J. Jing, A. Helal, and C. Lee, "Scalable Cache Invalidation Algorithms for Mobile Data Access," IEEE Trans Knowledge and Data Eng., vol. 15, no. 6, pp. 1498-1511, Nov. 2003.

[8] IEEE Standard 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, IEEE, 1999.

[9] J. Jing, A. Elmagarmid, A. Helal, and R. Alonso, "Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," Mobile Networks and Applications, vol. 15, no. 2, pp. 115-127, 1997.

[10] X. Kai and Y. Lu, "Maintain Cache Consistency in Mobile Database Using Dynamical Periodical Broadcasting Strategy," Proc. Second Int'l Conf. Machine Learning and Cybernetics, pp. 2389-2393, 2003.

[11] W. Li, E. Chan, Y. Wang, and D. Chen, "Cache Invalidation Strategies for Mobile Ad Hoc Networks," Proc. Int'l Conf. Parallel Processing, Sept. 2007.

[12] S. Lim, W.-C. Lee, G. Cao, and C.R. Das, "Performance Comparison of Cache Invalidation Strategies for Internet-Based Mobile-Ad Hoc Networks," Proc. IEEE Int'l Conf. Mobile Ad-Hoc and Sensor Systems, pp. 104-113, Oct. 2004.

[13] M.N. Lima, A.L. dos Santos, and G. Pujolle, "A Survey of Survivability in Mobile Ad Hoc Networks," IEEE Comm. Surveys and Tutorials, vol. 11, no. 1, pp. 66-77, First Quarter 2009.

[14] P. Papadimitratos and Z.J. Haas, "Secure Data Transmission in Mobile Ad Hoc Networks," Proc. ACM Workshop Wireless Security (WiSe '03), pp. 41-50, 2003.

[15] W. Stallings, Cryptography and Network Security, fourth ed. Prentice Hall, 2006.

[16] J. Xu, X. Tang, and D. Lee, "Performance Analysis of Location-Dependent Cache Invalidation Schemes for Mobile Environments," IEEE Trans. Knowledge and Data Eng., vol. 15, no. 2, pp. 474-488, Feb. 2003.

[17] J. Yuen, E. Chan, K. Lain, and H. Leung, "Cache Invalidation Scheme for Mobile Computing Systems with Real-Time Data," SIGMOD Record, vol. 29, no. 4, pp. 34-39, Dec. 2000.