International Conference on Systems, Science, Control, Communication, Engineering and Technology 2016 [ICSSCCET 2016]

ISBN	978-81-929866-6-1	**VOL**	02
Website	icssccet.org	**eMail**	icssccet@asdf.res.in
Received	25 – February – 2016	**Accepted**	10 - March – 2016
Article ID	ICSSCCET113	**eAID**	ICSSCCET.2016.113

Design of a Dynamic Architecture for Cache Management

C H Rammanogerlokiya[1], D Sharmitha[2], K Keerthivasan[3]
[1]Assistant Professor, Dept. of ECE, Karpagam Institute of Technology, Coimbatore
[2]Assistant Professor, Dept. of EEE, Kumaraguru College of Technology, Coimbatore
[3]Professor & Head, Department of Electronics and Communication Engineering

Abstract- *Mobile Ad Hoc NETworks (MANETs) are dynamic in nature, Links between nodes may constantly change as nodes move around, enter or leave the network, and therefore, a reliable caching scheme is more difficult to achieve. This project proposes a server based architecture to provide efficient and reliable caching in MANET environments which also minimizes the delays in answering node queries. Nodes can take on one of two possible roles: Caching nodes (CNs) and Query directories (QDs). A QD's task is to cache queries submitted by the requesting mobile nodes, while the CN's task is to cache data items (responses to queries). The heart of the system is the nodes that cache submitted queries. The queries are used as indexes to data cached in nodes that previously requested them. The consistency scheme is server-based in which control mechanisms are implemented to adapt the process of caching a data item and updating it by the server to its popularity and its data update rate at the server. The system implements methods to handle disconnections of QD and CN nodes from the network and to control how the cache of each node is updated or discarded when it returns to the network. This Architecture implements a pro-active Load distribution scheme to minimize the delay. Moreover, ns2 simulations were performed to measure several parameters, like the average data request response time, cache update delay, and hit ratio. The results demonstrate the advantage of the proposed scheme over existing systems.*

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

I. INTRODUCTION

As Mobile Ad Hoc Networks (MANETs) are becoming increasingly widespread, the need for developing methods to improve their performance and reliability increases. One of the biggest challenges in MANETs lies in the creation of effective algorithms to handle the acquisition and management of data in the highly dynamic environments of MANETs. 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. In a MANET, all messages sent between the server and the cache are subject to network delays, thus, impeding consistency by download delays that are considerably noticeable and more severe in wireless mobile devices. All cache consistency algorithms are developed with the same goal in mind: to increase the probability of serving data items from the cache that are identical to those on the server In many scenarios, mobile devices (nodes) may be spread over a large area in which access to external data is achieved through one or more access points (APs). However, not all nodes have a direct link with these APs. Instead, they depend on other nodes that act as routers to reach them. In certain situations, the APs may be located at the extremities of the MANET, where reaching them could be costly in terms of delay, power consumption, and bandwidth utilization. Additionally, the AP may connect to a costly resource (e.g., 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 MANETs is a topic that deserves attention. MANETs are dynamic in nature, and therefore, a reliable caching scheme is more

Cite this article as: C H Rammanogerlokiya, D Sharmitha, K Keerthivasan. "Design of a Dynamic Architecture for Cache Management". *International Conference on Systems, Science, Control, Communication, Engineering and Technology 2016*: 565-571. Print.

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 (e.g., processing power and storage capacity) and limited battery life. It follows that an effective caching system for MANETs needs to provide a solution that takes all of these issues into consideration. 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. With distribution, however, comes a new set of challenges. The most important of which is the coordination among the various nodes that is needed in order to store and find data. The rest of this paper is organized as follows: Section II describes the proposed system. Section III is dedicated to describing the simulation experiments and discussing the results. Finally, Section IV concludes the paper.

II. Proposed Framework

This section describes the proposed system, A Server-Based Architecture for Maintaining Cache Consistency and Minimizing Delay in Mobile Environments. The idea is to create a cooperative caching system that minimizes delay and maximizes the likelihood of finding data that is cached in the ad hoc network.

A. Basic Concepts

Nodes in MANET can take on one of two possible roles:

- Caching nodes (CNs) and,
- Query directories (QDs).

A QD's task is to cache queries submitted by the requesting mobile nodes, while the CN's task is to cache data items (responses to queries). The queries are used as indexes to data cached in nodes that previously requested them.

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. The server autonomously sends data updates to the CNs, meaning that it has to keep track of which CNs cache which data items.

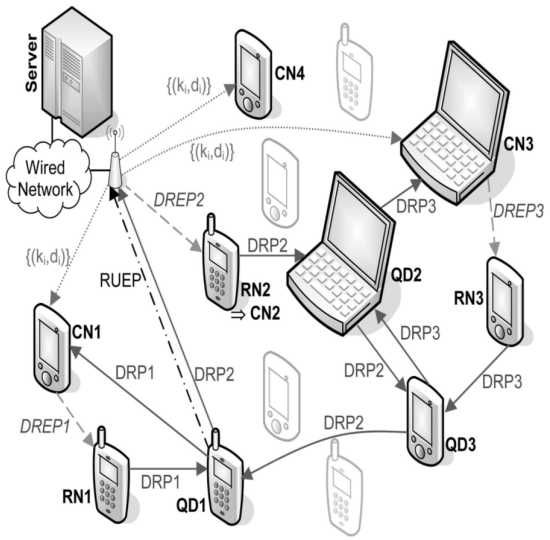

Fig 1 Basic operation of architecture

Fig 1 illustrates the Basic operation of this architecture. 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

Cite this article as: C H Rammanogerlokiya, D Sharmitha, K Keerthivasan. "Design of a Dynamic Architecture for Cache Management". *International Conference on Systems, Science, Control, Communication, Engineering and Technology 2016*: 565-571. Print.

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.

B. Search Algorithm

Fig 2 illustrates the Search Algorithm. Given that all nodes in the MANET have knowledge of all QDs, then when a node requires certain data, it sends a Data Request Packet (DRP) to the nearest QD. If this QD does not have a matching query, it adds its address to the DRP to indicate that it has already received the request and then sends this modified DRP to the nearest QD that has not been checked yet.

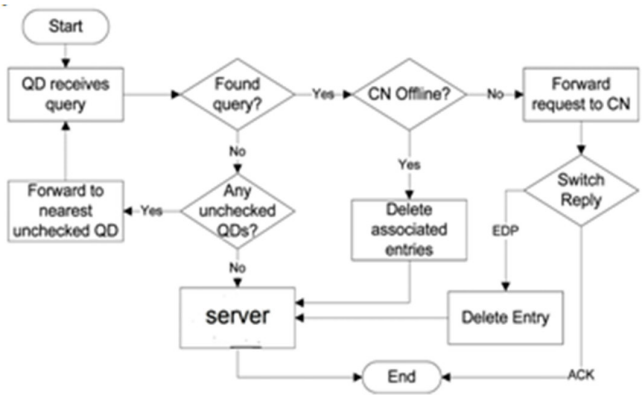

Fig 2 Search Algorithm

This continues until a hit occurs or until all the QDs have been checked, in which case an attempt is made to contact the data source. If a hit occurs at a QD, the QD visit list is removed from the DRP before the latter is sent to the CN that is caching the corresponding data, which will in turn send the reply data via a Data Reply Packet (DREP) directly to the RN, whose address is in the source field of the request packet. If a CN has gone offline, all QDs will be able to detect this when their routing tables get updated by the proactive routing protocol and will delete all the related entries in their memories. On the other hand, if a CN ever decides to replace an old cache item with a newer one, it informs the corresponding QD about it by sending an Entry Deletion Packet (EDP). In this case, the QD will delete the related entry from its cache to prevent misdirected requests for the data.

C. Caching System Formation

The QDs are the central component of the system and must be selected carefully. Preference should be given to nodes that are expected to stay the longest in the network and have sufficient resources. Nodes have to calculate and store a special score that summarizes their resource capabilities, including the expected time during which the device is in the MANET (TIME), the battery life (BAT), the available bandwidth (BW), and the available memory for caching (MEM). To be considered a candidate QD, the device must meet a minimum criterion in each category. The addition of a QD decreases the average load on existing QDs and potentially increases the available cache space (given that nodes can cache additional results) and, in turn, the hit ratio. At the same time though, this increases the response time of the system. Hence, the number of QDs should be chosen prudently in order to restrict the average delay of the system from increasing indefinitely while maintaining an acceptable load on QDs. When nodes join the network, they send HELLO packets so that other nodes know about them. After the HELLO packets are exchanged, the first node that needs to cache data item (i.e., after submitting a data request and getting the reply) sends a Caching Score Packet (CSP) containing its score, address, and caching capacity and an empty exhausted node list to one of its neighbours. When a node receives a CSP, it adds its score, address, and caching capacity to the table in the CSP and then chooses one of the nearest nodes from its routing table that is not contained in the list of addresses and the exhausted node list in CSP (implying that this node has not received this CSP yet) and sends the CSP to it. If the node receiving the CSP finds that all nodes from its routing table are present in the list of addresses in CSP, it adds itself to the exhausted node list and sends the CSP to a node that is in the list of addresses but not in the exhausted list. This strategy insures that the CSP will traverse all nodes in the network sequentially. Each node checks if the CSP table includes the scores and caching capacities of all nodes in the network excluding itself (the list of nodes can be retrieved from the underlying routing protocol). If yes, it plays the role of a QD Assigner (QDA) by sending the node that corresponds to the highest score a QD Assignment Packet (QDAP) containing the CSP table. Generally, a new QD is added to the system when a query needs to be cached but no QD agreed to cache it. The last QD to receive the caching request will initiate a CSP. Also, If a QD goes offline, the first node to discover this will initiate a CSP in order to find a new candidate QD. In both cases, the first node to compute the highest score from the CSP will be the QDA and sends a QDAP to the highest score candidate. If this node accepts, it broadcasts a (QIP) QD information packet with itself added to the QD list; else, it replies with a negative acknowledgment to the QDA. To protect against situations in which this candidate takes no action,

Cite this article as: C H Rammanogerlokiya, D Sharmitha, K Keerthivasan. "Design of a Dynamic Architecture for Cache Management". *International Conference on Systems, Science, Control, Communication, Engineering and Technology 2016*: 565-571. Print.

a timer is started at the QDA after each QDAP is sent. If the QDA receives a NACK or if it waits a period of T, it sends a QDAP to the second-highest-score candidate, and so on, until a candidate accepts the assignment. As discussed below, the CN holds for each cache entry, in addition to the response data, the query itself and a reference to the QD that caches it. This added information is used to rebuild QD entries when a QD goes offline. Upon receiving the CIP from the replacement QD, the concerned CNs will send it the queries that used to reference the lost QD using a Query Caching Request Packet (QCRP). The CIP will also serve to inform nodes about the change and prompt them to update their QD lists. If a CN goes offline, the QDs will detect its departure after the routing protocol updates their routing tables and will delete the associated entries in their cache. Note that in case an on-demand routing protocol is in place, each QD could send a special message to all other QDs periodically to discover if a QD has gone offline. Furthermore, every CN could be set up to return an acknowledgment when it is forwarded a request from a QD, so it could eventually be discovered when it goes offline. Additionally, the score of a QD may fall below the set threshold at any time due to its participation in the network and its use by the user. When it detects that its score is about to become low, it broadcasts a CSP packet, and upon receiving the CIP from the new QD, it transfers its cache to it, broadcasts a CIP not including itself, and then deletes its cached queries.

D. 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.

E. Dealing with Node Re-union

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 CN 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.

F. Adapting to Update Rate

This Architecture 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 τ. If this ratio is greater than or equal to τ , 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-to-request 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, 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, γ ($\gamma < \tau$), the server will reply to the RN with a DREP that includes the CACHE_NEW flag in the header. Upon receiving the DREP, the RN sends a 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 τ and γ the system can reduce unnecessary network traffic. However, the two thresholds allow for favoring bandwidth consumption over response time, or vice versa. This makes the proposed system suitable for a

Cite this article as: C H Rammanogerlokiya, D Sharmitha, K Keerthivasan. "Design of a Dynamic Architecture for Cache Management". *International Conference on Systems, Science, Control, Communication, Engineering and Technology 2016*: 565-571. Print.

variety of mobile computing applications: a large τ may be used when disconnections are frequent and data availability is important, while a low τ could be used in congested environments where requests for data are infrequent or getting fresh data is not critical.

G. Adapting To Request Rate

When demand for a particular data is too high. A caching node which consists of that data would get more requests. This makes the entire path to be flooded by packets, which paves a way for high network traffic. When request rate for "d" is more, CNd (The node that caches the data'd') will get more data request packets (DRP), results in delay in answering the nodes that requested for data'd'. And also unnecessary traffic is generated. In such case this Architecture implements a pro-active Load distribution scheme to minimize the delay. CNd Computes Ru/Rr value and compares with a threshold level "β".If this ratio is lesser than the threshold value β then, CNd Send LDRP to server which includes file distribution request. Servers in turn, reply with CAP to CNd which includes the distribution details. Now CNd distributes the "d" to CNd1 (A new node that caches the data'd' that is distributed from CNd).Now CNd1 send 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 CNd1 directly whenever the data items are updated.

III. Performance Evaluation

A. Network and Cache Simulation Parameters

A single database server is connected to the wireless network through a fixed access point, while the mobile nodes are randomly distributed. The client cache size was fixed to 200 Kb, meaning that a CN can cache between 20 and 200 items, while the QD cache size was set to 300 Kb, and therefore, a QD can cache about 600 queries.

Every second, the server updates a number of randomly chosen data items, equal to a default value of 20. The default values of τ and γ were set to 1.25 and 0.75, respectively, while the default number of node disconnections is 1 every two minutes with a period of 10 seconds, after which the node returns to the network. For experiment Parameter values are varied to study their effects on performance.

Performance of the system is evaluated by considering several parameters such as query delay, response delay, update delay, and hit ratio.

B. Varying the Number of Nodes

This section presents the effects of varying the node density in the fixed network area. Figure 3 shows the time taken by the server to update data items which is stored in caching nodes (CN).

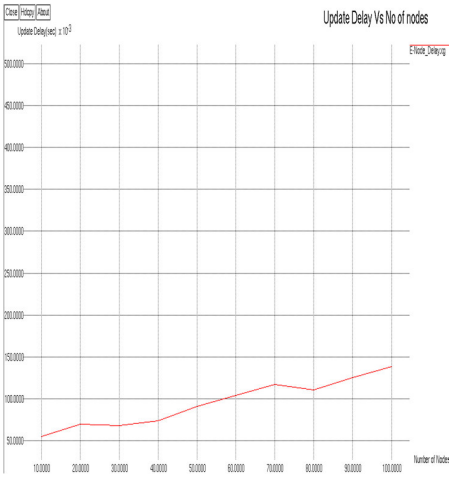

Fig 3 Update delay Vs Number of nodes

C. Varying Threshold Rate

When the ratio R u/R r of a data item reaches τ, the item is fetched from the server whenever requested until the ratio drops below γ, then the request is cached again. The values of τ and γ were set to 1.5 and 0.75, respectively. For experiments, three different update

Cite this article as: C H Rammanogerlokiya, D Sharmitha, K Keerthivasan. "Design of a Dynamic Architecture for Cache Management". *International Conference on Systems, Science, Control, Communication, Engineering and Technology 2016*: 565-571. Print.

rate (Ru) values 1.5, 20, and 50 updates per second are fixed. Figure 4 shows the delay in data update in various threshold rates. The result indicates that the performance of the architecture is not affected in major way while varying thresholds.

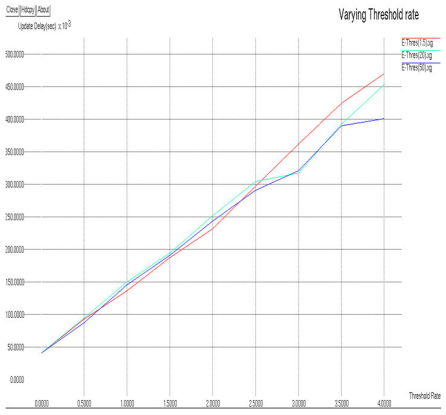

Fig 4 Threshold rate Vs update delay

D. Varying the Data Update Rate

The results of varying the update rate are shown in Figure 5.Results indicate that query delay of the system remains almost unchanged as the update rate increases. On the other update delay increases gradually as data update rate increases.

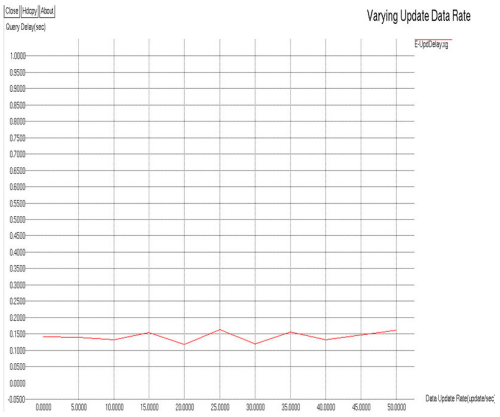

Fig 5 Varying update rate Vs Query delay

E. Varying the Query Request Rate

When the request rate is increased the update delay decreases initially and then settles down as shown in figure 6.Because as more items are cached, new CNs are set up. This increases the probability of having more CNs closer to the access point, which, in turn, results in smaller number of hops, on average, for the update packets to reach their destinations.

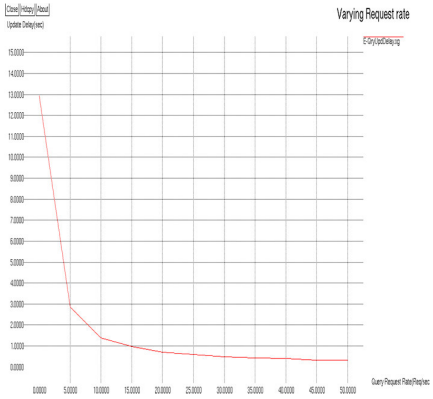

Fig 6 Request rate Vs update delay

Cite this article as: C H Rammanogerlokiya, D Sharmitha, K Keerthivasan. "Design of a Dynamic Architecture for Cache Management". *International Conference on Systems, Science, Control, Communication, Engineering and Technology 2016*: 565-571. Print.

IV Conclusion

In this paper we proposed a novel mechanism that provides efficient and reliable caching in MANET environments which also minimizes the delays in answering node queries.This architecture suspends server updates when it deems that they are unnecessary in order to maintain the network traffic.The consistency scheme is server-based in which control mechanisms are implemented to adapt the process of caching a data item and updating it by the server to its popularity and its data update rate at the server. Also deals disconnection/re-union of QD and CN nodes from/to the network and to control. This Architecture follows a pro-active Load distribution scheme to minimize the delay in answering nodes. Hence, it can be concluded that the system can scale to a moderately large network even when nodes are requesting data frequently.

References

1. H. Artail and K. Mershad, "SSUM: Smart Server Update Mechanism for Maintaining Cache Consistency in Mobile Environments" IEEE Trans. Mobile computing, vol. 9, no. 6, pp.778- 795 June 2010
2. B. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments," Proc. ACM SIGMOD, pp. 199-210, May 1995.
3. 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.
4. H. Artail and K. Mershad, "MDPF: Minimum Distance Packet Forwarding for Search Applications in Mobile Ad Hoc Networks," IEEE Trans. Mobile Computing, vol. 8, no. 10, pp. 1412- 1426, Oct. 2009.
5. J. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies for Mobile Environments," Proc. ACM SIGMOD, pp. 1- 12, May 1994.
6. K. Bettstetter and J. Eberspacher, "Hop Distances in Homogeneous Ad Hoc Networks," IEEE Proc. 57th IEEE Semiann. Vehicular Technology Conf., vol. 4, pp. 2286-2290, Apr. 2003.
7. 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.
8. N. Cai and K. Tan, "Energy-Efficient Selective Cache Invalidation," Wireless Networks J., vol. 5, no. 6, pp. 489-502, Dec. 1999.
9. N. 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.
10. R. 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.
11. R. 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.
12. S. Chung and C. Hwang, "Transactional Cache Management with Aperiodic Invalidation Scheme in Mobile Environments," Advances in Computing Science, pp. 50-61, Springer, 1999.
13. S. 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.
14. T. Jin, J. Cao, and S. Feng, "A Selective Push Algorithm for Cooperative Cache Consistency Maintenance over MANETs," Proc. Third IFIP Int'l Conf. Embedded and Ubiquitous Computing, Dec. 2007.
15. T. 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.
16. 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.
17. X. 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.
18. Y. 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.

Cite this article as: C H Rammanogerlokiya, D Sharmitha, K Keerthivasan. "Design of a Dynamic Architecture for Cache Management". *International Conference on Systems, Science, Control, Communication, Engineering and Technology 2016*: 565-571. Print.