A Hybrid Cache Invalidation Technique for Data Consistency in MANET

N. Sabiyath Fatima
Department of Computer Science and Engineering
BSA University, Chennai, India.

Dr. P. Sheik Abdul Khader
Professor & Head Department of Computer Applications
BSA University, Chennai, India.

ABSTRACT

Recent advances in computer and wireless communication technologies have led to an increasing interest in ad hoc networks, which are temporarily constructed by only mobile hosts. Data accessibility in ad hoc networks is lower than that in the conventional fixed networks.. Caching the frequently accessed data items on the client side is an effective technique for improving the performance in a mobile environment. This form of data caching significantly improves the efficiency of information access in a wireless ad hoc network which in turn reduces the access latency and bandwidth usage. The main objective is to avoid the stale data using invalidation policy based on TTL. This paper introduces an Extended Adaptive TTL (Ex-ATTL) algorithm, in which 1-hop distance nodes to data cache node maintain a hash table for cache invalidation. Data item name is used as key and TTL is used as its value. The network is simulated using NS-2 to evaluate the performance of the proposed algorithm, also it is compared with fixed TTL and Adaptive TTL schemes. The simulation results shows that Ex-ATTL algorithm can cut the query delay by a factor of 3 and double the throughput compared to the Adaptive TTL.

Keywords: ***Invalidation, MANET, Stale Data, TTL, Cache Consistency.***

1. INTRODUCTION

The recent advancement in computer and wireless communication technologies have led to an increasing interest in ad hoc networks, which are temporarily constructed by only mobile hosts. The ad-hoc network architecture can be used as a wireless extension for the users to access the Internet in an area without network infrastructure [1]. The features like multi-hop message relay, frequent link disconnection and power-constrained operation, complicate the design of network protocols [4] and the application development.

In order to ensure the consistency between source node and cached data at the clients efficient data cache invalidation mechanisms are essential. TTL based cache invalidation problem has been well studied in mobile computing environments [5] [1]. The Fixed TTL and Adaptive TTL based techniques are the existing strategies for cache invalidation. Maintaining cache invalidation process in cache node produces some additional overhead to cache and also decrease the performance of the network. Due to changes such as two nodes moving out of wireless transmission range of each other cache may become invalid [12]. In the proposed Ex-ATTL algorithm, cache node broadcast the data item's name and data item's TTL to its 1-hop neighbors. Each node invalidates the data according to data items TTL stored in cache invalidation hash table.

The rest of this paper is organized as follows. Section 2 reviews the related works for cache invalidation schemes in mobile ad hoc networks. Section 3, describes the system model for data service in a mobile ad hoc network (MANET). Section 4, gives the proposed system architecture and Extended Adaptive TTL algorithm. Section 5, discusses the performance evaluation, and simulation results for Ex-ATTL. Finally, Section 6 concludes the paper.

2. RELATED WORKS

As categorized in [3], [11], there are two kinds of cache invalidation methods. Location-dependent invalidation which is based on the geographical position of the mobile nodes supports single hop communication only. Temporal dependent invalidation is carried out by data updates. To carry out temporal-dependent invalidation, the data source keeps track of the update history and sends it, in the form of an invalidation report (IR), to the clients, either by periodic/a periodic broadcasting or upon individual requests from the clients [16], [10], [14], [9]. The mobile client, if active, listens to the IRs and updates its cache accordingly.

According to the authors of [15], [16], caching data or path of the data at mobile client side is an essential mechanism for improving system performance in a mobile computing environment.

In [6], Cache Path and Cache Data handle cache consistency using a simple weak consistency model based on the TTL mechanism. In this model, a routing node considers a cached copy up-to-date if its TTL hasn't expired. If the TTL expires, the node removes the map from its routing table (or removes the cached data). Author's of [8] optimize this model by allowing nodes to refresh a cached data item if a fresh copy of the same data passes by. In [13] [2] authors use the Adaptive TTL based cache invalidation scheme for Hybrid Cache and Cooperative Cache. Caching the data indicates the mobile node's interest in it. While performing data forwarding, if a mobile node finds an invalid copy of that data in its cache, it deletes the old copy and stores new copy for future use. If an expired path or data item has not been

refreshed for the duration of its original TTL time, it is removed from the cache.

3. NETWORK MODEL

The system constitutes n mobile nodes. N= {N_1, N_2, . . . , N_n} denote the set of mobile nodes. Every node holds a cache in its local storage, and issues queries for data items from the data source. The data source is called a server and it contains a database. It is assumed that there are m data items in the database. Data items from 1 to m serve as their identifiers. D= {D1, D2, . . ., Dm} denotes the set of data items in the database. In order to keep track of the version of a cached copy of a data item, each copy of a data item is associated with a TTL and also records the time T_c when the copy is created in the source, where T_c is a data item copy creation time.

4. PROPOSED INVALIDATION ALGORITHM

Caching is the concept of locally storing copies of remote data objects for faster retrieval. When multiple nodes cache the same data item, there is a need to maintain cache consistency. The policy employed to notify all caches about an update is called invalidation policy. TTL is a number indicating the time interval during which the data item is considered to be fresh. When the data item is to be referenced, its TTL value is checked and if the TTL is still valid then the object is retrieved otherwise a new value is requested from the server. The whether forecast, stock market, sports news and battle field conditions monitoring update the data item in every short interval of time periods. Some kind of data items like video films and geographical information of particular places are not updated in short time periods. There are two types of TTL which are categorized based on the updating nature of data item.

Architectural Overview

In Extended Adaptive TTL scheme, each and every node maintains the cache invalidation hash table. T_c is the time when successful copy of the data item is created in a mobile node. The Fig.1 shows the Architecture of EX-ATTL.

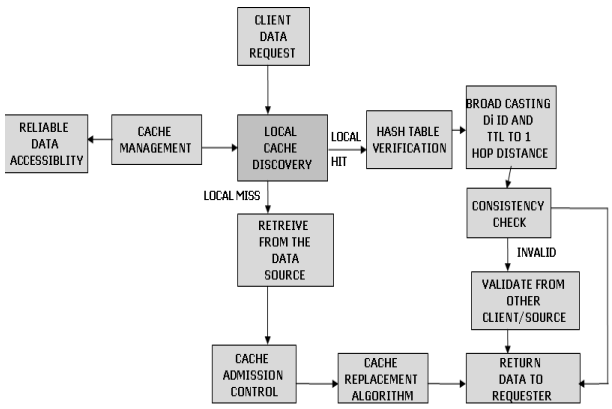

Fig.1. Architecture of EX-ATTL

The proposed scheme is divided into three phases. First phase of the algorithm does the cache invalidation hash table verification for current request. Second phase of the algorithm retrieve the data item from appropriate source. In third phase cache node broadcasts the data item name and its current TTL value to its one hop neighbor.

Phase 1

When a node generates a data request, it is verified from current nodes hash invalidation hash table. If cache invalidation hash table contains the requested data item then the algorithm retrieves the TTL and T_c from cache table. The request generation time T_{req} is also cached from the nodes timer then it subtracts the T_{req} from T_c. If this subtracted value is greater than TTL then this also removes the data item, its TTL and its T_c from hash table and forwards the request to the data source. The data source calculates the new TTL and sends with the data item. If the subtracted value is less than TTL it moves to the next phase.

Phase 2

Here, the algorithm checks the local cache for node request. If local cache contains the requested data item, then the request node simply uses its local cache. Other wise this phase uses the current implementing cache algorithm and retrieve the data item from nearest cache node or from the data source. Again, the TTL of the particular data item is recalculated.

$$\text{New TTL} = T_{req} - T_c$$

The new TTL is appended with the corresponding data item and sends the response to the client node. Data cache node also updates the new TTL in its cache table only for the particular data item.

Phase 3

When a successful copy of the data item is created in localized cache or in any intermediate cache node for future requests this phase cache the T_c from the cache node's timer and updates this T_c in its cache invalidation hash table. Cache node also broadcasts the new cached data items id, TTL and its T_c only to one hop neighbors.

Pseudo-code of the Algorithm

```
SET TTL, Tc, Treq /* declaration */
Treq = getCurrentTime () \* node's time when request is generated */
if (local hash table contains requested data item Di)
{
   /* phase 1*/
      TTL =  retrieveFromHashTable (Di's TTL)
      Tc  =  retrieveFromHashTable (Di's Tc)
      if (Treq - Tc > TTL)
      {
          deleteFromHashTable (Di's id, Di's TTL, Di's Tc)
          deleteFromLocalCache (Di)
```

```
                    SendRequestToDataSource (Di's id)
                    /* phase2*/
                    SET hopcount = 1
              NewTTL =  Treq - Tc  /* Done by data source*/
              ReplyFromServer (Nj id, Di , newTTL )
              dataCache (Di)

                    / phase 3 */
                 localHashTableUpdation (Di , newTTL,Tc)
              broadCast (Di , newTTL,Tc , hopcount)
      }
      else  /* phase 2 */
      {
           SET newTTL
        if (localCacheContains (Di) == TRUE)
        {
           retriveFromLocalCache (Di)
           }
           else
        {
             SET hopcount = 1
                    SendRequestToDataSource (Di's id)
                    newTTL =   Treq - Tc   /* Done by data
source*/
              ReplyFromServer (Nj id, Di , newTTL)
              dataCache (Di)
                    / phase 3 */
                 localHashTableUpdation (Di , newTTL,Tc)
              broadCast (Di , newTTL,Tc , hopcount)
        }
     }
}
else
{
                    /* phase2*/
                    SET hopcount = 1
              newTTL =  Treq - Tc  /* Done by data source*/
              ReplyFromServer (Nj id, Di , newTTL )
              dataCache (Di)
                    / phase 3 */
                 localHashTableUpdation (Di , newTTL,Tc)
              broadCast (Di , newTTL,Tc , hopcount)
}
```

5. PERFORMANCE EVALUATIONS

The Simulation Environment and Parameters

The system model consists of a single data source that serves multiple client nodes. The system parameters are shown in Table 1.

There are 1,000 data items in the database, which is divided into two subsets: the short TTL data subset and the Long TTL data subset. The short data subset includes data items from 1 to 150 (out of 1,000 items) and the long data subset includes the remaining data tems of the database. Clients have a large probability (75 percent) to access the data in the short TTL set and a low probability (20 percent) to access the data in the Long TTL set.

Table I Parameter Settings

Simulator	**Network Simulator (NS2)**
Simulation time	5000 Sec
Network size	1800m x 800m
Mobility model	Random waypoint
Database size n	2000 items
Number of nodes	50 to 100
Transmission range	200m
Speed of mobile host	1~20m/s randomly
Average Query Rate	0.5 / Second
Bandwidth(Mb/s)	2
TTL(secs)	300 to 1000
Client cache size(kb)	100 to 1000
Mean query generate timeT_{query}(secs)	1 to 100
Pause time(secs)	200

Results and Discussions

The performance analysis presented here is designed to find the effects of different workload parameters such as mean update arrival time, mean query generate time, and system parameters such as cache size, replicate times (m), and short TTL data access probability (p_q). Then the performance comparison is carried out among Adaptive TTL and Ex-ATTL. The performance is measured by the cache hit ratio, the query delay, and the throughput. Note that minimizing the number of uplink requests is a desirable goal as clients in a mobile environment have limited battery power and transmitting data requires a large amount of power. The performance metrics are defined as follows.

Query Delay: The time interval between the request and the response.

Cache Hit Ratio: The percentage of accesses that result in cache hits is known as the hit rate or hit ratio of the cache.

Impact of Query Delay

The query delay is measured as a function of the mean query generates time and the mean update arrival time. As shown in Fig.2, Ex-ATTL algorithm significantly outperforms the Adaptive TTL scheme. As explained before, each client generates queries according to the mean query generate time. If the queried data is in the local cache, the client can serve the query locally; otherwise, the client has to request the data from the server. Since the broadcast bandwidth is fixed, the server can only transmit a limited amount of data during one transaction interval, and then it can only serve a maximum number (α) of queries during one transaction interval. If the server receives more than (α) queries during one transaction interval, some queries are delayed to the next transaction interval. Fig. 2 shows the query delay as a function of the mean query generate time with T_u = 10s and c = 100 items. When the query generates

time is lower than 70s (e.g., 60s), the query delay of the Adaptive TTL algorithm becomes infinite long. However, when the query generation time reaches 30s, the query delay of proposed algorithm is still less than 20s.

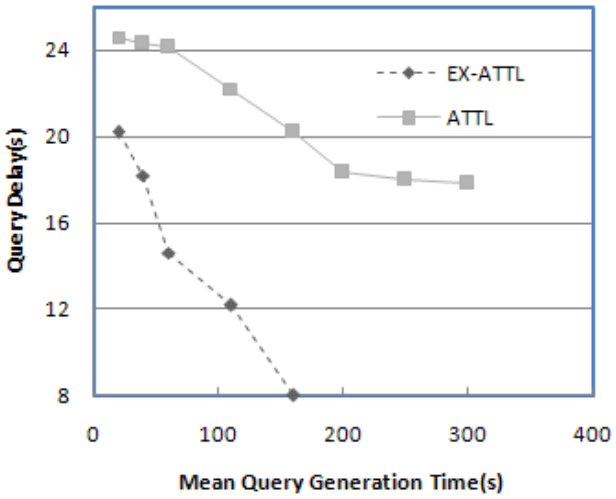

Fig.2. Query delay Vs Mean query generate time (Tu = 10s, c = 100 items).

The Query Delay Evaluation Versus Mean Update Arrival Time

As shown in Fig. 3, as the mean update arrival time increases, the cache hit ratio increases and the query delay decreases. Since the proposed algorithm has high cache hit ratio than the Adaptive TTL algorithm, the query delay of Ex-ATTL algorithm is shorter than the Adaptive TTL algorithm. For example, with Tu = 10,000s, Ex-ATTL algorithm reduces the query delay by a factor of 3 compared to the Adaptive TTL algorithm. Although the cache hit ratio of the Adaptive TTL algorithm is doubled from T_u = 10s to T_u = 33s, the query delay of the Adaptive TTL algorithm does not drop too much (from 18.7s to 16.1s). Since the query generated time is exponentially distributed, multiple queries may arrive at a client during one transactional interval. The requests in the back end of the queue will have much higher query latency, and it increases the average query latency.

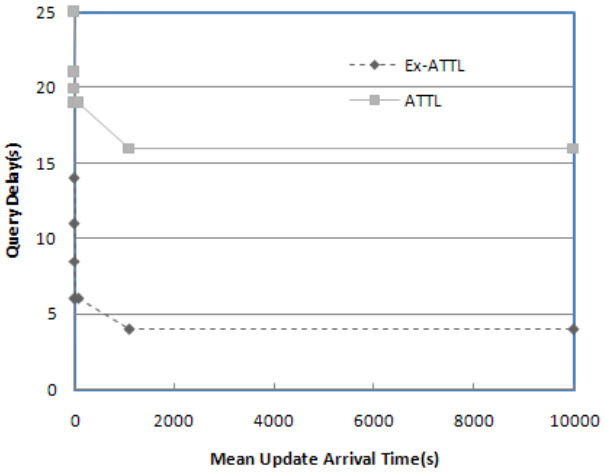

Fig.3. Query delay Vs Mean update arrival time (T_u = 100s, c = 100 items).

Cache Hit Ratio Comparison with Different Cache Size

The Fig.4. shows the cache hit ratio under different cache sizes when the number of clients is 100. It is easy to see that the cache hit ratio of Ex-ATTL algorithm is always higher than that of the Adaptive TTL algorithm for one particular cache size (cache size is 50 items, 100 items). For example, in the Adaptive TTL, when the update arrival time is 1s, the cache hit ratio does not have any difference when the cache size changes from 50 data items to 100 data items. However, under the same situation the cache hit ratio increases from about 40 percent to 58 percent in the proposed scheme. In Ex-ATTL algorithm, clients may need to download interested data for future use, so a large cache size may increase cache hit ratio.

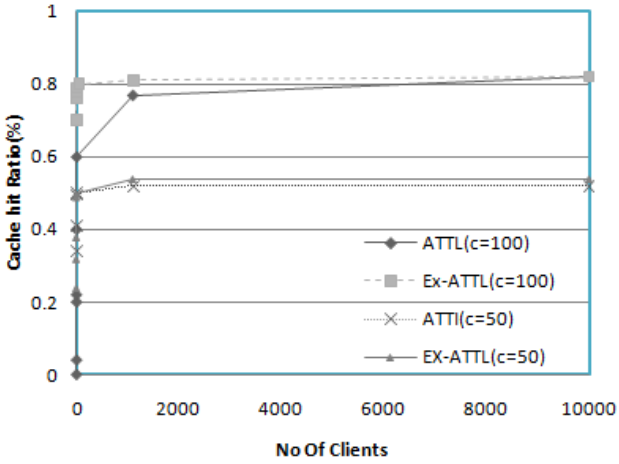

Fig.4. Cache hit ratio Vs Cache sizes (T_{req} = 100s , c = 100 items).

As shown in Fig. 4, the cache hit ratio drops as the update arrival time decreases. When the update arrival time is 10,000s, both algorithms have similar cache hit ratio for one particular cache size. With c = 300 items, as the update arrival time reaches 1s, the cache hit ratio of Ex-ATTL algorithm still keeps around 58 percent. When the update arrival time is very low (e.g., 1s), most of the cache misses are due to short TTL data access; when the update arrival time is very high (e.g., 10,000s), most of the cache misses are due to Long TTL access. Since Ex-ATTL algorithm is very effective to improve cache performance when accessing short TTL data, the cache hit ratio of the proposed scheme can be significantly improved when the update arrival time is low. However, as the mean update arrival time drops further (T_u < 1s), the cache hit ratio of the Ex-ATTL drops much faster than before.

*Impact of **Number** of Uplink Requests*

Fig.5. shows the uplink cost of both algorithms. Since Ex-ATTL algorithm has a lower cache miss rate than the Adaptive TTL and clients only send uplink requests when there are cache misses, proposed algorithm has lower uplink cost compared to the Adaptive TTL.

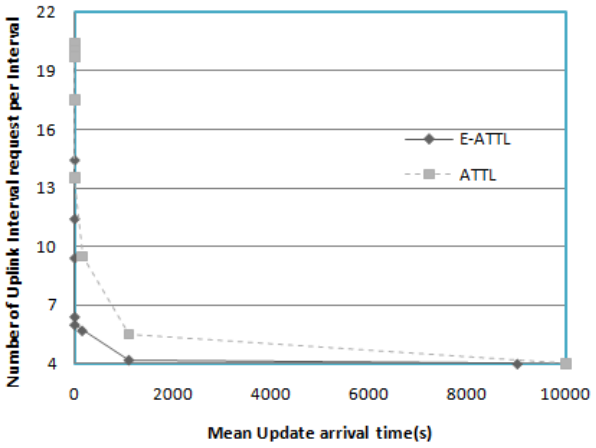

Fig.5. Mean update arrival time Vs the number of uplink requests per IR interval (T_q = 100s, c = 100 items).

Both the algorithms have similar uplink cost when the mean update arrival time is very high (e.g., 10,000s), but there is a significant difference when the mean update arrival time is 10s. From Fig. 5, it is found that both algorithms have similar cache miss ratio (1 - cache hit ratio) when T_u = 10,000s, but a significant difference when T_u = 10s. As shown in Fig. 6, Ex-ATTL algorithm can cut the uplink cost by a factor of 3 (with T_u = 10s) and, hence, the clients can save a large amount of energy and bandwidth.

6. CONCLUSIONS

TTL based cache invalidation technique is designed to efficiently support data reliability in ad hoc networks. TTL-based cache invalidation techniques have received considerable attention due to its reliability, scalability and simple implementation mechanism. The problem of cache staleness are studied and a new scheme called Ex-ATTL is implemented to overcome the above mentioned problem. The existing technologies, such as Fixed TTL and Adaptive TTL do not have clear TTL calculation mechanism and hence it does not support valid data accessing 100%. In the proposed scheme if any node caches the data item then it broadcast the data items id and its current valid TTL to its one hop neighbor for future request. This advanced mechanism improves the network performance and also reduces the cache nodes overhead.

7. ACKNOWLEDGEMENTS

We express our thanks to Dr. P. Kanniappan, the ViceChancellor, Prof. V. M. Periasamy, the Registrar and Prof.K.M.Mehtha, the Head, Department of CSE & Dean, School of Computer and Information Science, B.S.Abdur Rahman University, Chennai, Tamilnadu, India for the encouraging environment provided.

8. REFERENCES

[1] L. Yin and G. Cao, Supporting cooperative caching in ad hoc networks, IEEE Transactions on Mobile Computing, 5(1), pp.77-89, 2006.

[2] Y. Du, and S. K. S. Gupta, A cooperative caching service in MANETs,Autonomous Systems and International Conference on Networking and Services, ICAS/ICNS 2005.

[3] M. Abolhasan, T. Wysocki and E. Dutkiewicz, A review of routing protocols for mobile ad hoc networks, Ad Hoc Networks, Vol. 2, pp.1-22, 2004.

[4] G. Cao, L. Lin and C. Das, Cooperative cache-based data access in ad hoc networks, IEEE Computer, 37(2), pp.32-39, 2004.

[5] S. Lim, W. Lee, G. Cao, and C. Das, Performance Comparison of Cache Invalidation Strategies for Internet based Mobile Ad Hoc Networks, IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), pp.104-113, 2004.

[6] I. Chlamtac, M. Conti and J. J.-N Liu, Mobile ad hoc networking: imperatives and challenges, Ad Hoc Networks, Vol.1, 13-64, 2003.

[7] A. Kahol, S. Khurana, S.K.S. Gupta, and P.K. Srimani, "A Strategy to Manage Cache Consistency in a Distributed Mobile Wireless Environment," IEEE Trans. Parallel and Distributed Systems, Vol.12, No.7, pp. 686-700, July 2001.

[8] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," Proc. Sixth Ann. ACM/IEEE Int'l Conf. Mobile Computing and Networking (MobiCom 2000), pp. 200-209, August 2000.

[9] J. Xu, X. Tang, D.L. Lee, and Q.L. Hu, "Cache Coherency in Location-Dependent Information Services for Mobile Environments", Proceedings of First Int'l Conf. Mobile Data Access (MDA '99), pp. 182- 193, December1999.

[10] N. Vaidya and S. Hameed, "Scheduling Data Broadcast in Asymmetric Communication Environments," ACM / Baltzer Wireless Networks (WINET), pp.171-182, May 1999.

[11] S. Acharya and S. Muthukrishnan, "Scheduling On-Demand Broadcasts: New Metrics and Algorithms," Proc. ACM MobiCom '98, pp. 43-54, October 1998.

[12] J. Jing, A.K. Elmagarmid, A. Helal, and R. Alonso, "Bit-Sequences: A New Cache Invalidation Method in Mobile Environments," ACM/Baltzer J. Mobile Networks and Applications. (MONET), Vol. 2, No. 2, pp. 115-127, 1997.

[13] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments," Proc. ACM SIGMOD Conf. Management of Data, pp. 199-210, May 1995.

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

[15] G.H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," Computer, Vol. 27, No. 6, pp. 38-47, April 1994.

[16] C.Siva Ram Murthy, and B.S.Manoj. Ad Hoc Wireless Networks Architectures and Protocols, Pearson Education, 2005.