Chapter 4

E-LRU: A Cache Replacement Algorithm for Cooperative Caching

Contents

4.1 Introduction .................. 80
4.2 Formulation of Cache Replacement Problem . 80
4.3 Temporal Locality ................ 82
4.4 Cache Consistency ............. 83
4.5 The Proposed Scheme ............ 85
4.6 Cache Consistency and Cache Invalidation .. 86
4.7 Simulation Scenarios and Metrics .......... 86
    4.7.1 Performance Metrics ............. 87
4.8 Performance Evaluation .......... 87
4.1 Introduction

Cache replacement policy is a major design parameter of any caching policy. This is particularly true for MANETs since they are characterized by memory and resource constraints. The efficiency of the replacement policy affects both the hit rate and access latency of the system [Zahran, 2007]. The higher the memory constraints of the cache, the more vital the replacement policy becomes. As seen in the previous chapter, a lot of research work is geared towards finding the best cache replacement policy. But almost all the replacement policies proposed for ad hoc networks deal with LRU replacement policy. This chapter addresses the problem of cache replacement in cooperative caching for ad hoc networks and proposes a new algorithm for cache replacement.

Another issue in caching is consistency. The data supplied from the cache may not be the same when compared to the original copies on the data server, where the data is originated. In order to avoid giving stale data, caches have to update their contents within them so that they are not stale. Therefore attention is paid to the problem of maintaining cached data up to date with the server.

4.2 Formulation of Cache Replacement Problem

The objective of cache replacement is to maximize hit rate and minimize access delay of all nodes in ad hoc networks by appropriately replacing cached copies of data. A poor replacement policy can increase the number
of misses in the cache and cause a lot of traffic out of the cache and increase the miss penalty [Zahran, 2007].

Cache replacement becomes necessary when a cache miss occurs and no free slots exist in the local cache. In principle, the cache memory handler could select a data item randomly for replacement. However, another cache miss would result when the replaced data item is referenced again, so it is important to replace data items that are not likely to be referenced in the immediate future. Replacement strategies are concerned with deciding which data item or document to be displaced to make room for an incoming data when the cache storage is already full. Due to the space limitation, the mobile nodes can store only a subset of the frequently accessed data. A good replacement mechanism is needed to distinguish between the items to be kept in cache and that is to be removed when the cache is full.

Almost all of the replacement algorithms proposed for cooperative caching are LRU based. LRU is the simplest replacement policy suggested from early days for data caching. The disadvantage of LRU is, it considers only too little information for replacement. Another category for cache replacement is function based policies which involve the calculation of a value function based on different parameters, which use complex data structures that make the implementation difficult. The replacement policy introduced in this chapter, Extended-LRU (E-LRU), aims at replacing the data objects based on the time difference between the recent references. The novelty of this approach lies in the key based replacement where we first consider the time interval between the recent references for each data item and the data with longest reference interval time is dropped first. The advantage of this scheme is that the pages with
shortest access time interval, which have more probability of reference in the future, will be kept in the cache.

The basic idea of E-LRU is to keep track of the last two references to the cached data item and using this information the inter arrival time between requests are calculated for each data item. Since LRU drops the data from cache that has not been accessed for the longest time, it is unable to differentiate between data that have relatively frequent references. The basic frequency based strategy for replacement is Least Frequently Used (LFU) strategy. LFU removes the least frequently requested data. LFU based algorithms take more history information in to consideration but have no means to discriminate between recent and past reference frequency of a data item and is unable to cope with the evolving access patterns. E-LRU effectively addresses the limits of these algorithms by making use of inter reference time for making replacement decision. The major objective of the proposed approach is to effectively address the limits of LRU by retaining the low overhead merit of LRU.

4.3 Temporal Locality

Generally data availability is enhanced by different mechanisms like caching, replication and prefetching protocols. All these protocols exploit the property of locality of reference for its implementation. These protocols can be generally classified as server based, client based and network based systems [Jin, 2001]. The important characteristic that classifies these systems are web reference patterns. In each of these categories the request streams generated are different. In client based systems, the request streams are given to a limited and homogeneous community of users. On the other hand the request stream is limited to the objects offered by the
server in a server based systems. In network based systems the existence of these protocols are transparent to both server and client and therefore the above mentioned limitations does not exist. In each of the above mentioned categories the underlying determinant of temporal locality is different and an effective cache replacement policy should characterize the degree of locality present in typical web request streams.

Ad hoc networks can be considered as client based systems and previous studies have concluded that temporal locality is weakening for client based systems [Barford, 1999]. This is because the data requests generated for client based systems are generally the set of requests that missed in the client cache. Such a request stream is likely to exhibit weak temporal locality of reference in particular, a recently accessed object is unlikely to be accessed again in the future [Barford, 1999].

Using an independent reference model [Coffman, 1973, Breslau, 1999] showed that the Zipf-like popularity distribution of objects in web request streams can asymptotically explain other properties (namely, cache efficiency and temporal locality). In particular, they showed that the probability of referencing object $t$ units of time after it has been last referenced is roughly proportional to $1/t$. Thus, the probability distribution of reference inter-arrival times (or inter-request time) could be used to model temporal locality.

4.4 Cache Consistency

As discussed in chapter 2, there are mainly two techniques for ensuring cache consistency: strong cache consistency and weak cache consistency. The main difference between strong cache consistency and weak cache consistency is that in strong cache consistency the cache ensures
4.4. Cache Consistency

the freshness of content by cross checking with the original server every time the content is asked for. On the other hand, in weak consistency model the cache may employ heuristics for ensuring freshness and need not contact the server every time it serves the content. Hence, strong consistency could turn out to be more expensive for ad hoc networks than weak consistency.

The adaptive TTL consistency model which has its origin in Alex protocols [Cate, 1992] represents a weak consistency model. In this model, the TTL of a document is regulated by monitoring its life time. It uses a statistical property of file life time distribution. This distribution tends to be bimodal if a file has not been altered for a comparatively long time. The TTL dimension of an object is computed as a fraction of the document’s current age. The current age of a document is calculated by subtracting the earlier time of modification of the document from the prevailing time. Earlier studies have shown that the probability of stale documents occurring in adaptive TTL is about 5% [Cate, 1992].

Previous studies show that TTL-based caching consistency strategy is suitable for ad hoc networks [Fan, 2011]. In a TTL-based strategy, location of the cache nodes may not be tracked by the data server. Therefore it is not necessary for the data source to keep track of the locations of cache nodes. This makes it adaptable to the dynamic nature of ad hoc networks, where the mobile nodes join or leave the set of cache nodes at any time. The validity of the cached data item is determined autonomously and the validation of stale data can be performed either by data server or by the neighboring nodes having valid copies. The TTL based consistency also ensures flexibility. Different TTL values can be used for different consistency requirements that may vary with the nature
4.5  The Proposed Scheme

The replacement algorithm illustrated here considers the latest reference times for a data item and gives more importance to data items that are referenced more than once. The requested data items are divided into two categories: data items requested only once and data items referenced more than once. If we have data items referenced only once then that set is given priority for replacement. For this LRU policy is used. If an item is referenced more than once, the inter request arrival time (IRT) between the recent two references is considered for eviction. For each data item, IRT is taken as the recorded history information. IRT refers to the time interval between the last and penultimate (second to last) reference of a data item. It is assumed that if the IRT of a data item is large, the data is not likely to be referred in the near future.

Let $t_c$ be the current reference time and $t_r$ be the penultimate reference time then IRT, the inter request arrival time is given by,

$$\text{IRT} = t_c - t_r.$$ 

For items with one reference $t_r$ is taken as $\infty$.

If $t_c - t_r = \infty$, an item with least $t_c$ is replaced.

If $\text{IRT} \neq \infty$, then the replacement decision is made on the value of $K(i)$ which is given as,

$$K(i) = \max \sum_{i=1}^{n} t_c - t_r.$$
The data item with maximum IRT is considered for replacement.

4.6 Cache Consistency and Cache Invalidation

In cooperative caching, nodes share the cache contents of neighboring nodes to utilize the full advantage of caching. In order to improve the content diversity in the cooperative cache, the cache admission control scheme does not cache any data coming from the neighboring nodes. This increases the availability of information for the user, as more data items are cached and also avoids additional request to the server. Weak consistency model is used to maintain cache consistency. Each data item is associated with a TTL field which contains the allowed caching time and a time stamp. Data is considered as valid if the sum of TTL and time stamp is greater than the current time.

4.7 Simulation Scenarios and Metrics

In this work, to evaluate the performance of E-LRU the following network model is used. The nodes are randomly placed in the simulation area. Each node is identified by a node id and a host name. The data server is implemented as a fixed node in the simulation area. The data server contains all the data items requested by the mobile nodes. The nodes in the network move randomly based on a random path. The nodes within a transmission range of 100m are taken as the neighboring nodes. The nodes that generate data request are selected randomly and uniformly. Each mobile node generates a single stream of read only queries. After a query is sent out, the client does not generate new query until the pending query is served. The data access pattern follows a Zipf distribution
[Breslau, 1999] with a skewness parameter $\theta$ as 0.8. The simulation is implemented in JAVA.

4.7.1 Performance Metrics

Two metrics are taken for performance evaluation: cache hit ratio and average number of requests served. The evaluation of these parameters is done by varying the cache size of the mobile nodes. The hit ratio is defined as the percentage of requests that can be served from previously cached data. Since the replacement algorithm decides whether to cache the data or not, it affects the cache hits of future requests. The number of requests served for a particular period of time is taken. Average number of request is the average number of requests served for a particular period of time.

4.8 Performance Evaluation

The performance of the proposed E-LRU is compared with LRU for different cache sizes. Different cache sizes were used, ranging from 10% to 60% of the total size of the database. Fig 4.1 and Fig 4.2 show the result with a set of 6 cache sizes which are 10%, 20%, 30%, 40%, 50% and 60% of the total size of the database, respectively. Fig 4.1 depicts the comparison of cache hit ratio for different cache sizes. This figure shows that cache hit ratio of E-LRU is more for all cache sizes than LRU. At small cache sizes E-LRU shows significant improvement in cache hit ratio compared to LRU. For large cache sizes the difference in hit ratio of the two policies became less significant. Fig. 4.2 is the comparison of average number of requests served for different cache sizes and it shows that the
Figure 4.1: Cache Hit Ratio

Figure 4.2: Average number of requests
average number of request is higher for E-LRU than LRU. This is due to the fact that E-LRU uses the cache space more effectively and the number of data requests send to the server is reduced.

4.9 Conclusion

This chapter explored the cache replacement issues for ad hoc networks and presented a new cache replacement policy for cooperative caching. The proposed algorithm takes into account the inter-arrival time of recent requests for data items for replacement. In LRU only the last time of reference is taken and the numbers of references are not considered. Since the inter-arrival time of the recent request is taken more preference is given to the data items that have been accessed more than once. Hence we are able to distinguish between data that are frequently referenced with that of occasionally referenced. Since the algorithm is based on a key based approach it is simple to implement. Experimental results show that the proposed replacement algorithm can significantly improve the cache hit ratio and lower the data access delay when compared to LRU.
4.9. Conclusion