Journal of
Computer Science & Systems Biology

Harish and Thomas, J Comput Sci Syst Biol 2015, 8:1
DOI: 10.4172/jcsb.1000169

Research Article Open Access

A Novel Approach to Enhance the Efficiency of Distributed Cooperative Caching in SWNETs

Polanki Harish[1]* and Wilson Thomas[2]

[1]*Research Fellow, Computer Science and Engineering, Madanapalle Institute of Technology and Science, Andhra Pradesh, India*
[2]*Professor, Computer Science and Engineering, Madanapalle Institute of Technology and Science, Andhra Pradesh, India*

Abstract

This project helps to improve Performance of Distributed Cooperative Caching by choosing appropriate object replacement algorithms in the Social wireless networks (SWNET). E-Object caching in such SWNETs are shown to be able to minimize the Object provisioning and maintenance cost which is based on the pricing and service dependences among different stakeholders including network communication service providers, Object providers (Server) and End users. Here we consider replacement of objects with respect to two dimensions. One is Frequency and another is Recency (freshness) factor of object requests. The efficiency of the algorithm lies in choosing which items to discard to make room for the new ones. So we use the concept of knapsack problem with respect to frequency and freshness of the objects. So nodes maintain their cache memories with more frequent and latest objects.

Keywords: Knapsack; 2-Dimensional cooperative caching; Sharing; SWNET; Freshness

Introduction

Social Wireless Networks are wireless networks where individuals with similar likes gather together and connect with one another through their mobile nodes. Much like mobile social networking and web-based social networking occurs in virtual communities. The list of data applications may be e-books, magazine viewers and mobile phone Apps [1]. The data items may be anything like an music file, video clip, e-book, PDF files, mobile app's, software, etc., One of the example of mobile app explosion is, as of January 2014, Apple's App Store offered over 10,20,000 applications those can be accessed by smart phone users. When users carrying mobile devices physically gather in settings such as work place, University campus, Airports, Shopping Mall, hotels and other public places, Social Wireless Networks (SWNETs) can be formed using ad hoc wireless connections between the nodes. With the existence of such SWNETs, In such networks, the object is initially searched in local social wireless Network instead of directly downloading it from the Content Provider (CP)s server [2]. If the content is found in local SWNET, it will be sent to requesting node as shown in Figure 1. The content provisioning cost of such an approach was proved to be lower as it avoids download and usage of Internet resources. This mechanism is termed as cooperative caching [3].

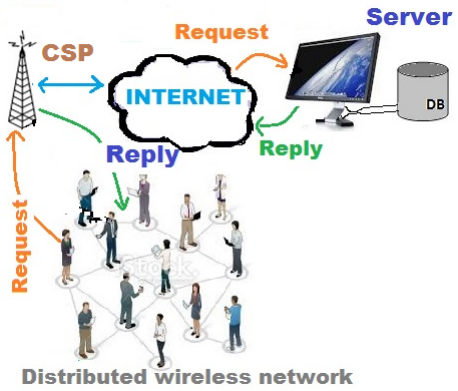

Figure 1: Content sharing and access in SWNET's.

In order to share the objects in SWNET, the end-users are to be encouraged to cache previously downloaded content by paying the rebate as shown in Figure 2. This mechanism can serve as motivation to the end-consumers for their energy and storage costs. This node-to-node rebate must be smaller than the content download cost paid to the Communication Service Provider (CSP). This rebate should be factored in the content provider's overall cost. For the measurement and arrangement of objects in cache, we are using the ZipF distribution, which says that with increase of population of object references, the rank of object is going to decrease. According to ZipF's law, the frequency of Objects is inversely proportional to its rank in the frequency table. The objects with highest popularity are given least rank.

To minimize the object maintenance and provisioning cost in the

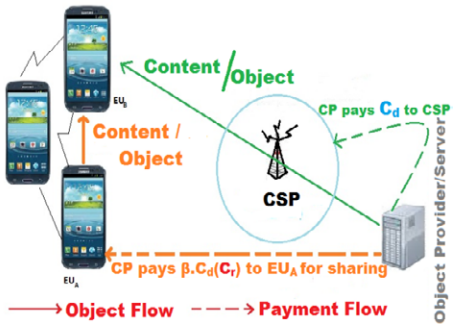

Figure 2: Price and Content Flow Model.

***Corresponding author:** Harish Polanki, Computer Science and Engineering, Madanapalle Institute of Technology and Science, Madanapalle-517325, Andhra Pradesh, India, Tel: 9154834147; E-mail: polankiharish@gmail.com

Received November 03, 2014; **Accepted** December 19, 2014; **Published** January 07, 2015

Citation: Harish P, Thomas W (2015) A Novel Approach to Enhance the Efficiency of Distributed Cooperative Caching in SWNETs. J Comput Sci Syst Biol 8: 045-048. doi:10.4172/jcsb.1000169

cache memory of mobile node, and to provide efficient provisioning and maintenance of objects, we are concentrating in using an optimal and appropriate object replacement algorithm. Here we consider the replacement of objects using the concept of Knapsack problem [4]. Knapsack problem says that fill the bag with objects having more value and less weights if there exists excess number of objects [5]. Here the Knapsack Concept uses 2 factors. First one is maintaining high Frequency of object requests in unit time and another is Recency (freshness) factor of object requests, i.e., keeping the objects which are recently requested. So nodes maintain their cache memories with more frequent and latest objects leading to more efficient results. This reduces cache miss rate, battery usage and unnecessary loading and removal of objects. Thus, the traffic and object maintenance cost is reduced by keeping more popular objects.

Literature Survey

We conducted literature survey on several object replacement algorithms and presenting the most appropriate one. One of the important performance factors of Web cache is the replacement strategy. There is huge number of methods for cache replacement. Some of them are

- Frequency-number of requests to an object,
- Recentness-time of the last reference to the object,
- Cost-cost to fetch an object from its origin server,
- Size-size of the Web object,
- Modification time-time of last modification [6].

As we conducted the study on replacement algorithms, the recency factor [**LRU** algorithm] is best suitable among the above. The recency factor is best suitable for situations having less cache memory [6,7]. The LRU (least recently used) algorithm gives best results and **LFU** (Least frequently used) algorithm [8] also gave approximately the same/less results as LRU [2,9]. The existing system uses either of above in cooperative caching replacements.

The Problem with these algorithms is that these two algorithms are single dimensional [10]. Both consider either frequency [LFU] factor or Recency factor [LRU]. The problem with least recently used (LRU) algorithm is it does not consider about the frequency of the objects, i.e., how many times that particular object was referenced. In the recently used one, according to the Belady's anomaly [11], the more objects the memory has in the recent time, the fewer object faults/hit miss a program will get. The problem with LFU algorithm is it does not consider when the objects are requested, it only considers the frequency of objects.

But in the situations like cooperative caching in SWNETs, we have to consider different aspects/dimensions like size of objects, frequency of each object requests, recentness of objects access, cost of objects, etc.,. But, considering all the aspects for provisioning appropriate object is leading to high performance issues [4,12]. So we consider only most useful aspects. The cost factor will not play big role with respect to object provision cost in network. The factors size, frequency and recentness play a vital role. But consider these three aspects which lead to performance problem in knapsack algorithm [13]. So, we consider the most essential factors frequency and recentness for caching more appropriate objects in the cache.

Objectives:

- To increases the Efficiency and decrease the object provisioning costs.
- To reduces cache faults, and unnecessary loading and removal of objects.
- To reduce the traffic in SWNET.
- To keep the latest and more appropriate popular objects in cache

Proposed System

After analyzing the problems associated with existing replacement algorithms [LRU, LFU] which considers only single aspect like frequency, freshness, etc., we are presenting a 2-dimensional replacement algorithm by using the concept of 0/1 knapsack problem. Knapsack problem says that fill the knapsack bag with objects such that they are having less weight and more value [5].

Here, we apply the concept of knapsack concept to cooperative caching with respect to frequency of object requests in unit time and recent access of objects [14-16]. The Figure 3 shows the algorithm for the knapsack problem. It selects the objects such that, objects that are recently accessed and having high request rate/frequency in unit time. On other way, by calculating probability of each individual objects, we can shortlist the object having higher frequency and recently accessed as shown in flow chart displayed in Figure 4.

In the above algorithm, the weights in knapsack concept are taken as freshness (time), t .i.e., very recently accessed and the values is assumed as frequency, f. The capacity of knapsack bag is cache capacity C and n is the number of distinct objects.

Implementation

We use a data structure, namely item, having 2 fields (frequency and freshness) to represent each item/object. Then we use an array of type item to store all objects in it, as shown in below Figure 5.

We implemented the work presented above using network simulator (ns2) tool having 30 nodes with existing and proposed replacement algorithms. Every node is having its own cache memory of each cache capable of storing up to 8 objects. The cost of download

```
Algorithm:
 INPUT: f //frequency
        t// recentness(time)
        C// Cache capacity
FOR y from 0 to C DO
  m[0, y] := 0
END FOR
FOR x from 1 to n DO
  FOR y from 0 to C DO
   IF (t[x] <= y)
     m[x, y] := max(m[x-1, y], m[x-1, y-t[x]] + f[x])
   ELSE
     m[x, y] := m[x-1, y]
   END IF
  END FOR
END FOR
```

Algorithm-1: 0-1 knapsack concept for object replacement in SWNETs

Figure 3: Knapsack Algorithm.

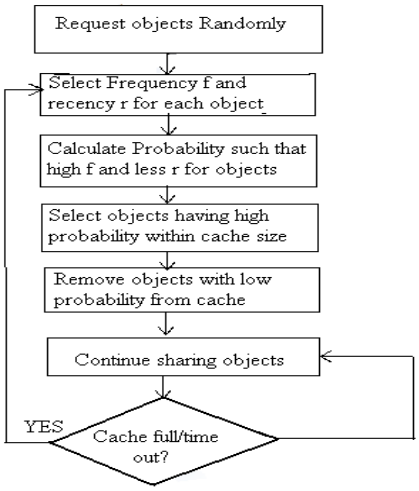

Figure 4: Flowchart for object replacement using 0-1 knapsack concept.

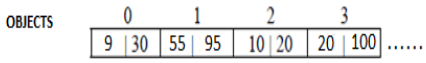

Figure 5: Objects with frequency and freshness.

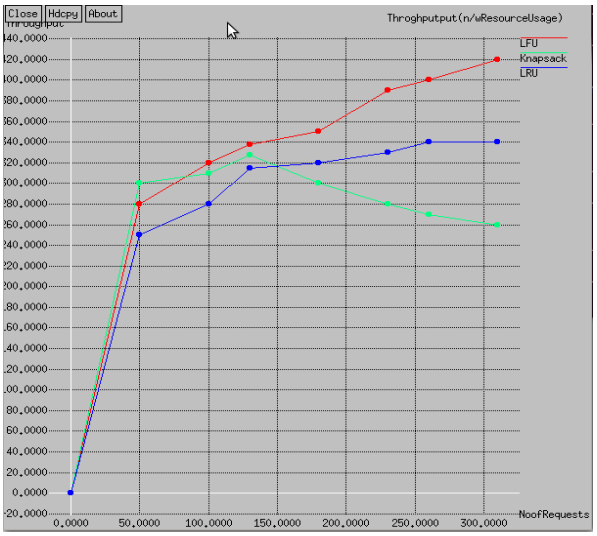

Figure 6a: Throughput.

per each object from CP is set as 10, the rebate-to-download cost set between 0 and 10 based on location of object, the number of objects in the SWNET are 250, and total no. of requests for objects are 300 requests. We performed the study on 3 algorithms LRU, LFU and Knapsack algorithm, and observed different parameters like Throughput, Energy usage by nodes, CPU usage and flow rate.

Results and Comparison

The results are shown in below graphs. Figure 6a shows the usage of network resources like bandwidth, cpu cycles and node energy versus increasing object requests. As Knapsack algorithm is little complex than

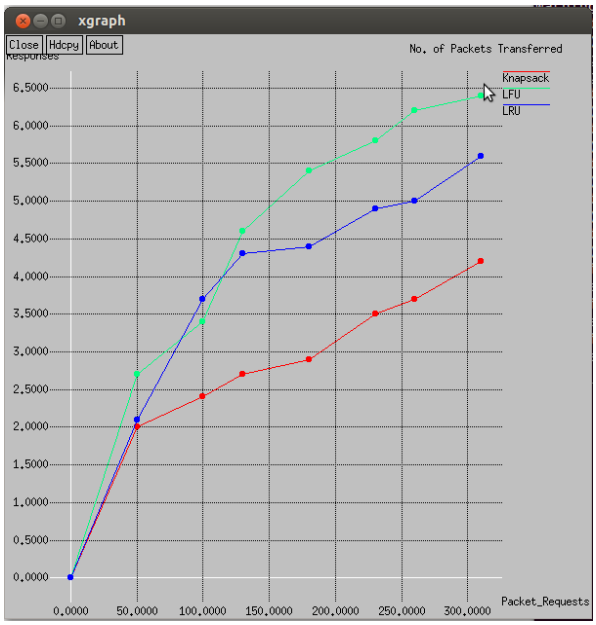

Figure 6b: Responses/Traffic as object Requests.

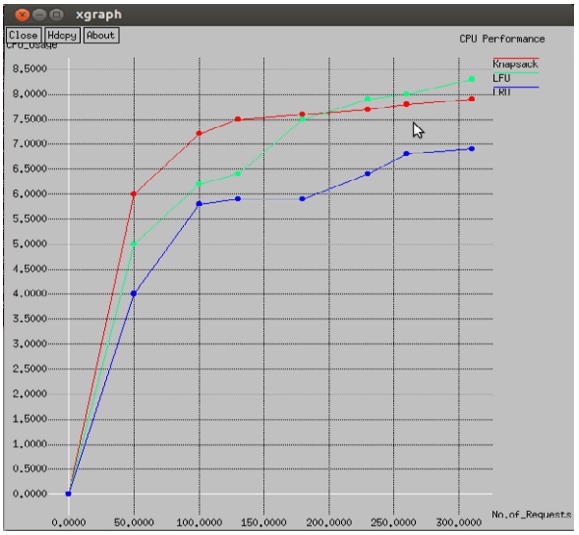

Figure 6c: CPU cycles usage.

others the cpu and battery usage is more initially. As time passes and number of requests increases, the usage of bandwith and traffic is more in LRU and LFU due to more number of cache faults. So the knapsack gives the best throughput as number of requests are increasing.

The below graph in Figure 6b shows the results of Response i.e., how many packets transferred for increasing requests for different algorithms. As knapsack algorithm maintains the appropriate objects from beginning, the number of responses in 2-dimensional knapsack algorithm is less than others.

Figure 6c shows the usage of cpu cycles versus increasing object requests. As Knapsack algorithm is little complex than LRU and LFU, the cpu usage is more initially. As time passes and number of requests are increases the usage of cpu is more in LRU and LFU due to more number of cache faults. So the knapsack uses cpu more initially and

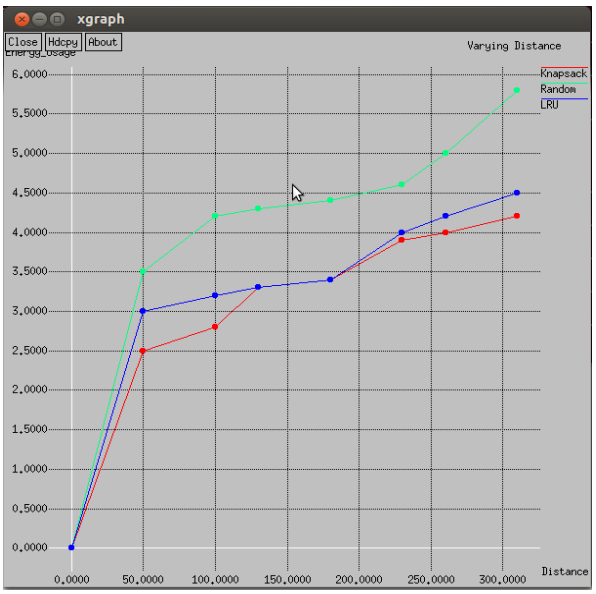

Figure 6d: Energy usage in nodes vs Distance between nodes.

Factor	LFU	LRU	Knapsack
Throughput	Good	Very Good	Excellent
Traffic	More	Average	less
CPU usage	More	Less	More at initial
Energy usage	More	Less	Less

Table 1: Comparison of factors.

as number of requests are increases, the cpu usage is less compared to LRU and LFU algorithms.

The below graph shows the results of Energy usage as distance between nodes increases for different algorithms. As knapsack algorithm maintains the appropriate objects from beginning and as the number of responses in 2- dimensional knapsack algorithm is less than others, the energy usage of nodes is also less due to less traffic, responses.

Finally, The below table compares the different factors like throughput, traffic, energy and CPU usage for 3 algorithms (Table 1).

By analyzing above results, we noticed that 2-dimensional knapsack algorithm in view of frequency and Recency factor is giving better results than existing one dimensional algorithms.

Conclusion and Future Work

The goal of this work is to reduce object provisioning cost by demonstrating 2-dimensional cooperative caching using knapsack algorithm. By analyzing the above results, we conclude that the 2-dimensional cooperative caching is providing the good and more accurate results than in the previous one dimensional one. The knapsack problem gave the efficient results than LRU and LFU for fixed time intervals of cache updates. The future work corresponds to replacements of objects with respect to flexible time intervals. i.e., cache should be updated when the congestion happens.

References

1. (2014) Mobile social network.
2. Yin L, Cao G (2006) Supporting Cooperative Caching in Ad Hoc Networks. Mobile Computing 5: 77-89.
3. Charles O, Mahmoud T, Eric T, Subir B, Kristopher M (2013) Distributed Cooperative Caching in Social Wireless Networks. Proc IEEE Trans mobile computing 12: 1037-1053.
4. Pisinger D (2005) Where are the hard knapsack problems? Computers and Operations Research 32: 2271-2284.
5. (2014) knapsack-problem.
6. Boszormenyi L, Podlipnig S (2003) A Survey of Web Cache Replacement Strategies. ACM Computing Surveys 35: 374- 398.
7. (2013) Least recently used.
8. (2013) Least frequently used.
9. Jongmoo C, Donghee L, Jong-Hun K, Noh SH, Sang Lyul M, et al. (2001) LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Transactions on Computers 50: 1352-1361.
10. Voelker M, Wolman A, Karlin A, Levy H (1999) On the Scale and Performance of Cooperative Web Caching. Proceedings of the seventeenth ACM symposium on Operating systems principles 16-31.
11. (2014) Belady's Anamoly.
12. Caccetta L, Kulanoot A (2001) Computational Aspects of Hard Knapsack Problems. Nonlinear Analysis 47: 5547-5558.
13. Martello S, Toth P (1990) Knapsack Problems: Algorithms and Computer Implementation. John Wiley and Sons, USA.
14. Maya H, Dipti S (2004) Solving the 0-1 Knapsack Problem with Genetic Algorithms 3rd International Conference on Communication Software and Networks (ICCSN) 591-595.
15. Plateau G, Elkihel M (1985) A hybrid algorithm for the 0-1 knapsack problem. Methods of Oper Res 49: 277-293.
16. Breslau L, Cao P, Fan L, Shenker S (1999) Web Caching and Zipf-Like Distributions: Evidence and Implications. Proc IEEE INFOCOM 1: 126 - 134.