Towards Optimal Cooperative Caching in Social Wireless Networks

Mahmoud Taghizadeh, Anthony Plummer Jr., Ali Aqel and Subir Biswas
Department of Electrical and Computer Engineering, Michigan State University, USA

Abstract—This paper introduces an optimal cooperative caching policy for minimizing electronic content provisioning cost in Social Wireless Networks (SWNETs). The SWNETs are typically formed by a collection of mobile devices, such as data enabled phones, net-books, electronic book readers etc., sharing common interests in electronic content, and physically gathering in settings such as University campuses, work places, malls, airports, train stations and other public places. Electronic object caching in such SWNETs are shown to be able to reduce the content provisioning cost which depends heavily on the service and pricing dependencies among various stakeholders including the content provider, the network service provider, and the end consumers. Drawing motivation from Amazon’s Kindle electronic book delivery business, this paper develops a practical network, service, and pricing model which are then used for creating the proposed optimal caching strategy. In addition to proving the optimality of the mechanism, the paper constructs extensive analytical and simulation models for analyzing the proposed caching strategy and its optimal operating points for the mobile ecosystem stakeholders.

Index Terms—Social Wireless Networks, Cooperative Caching, Content Provisioning, Ad Hoc Networks.

I. INTRODUCTION

Unprecedented penetration of data enabled mobile devices and wireless-enabled data applications have recently emerged. A list of such devices include Apple’s iPhone, Google’s Android™ and Microsoft’s Windows Mobile™ powered smart phones, Amazon’s Kindle™, and book readers from other vendors. The array of data applications includes electronic books and magazine readers and mobile phone Apps. As of October ’09, Apple’s Appstore offered over 85,000 Apps that are individually downloadable by the smart phone users.

With the conventional download model, a user downloads objects directly from the Content Provider’s (CP) (e.g. Amazon) server over a Communication Service Provider’s (CSP) (e.g. Sprint) network. Now consider a Social Wireless Networks (SWNET) that can be formed by ad hoc wireless links among data-enabled wireless devices in University campuses, work places, malls, airports, train stations, and other public places. With an SWNET, an alternative download model would be to search for the object (i.e. a book) within the SWNET first, before downloading it from the CP’s server, since the later would require the CP to pay a download cost to the CSP. This is currently the Amazon Kindle™ model in which Amazon, the CP, pays to Sprint, the CSP, for the cost of network usage due to downloaded books by Kindle™ users. This motivates cooperative caching within the SWNET so that the effective cost of object provisioning to the content provider can be reduced by avoiding expensive from-server downloads.

Such a cooperative caching model will need a rebate mechanism so that whenever an end-consumer (e.g. a Kindle owner) supplies a cached object to another end-user over the SWNET, it should get a rebate from the CP. Such rebates will serve as an incentive so that the end-consumers are enticed to participate in spite of the storage and energy costs leading to shorter battery life. This rebate should be factored in the Content Provider’s overall cost. This rebate can also be distributed among the provider end-consumer and the end-consumers of all the intermediate mobile devices that take part in object forwarding within the SWNET.

For objects with varying level of popularity, a greedy caching approach for each node would be to store as many distinct popular objects as its storage allows. This will amount to non-cooperation and give rise to heavy object duplications across the network. In the other extreme case, which is fully cooperative caching, a node would try to maximize the total number of unique objects stored within the SWNET by avoiding object duplications. However, we will show that none of the above approaches can minimize the content provider’s cost. In fact, for a given rebate-to-download-cost ratio, there exists an optimal caching policy which is somewhere in between those two extremes, and can minimize the content provider’s cost by striking a balance between the greediness and cooperation.

The overall contribution of this paper is to determine and analyze the best object placement in a SWNET for minimizing the object provisioning cost. The specific contributions of this paper are threefold. First, a stochastic model for the content provider’s cost computation is developed using a practical service and pricing model. Second, a cooperative caching strategy, Split-λ, is proposed, numerically analyzed, and theoretically proven to be optimal. Finally, the numerical results and their trends for Split-λ are validated using ns2 simulation and compared with a series of traditional caching policies.

Figure 1: Object access from a SWNET in a University campus

A. Network Model

Fig. 1 illustrates an example SWNET within a University campus. End Consumers (ECs) carrying mobile devices form the...
SWNET partitions, which can be either multi-hop ad hoc network (i.e. MANET) as shown for partitions 1, 3, and 4, or single hop access point based as shown for partition 2. A mobile device can download an object from the CP’s server using the CSP’s cellular network, or from the local SWNET partition. In this paper, we have modeled objects such as electronic books, music, etc., which are time non-varying, and therefore cache consistency is not explored in this work.

We considered two types of SWNETs. The first type involves in stationary [1] SWNET partitions. Meaning, after a partition is formed, it is maintained for sufficiently long so that the cooperative object caches can be formed and reach steady states. After the optimality and performance of the proposed caching mechanism is evaluated for this stationary case, we investigate the second type to explore as to what happens when the stationary assumption is relaxed. To investigate this effect, caching is applied to SWNETs formed using human interaction traces obtained from a set of real SWNET nodes [2].

B. Pricing Model

We use a pricing model similar to the Amazon KindleTMbusiness model in which CP(e.g. Amazon) pays a download cost \( C_d \) to the CSP when an End-Consumer(EC) downloads an object from the CP’s server through the CSP’s cellular network. Also, whenever an EC provides a locally cached object to another EC within its SWNET partition, the provider EC is paid a rebate \( C_r \) by the CP. This rebate can also be distributed among the provider EC and the ECs of all the intermediate mobile devices that take part in object forwarding. The primary objective is to design an optimal caching policy so that for a given \( \frac{C_r}{C_d} \) ratio, the provisioning cost for each object should be minimized.

Note that these cost items, namely, \( C_d \) and \( C_r \), do not represent the selling price of an object (e.g. book). The selling price is directly paid to the Content Provider (e.g. Amazon) by an End Consumer (e.g. a Kindle user) through an out-of-band secure payment system. \( C_d \) corresponds to the CP’s object delivering cost when it is delivered through the CSP’s network, and \( C_r \) corresponds to the rebate given out to an EC when the object is found within the SWNET.

A digitally signed rebate framework needs to be supported so that the rebate recipient ECs can electronically validate and redeem the rebate with the CP. Also, a digital usage right mechanism [3] is needed so that an EC which is caching an object (e.g. a book) should not necessarily be able to open/read it unless it has explicitly bought the object from the CP. We assume the presence of these two mechanisms on which the proposed caching mechanism is built.

C. Request Generation Model

We used a server-tagged object popularity [4] and a Zipf distribution [5] based object request model. This model is widely used in the literature for modeling popularity based online object request distributions. According to Zipf law, the popularity of the \( i^{th} \) popular object out of an N-object pool can be expressed as:

\[
p_i = \frac{\Omega}{i^\alpha} (p_1 > p_2 > \cdots > p_N), \text{where } \frac{1}{\Omega} = \sum_{i=1}^{N} \frac{1}{i^\alpha} \quad (1)
\]

and \( \alpha \) is a Zipf parameter that determines the skewness coefficient in request pattern. The quantity \( p_i \) indicates the probability that an arbitrary request is for the \( i^{th} \) popular object. As \( \alpha \) increases, the access pattern becomes more concentrated on the popular data items.

After an object request is originated by a mobile device, it first searches its local cache before searching within the local SWNET partition. If the above searches fail, the object is downloaded from the CP’s server using the CSP’s 3G/4G cellular network.

III. COST OF CONTENT PROVISIONING

At steady state, let \( P_L \) be the local hit rate, representing the probability of finding an object within a device’s local cache. And let \( P_V \), the SWNET hit rate, represent the probability of finding an object in the local SWNET partition. The parameter \( P_{LV} \) represents the probability of finding an object within a device’s local cache as well as in the SWNET partition. The overall miss rate \( P_M \) can be written as:

\[
P_M = 1 - P_L - P_V + P_{LV} \quad (2)
\]

The cost of object provisioning is: a) zero, when it is found locally, b) \( C_r \), when found in an EC in the SWNET, or c) \( C_d \), when downloaded from the CP’s server. The expected provisioning cost:

\[
Cost = (P_V - P_{LV})C_r + P_M C_d \quad (3)
\]

Substituting \( P_M \) from Eqn. 2 and letting \( C_r/C_d \) as \( \beta \), the cost can be expressed as:

\[
Cost = (1 - (1 - \beta)(P_V - P_{LV}) - P_L)C_d \quad (4)
\]

Let \( V \) be the number of devices within a partition, and \( S_j \) be the set of objects stored in device-\( j \) \((1 \leq j \leq V)\). With \( p_i \) \((1 \leq i \leq N)\) as defined in Eqn. 1, the probability of finding an object in device-\( j \)’s cache can be written as \( P_{LV}^j = \sum_{i \in S_j} p_i \).

The resulting probability of finding the object at any given device in the partition is \( \frac{1}{V} \sum_{j=1}^{V} P_{LV}^j \) or \( \frac{1}{V} \sum_{j=1}^{V} \sum_{i \in S_j} p_i \). This is the local hit rate \( P_L \), and can be simplified as:

\[
P_L = \frac{1}{V} \sum_{i=1}^{N} n_i p_i \quad (5)
\]

where \( n_i \) represents the number of copies of object-\( i \) within the partition. For a partition size, object population, and popularity distribution \( V, N, \) and \( p_i \), the \( P_L \) depends on \( n_i \) \((1 \leq i \leq N)\) which is determined by the adopted cooperative caching strategy.

If \( C \) is the available cache size (i.e. the number of objects that can be stored) at each mobile device, then the maximum number of objects that can be stored within a SWNET partition is \( VC \). With the Zipf popularity based object request model as presented in Section II, any meaningful cooperative caching approach must ensure the following constraint at steady state.

\[\text{Popularity storage constraint: An object should not be cached in a partition when at least one object of higher popularity is missing in the partition. Meaning, object } i \text{ cannot be cached while object } i-k(k=1,2,\cdots,i-1) \text{ is missing. With this constraint:}
\]

\[
P_L = \frac{1}{V} \sum_{i=1}^{VC} n_i p_i \quad (6)
\]

Let \( S \) represent the set of all stored objects in a partition, the probability of finding an object in the partition can be expressed...
as \( \sum_{i=S} p_i \), or \( \sum_{i=1}^T p_i \), where the \( T \)th popular object is the least popular one stored in the partition. The quantity \( \sum_{i=1}^T p_i \) represents the cache hit rate or \( (1-P_H) \). Substituting \( \sum_{i=1}^T p_i \) for \((1-P_H)\) and \( P_l \) from Eqn. 6 in Eqn. 2, we will have \( P_l - P_l V = \sum_{i=1}^T p_i - \frac{1}{V} \sum_{i=1}^C n_i p_i \). Using Eqn. 4 cost expression can be written as:

\[
Cost = (1 - (1 - \beta) \sum_{i=1}^T p_i - \beta \frac{1}{V} \sum_{i=1}^C n_i p_i) C_d
\]  

(7)

IV. OPTIMAL OBJECT PLACEMENT

Consider a caching state in which all \( n_i = 1, (1 \leq i \leq V) \). It means the entire \( VC \) storage in a partition is filled up with the most popular objects, and there is no object duplication. The minimum deviation from this state corresponds to object duplication by replacing an object by another that is already present in the partition. The objective is to identify the evictee and the evictor so that the overall cost in Eqn. 7 reduces after that replacement which is guaranteed to cause duplication.

The second term in Eqn. 7 reduces after any such replacement, since the summation \( \sum_{i=1}^T p_i \) decreases due to the removal of the quantity \( p_{evictee} \) from the summation. Since this reduction increases the overall cost, the goal is to minimize this reduction by evicting the object with smallest \( p_i \). Assuming that the replacement complies with the popularity storage constraint, as introduced in Section III, the popularity of the evictor (i.e. duplicating) does not increase amount of \( \sum_{i=1}^T p_i \), since \( p_{evictor} \) was already there in the summation before the replacement operation. After a replacement, the summation component of the third term in Eqn. 7 becomes \( \sum_{i=1}^{VC} n_i p_i - p_{evictee} + p_{evictor} \). So, the value of this summation can be increased by replacing an object by a more popular one. Since any increase in this summation would reduce the cost, the best duplication policy is to replace the least popular object by the most popular one. Since the number of copies of an object cannot be greater than the number of devices \( V \), after duplicating the most popular object at all devices, the second most popular object must be duplicated till its count reaches \( V \). This duplication process with the above replacement strategy should continue in this manner.

Considering the above effects of object duplication on the second and the third terms of Eqn. 7, it can be concluded that the best replacement policy for cost reduction is to replace the least popular object available in the partition by the most popular object that has less than \( V \) copies in the partition.

The next question is that having started from the no-duplication initial state of \( n_i = 1(1 \leq i \leq V) \), at which point the above iterative duplication/replacement process should stop. The replacement strategy causes the second term in Eqn. 7 to go down and the third term to go up. For given \( C_d \), \( \beta \), \( V \), and \( C \), when the increase of the third term dominates the decrease of the second term, the overall cost goes down. Otherwise it goes up. Therefore, the iteration should stop exactly when replacing the available least popular object by the most popular object with less than \( V \) copies actually increases the overall cost.

From the above discussion we conclude that in order to minimize object provisioning cost, certain number of objects should be duplicated in all devices and the remaining space in the partition should be filled with unique objects.

V. OPTIMAL CACHING REPLACEMENT POLICY

A. Split – \( \lambda \): Replacement Policy

To realize the optimal object placement as described in section IV we propose \( Split – \lambda \) policy in which the available cache space in each device is divided into a duplicate area (\( \lambda \) fraction) and a unique area (see Fig. 2). At steady state, all devices’ caches maintain the same object set in their duplicate areas.

For given \( C_d \), \( \beta \), \( V \), and \( C \), the factor \( \lambda \) needs to be dimensioned such that ”the least popular duplicated object” in the partition is adjusted to the value that minimizes the cost. When \( \lambda = 0 \), then for all \( i \), \( n_i = 1 \), indicating no duplication in the partition. This situation is referred to as exclusive caching, in which a device cannot cache an object that is retrieved from the \( SWNET \). The other extreme is when \( \lambda = 1 \), in which case all devices store the \( C \) most popular objects. Meaning, all \( C \) objects are duplicated across the devices in the partition.

Algorithm 1: Object replacement logic for the CSC policy

if (the new object comes from another device in the Domain) then

 candid = the least popular object in the "duplicate" area of the local cache;

else: /* object comes from outside of domain */

 candid = the least popular object in the entire cache;

end

if (new_object.popularity > candid.popularity) then

 replace candid with the new object;

else

 do not cache the new object;

end

B. Split – \( \lambda \): Provisioning Cost

Let \( f(k) \) be the probability of finding an arbitrary object within a device’s cache that is filled with the \( k \) most popular objects. \( f(k) \) can be expressed as \( \sum_{i=1}^k p_i \). Substituting \( p_i \) for the Zipf distribution (see Section III), \( f(k) = \sum_{i=1}^k p_i \approx \frac{k}{\Omega} = \frac{k^{1-\alpha}}{1 - \alpha} \). Now, considering \( \Omega = 1/\sum_{i=1} p_i \approx 1/\sum_{i=1}^{CN} p_i = \frac{1}{V^{1-\alpha}} \), we will have:

\[
f(k) = \frac{k^{1-\alpha} - 1}{N^{1-\alpha} - 1}
\]  

(8)

1) Local Hit Rate (\( P_L \)): Let \( \mu = VC(1-\lambda) \) be the number of unique objects in the partition. Therefore, the number of different objects in the partition is \( \lambda C + \mu \). With \( Split-\lambda \), the average local hit rate:

\[
P_L = f(\lambda C) + \frac{f(\lambda C + \mu) - f(\lambda C)}{V}
\]  

(9)

The two terms in Eqn. 9 correspond to the cache hits contributed by the objects in the duplicate and the unique areas respectively. The quantity \( f(\lambda C + \mu) - f(\lambda C) \) represents the hit rate contributed by the unique objects (in the partition) which are assumed to be uniformly distributed over all \( V \) devices’ caches.

2) \( SWNET \) Hit Rate (\( P_V \)): It is equal to the hit probability contributed by the objects stored in: a) local duplicate area, plus b) the unique area of all devices in the partition, minus c) the unique area of the local cache. This can be expressed as: \( f(\lambda C) + \)
Using Eqn. 8 to expand \( V_p \lambda \) when objects in the partition, which reduces with higher duplications (of unique objects in the partition. The case represents zero duplication, leading to the maximum number of unique objects within the local cache and the copies of the popular objects. Therefore, \( P_{LV} = f(\lambda C) \). Substituting this value of \( P_{LV} \), and \( P_L \) and \( P_V \) from Eqns. 9 and 10 in Eqn. 4, the cost can be simplified and written as:

\[
1 - f(\lambda C + \mu) + \beta V - \frac{1}{V} (f(\lambda C + \mu) - f(\lambda C)) \mid C_d \tag{11}
\]

Using Eqn. 8 to expand \( f(\lambda C + \mu) \) and \( f(\lambda C) \), Eqn. 11 can be written as a function of \( \lambda \). By equating the derivative of the cost expression to zero, we can compute \( \lambda_{\text{opt}} \) at which cost is minimized.

### VI. PERFORMANCE WITH STATIONARY PARTITIONS

#### A. Hit Rates and Optimal Cost

Performance of Split-\( \lambda \) for stationary partitions has been evaluated using the analytical expressions and also via ns2 network simulation. Performance with non-stationary partitions created using real human interaction traces, is presented in Section VII. A flooding based object search mechanism, as described in Section II, has been developed within ns2 using the baseline AODV [6] route discovery syntaxes. Parameters for the baseline experiments are summarized in Table 1.

| Number of ECs in a partition \( V \) | 40 |
| SWNET partition type | Stationary |
| Download cost \( C_d \) | 10 |
| Rebate-to-download-cost ratio \( \beta \) | \( 0 \leq \beta \leq 1 \) |
| Cache size in each mobile device \( C \) | 50 |
| Zipf parameter \( \alpha \) | 0.5 |
| Object population \( N \) | 5000 |
| Warm up phase to reach steady state | 2000 requests |
| Total simulation duration | 10000 requests |

*Table 1: Baseline simulation parameters*

Fig. 3(a) depicts the impacts of \( \lambda \) on the hit rates. The \( \lambda = 0 \) case represents zero duplication, leading to the maximum number of unique objects in the partition. The \( \lambda = 1 \) case causes maximum duplication. In this case, all nodes cache the same set of \( C \) (cache size) most popular objects. Smaller \( \lambda \) values lead to very few copies of the popular objects within the local cache and the subsequent low local hit rates. Since with larger \( \lambda \), more and more popular objects are duplicated, the likelihood of finding objects locally improves, leading to higher \( P_L \) values.

The miss rate \( P_M \) depends on the total number of unique objects in the partition, which reduces with higher duplications when \( \lambda \) is increased. Smaller \( \lambda \) values lead to less duplication and as a result \( P_M \) increases. The excellent agreement between the analytical and the simulation results for \( P_L, P_V, P_{LV}, \) and \( P_M \) in Fig. 3(a) indicates the correctness of the analytical formulations in Sections V.

Figs. 3(b) and 3(c) depicts the object provisioning cost as a function of \( \lambda \). When \( \beta = 0 \) (i.e. per object credit \( C_r = 0 \)), the cost expression in Eqn. 3 reduces to \( \text{Cost} = P_M C_d \). Meaning, for a given \( C_d \), the cost depends only on the miss rate \( P_M \) which reduces as \( \lambda \) reduces. Therefore, when \( \beta = 0 \), \( \lambda = 0 \) gives the minimum \( P_M \) and consequently the minimum cost. When \( \beta = 1 \), meaning the rebate is same as the download cost \( C_d \), the cost expression in Eqn. 4 reduces to \( \text{Cost} = (1 - P_L) C_d \), indicating that it depends only on the local hit rate for a given \( C_d \). This explains as to why the cost decreases with increasing \( \lambda \) where the local hit rate increases. Intuitively, when \( C_r = C_d \), there is no advantage of fetching objects from the SWNET. The only way to reduce cost in this situation is to maximize \( P_L \).

Observe in Fig. 3(c) that for both \( \beta = 0.5 \) and \( \beta = 0.7 \), the cost reduces initially by increasing \( \lambda \) but after a critical \( \lambda = \lambda_{\text{opt}} \), the cost starts to increase. This critical point can be found numerically from Eqn. 11. The reason for this \( \lambda_{\text{opt}} \) was explained in Section IV. It was established that starting from the state of zero partition-wide duplication, if the iterative duplication/replacement process stops at the correct point, the cost can be minimized. This translates to finding the appropriate level of duplication, which is decided by \( \lambda \). As shown in Fig. 3(c), \( \lambda_{\text{opt}} \) is 0.4 when \( \beta = 0.5 \), and it is 0.6 when \( \beta = 0.7 \). Thus, a larger \( \lambda_{\text{opt}} \) is needed when the rebate is larger with respect to the download cost from the CP’s server.

#### B. Comparison with Traditional Caching Policies

Fig. 4(a) shows the cost for Least Recently Used (LRU) [7], Least Frequently Used (LFU) [7], and Random (RNDM) [7] along with those for Split-\( \lambda \) with \( \lambda \) set to 0, 1, and \( \lambda_{\text{opt}} \). While LRU and LFU implicitly leverage object popularity by storing the most popular objects, RNDM policy is completely insensitive to object popularity. As expected, Fig. 4(a) depicts that Split-\( \lambda_{\text{opt}} \) provides the best cost. \( \lambda = 0 \) delivers near-best performance for small \( \beta \) values. This is because as shown in Eqn. 4, for small \( \beta \) (i.e. small rebate \( C_r \)), the cost depends mainly on \( P_M \). From Fig. 3(a), the miss rate is minimum for \( \lambda = 0 \), which corresponds to no-duplication exclusive caching.

When \( \beta \) is large (e.g. \( \beta > 0.7 \)), \( \lambda = 1 \) delivers near-best performance. This is because as shown in Eqn. 4, for large \( \beta \), the cost depends mainly on \( P_L \), which is maximized when \( \lambda = 1 \). All traditional policies perform in between Split-\( \lambda = 0 \) and Split-\( \lambda = 1 \). Since RNDM is insensitive to popularity, by uniformly distributing the objects in the partition, it is able to increase \( P_V \), which helps it outperform LRU and LFU for small \( \beta \). LFU, on the other hand, attempts to distinguish popular objects by keeping track of the number of hits for an object. This explains its performance proximity with Split-\( \lambda = 1 \).

#### C. Partition Object Density

As discussed in Section IV, in order to minimize cost, certain number of objects should be duplicated in all devices and the remaining space in the partition should be filled with unique objects. Therefore, the possible density values are \( V \) (number of ECs) for the duplicated objects, 1 for the unique objects, and 0 for the objects that are not stored in any node. This is confirmed in Fig. 4(b) which reports object density from simulation for different values of \( \beta \). With increasing \( \beta \), since \( \lambda_{\text{opt}} \) increases, more objects are duplicated, thus increasing the object density. We show results only up to object-id 50 (i.e. cache size \( C \)), because objects beyond \( C \) have only one or zero copy. When \( \beta = 0 \), since \( \lambda_{\text{opt}} \) is also zero, there is no duplication, causing the VC most popular objects to have one copy and the rest of the objects (i.e. \( N-VC \)) with zero copy.

Fig. 4(c) depicts simulated object densities for LFU, LRU, and RNDM. Certain amount of density skew (i.e. higher density...
for more popular objects) is generated by the Zipf based object requests, which favor more popular objects. Observe that for RNDM, the object densities are minimally skewed, since the policy itself is not at all sensitive to object popularities. LFU, on the other hand, shows a density pattern that is closest to the optimal Split-λ due to its effective sensitivity to object popularity. Similar to the cost results, the density pattern for LRU lies somewhere in between RNDM and LFU. This is because the effective sensitivity to object popularity for LRU is weaker than LFU, but stronger than RNDM.

VII. PERFORMANCE WITH NON-STATIONARY PARTITIONS

The stationary partition assumption is relaxed in this section. We evaluated Split-λ and the traditional policies on a dynamic 98-node SWNET formed by 98 individuals attending the INFOCOM ’05 conference [8]. We have extracted the SWNET partition dynamics from a pairwise interaction trace obtained from [2]. The trace contained synchronized time-stamped pair-wise individual interaction information with a granularity of 4 minutes, which is the Hello packet interval used by a small RF transceiver attached to all 98 individuals while attending the conference. Fig. 5(a) reports the extracted partition dynamics as the average partition size from individual nodes’ perspective. For example, at time 20, average partition size across all nodes is 12.

Fig. 5(b) depicts the simulated cost as a function of λ. Observe that the pattern in this graph is exactly the same as that observed for the stationary partition case in Fig. 3(c), indicating that the concept of optimal λ also holds for networks with dynamic partitions. Analytical computation of the λopt in this dynamic case, however, may not be as straightforward due to the wide variation of the partition size as shown in Fig 5(a). A heuristic approach would be to compute λopt for each node individually based on its own observed average partition size.

Also, unlike in the static case, it is relatively harder to keep consistency of duplication under the dynamic scenario. This is because when a node is in a small partition, it has to download a large number of objects from the CP’s server. In other word, from the standpoint of a node, which is in a small partition, those objects are unique. Later, when such a node enters a bigger partition, some of those unique objects may not remain unique anymore in the new partition. To avoid such situations, current partition size is stored along with the object in the cache and during cache replacement objects with smaller partition size are evicted before other objects.

Fig. 5(c) depicts that Split-λopt provides the best cost compared to the traditional policies even with dynamic SWNET partitions. Similar to the stationary case, λ = 0 and λ = 1 deliver near-best performance for small and large λ values respectively. Also note that the cost for all policies except Split-λopt grow linearly with β. This is because the quantities PL and PV grow linearly with β. Therefore, as seen in Eqn. 4, the cost is simply a linear function of β. One main difference between the dynamic and stationary network scenarios is that for the dynamic case, split policy with λ = 0 outperforms LRU and RNDM for all values of β. This is because as shown in Fig 5(a) in some durations of the experiment, the partition size is quite small. For small partitions, split with λ = 0 generates the best cost which reduces the average cost of the experiment.

VIII. RELATED WORK

There is a rich body of literature [9], [10] on several aspects of cooperative caching including object replacements, reducing cooperation overhead [11], and cooperation performance [9]–[11] in traditional wired networks. The Social Wireless Networks (SWNETs) explored in this paper, which are often formed using mobile ad hoc network protocols, are different in the caching context due to their additional constraints such as topological insatiable (see Fig. 5(a)) and limited resources. As a result, most of the available cooperative caching solutions for static networks are not directly applicable for the SWNETs.

Three caching schemes for MANET have been presented
in [12]. In the first scheme, CacheData, a forwarding node checks the passing-by objects and caches the ones deemed useful according to some pre-defined criteria. This way, the subsequent requests for the cached objects can be satisfied by an intermediate node. A problem with this approach is that storing large number of popular objects in large number of intermediate nodes does not scale well.

The second approach, CachePath, is different in that the intermediate nodes do not save the objects; instead they only record paths to the closest node where the objects can be found. The idea in CachePath is to reduce latency and overhead of cache resolution by finding the location of objects. This strategy works poorly in a highly mobile environment since most of the recorded paths become obsolete very soon. The last approach in [12] is the HybridCache in which either CacheData or CachePath is used based on the properties of the passing-by objects through an intermediate node. While all three mechanisms offer a reasonable caching solution, it is shown in [13]—[15] that relying only on the nodes in an object’s path is not the most efficient approach. Using a limited broadcast based cache resolution can significantly improve the overall hit rate and the effective capacity overhead of cooperative caching.

According to the protocols in [16]—[18] the mobile hosts share their cache contents in order to reduce both the number of server requests and the number of access misses. The concept is extended in [14] for tightly-coupled groups with similar mobility and data access patterns. This extended version adopts an intelligent bloom filter based peer cache signature to minimize the number of flooded messages during cache resolution. A notable limitation of this approach is that it relies on a centralized mobile support center to discover nodes with common mobility pattern and similar data access patterns. Our work, on the contrary, is fully distributed in which the mobile devices cooperate in a peer-to-peer fashion for minimizing the object access cost.

In summary, in most of the existing work, there is a focus on maximizing the cache hit rate of objects, without considering its effects on the overall cost which depends heavily on the content service and pricing models. This paper formulated an object replacement mechanism to minimize the provisioning cost, instead of just maximizing the hit rate. Also, the validation of our protocol on a real SWNET [2] with dynamic partitions is unique compared to the existing literature.

IX. SUMMARY AND ONGOING WORK

The objective of this paper was to develop a cooperative object caching strategy for provisioning cost minimization in Social Wireless Networks (SWNETs). The key contribution was to demonstrate that the best cooperative caching for provisioning cost reduction requires an optimal split between object duplication and uniqueness. The paper analytically develops this optimal split point and subsequently develops the caching performance using a practical network, service and cost formulation that is motivated by Amazon’s Kindle electronic book delivery model. Ongoing work on this topic includes its extensions for other service and pricing models, and applications to more dynamic networks such as Vehicular Ad Hoc Networks (VANETs).

REFERENCES