Distributed Content Backup and Sharing Using Social Information

Jin Jiang, Claudio E. Casetti

Distributed Content Backup and Sharing using Social Information

Jin Jiang and Claudio E. Casetti

Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino, Italy

Abstract. This paper addresses the need for content sharing and backup in household equipped with a home gateway that stores, tags and manages the data collected by the home users. Our solution leverages the interaction between remote gateways in a social way, i.e., by exploiting the users' social networking information, so that backup recipients are those gateways whose users are most likely to be interested in accessing the shared content. We formulate this problem as a Budgeted Maximum Coverage (BMC) problem and we numerically compute the optimal content backup solution. We then propose a low-complexity, distributed heuristic algorithm and use simulation in a synthetic social network scenario to show that the final content placement among "friendly" gateways well approximates the optimal solution under different network settings.

Keywords: Content sharing, social networks, federated homes

1 Introduction and Motivation

The wealth of digital devices and appliances in everyday's life has brought about dramatic changes in our habits. Perhaps one of the most remarkable is the reliance on digital storage for whatever information content we own or produce. Of course, no savvy user would rely solely on storing precious, irreplaceble data in a single device and backup systems are now common in most households. More recently, the availability of "cloud" storage services, aimed at consumers and companies alike, such as Dropbox, Box.net, Ctera to name a few, has introduced a new opportunity. In the latter case, a wideband Internet connection can be exploited during idle periods to run background backups onto cloud storage.

One of the drawbacks of personal or cloud backup approaches is the fact that data of potential interest of other users sit unused in a storage device. Let us consider the following example. George has a set of pictures of the latest family vacations and he wants to show them to his friend John, while, at the same time backing them up. George remotely uploades the pictures to John's NAS, where a storage quota is reserved for such purpose; John is then notified that a copy of the pictures now exists in his NAS and that he is welcome to have a look, while keeping it in its NAS as a backup. For fairness, a similar quota for John's backups should be set aside at George's. The example could be extended to a close group of friends, as defined within social networks, and the potential of such a scheme instantly become apparent. By leveraging typical social networks

indicators, such as interests, hobbies and preferences, and by having all personal digital data appropriately tagged, the matching of remote users and content to backup would allow to catch two birds with a stone: safe, redundant online backup and social content sharing.

In this paper we will outline an architecture to realize this vision. In line with recent "federated homes" networking architectures [1, 2], we assume that home is equipped with a gateway and a large number of interconnected devices within the household. The gateway allows any content to be downloaded from outside the household, stored on it, and accessed by satellite devices in communication range of the home gateway. We also assume that, in keeping with the federated home vision, multiple neighboring or remote home gateways can be connected in a collaborative fashion, and can exchange various information. Although not explored in this paper, let us assume that a home gateway can collect its users' social networking data, e.g., a list of user's friends and interests, friends' locations and whether they are in the federated home network or not.

The outlook is not as simple as the description implies, though. Firstly, there are gateway selections issues. Choosing a friend's gateway to back up data only because mutual user interests match is not a sound policy from a networking point of view. The remote gateway could have poor connectivity or it could be overloaded. Even though friends in social networks are more likely to be in nearby areas [3], the gateway could be located in a far away country. The remote gateway should enforce a rigid quota management to avoid being swamped by friends' uploads. Additionally, there are management details to address: the user must rely on the backed-up content to be readily available on the remote gateway and it should be notified when the content is about to be deleted. If the content is deleted, a second-best choice should be identified, based on the same criteria that guided the former selection.

Our work falls into the same category as several recent research efforts tacked the problem of multiple backups across different resources [4–6]. None of these works, though, leverages the potential of social networking. Related to our problem are also the works on content placement exploiting information from social networking. [7] proposes ContentPlace, which is a social-oriented framework for data dissemination taking into consideration user interest with respect to content. A similar approach is taken in [8] where it is shown that mobility and cooperative content replication strategies can help bridge social groups. Another relevant work on an efficient social-aware content placement in opportunistic networks is [9] in which the authors model the content placement as a facility location problem.

In our paper, we devise an efficient content placement scheme to determine where to back up the content from a user's gateway to remote gateways belonging to his/her social friends. As remarked above, placing content replicas "outside" the home (i) consumes transmission bandwidth for uploading the content and (ii) incurs a storage cost on the remote friends' home gateways. So we aim at a strategy *that maximizes the friends' benefit by trying to match content type and friends' interests while taking into account both the bandwidth constraints*

between gateways and the storage space at the remote gateway. Thus, we model this optimization problem as a Budgeted Maximum Coverage (BMC) problem, as preliminary introduced in [10], and numerically obtain the optimal content placement solutions under a synthetic social networking scenario. Next, we propose and evaluate some heuristic distributed algorithms that federated gateways can implement to realize a social backup strategy. We evaluate the heuristics and discuss the conditions under which they can approximate the optimal solution.

2 Model description

We consider N federated home gateways, GW_i $(i = 1, 2, \ldots, N)$, each reachable through an Internet connection. A home gateway stores content for all users in the corresponding household, setting aside a "friend quota", Q_i, defined as the available storage capacity for the content uploaded by friends of its users. We will generically indicate by C_{ih} the bandwidth from gateway GW_i to gateway GW_h, and by C_{hi} the bandwidth in the opposite direction.

We then assume that totally there are M users in the network, and each one is registered on a single home gateway through which the user can access/store the content. For the purpose of identifying which users are registered on which gateway, we define an $N \times M$ matrix $\mathbf{P}$ whose generic element is given by:

$$P_{ij} = \begin{cases} 1 & \text{if } U_j \text{ registered on } GW_i \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

where i indicates the gateways, $i = 1, 2, \ldots, N$, while j is the user index, $j = 1, 2, \ldots, M$.

As explained earlier, we assume that home gateways somehow collect the social information of their users. In particular, we are interested in collecting *user's friend list* and *user's interests*. Also, gateways can collect friends' registration information (i.e., which friend of U_j is registered on which gateway).

We define E_j as friend list of user U_j:

$$E_j = \{U_f : F(j, f) = 1\} \tag{2}$$

where a friendship function $F(j, f)$ tracks whether user U_j and user U_f are friends:

$$F(j, f) = \begin{cases} 1 & \text{if } U_j \text{ and } U_f \text{ are friends, } j \neq f; \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

As we have seen, a user and its friends cooperate to back up content, which we generically represent as *items*. Items can be videos, photos or any other digital content. In order to handle the mathematics, we assume there is a maximum of K different items. A generic item k, $k = 1, 2, \ldots, K$, is characterized by its size $D^{(k)}$ and classified into a content type l (e.g., movies, books, outdoor photography ...). Content types too are finite, , i.e. $l = 1, 2, \ldots, L$, where L is the interest area size, i.e., the total number of content types considered in our system. The

association between an item and its type is assigned according to a uniformly random distribution.

The user's interests, i.e., the distribution of content preferences of the user, is captured by an *interest vector*, defined as follows. Given an item of type l let I_{jl} denote the interest factor that user U_j has for it, with $0 \leqslant I_{jl} \leqslant 1$. Thus, we can outline user U_j's profile through its interest vector:

$$\overline{I}_j = (I_{j1}, I_{j2}, \ldots, I_{jl}, \ldots, I_{jL}) \tag{4}$$

where $\sum_{l=1}^{L} I_{jl} \stackrel{\Delta}{=} 1 - r^j$. r^j is the probability that user U_j is interested in a content type out of the interest area L. Without loss of generality, we will just assume $\sum_{l=1}^{L} I_{jl} = 1$ or $r^j = 0$, i.e., users do not have interests outside the area L.

For the sake of notation simplicity, we also assume that every user has the same average number of items to back up.

Following the definitions above, each user has the objective of finding a selection of friends from the friend list, on whose gateways to back up its items. The matching of item and remote gateway should benefit both the hosting users, i.e., by closely matching their interests, and, it should transfer data effectively, i.e., by maximizing the utilization of available bandwidth between the respective gateways. Clearly, these objectives are not the only possible choices, and they lead to somewhat arbitrary weight formulations in the optimization problem. Through far from unique, such formulations will attempt to enhance the benefits we have outlined above.

As previously observed, we cast the optimization problem as a BMC problem. In BMC problems, a collection of sets $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ have associated costs $\{c_i\}_{i=1}^{m}$. The σ sets are defined to comprise elements $X = \{x_1, x_2, \ldots, x_n\}$ whose associated weights are $\{w_j\}_{j=1}^{n}$. The goal is to find a collection of subsets $\sigma' \subseteq \sigma$, such that the total weight of elements in σ' is maximized and their total cost is bounded by a budget L. The BMC problem is known to be NP-hard [11].

Gateway GW_i is assumed to have already collected user U_j's friend list E_j. We define $\sigma_j = \{\sigma_{j1}, \sigma_{j2}, \ldots, \sigma_{jh}, \ldots, \sigma_{jN}\}$ the collection of subsets σ_j for user U_j; here, subset $\sigma_{jh} = \{U_f \in E_j : P_{hf} = 1\}$ denotes the set of friends of user U_j who are registered on gateway GW_h, $h = 1, 2, \ldots, N$. We recall that $P_{hf} = 1$ means that user U_f is registered on gateway GW_h, and that $U_f \in E_j$ means that user U_f is in the friend list of user U_j, so $\sigma_{jh} \subseteq E_j$. The cost $c^{(k)}_{(\sigma_{jh})}$ of selecting the subset σ_{jh} is defined as the cost of uploading the content item k of size $D^{(k)}$ onto the gateway GW_h, which can be defined as: $c^{(k)}_{(\sigma_{jh})} = D^{(k)}$.

The *element* set in our problem obviously is the user U_j's friend list E_j. For each element/user $U_f \in E_j$ (user U_f is a friend of user U_j), we can define the weight as the benefit $w^{(k)}_{(U_f)}$ that element U_f can obtain when item k is uploaded onto gateway GW_h where U_f is registered. Such benefit will chiefly depend on the interest I_{fl} that U_f has in the uploaded content. The friend's interest also has a subtle implication for the content owner U_j: the more U_f is interested in the backed-up items, the longer it is likely to store them. Additionally, we factor

in the ease of accessibily of the content when its owner U_j wants to retrieve it, i.e., the bandwidth between the hosting gateway and the uploader gateway. The ease of accessibility is also a plus for U_f, because shorter retrievals are less penalizing for its uplink capacity. We can thus define $w^{(k)}_{(U_f)}$ as:

$$w^{(k)}_{(U_f)} = I_{fl} \cdot C_{hi} \tag{5}$$

where I_{fl} denotes the interest of U_f in items of type l which content k belongs to and C_{hi} is the bandwidth of the link from the hosting gateway GW_h to the uploader gateway GW_i. Although bandwidth and user interests are, by definition, quantities that may vary over time, we can safely assume that the time scale of their variations is larger than the time scale of the algorithm execution.

Our constraint is the gateway friend quota, Q_i, which we recall is the available storage capacity for data uploaded by friends.

Finally, our problem can be formulated as follows:

$$\begin{aligned} &maximize && \textstyle\sum_{k=1}^{K} \sum_{U_f \in E_j} w^{(k)}_{(U_f)} \cdot y^{(k)}_f \\ &subject\ to && \textstyle\sum_{k=1}^{K} D^{(k)} \cdot x^{(k)}_h \leqslant Q_h \\ & && \textstyle\sum_{P_{hf}=1} x^{(k)}_h \geqslant y^{(k)}_f \\ & && x^{(k)}_h, y^{(k)}_f \in \{0,1\} \end{aligned} \tag{6}$$

where $x^{(k)}_h = 1$ indicates that gateway GW_h is selected to host a backup of content item k, while $y^{(k)}_f = 1$ means that user U_f's associated home gateway is chosen as backup for the content item k. The first constraint limits the limitation of available friend quota on each home gateway; and the second one applies to the case of one gateway with multiple associated users: if one user is chosen as backup for one item k, its associated gateway must be selected too.

The number of Boolean decision variables ($x^{(k)}_h$ and $y^{(k)}_f$) is $O(K\langle N\rangle)$, where $\langle N\rangle$ denotes the average number of friends per user. The number of constraints is $O(K\langle N\rangle + M)$. The solution time required by the Gurobi solver for an instance with approximately $1,000$ gateways, $3,000$ users and an average of 5 content items for each user to share or backup, is about 30 minutes using a 4-core 2.3 GHz system and a 4 GB RAM.

3 Distributed Heuristics

The greatest hurdles in translating the optimization model into a working implementation are (i) that the model paints a *static* picture, where all users take instantaneous decisions and (ii) that decisions are taken by a centralized, knowledgeable entity.

In this section we propose a set of distributed heuristic algorithms that strive to achieve the same goal as the model outlined in the previous section. The algorithms take two different viewpoints: that of participating content owners who have data to share or back up and that of remote gateways who provide their own storage space for their social friends. In both cases, we follow the same arguments used in the optimization problem definition.

From the viewpoint of content owners, not only do they wish to back up or restore the data as fast as possible, but, in the long run, they also wish that the remote gateway keeps the content for as long as possible. Therefore, content owners are naturally disposed to choose friends from whose gateways they can retrieve the backed up items more quickly. Also, they would like friends to be interested in the content they upload, because such friends are more inclined to store it for a long time. If one wants to back up their kid's pictures, what better place than the grandparent's gateway?

At the receiving end, the remote gateway can display two types of behavior that are arguably worth investigating. One is a selfish behavior: regardless of the backup requests received by friends of its users, the remote gateway will devote its "friend quota" only to maximize the interests of its associated users, i.e., based only on the first factor in the benefit $w^{(k)}_{(U_f)}$ of eq. (5). The other one is a cooperative behavior: the remote gateway fills up its friend quota while trying to maximize the whole benefit of eq. (5), hence accounting for both its users' interest and the bandwidth toward the content owner's gateway.

We will address either viewpoint through a specific distributed algorithm: a Greedy Placement Algorithm (GPA) run by content owner gateways in order to identify the most suitable places where to back up their items, and the RePlacement Algorithm (RPA), run by each remote gateway upon receiving a backup request.

3.1 The Greedy Placement Algorithm

We assume that a user has available all items it wants to back up when the GPA procedure is started. Further, we assume time to be slotted in intervals of fixed length and that the starting time slot of GPA procedure on a gateway is random. On each gateway, the sequence in which users start GPA is also randomly determined. To achieve the fairness among all the users in the system, each user can run GPA *only once* per time slot.

When starting the GPA, a gateway GW_i will have already collected the following information *from each of its associated users* U_j:

- the friend list E_j;
- the remote friend gateway list RG_j which includes the remote gateways on which user U_j's friends are associated;
- list K_j of items to back up;
- for each item $k \in K_j$, the benefit $w^{(k)}_{(U_f)}$ of each friend $U_f \in E_j$ as defined in Section 2;

- for each remote friend gateway $GW_h \in RG_j$, a quantity referred to as *gateway aggregate benefit* $w_h^{(k)} = \sum_{U_f \in E_j} w_{(U_f)}^{(k)} \cdot P_{hf}$ (recall that $P_{hf} = 1$ indicates the friend U_f is associated to gateway GW_h);
- a query list Z_j where each element is a pair (k, GW_h) representing an item and the IDs of a remote friend gateways, sorted by their gateway aggregate benefit $w_h^{(k)}$.

The main idea behind GPA, detailed in Algorithm 1, is the following: every time the algorithm is scheduled, user U_j sends a backup request to the remote friend gateway whose ID is in the element that tops the query list Z_j. Such element is then removed from the list if the request is accepted; otherwise, it is pushed back to the bottom of Z_j. After sending backup requests on behalf of a user U_j for a total item size of S bytes, GPA stops and it is rescheduled randomly in the next time slot.

Algorithm 1 Greedy Placement $GPA(U_j, Z_j)$

Require: RETRY counters for all elements of Z_j
  $size \leftarrow 0$
  loop
    pop_front element (k, GW_h) from Z_j
    if RETRY$(k, GW_h) >$ MAX_RETRY **then**
      continue
    end if
    if $size + D^{(k)} > S$ **then**
      insert_head element (k, GW_h) into Z_j
      break loop
    else
      $size = size + D^{(k)}$
    end if
    send Backup_REQ to GW_h for k
    if Backup rejected **then**
      push_to_back element (k, GW_h) into Z_j
      RETRY(k, GW_h) = RETRY(k, GW_h) +1
    end if
  end loop
  if $Z_j != \emptyset$ **then**
    schedule $GPA(U_j, Z_j)$ next time slot
  end if
  return RETRY counters for all elements of Z_j

Since the query list Z_j of user U_j is sorted by the gateway aggregate benefit for the corresponding remote gateway, the pop_front operation corresponds to extracting from the list the item k and the ID of the best candidate gateway where it can be backed up in the current time slot. A *Backup_REQ* message is then sent to such gateway.

Once a gateway receives the *Backup_REQ*, it will first check whether it has already cached this item. If not, and there is enough free space in its friend quota[1], it will set aside the corresponding size for this item in its storage space. A *Backup_REP* message is returned to the item owner notifying it whether the backup request was accepted. If the request is accepted, the content owner will start the upload. If the request is denied, or no reply is received, the corresponding list element is pushed at the bootom of the query list, for a later retry, up to a limit of MAX_RETRY times.

Upon reaching the S bytes backup limit, and if the query list is not empty, the gateway schedules the next run of GPA, and the next batch of backup requests for user U_j, at the next time slot. In order to achieve fairness across the federated network, all users should use the same upper backup limit S.

We finally remark that GPA can easily be modified so that the gateway attemps to back up a single item k only onto a limited set of friends' gateway. In this paper, we have only considered the most general (and most challenging) case in which the gateway tries to back up all items on all the friends' gateways. Using the above notation, when GPA evantually stops, an always successful gateway will have dislocated $|RG_j| \cdot |K_j|$ items across the federated network, for each of its users.

3.2 The Replacement Algorithm

If remote gateways "passively" accepted all backup requests until their quota is filled up, their collection of backed up items would not match the optimum allocation, being strongly dependent on which users start the GPA procedure first, hence which greedy requests they receive first. After all, the friends of the users associated to a gateway share the quota on this gateway by competing with each other. In order to alleviate such unbalancement, we introduce a second algorithm, called RPA, RePlacement Algorithm, to be run by every gateway upon receiving a backup request and discovering that the quota is already filled up. We assume that a gateway GW_i holds a list B_i of items backed up in its storage space, sorted by benefit, while we indicate by q_i the free space still available for backups out of the friend quota.

As explained above, when a gateway GW_i receives a *Backup_REQ* message for a new item k of size $D^{(k)}$, it will check whether it has the enough storage space for it; if the space is not sufficient, the gateway will compute the aggregate benefit $w_i^{(k)}$ of the item according to eq. (5) and start the RePlacement Algorithm, described in Algorithm 2.

The gist of the RPA procedure is the following. In order to maximize the benefit of the users associated to the receiving gateway, the replacement strategy considers the removal of backed up items with lower benefit than the incoming item. The B_i list is sorted by benefit, and RPA checks if there are enough items with lower benefit than $w_i^{(k)}$ that can be dropped to leave room for the incoming item. A second check verifies if the product of the total benefit and total size of

[1] We will discuss the case of no free space in the next subsection.

Algorithm 2 The RePlacement Algorithm $RPA(GW_i, k)$

Require: $B_i, q_i, w_i^{(k)}, D^{(k)}$
  $replace \leftarrow$ **false**, $FreeSpace \leftarrow q_i$
  $DropWeight \leftarrow 0, DropSize \leftarrow 0$
  while $B_i \neq \emptyset$ **do**
    select $k' \in B_i$ with the lowest benefit
    $DropWeight \leftarrow DropWeight + w_i^{(k')}$
    $DropSize \leftarrow Dropsize + D^{(k')}$
    $FreeSpace \leftarrow FreeSpace + D^{(k')}$
    if $w_i^{(k)} < DropWeight$ **then**
      $replace \leftarrow$ **false**
      break
    else if $w_i^{(k)} == DropWeight$ **then**
      if $DropSize \leqslant D^{(k)}$ **then**
        $replace \leftarrow$ **false**
        break
      else
        $replace \leftarrow$ **true**
        break
      end if
    else
      if $D^{(k)} > FreeSpace$ **then**
        $B_i \leftarrow B_i \setminus k'$
        continue
      else
        $replace \leftarrow$ **true**
        break
      end if
    end if
    if replace **and** $D^{(k)} \cdot w_i^{(k)} \leqslant DropSize \cdot DropWeight$ **then**
      $replace \leftarrow$ **false**
    end if
  end while
  return $replace$

the items selected for dropping is smaller than the benefit/size product on the incoming item. If so, the latter replaces the dropped items in the storage space of GW_i. Ideally, the second check is aimed at preserving the network efficiency, so that a large backed up item is not easily dislodged by much smaller item with a marginally higher benefit.

If the remote gateway GW_i replaces content item k', a $Backup\_DEL$ message must be sent to inform the content owner that it needs to find a new gateway where to store k'. The content owner will thus place a corresponding element in the Z_i queue of algorithm GPA (or start a new instance of GPA if Z_i has been emptied in the meanwhile).

The running time cost of GPA and RPA is minimal. For each user for which GPA is run, the length of the query list Z_j is $O(\hat{K}\hat{E})$, where $\hat{K}$ is the average number of items per user and $\hat{E}$ is the average number of remote friend gateways. So for the individual gateway the running time is $O(\hat{K}\hat{E}m/n)$ with m/n denoting the average number of users per gateway. As for the running time of RPA, the algorithm searches the whole the backed up item list B_i to check what can be replaced, while the maximum number of iterations depends on the number of remote friends and their own items to back up. So the complexity of RPA for one gateway also is $O(\hat{K}\hat{E}m/n)$.

4 Evaluating optimal model and distributed heuristics

We will now investigate the validity of our approach by following two main directions. Firstly, we numerically solve the model and derive the maximal benefit according to eq. (6). The results will be benchmarked against two other, simpler backup (re)placement strategies, to evaluate the gain that comes at the expense of extra complexity in the backup strategy. Secondly, we compare the content allocation resulting from the optimal solution with what is achieved through the distributed heuristics. In this case too, we will explore variants of GPA and RPA.

Our evaluation will necessarily target a synthetic scenario. Recreating all the conditions and variables of an actual online social network would be a daunting task. We thus extrapolate its essential features and create a scaled-down version for our simulation following the procedure we outlined and validated in [10].

4.1 Optimization, heuristics and variants thereof

In order to extract meaningful comparisons between the optimization approach and the distributed heuristics, sharing the same scenario is not enough. On the one hand, we used Gurobi [12], which runs a variant of the branch-and-cut algorithm, to numerically solve the BMC problem in eq. (6). The solution yielded an optimal joint content item placement, i.e., the set of candidate home gateways to be selected for each content item, as well as the optimal benefit value that each user can obtain by being selected. On the other hand, we simulated the heuristic approach on an ad-hoc simulator. The GPA and RPA algorithms were stripped to their bare bones (i.e., not considering the actual content file transfer while working on GPA timeslot granularity) so that we could focus on the resulting allocation after requests, allocations and replacements have settled. Finally, we compared the steady-state outcome to what the optimization had predicted.

We have tried to gauge the effectiveness of optimization and heuristics not only by comparing one against the other, but also by running some variants of either approach with the aim of catching a glimpse of what we would stand to lose or gain, if we chose a simpler (or a more convolute) strategy than the ones outlined in Sec. 2 and 3.

As far as optimization was concerned we evaluated three different content placement strategies. The first strategy is the *joint* optimization method, described in Section 2, in which the friends who have the largest interest in the

corresponding content item and the highest uplink bandwidth will be selected first (assuming their quota is not used up). The second strategy is a *bandwidth-based* method, in which friends reachable through gateways with the highest bandwidth will be selected, regardless of their interests in the uploaded items. The optimal bandwidth-based placement is still obtained from eq. (6) by changing the benefit definition into $w^{(k)}_{(U_f)} = C_{hi}$. The last one is a *random* method, where users randomly choose what friends to share the content item with, as long as enough quota is available, not considering any other factors.

Concerning the heuristics, we considered three versions of GPA:

- GPA-j, which corresponds to the definitions outlined for Algorithm 1;
- GPA-b, where the benefit of a friend U_f on GW_h just depends on the uplink bandwidth from GW_h to GW_i (where U_j is located) , i.e., $w^{(k)}_{U_f} = C_{hi}$;
- GPS-r, where elements in Z_j are randomly sorted.

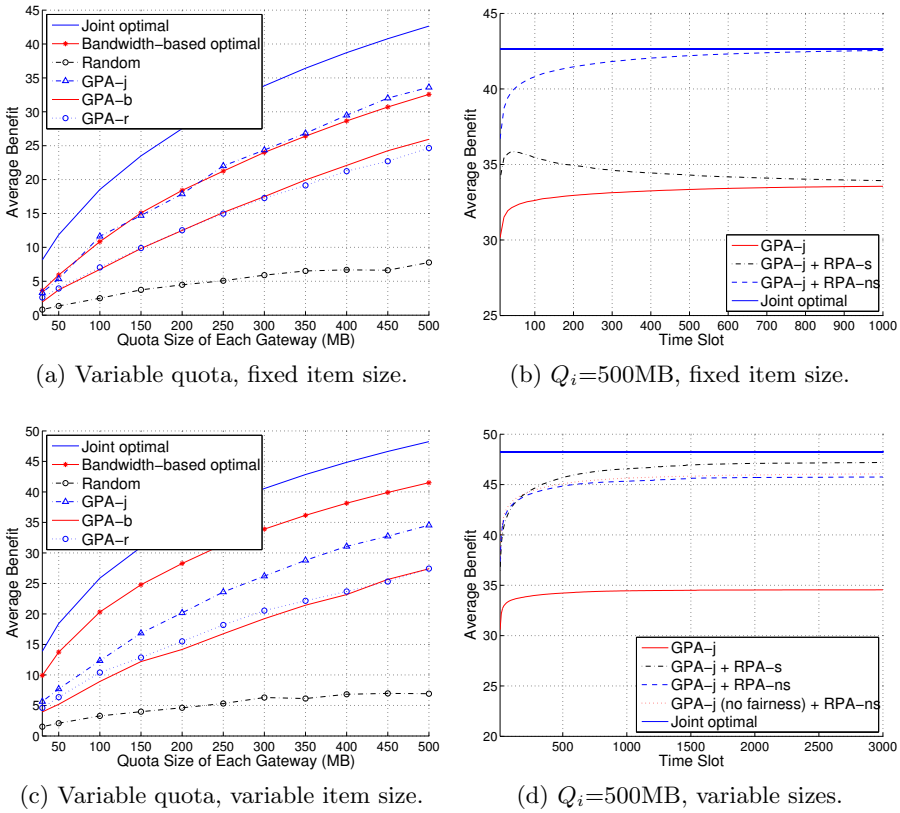
Likewise, we evaluated two versions of RPA, RPA-ns and RPA-s differing by the sorting of B_i. The former corresponds to the definitions outlined for Algorithm 2. In the latter, the benefit is defined just as the sum of interests of the associated users who are also friends of the content owner, disregarding the uplink bandwidth to the owner gateway; This behavior is "selfish", hence RPA-s, because the receiving gateway only tries to maximize the interest of its own users.

One final variant, which affects the GPA procedure, concerns the size of items to back up. At first, we assumed that all items have the same size. While not realistic per se, it could be meaningful if the implementation of the backup system were limited to a single class of items. In this case, tagged as "fixed size" in our results, we assumed all items to have a 10 MB size. Then, we considered items of any possible size within a bound. Such "variable size" case features random item sizes following a truncated exponential distribution with expected value of 10 MB and maximum size of 50 MB. While studying the latter case, though, we soon found out that allocation results were dangerously biased toward smaller items, so we introduced a *fair item size balancing* mechanism in GPA. Items were divided into size groups of 10 MB each and GPA was modified so that a backup request for one item was sent only if the total amount of data already backed up in the item group did not exceed the total amount already backed up for items of the first size group bigger than it. For instance, if a 15 MB item in the 10-20 MB size group increased the total amount of data already backed up for that size group to 95 MB, and the total amount of the 20-30 MB size group were still 90MB, the request would be put on hold.

4.2 Performance Evaluation

Our first set of plots aims at comparing the optimization results, in their three variants, with the corresponding three variants of the distributed heuristics in which the GPA algorithm alone in employed. The rationale of such comparison is to show the importance of the replacement management introduced by RPA (which is not used in these first results). For reason of space, we cannot show

the whole possible parameter space, so we will just focus on a few representative cases to prove our point. Throughout the section, the number of gateways is $N = 1000$, the number of users is $M = 3000$ and the number of item types is $L = 10$. Results with $L = 50$ and $L = 100$ were qualitatively similar.



(a) Variable quota, fixed item size.

(b) Q_i=500MB, fixed item size.

(c) Variable quota, variable item size.

(d) Q_i=500MB, variable sizes.

Fig. 1: Avg. user benefit obtained by different methods under different cases.

The plot in Fig. 1a clearly ranks the optimization and heuristics variants in terms of average user benefit as defined in eq. (5), for various quota constraints and fixed item size. It shows that jointly optimizing bandwidth and benefit is a clear winner. Even though the average benefit is low (due to the an average of three users per gateway sharing the same quota), the advantage of finding an optimal allocation for backed-up content depending on interest and bandwidth is clearly visible. Additionally, GPA heuristics alone do not match the joint optimization results, as expected. Similar conclusions can be drawn for the variable item size case in Fig. 1c, where GPA-j fares even worse.

In the next set of results, we let receiving gateways run the replacement algorithm, RPA, in its two (selfish and non-selfish) versions. Fig. 1b plays out the $Q = 500MB$ quota case over time, in the same scenarios just examined. It shows that, given some time to converge, GPA-j and RPA-ns together yield an allocation that progressively corrects the initial uneven backup distribution provided by the distributed implementation of GPA-j alone. The selfish version of RPA, RPA-s, instead shows that if the owner and the storing gateway are not on the same page, as it were, when deciding what items are preferable to backup, the performance reverts to that of GPA-j alone. The selfishness of RPA, however, seems to have a lesser impact in the variable item size case, shown in Fig. 1d, where we also plotted a run of GPA-j without the fair item size balancing (tagged "no fairness" in the legend). The use of size balancing, though of marginal impact on the average user benefit, comes in handy in order to control the size of backed up items, as will be shown below.

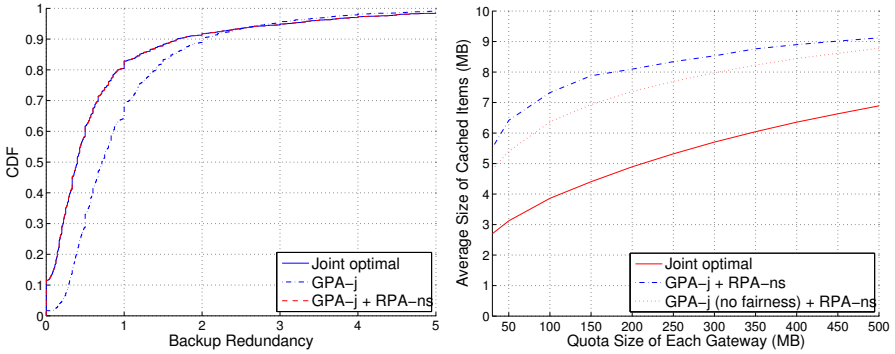


Fig. 2: Backup redundancy CDF; fixed item size and $Q_i = 500$MB (left). Avg. backed-up item size as a function of gateway quota; variable item size (right).

We next plot the *backup redundancy*, i.e. the average number of a user's own items that have been replicated onto its friend gateways, upon reaching convergence of the distributed heuristics. The CDF of the redundancy is useful to understand the thoroughness of the backup process. In the left plot of Fig. 2, RPA-ns allows the remote gateway to replace the items (all of the same size) with smaller benefit and improve the storage thoroughness as much as the optimal method (the two curve indeed overlap). Finally, the right plot of Fig. 2 reports the average size of items backed up in the "friend" storage quota at remote gateways, when convergence is reached. Recalling that the average item size in the variable item case is 10MB, we can conclude that the loss of average user benefit with respect to the optimal case shown in Fig. 1d is offset by a fairer distribution of item sizes in the gateway storage across the federated network (i.e., the average size of backed up item achieved by GPA-j and RPA-ns is just 10% smaller than the average item size in the system, as opposed to 30% smaller

in the joint optimization case). Also, the use of fair item size balancing with GPA proves of some consequence to achieve this result.

5 Conclusions

In this work we addressed a novel approach to content sharing and backup in home networks. We envisioned a system where content is placed "outside the home" in a cloud formed by federated home networks and its location is selected exploiting the user's social networking information. We studied its performance in terms of the benefits that remote gateways and their users can enjoy when friends choose them to back up (and share) their content. We defined an optimization problem and compared its numerical solution with several flavors of distributed heuristics.

6 Acknowledgments

This work was partly funded by the European Union FP7, under grant agreement 258378 - FIGARO project and by the Italian Ministry of Education and Research under the PRIN project GATECOM.

References

1. The FIGARO FP7 ICT project. http://www.ict-figaro.eu/
2. S. Defrance, R. Gendrot, J. Le Roux, G. Straub, T. Tapie. Home Networking as a Distributed File System View. *HomeNets*, August 2011.
3. L. Backstrom, E. Sun, and C. Marlow. Find me if you can: improving geographical prediction with social and spatial proximity. *19th ACM WWW*, April 2010.
4. N. Belaramani, M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula, and J. Zheng. PRACTI replication. *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, May 2006.
5. V. Ramasubramanian, T. Rodeheffer, D. Terry, M. Walraed-Sullivan, T. Wobber, C. Marshall, and A. Vahdat. Cimbiosys: A platform for content-based partial replication. *NSDI*, August 2009.
6. A. Post, P. Kuznetsov, and P. Druschel. PodBase: Transparent storage management for personal devices. *USENIX International Workshop on Peer-to-peer Systems*, February 2008.
7. C. Boldrini, M. Conti, and A. Passarella. ContentPlace: social-aware data dissemination in opportunistic networks. *ACM MSWiM 2008*, ACM, October 2008.
8. E. Jaho and I. Stavrakakis. Joint interest-and locality-aware content dissemination in social networks. *IEEE/IFIP WONS 2009*, February 2009.
9. P. Pantazopoulos, I. Stavrakakis, A. Passarella, and M. Conti. Efficient Social-aware Content Placement in Opportunistic Networks. *IEEE/IFIP WONS 2010*, February 2010.
10. J. Jiang, C. Casetti. Socially-aware Gateway-based Content Sharing and Backup. *HomeNets*, August 2011.
11. S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.
12. Gurobi optimizer. http://www.gurobi.com/html/products.html.