# Compact Samples for Data Dissemination [⋆]

Tova Milo, Assaf Sagi, and Elad Verbin

School of Computer Science
Tel Aviv University
{milo,sagiassa,eladv}@post.tau.ac.il.

**Abstract.** We consider data dissemination in a peer-to-peer network, where each user wishes to obtain some subset of the available information objects. In most of the modern algorithms for such data dissemination, the users periodically obtain samples of peer IDs (possibly with some summary of their content). They then use the samples for connecting to other peers and downloading data pieces from them. For a set $O$ of information objects, we call a sample of peers, containing at least $k$ possible providers for each object $o \in O$, a *k-sample.*
In order to balance the load, the k-samples should be *fair*, in the sense that for every object, its providers should appear in the sample with equal probability. Also, since most algorithms send fresh samples frequently, the size of the k-samples should be *as small as possible*, to minimize communication overhead. We describe in this paper two novel techniques for generating fair and small k-samples in a P2P setting. The first is based on a particular usage of uniform sampling and has the advantage that it allows to build on standard P2P uniform sampling tools. The second is based on non-uniform sampling and requires more particular care, but is guaranteed to generate the smallest possible fair k-sample. The two algorithms exploit available dependencies between information objects to reduce the sample size, and are proved, both theoretically and experimentally, to be extremely effective.

## 1 Introduction

We consider in this paper data dissemination in a peer-to-peer network, where each user wishes to obtain some subset of the available information objects. In most of the modern algorithms for such data dissemination, the users periodically obtain samples of peer IDs, (possibly with some summary of the peers' content). They then use the samples for connecting to other peers and downloading data pieces from them. It is desirable that the peer samples (1) contain enough providers for each requested object, so that users have a sufficient choice of data sources, (2) are 'fair', so that the requests for objects are spread evenly over their providers, and (3) are as small as possible, so that the communication overhead is minimized when samples are sent frequently. The goal of this paper

[⋆] The research has been partially supported by the European Union Project EDOS and by the Israel Science Foundation.

is to devise peer-sampling algorithms that achieve the above three goals. Our algorithms take advantage of "correlations" between data objects to improve performance. To motivate the study of this problem, let us briefly describe its practical origin and explain how a good solution to the problem can contribute to better performance of data dissemination platforms.

**Motivation** In a data dissemination scenario, various information objects are to be disseminated to peers in the network. The original distributors of the objects are one or more *source* peers who initially hold the data. The other peers are interested in obtaining some subset of the available objects. When the number of peers to which the data is to be disseminated is large, it is practically impossible to have all peers downloading the data directly from the original sources. Indeed, most data dissemination platforms are based on peer cooperation, where each peer provides to other needing peers the objects (or parts thereof) which she has already acquired [3].[1] In such a setting, all peers serve essentially both as information consumers *and* information providers - a peer interested in obtaining a certain object serves also as a provider for it.

In order for a peer to connect with peers that may assist her in obtaining a certain object, she needs to obtain information about other peers in the system holding it. For this end, data dissemination algorithms typically supply the requesting peer with information about a set of $k$ peers (for some constant $k$), chosen randomly from the set of all peers in the system that hold the object [14]. The peer then chooses a subset that is most beneficial to her, e.g. in terms of bandwidths or available object pieces, and connects to those peers to obtain the data. New peer samples may be supplied periodically (or upon request) to the peers to allow them to acquire new, possibly more suitable, data sources. Depending on the particular data dissemination algorithm being used, the details of which object pieces are available at each of the sample peers may be either encoded as summary information given in the sample or, alternatively, may be obtained by querying the given peer [18]. To abstract this and ignore the specific implementation details, we assume the existence of a function $objects(n)$ that, given a network peer $n$, tells which (pieces of) objects may be provided by $n$.

The sampling domain for an object $o$ may consist of those peers that actually hold (pieces of) of $o$, or of the peers that declared their wish to obtain that object. The rationale for the latter is that such peers, being interested in $o$, are likely to have already obtained some of its pieces (or will soon manage to). Most algorithms take the latter approach as it guarantees the sampling domain to be fairly stable (it is determined once the peers declare their wishes) [5]. This is also what we assume here. In order not to overload certain peers, the samples are required to be *fair*, in the sense that for every object $o$ the peers (potentially) providing $o$ should appear in the sample with equal probability [8]. Also, since most algorithms send fresh samples *frequently* it is desirable that their size be as small as possible, to minimize communication overhead. Finally, to be able

[1] Data objects are typically fragmented into *blocks*. A peer who shares an object might not have yet completed its download and hence shares only the blocks downloaded thus far.

to guarantee that a certain number of samples can be sent within a *fixed* time period, the dissemination algorithms need to know the *bound* on the samples' size. To put this in terms of the problem mentioned at the beginning of the section, *one would like the samples sent to a given peer to contain at least k providers for each of the objects that she requested, be fair, and have the worst-case bound on their size be as small as possible.*

For a set of objects $O$, we call a fair sample of the network peers, containing at least $k$ providers for each object $o \in O$, a *k-sample* (the formal definition is given in sec. 2). Our goal is to devise sampling algorithms that minimize the worst-case bound on the k-samples' size. Before presenting our results, let us consider a simple example.

**Example** Consider a peer-to-peer network consisting of a set $N$ of peers and holding a set $\mathcal{O}$ of distinct information objects. Let $n$ be some network peer that is interested in obtaining a set of objects $O \subseteq \mathcal{O}$. For simplicity, assume a simple network architecture where some *coordinating* peer is informed of the peer's request and is in charge of providing her with corresponding k-samples. (In more general architectures the task may be distributed among several peers). Consider first a simple method that the coordinator peer can use for generating k-samples for $O$: For each $o \in O$, sample (uniformly) $k$ peers among the providers for $o$ (we will see in the following section standard techniques to perform such sampling). The k-sample then consists of the union of the sampled sets. Clearly its size is bounded by $k|O|$. Interestingly, although rather naïve, the bound obtained by this simple algorithm is in fact the tightest we could find in the existing literature since current systems treat the dissemination of distinct information objects separately (see more on that in Section 5). So the question motivating the present work is can one do better?

It turns out that the answer is often positive. The key observation is that, in practice, one can typically detect *correlations* between object requests, which can be used to significantly reduce the k-sample size. As a simple example, consider three objects $A, B$ and $C$. The naïve sampling algorithm described above yields for them a k-sample of size $3k$. Now, assume that the coordinator knows that every peer interested in obtaining $B$ also wants $A$ (implying that the set of potential providers of $B$ is included in that of $A$). If the two sets of providers happen to be identical then clearly a k-sample of size $2k$ suffices: the same set of $k$ peers sampled for $A$ can also be used for $B$. Even if the $B$'s are not provided by all the $A$-peers but only by say, 75% of them, a k-sample of size $2\frac{1}{3}k$ still suffices: a sample of $1\frac{1}{3}k$ $A$-peers contains on the average $k$ $B$-peers.

**Our results** Based on the above observation, we present in this paper two classes of algorithms for generating compact k-samples. The first employs uniform peer sampling. Its main advantage is that it allows to build on standard P2P uniform sampling tools e.g.[16, 13]. The sampling procedure here amounts to (1) grouping the requested objects, based on the correlations between their providers, and (2) uniformly sampling providers for each object group. The crux is to determine the optimal objects' grouping, i.e. the one that minimizes the resulting samples' size. We show the problem to be NP-hard but provide a linear time, constant-factor *approximation algorithm* for it. Furthermore, we show, experimentally, that our approximation algorithm yields in practice results much better than its worse case bound - indeed, the generated k-samples are of size very close to the minimal possible. We next consider non-uniform sampling. We first show that for k-samples generated using non-uniform sampling, the size of the minimal possible k-sample can be determined in linear time. We then propose a new simple distributed sampling technique that allows to generate such minimal k-samples in a decentralized P2P environment. To illustrate the benefit that our new sampling techniques can bring to existing data dissemination platforms, we tested experimentally the performance improvement that can be obtained by incorporating them in the popular BitTorrent[2], showing significant gain.

The paper is organized as follows. Section 2 introduces the basic formalisms used throughout the paper. Section 3 studies the generation of compact k-samples via uniform sampling. Non-uniform sampling is considered in Section 4. We conclude in Section 5 considering related work. The full proofs of the results as well as a detailed description of the experiments can be found in the full version of the paper [17].

## 2 Preliminaries

We introduce here the basic formalisms used throughout the paper, including our abstraction of the P2P network as an objects-providers bipartite graph, and the notions of k-samples and uniform peer sampling.

**Objects-Providers graph** As mentioned in the Introduction, peer requests serve as a good indication for the availability of objects on the given peers. The rationale is that peers that are interested in a given object are likely to quickly obtain (at least some pieces of) the object and be able to provide them to other peers. Consequently we consider from now on each network peer $n$ that requested a certain object $o$ as a provider of $o$. Consider a peer-to-peer network consisting of a set $N$ of peers and holding a set $\mathcal{O}$ of distinct information objects. The availability of the objects in the network peers can be represented as a bipartite graph consisting of two sets of nodes, one representing the objects and the other representing peers, with edges connecting each peer node to the nodes of the objects that it provides. Overloading notation, we will use $\mathcal{O}$ for both the objects and the nodes representing them. Peers that provide the same set of objects are grouped together and represented in the graph by a single node. We associate with each such node a *weight* that reflects the number of peers that it represents (as a fraction of the overall number of all peers). More formally,

**Definition 1.** *An* objects-providers *graph* $g = (\mathcal{O}, P, E, w)$ *is a weighted bipartite graph where* $\mathcal{O}$ *and* $P$ *are two disjoint sets of nodes called the* object nodes *and the* provider nodes*, resp.;* $E \subseteq \mathcal{O} \times P$ *is the set of edges of the graph; and* $w : P \rightarrow [0, 1]$ *is a* weight *function, associating to each provider node* $p \in P$ *some weight* $w(p)$*, s.t.* $\Sigma_{p \in P} w(p) = 1$.

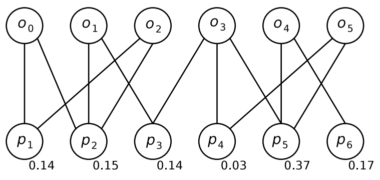

**Fig. 1.** An objects-providers graph.

We will use below $o, o_1, \ldots$ to denote object nodes as well as the information objects that they represent. We use $p, p_1, \ldots$ to denote provider nodes and $n, n_1, \ldots$ to denote the actual network peers. We use $v$ to denote an arbitrary graph node. Consider the objects-providers graph in Fig. 1, which will serve as a running example throughout this paper. $\mathcal{O}$ here consists of six information objects $o_0 \ldots o_5$ which are provided by six types of providers. Here, the peers represented by the node $p_2$ provide the objects $o_0$, $o_1$ and $o_2$ and form 15% of the overall set of peers providing objects in $\mathcal{O}$.

For a set of nodes $s$ in the graph $g$, we denote by $N(s)$ the set of nodes in $g$ that are *neighbors* of some node in $s$. When $s$ is a singleton set consisting of a single node $v$, we use $N(v)$ as a shorthand for $N(\{v\})$. Observe that for any information object $o \in \mathcal{O}$, the nodes in $N(o)$ represent the set of peers that provide $o$. Indeed, the sum of the neighbors' weight, $\sum_{p \in N(o)} w(p)$, describes precisely the number of $o$'s providers (as a fraction of the overall number of the providers of $\mathcal{O}$). We refer to this sum as the *popularity of o* (in $g$). To continue with our running example, the popularity of $o_4$ is $w(p_5) + w(p_6) = 0.54$.

The objects-providers graph for a given network may be constructed in different ways, depending on the particular application setting: for instance by considering the full set of peers' requests (e.g. in a centralized setting or when the number of peers is not too big); by drawing a random sample of the network peers as representatives for the requests distribution (e.g. in a distributed setting with a large number of peers); using logs and history information (in environments where peers tend to repeatedly request similar data); using known dependencies between information objects (when such dependencies are available); or by some combination of the above.

In the remainder of this paper we assume that we are given an objects-providers graph, and ignore the particular method used for its construction. We will return to this topic in Section 3.3, showing that the algorithms that we propose *do not require in practice to actually build the objects-providers graph but use only a very partial knowledge of its shape, which can be easily obtained in a distributed manner by simple uniform sampling.*

**k-samples** Consider a network peer that is interested in obtaining a set of objects $O \subseteq \mathcal{O}$. A *k-sample* for $O$ is a fair sample of the network peers containing at least $k$ providers for each object $o \in O$. More precisely,

**Definition 2.** *A* k-sample *(for a set $O$ of information objects) is a randomly generated entity consisting of a subset $K \subseteq N$ of the network peers and supporting a function* Providers$(o)$ *which returns, for each object $o \in O$, a subset $K_o \subseteq K$ of providers for $o$, where the following two properties hold:*

1. *(*sufficient number of providers*) For each object $o \in O$, the expected size of* Providers$(o)$ *is at least $k$, i.e. $E[|K_o|] \geq k$;*
2. *(*fairness*) For each object $o \in O$, each of $o$'s providers has an equal probability of appearing in* Providers$(o)$*.*

Our goal here is to design sampling algorithms that - given as input, an objects-providers graph $g$ describing the availability of objects in the network peers, a request for a set of objects $O$ and a number $k$ - generate small k-samples (for $O$). When measuring the quality of a sampling algorithm, we look at the maximal size of $K$ in the samples generated for the given input. *We are interested in devising sampling algorithms where for any $g$, $O$, and $k$, the worst-case bound on the size of $K$ is minimal.*

**Uniform sampling** For the generation of k-samples, we naturally need a method to sample the network peers. We present in the following sections some particular sampling techniques aimed at minimizing the sampled set's size. But before doing so, let us consider some of the standard methods used nowadays for peer sampling in P2P networks. Peer sampling has received much attention in recent research on P2P networks and is used in various applications for data dissemination, gossip-based communications and querying (see e.g. [11, 15, 9]). Much of the work has focused on *uniform sampling* of network peers, proposing various techniques that vary in their resilience to failures, communication overhead, etc. [16, 13]. Ignoring the particular algorithmic details, a uniform peer sampling technique can be viewed as a function $GetProvidersSample(R, l)$, where $R$ is a Boolean predicate on peers and $l$ is the size of the required sample. $GetProvidersSample(R, l)$ returns $l$ peers, drawn with uniform probability, from the set of all network peers that satisfy the predicate $R$. In our context we are interested in sampling peers that provide a certain set of objects. For a set of information objects $O$, $R_O$ will denote the predicate that is true for those peers that provide some object in $O$. Namely, for a network peer $n$, $R_O(n) = True$ iff $O \cap objects(n) \neq \emptyset$.

## 3 Uniform k-sampling

Given an objects-providers graph $g$ and a set of objects $O$ in $g$, our goal is to generate the smallest possible k-samples for $O$. Namely k-samples where the maximal size of the set $K$ of peer IDs (in any of the random instances) is minimal. The first sampling method that we present is based on uniform sampling. It has the advantage that it can employ any of the standard P2P sampling techniques mentioned in the previous section (hence enjoy whatever benefits they bring, such as resilience to failures, communication efficiency, etc.). To get some intuition, let us first describe two simple (not necessarily optimal) ways to use uniform sampling for the generation of a k-sample. Our novel sampling method is presented next (first intuitively and then formally). In all the examples below we assume that we are given the objects-providers graph of Figure 1 and we want to generate a k-sample for the set of objects $O = \{o_1, o_2, o_3, o_4\}$.

**Method 1:Individual sample for each object** The naïve sampling algorithm, described in the introduction, samples $k$ providers for each object $o \in O$, by running $GetProvidersSample(R_{\{o\}}, k)$. The set $K$ of the k-sample then consists of the union of the sampled sets, with the function $\textsc{Providers}(o)$ returning $o$'s sample. Clearly the size of the k-sample here is bounded by $k|O|$. In our running example, for $k = 3$ the size is bounded by $3 \cdot 4 = 12$.

Implementation wise, the $K$ peers are transmitted in an array, containing essentially a concatenation of the individual object samples. $\textsc{Providers}(o)$ returns for each object $o$ the corresponding array fragment. It is important to note that a simplistic implementation of $\textsc{Providers}(o)$ that simply returns *all* the peers in $K$ that provide $o$ would be inadequate as it may violate the *fairness* of the k-sample. To see this, consider the following simple example. Assume that the set $O$ contains two objects, $o_u$, an unpopular object provided by a very small fraction of the peers, and $o_p$, a very popular object provided by most peers. Consider a peer $n$ that happens to provide both objects. $n$ has a high probability of being sampled for $o_u$ (hence of appearing in $K$), a higher probability than any of the other providers of $o_p$. To guarantee the fairness of $\textsc{Providers}(o_p)$, it is essential to restrict its answer to those peers sampled (uniformly) for $o_p$, ignoring those other peers in $K$ (like $n$) which happen to also provide $o_p$.

**Method 2: One sample for all objects** An alternative method is to run only one sampling instance, $GetProvidersSample(R_O, l)$, drawing a single sample of size $l$ (for some constant $l$, to be defined below), from the set of *all the providers* of objects in $O$. The set $K$ here consists of all the peers in this sample, with the function $\textsc{Providers}(o)$ returning the network peers $n \in K$ for which $o \in objects(n)$. The use of uniform sampling guarantees that requirement 2 (fair sample) holds. To satisfy requirement 1 (at least $k$ providers for each object), the size $l$ of the sample should be large enough to contain $k$ providers even for non-popular objects (i.e. objects provided only by few peers). Clearly if the least popular object $o \in O$ is provided by $\beta$ of the providers of $O$, to assure an expectancy of $k$ providers for $o$, the sample size should be at least $l = \left\lceil k \cdot \frac{1}{\beta} \right\rceil$.

In our running example, the least popular objects $o_1$ and $o_2$ are each provided by 0.29 of the providers of $O$. Consequently, the size of the required sample (hence also the bound on size of the k-sample) is $\lceil 3 \cdot \frac{1}{0.29} \rceil = 11$, a bit smaller than the one obtained with the previous naïve construction.

In general this method beats the naïve construction whenever $|O| > \frac{1}{\beta}$, $\beta$ being the relative popularity of the least popular object in $O$. It performs particularly well when there are no “very unpopular” objects. For instance, in the extreme case where all objects are provided by all providers, $\beta = 1$ and a k-sample of size $k$ suffices. The naïve construction, on the other hand, is superior when some objects are provided by only a very small fraction of the peers.

**Method 3: Object Partitioning** The new method that we propose in this paper combines the advantages of the two previous ones. It partitions the objects into several sets $s_1, \ldots, s_m$. A sample is then drawn for each set $s_i$, from the set of all the providers of objects in $s_i$. The size of the sample for $s_i$ is dictated, as above, by the popularity of the least popular object in $s_i$. The set $K$ of the k-sample consists of the union of the $s_i$ samples. Finally, for any object $o \in s_i$, the function PROVIDERS($o$) returns the peers, in $s_i$'s sample, which provide $o$.[2] Observe that the previous two methods are in fact special cases of this new method: In the first naïve sampling we have $|O|$ singleton sets, one per each object in $O$; in the second method there is a single set $s_1$ consisting of all the objects in $O$. If the least popular object in a set of objects $s_i$ is provided by $\beta_{s_i}$ of the providers of $s_i$, then a call to $GetProvidersSample(R_{s_i}, \left\lceil k \cdot \frac{1}{\beta_{s_i}} \right\rceil)$ will draw a sample with an expectancy of at least $k$ providers for each object $o \in s_i$. The maximal size of the obtained k-sample is thus $\sum_{s_i} \left\lceil k \cdot \frac{1}{\beta_{s_i}} \right\rceil$. The challenge is *to find the optimal partitioning of objects into sets so as to minimize this sum.*

Consider again our running example and assume that we partition the objects in $O$ into two sets, $s_1 = \{o_1, o_2\}$ and $s_2 = \{o_3, o_4\}$. The providers of objects in $s_1$ (represented in the graph by $p_1$, $p_2$ and $p_3$) form 0.43 of the overall set of peers. $o_1$ and $o_2$ are each provided by 0.67 of these providers, hence the required size of $s_1$, for $k = 3$, is $\lceil 3 \cdot \frac{1}{0.67} \rceil = 5$. Similarly, the providers of objects in $s_2$ (represented by $p_3, \ldots, p_6$) form 0.61 of the overall set of peers. $o_3$ and $o_4$ are each provided by 0.76 of these providers, hence the required size of $s_2$ is $\lceil 3 \cdot \frac{1}{0.76} \rceil = 4$. Thus, the size of the k-sample here is bounded by $5 + 4 = 9$, smaller than in any of the previous two methods. In this example this is also the optimal partitioning.

Observe that the reduction in size here, relative to the previous two methods, is not too big because our running example contains, for simplicity, only very few information objects. In practice, results on real-life scenarios demonstrate significant size reduction [17].

## 3.1 Formal Problem Statement

Consider a partitioning of the objects in $O$ into (not necessarily disjoint) sets $s_1, \ldots, s_m$. As explained above, the size of the k-sample generated for such a partitioning is bounded by $\sum_{s_i} \left\lceil k \cdot \frac{1}{\beta_{s_i}} \right\rceil$, where $\beta_{s_i}$ is the fraction of peers, among the providers of $s_i$, providing the least popular object $o$ in $s_i$. To make this more precise we use the following notations.

Given an objects-providers graph $g$ and a set $s$ of information objects in $g$, the *popularity of the set* $s$ (in $g$), denoted $SPop(s)$, is the fraction of peers,

[2] Implementation wise, the sent k-sample is an array containing the concatenation of the samples drawn for the $s_i$ sets. Given the partitioning details (the object sets and their samples' size), PROVIDERS($o$) returns for each $o \in s_i$, the peers $n$ in the $s_i$ fragment of the array s.t. $o \in objects(n)$.

among all providers, that provide some object in $s$. Putting this in terms of the graph $g$, $SPop(s)$ is the sum of the weights of the provider nodes in $g$ that are neighbors to the object nodes in $s$. Namely, $SPop(s) = \sum_{p \in N(s)} w(p)$.

Observe that when $s$ contains a single object $o$, the set's popularity is precisely the popularity of $o$, as defined in Section 2, namely the fraction of peers, among all providers, that provide $o$. The *relative-popularity of $o$ w.r.t $s$*, denoted $relPop(o, s)$, is the fraction of peers, among the providers of $s$, that provide $o$. Namely, $relPop(o, s) = \frac{SPop(\{o\})}{SPop(s)}$.

Going back to the generation of our k-sample, the sample size for a set $s_i$ is dictated by the object $o \in s_i$ with the least relative-popularity. Namely, the sample size for $s_i$ should be at least $\left\lceil k \cdot \frac{1}{min_{o \in s_i} relPop(o,s_i)} \right\rceil$. Consequently, for a partitioning $S$ of the objects in $O$ to sets, the overall size of the k-sample is bounded by

$$(*) \sum_{s_i \in S} k \cdot \frac{1}{min_{o \in s_i} relPop(o, s_i)} = \sum_{s_i \in S} \frac{k \cdot SPop(s_i)}{min_{o \in s_i} SPop(\{o\})}$$

The value of (*) naturally depends on the particular partitioning of objects to sets. We denote by $size^{\lceil\rceil}(S)$ the value of (*) for a given partitioning $S$. For an objects-providers graph $g$, a set of objects $O$ in $g$, and an integer $k$, we refer to the problem of finding a partitioning $S$ of the objects in $O$ for which $size^{\lceil\rceil}(S)$ is minimal as the *Object Partitioning Problem* (denoted OPP$^{\lceil\rceil}$). We will call such a partitioning $S$ an *optimal* solution (for $g$, $O$, and $k$) and denote the value of $size^{\lceil\rceil}(S)$ for it by $opt^{\lceil\rceil}$.

We will also define a variant of this problem, denoted OPP (with no ceiling), where the objective is to minimize the value of

$$(**) \ size(S) = \sum_{s_i \in S} \frac{k \cdot SPop(s_i)}{min_{o \in s_i} SPop(\{o\})}$$

The size of an optimal solution for OPP is denoted by *opt*. For some purposes, such as to prove NP-Hardness of OPP$^{\lceil\rceil}$, we will go through OPP first.

### 3.2 Observations

We provide next some observations about the possible structure of an optimal solution. These will be useful later for studying the complexity of the problem and for proposing algorithms to solve it.

**Provider contribution** Consider some objects-providers graph $g$, a set of objects $O$, and a partitioning $S$ of the objects of $O$ into sets. Let $s_i \in S$ be some objects set and let $o_i \in S_i$ be the least popular object in $s_i$. Looking at the formula (*) from the previous subsection, it is easy to see that for every object $o \in s_i$, each of its providers $p$ (neighbor nodes in the graph $g$) contributes to $size^{\lceil\rceil}(S)$ a value $\frac{k \cdot w(p)}{SPop(\{o_i\})}$ (not yet regarding the ceiling we have to take). If a provider provides objects that belong to different sets, then the provider will contribute to $size^{\lceil\rceil}(S)$ such a value for each of these sets. We can therefore see that a provider's contribution to the value of $size^{\sqcap}(S)$ depends on the number of sets that the objects that it provides participate in, and on the popularity of the least popular object in each of these sets.

Looking at things from the provider's view point, the partitioning of the objects into groups can be viewed as *labeling* each provider with labels that identify the sets to which the objects that it provides belong. Each label placed on a provider induces a cost that he has to pay, and all providers who provide a given object must have at least one label common to all of them (describing the set(s) that contain the object). Our problem can thus be described as a labeling problem: *we want to find an optimal labeling for the providers, namely one that minimizes their overall contribution to* $size^{\sqcap}(S)$. This alternative description of the problem will prove useful later on in the various proofs.

**"Nice" partitioning** Given an objects-providers graph $g$ and a set of objects $O$, a partitioning $S$ for the objects in $O$ is called a *nice partitioning* if (1) all the sets $s \in S$ are pairwise disjoint and (2) there are no two distinct sets $s, s'$ in $S$ whose least popular objects are equally unpopular. Namely for all $s, s' \in S, s \neq s' \rightarrow min_{o \in s} SPop(\{o\}) \neq min_{o' \in s'} SPop(\{o'\})$.

The following lemma shows that when searching for an optimal solution for OPP$^{\sqcap}$ (resp., OPP) it is sufficient to look at nice partitioning.

**Lemma 1.** *For every objects-providers graph $g$, a set of objects $O$ in $g$, and an integer $k$, there always exists a nice partitioning for the objects in $O$ which is an optimal solution for* OPP$^{\sqcap}$ *(resp., for* OPP*).*

*Proof.* We prove the lemma for OPP$^{\sqcap}$. The proof for OPP follows exactly the same lines. Let us look at some optimal solution $S$ for OPP$^{\sqcap}$. Consider the nice partitioning $S'$ obtained from $S$ by removing redundant objects and unifying sets with equal least object popularity. It is easy to see from formula (∗) that $size^{\sqcap}(S') \leq size^{\sqcap}(S)$. Since $S$ is an optimal solution it must be the case that $size^{\sqcap}(S') = size^{\sqcap}(S)$, hence $S'$ is an optimal solution as well.

The above lemma is interesting since it allows to reduce the search space when searching for an optimal solution. Specifically, in every nice partitioning the number of sets is bounded by the number of the distinct popularity values for objects in $O$. We will use this extensively in our analysis of the problem.

### 3.3 Algorithms and Complexity

We will see below that both OPP$^{\sqcap}$ and OPP are NP-Hard and will propose linear and polynomial approximation algorithms for them. But before we do that, let us first consider a restricted case that can be solved in polynomial time and can shed some light on the structure of optimal solutions.

Consider an objects-providers graph $g$ and a set $O$ of information objects in $g$, where the objects in $O$ each have one of two possible popularity values. This is for instance the case in our running example, where objects in $O$ have popularity 0.29 or 0.54. Indeed, the popularities of $\{o_1, o_2, o_3, o_4\}$ are respectively $\{0.29, 0.29, 0.54, 0.54\}$.

By Lemma 1 we know that in this case there exists a nice optimal partitioning for $O$ consisting of (at most) two sets, $s_1$ and $s_2$, such that all the less popular objects belong to $s_1$ while the more popular objects may be spread between $s_1$ and $s_2$. To find the optimal partitioning we only need to to determine which popular objects belong to $s_1$ and which to $s_2$. We show next that this can be determined in PTime. We first consider OPP and then OPP$^{\lceil\rceil}$.

**Theorem 1.** OPP *can be solved in polynomial time when the information objects in $O$ each have one of two possible popularity values.*

*Proof.* (sketch) We provide here an intuition for the polynomial algorithm. The full algorithm and its correctness proof appear in the full version of the paper [17]. The algorithm works as follows. First, given the objects-providers graph $g$ and a set of objects $O$, we construct a new weighted bipartite graph $G$. For every provider node $p \in g$ that provides some object in $O$, the graph $G$ contains two nodes, $v_p^1$ and $v_p^2$. The node $v_p^i$, $i = 1, 2$ will be used to represent the case where none of the $O$ objects provided by $p$ belongs to the set $s_i$ (The graph edges and the weights of the nodes are described later on).

Next, we find a maximum-weight independent set[3] $W$ for $G$ (known to be polynomially solvable for bipartite graphs [12]). Finally, we use $W$ to determine the partitioning of objects of $O$ into the sets $s_1, s_2$: For every provider node $p$ s.t. $v_p^1 \in W$, all the objects in $O$ provided by $p$ are placed in $s_2$. For every provider $p$ s.t. $v_p^2 \in W$, all the objects in $O$ provided by $p$ are placed in $s_1$. The remaining objects are placed arbitrarily.

**Theorem 2.** *The same algorithm is an approximation algorithm for* OPP$^{\lceil\rceil}$ *that gives a solution whose value is at most $opt^{\lceil\rceil} + 1$, where $opt^{\lceil\rceil}$ is the size of an optimal solution.*

*Proof.* Let $opt^{\lceil\rceil}$ be the optimal value of OPP$^{\lceil\rceil}$ for an objects-providers graph $g$ and a set of objects $O$ in $g$. Let $S$ be the solution produced by the above algorithm. Let $opt$ be the optimal value of OPP on the same instance. We have shown that $opt = size(S)$. We wish to prove that $size^{\lceil\rceil}(S) \leq opt^{\lceil\rceil} + 1$. Obviously, $opt \leq opt^{\lceil\rceil}$. On the other hand, $size^{\lceil\rceil}(S) < opt + 2$, because the only change in the objective function is the ceiling sign added to each of the two summands. Therefore, $size^{\lceil\rceil}(S) < opt^{\lceil\rceil} + 2$. Since both expressions here are integral, it follows that $size^{\lceil\rceil}(S) \leq opt^{\lceil\rceil} + 1$.

It is open whether OPP$^{\lceil\rceil}$ has an exact polynomial solution in the case of two object popularities. We can show however that as soon as objects have more than two distinct popularities, both OPP and OPP$^{\lceil\rceil}$ become NP-Hard. The proof appears in [17].

[3] An *independent set* is defined as a subset $W$ of the nodes in the graph $G$ such that no pair of nodes in $W$ is connected by an edge in $G$.

**Theorem 3.** $\text{OPP}^{\lceil\rceil}$ *and* OPP *are both NP-Hard, even for objects-providers graphs where the object nodes have only three popularity values and all the weights of the provider nodes are equal.*

*Proof.* (sketch) We prove that the problems are NP-Hard by reduction from the problem of minimum unweighted 3-multiway cut (3MC) (also referred to in the literature as 3-Terminal cut or 3-multiterminal cut), known to be NP-Hard [6]. In the 3MC problem a graph $G$ and three distinguished nodes $v_1, v_2, v_3$ are given. The objective is to find a partitioning of the nodes in $G$ into three sets $V_1, V_2, V_3$ s.t. for $i = 1, 2, 3$ node $v_i$ is in partition $V_i$, so as to minimize the number of edges going between different partitions. The full reduction is given in [17].

Clearly an optimal solution can be found in exponential time by enumerating all possible solutions. As this is too expensive, we propose next two simple algorithms that approximate the optimal solution up to a constant factor.

**The PartitionByPopularity algorithm** Our first approximation algorithm partitions the information objects into sets based on their popularity. It has several advantages: (1) It is simple and runs in linear time. (2) It does not require knowing the exact structure of the objects-providers graph but only the popularity of the objects.[4] (3) It is *on-line* for the objects, namely if a new object is added then the partitioning of already-existing objects does not change.

To describe the algorithm we use the following notation. For two numbers $c > 1$ and $x > 0$, let $IPower_c(x) = c^{\lfloor \log_c x \rfloor}$. That is, $IPower_c(x)$ is the integral power of $c$ smaller than $x$ and closest to it. Given a constant number $c$, the algorithm PARTITIONBYPOPULARITY($c$) partitions the objects in $O$ into sets based on the value of $IPower_c(SPop(\{o\}))$. Formally, we define $s_i = \{o : IPower_c(SPop(\{o\})) = c^i\}$, and the solution $S$ is simply the collection of all non-empty $s_i$'s.

**Theorem 4.** PARTITIONBYPOPULARITY($c$) *is a* $\frac{c^2}{c-1}$*-approximation algorithm for* OPP *for any* $c > 1$*. Namely, it gives a solution whose value is at most* $\frac{c^2}{c-1} \cdot opt$*, where opt is the size of an optimal solution.*

**Corollary 1.** *(Optimizing on the value of c)* PARTITIONBYPOPULARITY(2) *is a* 4*-approximation algorithm for* OPP.

The proof is omitted for space constraints. We have a slightly weaker bound for $\text{OPP}^{\lceil\rceil}$.

**Theorem 5.** PARTITIONBYPOPULARITY($c$) *is a* $\frac{2c^2}{c-1}$*-approximation algo. for* $\text{OPP}^{\lceil\rceil}$ *for any* $c > 1$.

**Corollary 2.** *(Optimizing on the value of c)* PARTITIONBYPOPULARITY(2) *is an* 8*-approximation algorithm for* $\text{OPP}^{\lceil\rceil}$.

[4] This can be easily obtained in a distributed manner by a simple uniform peers sampling.

We conclude with a remark about the tightness of the approximation factor of PartitionByPopularity(2). We provide in [17] a particular example of inputs to OPP for which the algorithm indeed produces a result 4 times the value of the optimal solution (it is open whether the 8 factor for $\mathrm{OPP}^{\lceil\rceil}$ is indeed tight). Our experiments[17] show however that this algorithm yields in practice results much better than its worst case bound.

**The OrderByPopularity algorithm** Our second approximation algorithm is of polynomial time complexity. While its worst-case constant factor is the same as that of PartitionByPopularity, it yields in practice even better results. The algorithm first sorts the information objects according to their popularity, and puts them (in sorted order) into an array. To partition the objects it splits the array into non-overlapping intervals. Each partition contains the objects in the corresponding interval. To find the optimal splitting it employs standard dynamic programming. The overall complexity of the dynamic programming phase here is $o(|g| \cdot |O|^2)$: There are $o(|O|^2)$ possible intervals (i.e. partitions) to check, and the contribution of each partition to the overall length of the k-samples can be computed in time linear in $g$.

It is easy to see that partitions obtained by Methods 1 and 2 described previously, as well as by PartitionByPopularity, all belong to the search space of this algo.: in all these algorithms, each partition, when sorted internally by object popularity, forms a continuous interval of the overall popularity order of objects. The k-samples constructed by OrderByPopularity are thus assured to be at least as compact as those generated by the previous algos.

## 4 Non-Uniform Sampling

The sampling methods described in the previous section build on standard P2P uniform sampling tools. An alternative approach, based on non-uniform sampling, is considered briefly next. We first show that, given an objects-providers graph $g$ and a set $O$ of requested objects, the minimal bound on the size of the k-samples (for $O$) can be determined in time linear in $g$. Next we describe a non-uniform sampling method achieving this bound.

**Size** Consider a P2P network with a set $N$ of peers providing some object in $O$. Let $p$ be some provider node in the object-providers graph $g$. We denote by $o_p$ the least popular object in $O$ that $p$ provides. Note that for any object $o \in O$, $SPop(\{o\}) \cdot |N|$ is the absolute number of nodes that provide $o$ (for definition of $SPop$ see Sec. 3). Define $\bar{X}_p = \frac{k \cdot w(p)}{SPop(\{o_p\})}$. Denote $\bar{l} = \sum_{p \in g} \bar{X}_p$, and $l = \lceil \bar{l} \rceil$. We claim that $l$ is the minimal possible bound on the size of the k-sample for $O$. A proof to the claim is in [17].

**Sampling** To generate a sample of this size, we sample the network peers, non-uniformly: For every provider node $p$ in the graph $g$, each of the nodes $n$ that it represents is selected with probability $X_n = \frac{k}{SPop(\{o_p\}) \cdot |N|}$. In [17] we describe a simple distributed P2P algorithm to perform such sampling. The algorithm is an adjustment of an existing P2P uniform-sampling method (RanSub [16]) to non-uniform sampling. Let $K$ be the set of sampled peers. To complete the k-sample's definition, we define what subset of peers ($K_0 \subseteq K$) is returned by $\textsc{Providers}(o)$ for every object $o \in O$. To ensure fairness, $\textsc{Providers}(o)$ samples peers from $K$ as follows: Each peer $n \in K$ that provides $o$ is chosen to be in $K_o$ with probability $\frac{k}{SPop(\{o\}) \cdot |N|}/X_n$ [5]. We prove in [17] that the sampling is fair and that $E[|K_o|] \geq k$.

## 5 Related Work and Conclusion

We studied in this paper the problem of peer sampling for the dissemination of data in a peer-to-peer network. We introduced the notion of k-samples - fair samples that contain a sufficient number of providers for each requested object. We then proposed algorithms with varying complexity for the generation of compact k-samples. To illustrate the benefit that our new sampling techniques can bring to existing data dissemination platforms, we tested experimentally the performance improvement that can be obtained by incorporating them in the popular BitTorrent[2] platform, showing significant gain. The experimental result are reported in[17]. Two questions that remain open are the existence of an exact polynomial solution for $\text{OPP}^{\sqcap}$ in the case of two object popularities, and the tightness of the 8 factor in the approximation algorithm for $\text{OPP}^{\sqcap}$.

Peer sampling is used extensively in epidemic/gossip-based methods for disseminating information between network nodes. Such data dissemination is the basis of a wide range of application, including replicated databases [1], content distribution [11], failure detection [20] and probabilistic multicast [9]. Those methods depend on nodes acquiring information about other nodes in the network, or a *sample* in large networks. Peer sampling is also used in a variety of P2P systems (e.g. [4]) and for load balancing [7]. The compact samples generated by our algorithms may help to reduce the communication overhead in such applications. There has been an extensive previous work on uniform peer sampling in P2P networks, ranging from dedicated techniques developed for a particular system or protocol (e.g. [8]), to general-purpose sampling algorithms (e.g. [16, 13]). All these methods sample each requested object separately, yielding, for a set $O$ of requested objects, samples of size bounded by $k|O|$. To our knowledge the present work is the first attempt to use correlations between object requests to reduce the sample size. Non-uniform sampling has received relatively little attention. An algorithm in which a probability for each node to appear in the sample may be defined, is given for instance in [21]. Our non-uniform k-sampling method from Section 4 can be built on top of such an algorithm.

We are currently incorporating our k-sampling algorithms in the Information Dissemination Platform (IDiP) developed in Tel Aviv University. More generally we believe them to be useful in the general context of publish/subscribe systems [10, 19], where users inherently have heterogeneous needs.

[5] This probability is $\leq 1$ because $\frac{k}{SPop(\{o\}) \cdot |N|}/X_n = \frac{SPop(\{o_p\})}{SPop(\{o\})}$ and we assumed that $SPop(\{o_p\}) \leq SPop(\{o\})$.

## References

1. D. Agrawal, A. El Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases. *PODS'97*.
2. Bittorrent. `http://bittorrent.com`.
3. J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. *SIGCOMM'02*.
4. M. Castro, P. Druschel, A.M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC, 20(8)*, October 2002.
5. B. Cohen. Incentives build robustness in BitTorrent. *In Proc. of the Workshop on the Economics of P2P Systems, Berkeley, CA*, 2003.
6. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM*, 23(4):864–894, 1994.
7. M. Dahlin. Interpreting stale load information. *The 19th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, May 1999.
8. P. Eugster, S. Handurukande, R. Guerraoui, A. Kermarrec, and P. Kuznetsov. Lightweight probabilistic broadcast. *In Proc. of The Int. Conf. on Dependable Systems and Networks (DSN 2001)*, July 2001.
9. P. T. Eugster and R. Guerraoui. Probabilistic multicast. *In Proc. of the Int. Conf. on Dependable Systems and Networks (DSN'02)*, June 2002.
10. F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. *In Proc. of ACM SIGMOD'01*.
11. M. J. Freedman, E. Freudenthal, and D. Maziéres. Democratizing content publication with Coral. *In Proc. 1st USENIX/ACM Symp. on Networked Systems Design and Implementation (NSDI '04)*, 2004.
12. M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.
13. C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. *INFOCOM'04*.
14. M. Jelasity, R. Guerraoui, A. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. *5th Int. Middleware Conference, Toronto*, October 2004.
15. D. Kostic, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat. Maintaining high bandwidth under dynamic network conditions. *USENIX*, 2005.
16. D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat. Using random subsets to build scalable network services. *In Proc. of USITS'03*, 2003.
17. T. Milo, A. Sagi, and E. Verbin. Compact samples for data dissemination (full version). Tech. Report. `http://www.cs.tau.ac.il/ milo/work/Samples.pdf`.
18. Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *Int. Symp. on Information Theory*, June 2001.
19. M. Petrovic, H. Liu, and H. Jacobsen. CMS-ToPSS: efficient dissemination of RSS documents. *VLDB'05*.
20. S. Ranganathan, A. D. George, R. W. Todd, and M. C. Chidester. Gossip-style failure detection for scalable heterogeneous clusters. *Cluster Computing*, 4(3):197–209, July 2001.
21. M. Zhong, K. Shen, and J. Seiferas. Non-uniform random membership management in peer-to-peer networks. *INFOCOM'05*.