Informed-Source Coding-On-Demand (ISCOD) over Broadcast Channels

Yitzhak Birk and Tomer Kol
Electrical Engr. Dept, Technion
Haifa 32000, ISRAEL
birk@ee, tkol@psl.technion.ac.il

Abstract— **We present the Informed-Source Coding-On-Demand (ISCOD) approach for efficiently supplying non-identical data from a central server to multiple caching clients through a broadcast channel. The key idea underlying ISCOD is the joint exploitation of the data already cached by each client, the server's full awareness of client-cache contents and client requests, and the fact that each client only needs to be able to derive the items requested by it rather than all the items ever transmitted or even the union of the items requested by the different clients. We present a set of two-phase ISCOD algorithms. The server uses these algorithms to assemble ad-hoc error-correction sets based its knowledge of every client's cache content and of the items requested by it; next, it uses error-correction codes to construct the data that is actually transmitted. Each client uses its cached data and the received supplemental data to derive the items that it has requested. This technique achieves a reduction of up to tens of percents in the amount of data that must be transmitted in order for every client to be able to derive the data requested by it. Finally, we define *k*-partial cliques in a directed graph, and cast the two-phase approach in terms of partial-clique covers. As a byproduct of this work, bounds and a close approximation for the expected cardinality of the maximum matching in a random graph have been derived and are outlined.**

Key words and phrases: *ISCOD, caching clients, information-dissemination, multicast, k-partial clique, maximum matching, error correcting codes.*

I. Introduction

Consider a setting of many *caching* clients being fed by a common server via an expensive, high-speed *forward* broadcast channel. A slow *reverse* channel is also available for control and meta-data. Transmissions are typically initiated by the server which may, for example, be broadcasting a daily newspaper, and each client "caches" part of the received information. "Pushing" information in advance rather than supplying it on demand helps convert latency requirements into (typically easier) throughput demands; it also helps (temporally) balance the communication load. Recent commercial examples of information-*pushing* are [1] [2]. The combination of pushing, a broadcast channel and caching clients also reduces the total amount of traffic.

At any given time, each client's cache contains some subset of the transmitted items. An item may be missing from a client's cache due to reception problems, insufficient storage capacity, the lack of permission to record it, or lack or interest. As a need arises, a client may request the retransmission of one or more items. Possible applications of such systems include information-dissemination to static as well as mobile clients.

Fig. 1 depicts such a system, wherein the forward channel is a satellite link and the reverse channel is a slow terrestrial link. Other implementations might entail the use of cable-TV infrastructure for both directions or, alternatively, the use of telephone lines for the reverse links.

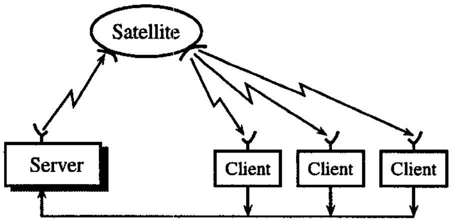


Fig. 1. Data disseminated by a server to caching clients via a broadcast channel. A separate, usually much slower, return channel, is also included.

The combination of broadcast channels, caching clients and data pushing is unquestionably very attractive, provided that the forward broadcast channel can be utilized effectively. The cost of transmission over such a channel, in turn, is amortized over the number of interested receivers, so using it for granting individual requests is very inefficient. Our focus in this paper is therefore on minimizing the forward-channel bandwidth required for "filling holes" in the client caches, i.e., for replenishment of previously-transmitted items, rather than on the original transmission of the information. Thus, our work is complementary to schemes such as the multicast file-transfer protocol (MFTP) [3], whose focus is on an efficient implementation of the original dissemination over point-to-point links.

Throughout the paper, we assume a system with the following characteristics.

- A broadcast erasure forward channel.
- A slow, error-free reverse channel.
- No peer-to-peer data communication.
- Caching clients with some computing power; perfect storage.
- Some correlation among clients' interest profiles.

As the main contribution of this paper, we present and analyze *Informed-Source Coding-On-Demand.* ISCOD uses full knowledge of the clients' states and of their exact needs to reduce the amount of information that must be transmitted over the forward channel. When used for "filling holes" in client caches, the savings relative to previous schemes may be as high as tens of percents. The paper is organized as follows. In section II, we flesh out the problem space in order to fully understand the possibilities, and introduce a taxonomy for classification of similar problems. Section III surveys and classifies existing approaches. In section IV, we present the ISCOD approach and provide some indication of the prospective savings that it may offer. In section V, a 2-phase method for ISCOD algorithms is proposed and is cast in graph-theoretic terms. A family of 2-phase ISCOD algorithms is then presented, analyzed and compared with prior art. Section VI discusses scalability, the reverse channel and other implementation-related issues. Section VII summarizes the paper and suggests directions for further research.

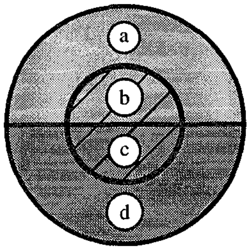
This research was supported in part by a grant from News Data Systems Ltd.

II. Problem Statement and a Classification Taxonomy

A. Problem statement

The state of a client's cache. At any given time, a subset of the items (that were ever transmitted) is *present* in a given client's cache. The remaining items are *missing*. By another classification, some of the transmitted items are *needed* by the client while others are not. We further classify missing items into items which are needed by the client and are therefore *requested* from the server, and *absent* items which are neither cached nor needed by the client.

Fig. 2 illustrates a client's classification of items. (Note that a client need not be aware of absent items. Moreover, items that are absent from *all* caches are ignored altogether.)



Legend:

All items ever transmited	⇔	$a \cup b \cup c \cup d$
Missing items	⇔	$a \cup b$
Present items	⇔	$c \cup d$
Needed items	⇔	$b \cup c$
Requested items	⇔	b
Absent items	⇔	a

Fig. 2. Classification of transmitted items' status in relation to a given client's needs and cache content.

Requirement. The server must transmit information that will enable each client to derive the items requested by it.

Design goal. For any given state of the clients, we want to minimize the amount of data that must be transmitted in order to meet the requirement.

Definition 1 *Given the total number of unique requested items NumUniqReq and the amount of communication ReqComm required by a proposed transmission scheme, the* ***savings effected by the transmission scheme*** *is*

$$savings(scheme) = \frac{NumUniqReq - ReqComm}{NumUniqReq} \tag{1}$$

Remark. The savings can also be defined relative to the number of (not necessarily different) requests. However, we will refer to the trivial optimization of broadcasting a single copy of an item that was requested by multiple clients as the baseline.

B. A taxonomy for problem classification

The problem space has three main dimensions:

- The *initial state* of the clients. (Empty or non-empty caches.)
- The *target state* of the clients. Possibilities for the guaranteed contents are: the items requested by the client; the union of the items requested by the clients; all the items that have been transmitted since a specified time.
- The *server's knowledge* of client states. Possibilities are (later ones include earlier ones): erasure statistics of the communication channel; number of items missing in each cache; number of items requested by each client; identity of items requested by each client; full state of each client.

Clearly, the initial state of a client's cache cannot be regarded as non-empty if it is empty; however, it may be treated as empty even if it is not. Similarly, in its target state, a cache must contain at least the items that were requested by its owner (in addition to the present ones), but it may contain additional ones. These issues are related both to the knowledge available to the server about the initial states of the client caches and to the transmission scheme. As we shall see, existing schemes use this "slack", rendering them relatively inefficient when applied to our problem.

III. Possible approaches based on prior art

In this section, we survey existing approaches that can be applied to our problem. They are ordered by decreasing "slack" in server knowledge and target state. (The baseline mentioned earlier is a variant of Case 4 below.)

Case 1. This is the classical case of communication over a lossy channel: every client needs all the data ever transmitted and has the required storage capacity. However, some of the transmitted items may be missing due to channel errors. Assuming at most N_{err} errors, we can construct an error correcting code (ECC) for the desired level of protection. After the application of *source coding*, the data is broadcast by the server. It is implicitly assumed that the caches are initially empty, the target state is for all caches to contain everything, and there is no meaningful client state for the server to know.

Case 2. Here, the client caches aren't empty, and the server knows the number of items missing from each cache (as well as the number of items present in it). Let *max_missed* denote the maximum (over clients) of this number. The best the server can do is construct a systematic error correcting code (ECC) that can correct *max_missed* errors, and send the additional ECC datagrams. The communication cost is thus *max_missed.* Due to lack of knowledge, the only possible target state of the caches is again for every cache to contain all items. This case differs from Case 1 in a fundamental way: the choice of code by the server can be based on definite information rather than on channel statistics.

Such a use of error-correcting codes in this case was suggested by Metzner [4]. As shown there, this approach performs and scales much better than the baseline. (More recent extensions of Metzner's idea entail the use of generalized minimum distance decoding instead of erasure-only decoding [5], as well as the use of adaptive forward error correction using BCH codes [6]).

Case 3. In this case, the server also knows the number of items *requested* by each client. Unfortunately, this additional information is of no real use to the server, as it doesn't know which of the missing items are requested. The server must therefore enable each client to restore all the *missing* items, so the situation is identical to Case 1 in this regard.

Case 4. In this case, which is the typical client-server case, the server also knows the *identity* of the requested items. This can be used to reduce communication by *duplicate elimination*, i.e., transmitting only one copy of an item that was requested by multiple clients. This is our baseline for comparison. (In the baseline case, we do not exploit any knowledge pertaining to the cache contents.) The following example describes a situation in which duplicate-elimination is efficient.

Example 1. *m* items are *missing* from each cache, but all clients only *request* the same single item. Here, it suffices for the server to transmit this item (once). In contrast, in Cases 2 or 3 it would have been required to construct an ECC that can reconstruct any m lost items and to transmit m items worth of data.

With this level of knowledge, however, one can often to better by using ECC, as illustrated by the following example.

Example 2. Each client is missing two unique items (and thus has all the others), of which one is requested (no duplicates). Here, computing from all the items an ECC that can tolerate two erasures enables all clients to reconstruct the items they requested while requiring the transmission of only two items worth of data; as a byproduct, each client can also reconstruct its absent item. This example illustrates that in some cases it is better to raise the goal (in terms of the target state), and the best approach would have already been possible in Case 2.

Fig. 3 summarizes the problem/design space. The numbers refer to the cases. "4(B)" and "4" refer the the baseline and ECC versions of Case 4, respectively.

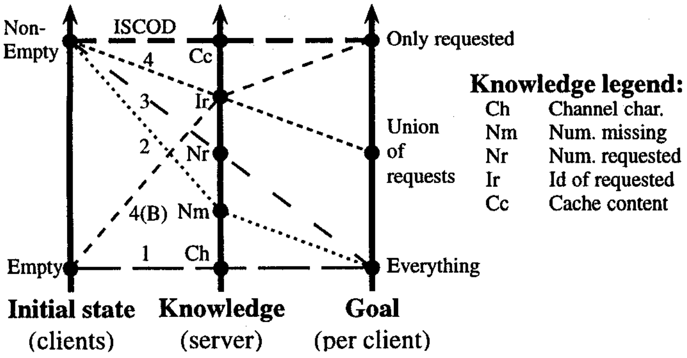


Fig. 3. Classification of the different problem variants and approaches according to the clients' initial state, the server's knowledge about the clients' states and needs (this axis is cumulative, except for 4(B)), and the target client states.

In addition to the approaches just surveyed and with which ISCOD will be compared, there are several additional relevant problems and methods. These include the *Multiple Descriptions Problem (MDP)* and the related *Multiple Diversity Coding* [7] [8], as well as that of coding with *Non zero initial state* [9] [10]. However, these methods can be viewed as complementary to the main thrust of our work: the (new) idea of an ad-hoc creation of coding groups for efficient handling of client requests.

IV. The ISCOD approach

In this section, we present *Informed-Source Coding-On-Demand* (ISCOD). We begin with a brief overview that places this approach in the design space, lays out the general scheme, and provides a motivating example. Next, we discuss client-access models and use them to get an idea of the prospective savings.

A. Overview

Informed-Source Coding-On-Demand entails exploiting full knowledge of client states to achieve the *required* target. (As a side-effect, some caches may end up with additional items.)

It is important to note the subtle difference between the client states known to the server in Case 4 and in ISCOD. In Case 4, the server knows the identities of the requested items but only the *number* of items present in each cache. Accordingly, other than in extreme situations like the one in Example 2, it is unable to beneficially exploit the contents of these caches. ISCOD, in contrast, does so extensively. Moreover, as will be seen later, this does not necessarily imply scalability problems.

Having a fully-informed server and focusing on the true requirement, namely enabling each client to derive its *requested* items, presents an opportunity for a novel approach, as illustrated by the following example.

Example 3. Client A requests item a and has item b in its cache, while client B requests item b and has item a in its cache. Knowing the cache contents enables us to send $a \oplus b$ (bit-by-bit XOR) instead of sending both a and b, thereby reducing the amount of communication by 50%.

Generalizing the above example, ISCOD entails the use of error-correcting codes by the informed source on an Ad Hoc or "on-demand" basis, hence the name. We refer to clients that can be served jointly by using an error-correcting code as *ECC-matched*.

Before continuing our discussion of ISCOD, let us digress to discuss client access patterns and models.

B. Client access model

Two major categories of reference models exist: 1) *dependent* reference models, such as the stack reference model (SRM), whereby item reference probabilities depend on the item's location in an LRU stack and thus on the past, and 2) the *independent reference model* (IRM) [11], whereby the probability p_{item} of referencing any given item is constant over time. This simplification makes the analysis of IRM more tractable [12]. In [13] it is claimed, based on [14], that an actual caching system with dependent references can be closely approximated by the independent reference model with modified *virtual* access probabilities computed in an appropriate way, as described in [14].

The IRM abstraction has been used successfully in many cases involving caches [12], including World-Wide Web traffic [15], [16]. We will use the IRM since, due to local caches, frequent dependent re-references of items are likely to be found in the local cache and to not influence the rest of the system. If each client is actually a proxy serving several users, a client's request stream is likely to exhibit much less temporal locality than a single user's requests. For simplification, the actual access pattern may be approximated to an arbitrary degree of accuracy by grouping items with similar access probabilities into a *segment* [12]. All items within a segment have the same access probability.

According to studies of Internet use, access patterns tend to follow Zipf's law, i.e., have a hyperbolic distribution function [17], [18]. In order to demonstrate the prospective communication savings attainable with ISCOD, we use a 90–10 approximation for each client, whereby 10% of the items referenced by a client belong to its *hot* segment (hot-set) and account for 90% of its references; the remaining 90% form the client's *cold* segment (cold-set) and account for 10% of its references.

There are situations wherein many clients request the same items. Often referred to as the *flash-crowd* phenomenon, one of its instances was caused by the use of the WWW to distribute Shoemaker-Levy comet pictures. This situation is apparently rather common [19] and deserves treatment. We will nonetheless explore the prospects for savings under an assumption of unique requests, since a *flash-crowd* situation can be dealt with very efficiently by *duplicate elimination.*

C. The prospects for savings

Consider initially a setting of m data items, two clients with a 90–10 access pattern, each with an LRU cache and a hit-rate of 0.9 for "hot" items, and thus about 0.028 for "cold" items. (The hit-rate for the hot and cold sets are related through the size of the cache [12].) If such a client needs an item, the probability that it will have to be requested from the central server (cache miss) is

$$P_{req} = P^{h}_{need} \cdot P^{h}_{miss} + P^{c}_{need} \cdot P^{c}_{miss} \approx 0.09 + 0.097 = 0.187. \quad (2)$$

(Superscripts refer to hot and cold segments.)

Assuming identical hot-sets, if each client requests a single item, the probability that we can save by duplicate elimination (both clients request the same item) is

$$\begin{aligned} P_{dup\ saving} &\approx \left(0.09^2/0.1m + 0.097^2/0.9m\right)/0.187 \\ &\approx 0.49/m = o(1/m). \end{aligned} \quad (3)$$

The probability that we can *match* the two requests, i.e., each client has the other's request cached, is

$$P_{match\ saving} \approx ((0.09P^{h}_{hit} + 0.097P^{c}_{hit})/0.187)^2 \approx 0.2 = o(1) \quad (4)$$

This looks promising, but let us look into the case of disjoint hot-sets. In this case, similar calculations yield $0.57/m$ and 0.006, respectively.

The 0.006 probability does not look very promising, at least for $n = 2$ clients. However, for reasons discussed below, for $n = 1000$ clients, this slim probability yields an expected savings of at least 50%, i.e., a factor 2 reduction in communication! Besides, for larger values of n the hot-sets are unlikely to be disjoint.

Having demonstrated the promise of ISCOD, we next present a family of efficient ISCOD algorithms for use by the server.

V. A FAMILY OF ISCOD ALGORITHMS

A. The two-phase approach

For facility of algorithm development and analysis, we only consider 2-phase ISCOD approaches. In the first phase, the server uses information about the clients' states and their requests to find a "good" partitioning of the clients; in the second — it treats each subset separately as one of the previous cases, and constructs a source code. In other words, the Informed-Source uses its knowledge to perform efficient Coding-On-Demand, or *ISCOD.* The server may still use any of the other approaches, and particularly *duplicate elimination* as a preprocessing phase. Our focus will be on the first phase; for the second one, we will initially assume an application of Case 2 independently for each subset of clients, enabling every member of a given subset to derive the union of the items requested by the subset's members.

B. Problem statement – revised

We next somewhat constrain the coding problem in order to facilitate algorithm design and analysis. Problem instances that do not adhere to the constraints can always be transformed into ones that do; the only implication is that we may be giving up optimal solutions. However, the problem in its general form (and, in fact, also in the constrained form) is NP-hard, so we may not be losing anything in practice.

A single request per client. A client that issues $r > 1$ requests can be "split" into r clients, each having the same cache content and issuing a single request. Any solution of the "split" problem will solve the original problem and vice versa due to the broadcast nature of the forward channel, so there are no adverse effects.

Unique requests. Any item is requested by at most one client. In practice, this is equivalent to the use of a preprocessing phase of *duplicate elimination.* To illustrate the potential adverse effects of this constraint, consider the following example.

Example 4. Two clients possess item a and request item b, and two other clients have item b and request item a. The best solution is to send the single datagram $a \oplus b$; however, a simplistic duplicate-elimination preprocessing step would mandate the transmission of both a and b. It should nonetheless be noted that duplicate elimination saves at least 50% for the items to which it is applied. Moreover, a more sophisticated duplicate-elimination step, whereby the intersection of the cache contents of clients with identical requests is used to represent them, would prevent this problem [20].

Equisized items. One can always partition an item into fixed-size units and split the client accordingly. We ignore fragmentation overheads.

C. A graph model

The problem of each of several clients requesting a single unique item can be represented as a graph in which each vertex corresponds to a client and a directed edge (u,v) exists iff u's cache contains the item being requested by v.

Maximum matching. The two requests generated by a pair of matched vertices can be granted by transmitting the bit-by-bit XOR of the two requested items. A maximum matching thus directly yields the highest savings when using only XOR operations among two items.

Minimum clique cover. All the members of a clique (a complete subgraph) can derive their requested items from the bit-by-bit XOR (parity) of those, i.e., a single datagram. Therefore, finding the minimum clique cover would yield the highest savings that can be attained when using only XOR operations. Since this problem is NP-complete [21], we will look into heuristic algorithms for finding small covers.

Partial cliques. We generalize the notion of a clique as follows:

Definition 2 *A directed subgraph $G'(V',E')$ is a k-partial clique $Clq(s,k)$ iff: $|V'| = s$; $\forall v \in V', outdeg(v) \geq (s-1-k)$, and $\exists v \in V', outdeg(v) = (s-1-k)$.*

A cover of a graph by partial cliques is simply a partitioning of its vertices. We next define the cost of a partial-clique cover.

Definition 3 *The cost of covering a given graph with a given set S of k-partial cliques whose parameters are $k_1 \ldots k_{|S|}$ is*

$$CoverCost(S) = \sum_{i=1}^{|S|}(k_i + 1) \quad (5)$$

In our context, $Clq(s,k)$ corresponds to a group of s clients, each missing (not having in its cache) at most k of the $(s-1)$ items jointly requested by the other members of the group, with at least one client missing exactly k of those items. $Clq(s,0)$ is thus an ordinary clique. To "handle" a k-partial clique with $k > 0$, one can use systematic $(k+1)$-erasures correcting codes, e.g., *Reed-Solomon* codes. The communication cost with this approach is thus bounded from above by the *CoverCost*.

Proposition 1 *For* ***undirected*** *graphs, the minimum cost of a cover is equal for ordinary and partial cliques.*

Proof. Ordinary cliques are a special case of partial cliques, so we only have to prove that the use of partial cliques does not result in a lower cover cost.

Let $C(V,E) = Clq(s,k)$ be a k-partial clique that is part of a cover of a given graph G, and let $\bar{C}(V,\bar{E})$: $\bar{E} = \{(u,v)|(u,v) \notin E\}$ be its complementary graph. The cover cost of C is $k+1$. By definition, $\exists v \in C\, s.t.\ \deg(v) = s-k-1 = \delta(C)$. Thus, the highest vertex degree in $\bar{C}$ is $\Delta(\bar{C}) = \deg_{\bar{C}}(v) = (s-1)-(s-k-1) = k$. As the chromatic number of $\bar{C}$, $\chi(\bar{C}) \leq 1+\Delta(\bar{C})$ [22], $\bar{C}$ can be colored by $k+1$ colors. Since the chromatic number of the complementary graph is equal to the size of the minimum clique cover of the original graph, it follows that C can be partitioned into $k+1$ (ordinary) cliques, yielding a cost of $k+1$. □

While the use of partial cliques does not reduce the minimum cover-cost of undirected graphs, it may be beneficial for directed graphs, as demonstrated by the following example.

Example 5. Consider a graph comprising n vertices arranged in a circle, such that there is an edge from each vertex to its $n/2$ nearest neighbors in the clockwise direction. This is an $(n/2-1)$-partial clique, yet it contains no ordinary cliques of cardinality greater than one. So, the minimum cover-cost with partial cliques is $n/2$ as compared with n for ordinary cliques, a 50% savings.

D. First-phase algorithms

Having demonstrated the prospective savings with ISCOD, we next address the problem of finding error-correction sets. The discussion is restricted to sets that can be "handled" by transmitting the XOR of the items requested by the sets' members. Accordingly, we are in search of various (ordinary) clique covers, and may use undirected graphs. Algorithms for finding more general error-correction sets, which entails finding partial clique covers in directed graphs, are beyond the scope of this paper.

The proposed algorithms are evaluated and compared for undirected *random graphs* $G(n,p)$ with with n vertices and a probability p for any two vertices to be connected by an edge. The value of p in a random graph corresponding to an ISCOD scenario depends on the skew of clients' access patterns, cache sizes and hot-set commonality. Its behavior is not necessarily monotonic in these parameters (See [20] for further details).

Maximum matching

The problem of finding a maximum matching in a graph has been dealt with extensively and is well understood (e.g., [23]). Algorithms with polynomial complexity have been developed, E.g., $O(V^{2.5})$ [24] and $O(\sqrt{V}E)$ [25], [26]. Therefore, we do not attempt to develop any new ones.

The mean cardinality of the maximum matching of a random graph $G(n,p)$ can be determined by generating graphs and running the algorithm. However, a close analytical approximation is more convenient. To this end, past research on random graphs has focused on conditions for the existence of a perfect matching with very high probability. Specifically,

Theorem 2 ([27], theorem VII.14) *Let* $p \geq (\log(n)+2\log\log(n)+\omega(n))/2n$, *where* $\omega(n) \to \infty$ *(slowly as we wish). Then almost every (random graph)* $G(n,p)$ *has a matching covering every vertex of degree at least 1, with the exception of at most one vertex.* □

For values of p greater than this threshold, the graph is fully connected with a probability tending to 1 ([27]). Consequently, a perfect matching is expected beyond the threshold, corresponding to a 50%, or a factor of 2, savings in communication.

Our focus is on deriving or bounding the expected size of the matching, and thus the expected savings, across all values of p.

Expected savings by matching – upper bound

We have developed [20] upper and lower bounds on the expected savings of using ISCOD with maximum matching algorithms. For brevity, only some of the results are presented.

Proposition 3 *[20] Given an undirected random graph* $G(n,p)$. *Let* $Isol(n,p)$ *and* $UM_{star}(n,p)$ *denote, respectively, the expected values of the number of vertices with degree zero (isolated) and those with degree one that are left unmatched because they are part of a "star" (several vertices with degree one connected to the same vertex of a higher degree). Then, the expected savings by matching is bounded from above by*

$$UB = \frac{1}{2n}(n - Isol(n,p) - UM_{star}(n,p))$$

where

$$UM_{star}(n,p) = n\left((n-1)p(1-p)^{n-2} + (1-p(1-p)^{n-2})^{n-1} - 1\right)$$

$$Isol(n,p) = n(1-p)^{n-1} \tag{6}$$

□

This upper bound was found by simulation to be within 5% of the mean cardinality for $p < 1/(n-1)$.

Expected savings by matching – an empirical approximation.

During the simulation study, we observed that the expected savings by matching as a function of the mean nodal degree $p \cdot (n-1)$ is virtually identical for all values of n. The savings behaves (within 3% for $p > 1/(n-1)$) as *Savings* $\approx 0.5(1-e^{ap})$. Furthermore, the parameter a was found to be linear (within 5%) in n. To summarize our observation:

Observation 4 *The expected cardinality of the maximum matching in a random graph* $G(n,p)$ *behaves for* $p > 1/(n-1)$ *as*

$$\mathbb{E}(|M|) \approx 0.5n(1-e^{-k(n-1)p}) \tag{7}$$

The parameter k was found to be $k \approx 0.78$. It is worth noting that the range of p to which the above observation applies corresponds to the existence of a giant component in the graph [28].

The combination of the upper bound of Proposition 3 and Observation 4 provides a very close approximation of the expected cardinality of the maximum matching in a random graph.

Clique cover – algorithms.

As mentioned above, finding a minimum clique cover in a graph is NP-complete. One of the simplest heuristics for solving this problem is the greedy algorithm (picking items and then trying to match the rest of the items to those already in the forming clique), and in practice it yields reasonable results (see Fig. 4). The computational complexity is $O(V^3)$

The Least Difference Greedy (LDG) algorithm. The LDG algorithm is a heuristic algorithm that we developed for partitioning a given graph into a small number of cliques (ideally a minimum clique cover). The ISCOD intuition underlying this algorithm is that an item x which is *present* in the caches of all members of a clique that is under construction represents a degree of freedom, since this is a necessary condition for being able to add to this clique a client that is requesting x. The heuristic employed by the LDG algorithm is to try and *minimize the loss of degrees of freedom* whenever we look for candidates for enlarging a clique, as illustrated by the following example.

Example 6. Consider four clients, A,B,C,D, requesting items a,b,c and d, respectively. The cache contents of A,B,C are $\{b,c,d\},\{a\}$, and $\{a,d\}$, respectively. We begin our construction by choosing A. Upon consideration of B as the next member of the clique being formed, it is immediately evident that no further growth would be possible if it is chosen, since the caches of A and B contain no common items. Choosing C, in contrast, leaves a degree of freedom that may be useful. Specifically, if D's cache is found to contain a and c, it can be added to the clique.

The LDG algorithm performed better than the simple greedy clique cover in our simulation study for values of p up to 0.5, as can be seen in Fig. 4. The algorithm's computational complexity is $O(V^3)$, and thus it is a practical option for a real system of moderate size. For further details, see [20].

E. Comparison among different schemes

In this section, we present a comparison among prior art and ISCOD variants for undirected random graphs. For all but the maximum-matching ISCOD, these are simulation results. Results for $n = 500$ and various values of p are presented in Fig. 4, but the relative behavior of the various schemes is similar at least for graphs with up to 5000 vertices. The second-phase algorithm (ECC) used in the ISCOD schemes has a cost equal to that of the cover generated by the first phase.

In MFTP, following the initial attempt to transfer the file, the server sends again every item that is requested by one or more clients. The savings (in retransmissions to fill gaps) for this scheme is thus zero by definition.

The approach proposed by Metzner, which can be viewed as tailored to the case wherein each client needs all the items, results in an expected communication cost of $\mathbb{E}(comm) = n - \min_{v \in V}\{\deg(v)\}$. The savings for values of p that are not close to 1 are strikingly lower than with our ISCOD algorithms.

The curve for maximum matching exhibits a *"knee"* at the threshold value of p, and the saving approaches this technique's natural bound of 0.5. For values of p up to, and a little over, this threshold value, the maximum matching algorithm compares favorably with the other polynomial-complexity algorithms.

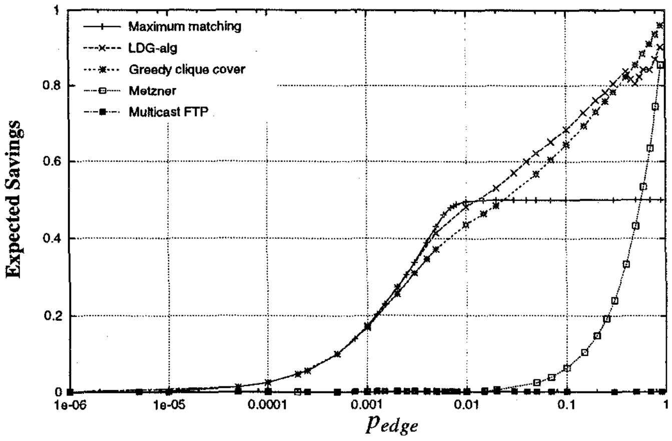


Fig. 4. Expected savings for the various algorithms for $n = 500$

The minimum clique cover algorithms perform as well as maximum matching for small values of p, when the graph is sparse and the minimum clique cover comprises mainly cliques of cardinality two (pairs). They perform better for high probabilities, when they can easily find larger cliques. In the intermediate region, around the perfect matching "knee", the probability of finding larger cliques is not sufficiently high to offset the suboptimal partitioning into cliques of size two by the clique algorithms, hence the crossover. Of the two heuristic algorithm presented, LDG is a little better due to its use of a heuristic estimation of the "damage" caused (to the subsequent iterations) by a decision to group certain clients together [20].

The most important aspect of Fig. 4 is the substantial advantage of the algorithms in the ISCOD family over previous approaches when applied to systems that are the target of this paper: caching clients fed via a broadcast channel. Achieving this saving comes at a certain price, but this overhead can be quite reasonable as we shall see in the next section.

F. More powerful second-phase schemes

The sub-optimality of two-phase ISCOD schemes is obvious in view of the requirements to partition the operation into two steps and to consider the subsets of vertices individually. However, even when taking the results of the partition as a given, one can sometimes attain a lower cost than the (partial-) clique cover cost.

Example 7. Consider the following case:

Client	requested item	cached items
1	a	b,x
2	b	a,x
3	c	b,d
4	d	b,c
5	x	a,c,d

The minimum-cost partial-clique cover has a cost of 3, requiring the transmission of three datagrams. However, there is a solution that requires the transmission of only two datagrams: $M_1 = a \oplus b \oplus x$ and $M_2 = b \oplus c \oplus d$ (client 5 will use $M_1 \oplus M_2$).

The fundamental reason for the apparent "miracle" is that, unlike the straightforward 2nd-phase algorithms, we exploited the fact that, even within a given partial clique, we are only required to enable each client to derive its requested item rather than the union of those requested by all the members of its partial clique.

One way of viewing the approach demonstrated by Example 7 is that the full knowledge of client states permits that beneficial use of error-correction codes with unequal protection (different clients in the same clique are able to derive different numbers of items in addition to their requested one). At a more general level, one can think of this as a recursive application of ISCOD, which entails the use of simple schemes while the problem is large in conjunction with more powerful (but computationally-complex) ones for small sub-problems.

VI. Scalability and other practical considerations

The scalability of the ISCOD family of algorithms may be hindered by the need for acquisition, storage and maintenance of *metadata* as well as the processing of clients' requests (and relevant metadata) and communication bandwidth. In this section, we examine these issues.

A. Computational complexity

As already mentioned, finding the optimal solution is usually impractical except when the number of clients is small. Even the polynomial-complexity heuristic algorithms do not scale very well. Fortunately, however, there is no need to jointly consider very large sets of clients and requests. One reason is that there is normally a limited time for accumulating requests (this is a system parameter based on customer requirements). Another reason is the diminishing return beyond a certain set size. This is most pronounced when the server uses a maximum matching algorithm. The savings in this case has a "knee" when it reaches 0.5. (see theorem 2 and Fig. 4). As an example, consider a case of 1000 requests with an edge probability equal to the threshold for n=100. If we split the 1000 requests into 10 groups of 100 and process each group separately, we are likely to achieve about a 50% reduction in communication. Applying maximum matching to all requests at once would take longer and would most likely yield the same savings. The other algorithms also exhibit a diminishing return, albeit not as pronounced as maximum matching.

The server's task can be divided into stages of collecting requests, calculating the ISCOD solution and transmitting the appropriate datagrams. These stages can moreover be pipelined for higher throughput.

Another way to improve scalability is to use an *hierarchical* solution. Intermediate agents, either regional or part of a central cluster, can collect requests from a certain group of clients (groups can be defined by geographic location, similarity in access patterns or ad-hoc). Each agent will process a group's requests and forward a partial solution to the central server. The central server may do some additional post processing, e.g., perform duplicate elimination, and will then transmit.

B. The reverse channel and metadata

Metadata. Acquiring and maintaining metadata for a large number of clients and items is a problematic chore, in terms of both storage and reverse-channel communication. The number of items is likely to be large regardless of the number of clients, thus if we keep all metadata in the center, we will have to store and maintain a large database. Many clients may be occasional, with long periods of inactivity. Keeping metadata for these will cause a lot of overhead with very little contribution to performance. Identifying such clients, or clients that alternate between periods of activity and inactivity, is not a trivial task.

The reverse channel. While the broadcast forward channel scales well with the number of clients, and traffic is proportional to the number of the actively requesting clients, the reverse channel is more problematic due to its different characteristics. It is a many-to-one channel, thus getting an ACK from each client per item that it adds to its cache, as well as cache-flush updates, may congest the center.

The reverse channel will most probably be much slower than the forward channel for several reasons: it may use a slower physical channel, e.g., dial-up lines; it may also be a contention channel, which reduces the effective bandwidth. Finally, the modulation used on the reverse channel is likely to be less aggressive, e.g., QPSK, in order to expedite locking onto the short messages that are transmitted asynchronously by different clients.

We next present several algorithms for information-gathering by the server, beginning with a trivial algorithm that will serve as a baseline.

Algorithm 1 (Trivial reverse channel approach.) *The server keeps the full state of every client. The client must constantly update the server.*

The naive algorithm requires $O(n \cdot m)$ storage in the server, and the transmission of a commensurate number of bits to the server by the clients. These demands can be reduced, e.g, by using compression or combining of ACKs by intermediate agents. We can also try and keep metadata only for currently active clients, but identifying those when the activity rate may change rapidly, and getting metadata regarding a client that becomes active creates other problems. Using a hierarchical implementation, as mentioned above, will also limit the reverse channel load per intermediate agent. However, even with such improvements, the reverse channel is still a serious limiting factor to scalability.

Fortunately, when the server handles a batch of requests, it actually needs only the metadata related to the items and clients in that batch. Noting this and the above problems in the trivial approach, we present the following alternative algorithm:

Algorithm 2 (An alternative approach) *The request-response cycle is broken into four phases:*

1. *The server collects several client-requests.*
2. *The server broadcasts the identities of the requests that it is about to fulfill.*
3. *A client whose request is about to be fulfilled, responds with a bitmap identifying which of these items it has in its cache.*
4. *The server collects the data and uses an ISCOD algorithm to decide what to transmit.*

The drawback of Algorithm 2 is a longer zero-load response time. On the other hand, it offers substantial advantages over Algorithm 1. First of all, the central server becomes *stateless* — it does not need to maintain and store the previously required amount of metadata. If we limit the number of requests handled as a group, which will probably be done for reasons explained earlier, the algorithm will scale easily with the number of clients and items. The stateless server can also easily handle occasional clients in the *mobile computing* model, and clients that alternate between periods of high and low activity.

Metadata will be transferred only on-demand, so items requested once and eventually flushed will cause no unnecessary metadata transfer. However, some metadata may be transmitted repeatedly, e.g., if there are some very active clients and some popular items that are requested frequently (by various clients). The metadata related to the presence of the popular items in the active clients' caches will be transmitted repeatedly. Such scenarios can be handled by a modification of the algorithm whereby the server may cache some metadata for a limited period of time.

In summary, if scalability is a factor and some latency is tolerable, algorithm 2 seems a much better option for ISCOD systems. It should moreover be noted that, with the exception of a very lightly-loaded system, the reduction in traffic is likely to result in an overall reduction in response time to client requests, despite the longer "service time" of the algorithm.

VII. CONCLUSIONS AND OPEN PROBLEMS

This paper takes a first step into a new and interesting area. We have introduced a new coding situation which is of practical interest, and presented ISCOD as a general approach along with practical two-phase ISCOD algorithms that substantially reduce the required communication relative to prior art. Practical issues such scalability of the computational complexity, storage and communication have also been addressed.

Graph-theoretic byproducts of this research include the definition of a k-partial clique in a directed graph as well as bounds and a close approximation for the *size* of a maximum matching in a random graph.

Finally, it should be noted that ISCOD algorithms offer a substantial bandwidth savings at the most crucial time, namely when many clients request incremental information to supplement the bulk of data that was transmitted in the past in response to requests or pushed to the clients during periods of low load such as nights.

In addition to the obvious room for improvement in the various heuristic algorithms and performance bounds, the exploitation of error correction codes with unequal protection seems to be of particular interest.

Another interesting topic for research is ISCOD-aware cache replacement policies. As a motivating example, consider two caches with similar content that have to flush part of their content. All things being equal (same expected probability for future access), ISCOD-aware caches should probably flush disjoint sets of items. If each client subsequently requests one of its missing items, there is a good chance that a requested item resides in the other cache, so the two clients' requests can be granted by a single datagram.

Acknowledgments. The authors wish to thank Roy Meshulam for insights pertaining to the unconstrained ISCOD problem, and Noga Alon for pointing out the relationship between partial- and ordinary-clique covers in an undirected graph.

REFERENCES

[1] "Backweb homepage," URL:http://www.backweb.com.

[2] "Pointcast homepage," URL:http://www.pointcast.com.

[3] "Starburst communications homepage," URL:http://www.starburstcom.com.

[4] John J. Metzner, "An improved broadcast retransmission protocol," *IEEE Trans. Commun.*, vol. 32, no. 6, pp. 679–683, 1984.

[5] Katsumi Sakakibara and Masao Kasahara, "A multicast hybrid arq scheme using mds codes and gmd decoding," *IEEE Trans. Commun.*, vol. 43, no. 12, pp. 2933–2940, Dec. 1995.

[6] Akira Shiozaki, "Adaptive type-ii hybrid broadcast arq system," *IEEE Trans. Commun.*, vol. 44, no. 4, pp. 420–422, Apr. 1996.

[7] Abbas A. El-Gamal and Thomas M. Cover, "Achievable rates for multiple descriptions," *IEEE Trans. Inform. Theory*, vol. 28, no. 6, pp. 851–857, Nov. 1982.

[8] Raymond W. Yeung, "Multilevel diversity coding with distortion," *IEEE Trans. Inform. Theory*, vol. 41, no. 2, pp. 412–422, Mar. 1995.

[9] Noga Alon and Alon Orlitsky, "Source coding and graph entropies," *IEEE Trans. Inform. Theory*, vol. 42, no. 5, pp. 1329–1339, Sept. 1996.

[10] Alon Orlitsky, "Worst-case interactive communication I: Two messages are almost optimal," *IEEE Trans. Inform. Theory*, vol. 36, no. 5, pp. 1111–1126, Sept. 1990.

[11] W. F. King, "Analysis of paging algorithms.," in *IFIP Congresss*, 1971, pp. 485–490.

[12] Duane Buck and Mukesh Singhal, "An analytic study of caching in computer systems," *Journal of Parallel and Distributed Computing*, vol. 32, pp. 205–214, 1996.

[13] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier, "Birthday paradox, coupon collectors, caching algorithms and self-organizing search," *Discrete Applied Mathematics*, vol. 39, pp. 207–229, 1992.

[14] F. Baskett and A. Rafii, "The a0 inversion model of program paging behaviour," Tech. Rep. CS-76-579, Stanford University, 1976.

[15] Kurt J. Worrell, "Invalidation in large scale network object caches," M.S. thesis, University of Colorado, Boulder, 1994.

[16] James Gwertzman and Margo Seltzer, "World-wide web cache consistency," in *Proc. 1996 Usenix Technical conf. San Diego CA*, Jan. 1996.

[17] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella, "Characteristics of www client-based traces," Tech. Rep. BU-CS-95-010, CS Dept. Boston University, Apr.(modified Jul.) 1995.

[18] Vigilio Almeida and Adriana de Oliveira, "On the fractal nature of www and its application to cache modeling," Tech. Rep. BU-CS-96-004, CS Dept. Boston University, 1996.

[19] W3C, "Propogation, replication and caching," Web page, URL: http://www.w3.org/pub/www/propogation/activity.html, 1996.

[20] Yitzhak Birk and Tomer Kol, "Informed-Source Coding-On-Demand (ISCOD) for efficient dispersal of information over a broadcast channel.," Tech. Rep. CC Pub #207 (EE Pub #1107), Electrical Engineering Dept, Technion, 1997.

[21] Michael R. Garey and David S. Johnson, *Computers and Intracability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979.

[22] Ronald Graham, Martin Grötschel, and László Lovász, Eds., *Handbook of combinatorics*, vol. I, Elsevier Science B.V., 1995.

[23] L. Lovász and M. D. Plummer, *Matching Theory*, Number 29 in Annals of Discrete Mathematics. Elsevier Science B.V., 1986.

[24] S. Even and O. Kariv, "An $o(n^{2.5})$ algorithm for maximum matching in general graphs," in *16th Ann. Symp. on Foundation of Computer Science, (Berkeley,1975)*. 1975, pp. 100–112, IEEE Computer Society Press.

[25] Norbert Blum, "A new approach to maximum matching in general graphs," in *International Colloq. Automata Language and Programming*, 1990, number 443 in Selected notes in Computer Science, pp. 586–597.

[26] Norbert Blum, "A new approach to maximum matching in general graphs," Tech. Rep. 8546-CS, Institut für Informatik der Universität Bonn, June 1994.

[27] Béla Bollobás, *Random Graphs*, Academic Press, 1985.

[28] Svante Janson, Donald E. Knuth, Tomasz Luczak, , and Boris Pittel, "The birth of the giant component," .