Coding On Demand by an Informed Source (ISCOD) for Efficient Broadcast of Different Supplemental Data to Caching Clients

Yitzhak Birk, *Senior Member, IEEE,* and Tomer Kol, *Member, IEEE*

***Abstract*—The Informed-Source Coding On Demand (ISCOD) approach for efficiently supplying nonidentical data from a central server to multiple caching clients over a broadcast channel is presented. The key idea underlying ISCOD is the joint exploitation of the data blocks already cached by each client, the server's full knowledge of client-cache contents and client requests, and the fact that each client only needs to be able to derive the blocks requested by it rather than all the blocks ever transmitted or even the union of the blocks requested by the different clients. We present two-phase ISCOD algorithms: the server first creates *ad-hoc* error-correction sets based on its knowledge of client states; next, it uses erasure-correction codes to construct the data for transmission. Each client uses its cached data and the received supplemental data to derive its requested blocks. The result is up to a several-fold reduction in the amount of transmitted supplemental data. Also, we define k-partial cliques in a directed graph and cast ISCOD in terms of partial-clique covers.**

***Index Terms*—Caching clients, clique cover, communication complexity, error-correcting codes, information dissemination, Informed-Source Coding On Demand (ISCOD), k-partial clique, maximum matching, multicast.**

I. Introduction

Consider a server sending data blocks to many *caching* clients via a high-speed *forward* broadcast channel. A slow *reverse* channel is used for control and metadata. There is no communication among clients. Transmissions are 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 hides latency requirements; it also helps (temporally) balance the communication load, and can even reduce the total amount of communication. Commercial examples include [1], [2].

At any given time, a client's cache contains some subset of the transmitted blocks. A block may be missing from a client's cache due to intermittent connectivity, insufficient storage capacity, the lack of permission to record it, or lack of interest in it. As a need arises, a client may request the retransmission of one or more blocks. Applications of such systems include information dissemination to static as well as mobile clients.

Fig. 1 depicts such a system with a satellite-based forward channel and a slow terrestrial reverse channel (e.g., telephone line). Alternatively, cable-TV infrastructure can be used for both directions.

A broadcast channel is very efficient for sending the same data to all clients, but inefficient for granting individual requests. This work therefore focuses on increasing the efficiency of the forward broadcast channel when used for "filling holes" in the client caches, i.e., for replenishment of previously-transmitted blocks. This 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. It is moreover critically important to realize that here, unlike in the common setting of broadcasting (or multi-casting) information to multiple clients, it suffices to enable every client to derive those blocks that it is actually requesting rather than to provide all the blocks to every client.

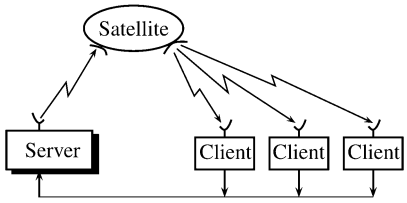

Fig. 1. Data disseminated by a server to caching clients via a broadcast channel. A separate, usually much slower, return channel, is also included.

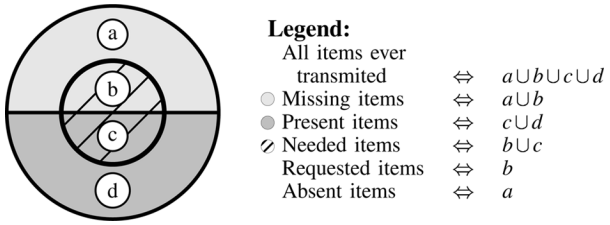

Fig. 2. Classification of transmitted blocks' status from a given client's perspective.

An important contribution of this work is the idea of coding on demand by a fully informed source, ISCOD. This approach uses full server knowledge of the clients' cache 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, one may achieve a several-fold reduction in the required amount of communication relative to previously proposed schemes.

The correspondence is organized as follows. In Section II, we flesh out the problem space and introduce a taxonomy. Section III surveys and classifies existing approaches. Section IV presents ISCOD and a two-phase approach, along with a graph model and the definition of a k-partial clique. Section V presents an effective heuristic clique-cover algorithm based on ISCOD insights. A comparison among our algorithms and prior art is presented in Section VI. Section VII discusses scalability and metadata, and presents scalable ISCOD protocols. Section VIII offers concluding remarks.

II. Problem Space and Problem Statement

A. Information Dissemination—Problem Space

The state of a client (Fig. 2.) At any given time, some (possibly empty) subset of the blocks that were ever *transmitted* are *present* in the client's cache; the others are *missing*. Also, some of the transmitted blocks are *needed* by the client whereas others are not. Those missing blocks that are needed by the client are *requested* by it from the server; the remaining ones are simply *absent*, and the client need not be aware of them.

An information-dissemination problem can be specified along three primary dimensions.

- The (actual or assumed) *initial state* of the client caches.
- The *target state* of client caches. Possibilities are: the blocks requested by the respective clients; the union of the blocks requested by the clients; all blocks ever transmitted.

Manuscript received March 14, 2005; revised January 15, 2006. This work was supported in part by a grant from News Data Systems Ltd. The material in this correspondence was presented in part at INFOCOM, San Francisco, CA, March 1998.

Y. Birk is with the Technion–Israel Institute of Technology, Technion City, Haifa 32000, Israel (e-mail: birk@ee.technion.ac.il).

T. Kol was with the Electrical Engineering Department, Technion–Israel Institute of Technology, Technion City, Haifa 32000, Israel. He is now with the IBM Research (e-mail: tomer@tx.technion.ac.il).

Communicated by R. Srikant, Guest Editor.

Digital Object Identifier 10.1109/TIT.2006.874540

- The *server's knowledge* of client states. Possibilities (later ones include earlier ones): maximum (over clients) number of missing blocks; number of blocks missing from each cache; number of blocks requested by each client; identity of blocks requested by each client; full state of each client.

B. Problem Statement

We begin with several important observations.

Lemma 1: Without loss of generality, one need only consider the case of a single requested block per client.

Proof: A client that issues $r > 1$ requests can be "split" into r "sub-clients," each having the same cache content as the original client and issuing a single request. The unchanged initial cache content and the use of an error-free broadcast channel jointly guarantee that any solution of the "split" problem will solve the original problem and *vice versa*. □

With no direct client-to-client communication, clients that are not requesting any blocks can be ignored. With a server that has full knowledge of client states, blocks that are not requested by any client can be ignored. Finally, variable block sizes can be closely approximated by a sufficiently small fixed block size. The problem addressed in this correspondence (originally proposed in [4]) can now be stated as follows.

Given: a set of n equisized data blocks and a set of n clients, each of whose caches contains some (possibly empty) subset of the blocks and each of which is requesting a single block; a server that has full knowledge of the contents of all client caches and their requests, and an error-free broadcast channel over which the server may transmit.

Goal: minimize the number of block transmissions required in order to enable every client to derive its requested block.

C. A Measure of Communication Savings

Our baseline for comparison is the transmission of every unique requested block, namely *duplicate elimination*.

Definition 1: The savings effected by a given scheme is

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

NumUniqReq and ReqComm are the number of distinct requested blocks and of transmitted supplemental blocks, respectively.

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 increasing server knowledge of client state (see Fig. 3). For brevity, we use IS and TS to denote the initial and target client states as known to the server. Implicit assumptions or by-products (brought about by the limited knowledge of the server) appear in parentheses. *Notation:* $X[i]$ denotes that the server knows the value of X for every client.

Case 1) IS: max_missing, the maximum (over clients) number of missing blocks in the cache. The server can compute a systematic max_missing-erasure correcting code over the set of all transmitted blocks and transmit only the max_missing "redundant" blocks, which is optimal. This is similar to a communication channel with at most max_missing erasures. (TS: all clients have everything.)

Case 2) IS: $N_{\text{missing}}[i]$. Unfortunately, the server cannot exploit the more detailed information and must act as in case 1. Such use of erasure-correcting codes in this case was suggested by Metzner [5]. 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 [6], as well as the use of adaptive forward error correction using BCH codes [7]).

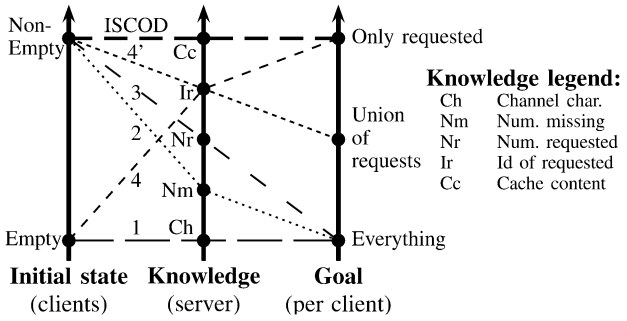

Fig. 3. Classification of the different problem variants and approaches according to the clients' initial state, the server's knowledge about the clients' states and needs (this axis is cumulative, except for 4'), and the target client states.

Case 3) IS: $N_{\text{missing}}[i]$, $N_{\text{requested}}[i]$. Again, the server cannot exploit the more detailed information and must act as in Case 1).

Case 4) IS: $N_{\text{missing}}[i]$ and the identities of the blocks requested by each client. Here, the server only needs to consider the union of the requested blocks, and can transmit each of them once (duplicate elimination). (TS: union of initial client state and the union of the requested blocks.) In this typical client-server case, the server has full knowledge of the requests but does not exploit the initial cache contents. The following example describes a situation in which duplicate elimination is efficient.

Example 1: m blocks are missing from each cache, but all clients only *request* the same single block. Here, it suffices for the server to transmit this block (once). In Cases 1)–3), m blocks must be transmitted.

With the level of knowledge of Case 4), one can sometimes do better than duplicate elimination by using ECC.

Example 2: Each client is missing two unique blocks (and has all the others), of which one is requested (no duplicates). Here, computing from all the blocks an ECC that can tolerate two erasures enables all clients to derive the blocks they requested while requiring the transmission of only two blocks; as a byproduct, each client can also derive its absent block. This example illustrates that in some cases it is better to raise the TS goal, 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* and *4'* refer to the baseline and ECC versions of Case 4), respectively.

Problems and methods such as Multiple Descriptions Problem (MDP) and the related Multiple Diversity Coding [8], [9], as well as that of coding with Non zero initial state [10], [11], are complementary to the main thrust of our work, namely *ad-hoc* creation of coding groups for efficient handling of client requests.

IV. The ISCOD Approach

A. Overview

ISCOD entails exploitation by the server of its full knowledge of client states (IDs of blocks in all categories), and only guarantees that every client be able to derive its *requested* block. (Some clients may be able to derive additional missing blocks.) This presents interesting opportunities, as illustrated by the following example.

Example 3: Clients A, B, C, D are requesting blocks a, b, c, d, respectively, and their caches contain blocks b, a, d, c, respectively. The state-aware server transmits $a \oplus b$ (bit-by-bit XOR) and $c \oplus d$, two

blocks in total, instead of sending a, b, c and d, thereby reducing the amount of communication by 50%. Note that A and B cannot derive c or d, and C and D cannot derive a or b, but that is not required!

Generalizing the example, ISCOD entails the use of erasure-correcting codes by the server on an ad hoc or "on demand" basis, hence the name. We refer to clients that can be served jointly by using an erasure-correcting code as *ECC-matched*.

B. Client Access Patterns and Prospects for Savings

Client access patterns: One interesting pattern is *flash flood*, wherein many clients request the same block [12]. This can be handled efficiently by simply transmitting such blocks, i.e., duplicate elimination. We focus on another common situation, wherein clients issue unique requests in an Internet access pattern that follows Zipf's law [13], [14]. Specifically, let us consider a 90–10 situation, wherein 90% of a client's references are to 10% of the blocks, which constitute its *hot* segment; the remaining blocks constitute its *cold* segment.

The prospects for savings: Consider initially a setting of m blocks and two clients with a 90–10 access pattern. Each client has an LRU cache, with certain miss rates for "hot" blocks and "cold" blocks. (The miss rates for the two segments are related through the size of the cache [15].) The probability that a block needed by such a client will have to be requested from the server is $P_{req} \approx o(1)$; the probability that we can save by duplicate elimination (both clients request the same block) is $P_{dup\ saving} \approx o(1/m)$ (The constants depend on the parameters and the similarity of the hot sets [4]). The probability of an *ECC-match*, i.e., that each client possesses the other's requested block, is $P_{match\ saving} \approx o(1)$

For a sample set of numbers, with (a worst case scenario of) disjoint hot sets, calculations ([4]) yield $0.57/m$ and 0.006, respectively. A pairwise *ECC-match* probability of 0.006 seems very low. For $n = 1000$ clients, however, this slim pairwise probability yields an expected savings of at least 50%, i.e., a two-fold savings in communication! For larger n, the hot sets are moreover unlikely to be disjoint.

To understand the underlying reason for the promise of ISCOD relative to duplicate elimination, consider two clients. For duplicate elimination, they must both request the same block, whereas for ISCOD each must possess the block that the other requests. In the typical ISCOD scenario, wherein a client's cache contains many blocks while it only requests very few, an *ECC-match* is much more likely than an *identity match*!

Having demonstrated the promise of ISCOD, we next present a two-phase approach to ISCOD algorithms.

C. The Two-Phase ISCOD Approach

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 tailors an erasure-correcting code. In other words, the informed source uses its knowledge to perform efficient coding on demand, or ISCOD. The server may still use other approaches, e.g., *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 ECC-matched subset to derive the union of the blocks requested by the subset's members. Throughout the remainder of the correspondence, we will refer to a slightly restricted client state space, whereby each client requests a different block. The case of duplicate requests may be treated by simply transmitting such a block once (duplicate elimination), thereby rendering the remaining requests unique. This approach is efficient, albeit not always optimal [4].

Before proceeding to consider specific problems and algorithms, we next introduce a useful graph model.

D. A Graph Model for ISCOD

Consider a graph in which each vertex corresponds to a client and its requested block, and a directed edge (u, v) exists iff u's cache contains the block being requested by v.

Clique cover: All the members of a clique (a complete subgraph) can derive their requested blocks from the bit-by-bit XOR of those, i.e., a single block transmission. The communication cost thus is bounded from above by the size of a Minimum clique cover of the graph.

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 0-partial clique is a conventional clique.

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

$$\text{CoverCost}(S) = \sum_{i=1}^{|S|} (k_i + 1). \tag{2}$$

(An isolated vertex is considered a 0-partial clique, as a single block transmission is required for it.)

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

Proposition 2: 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. $\deg(v) = \delta(C) = s - k - 1$. 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 $\bar{C}$, $\chi(\bar{C}) \leq 1 + \Delta(\bar{C})$[16], $\bar{C}$ 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 4: Consider a graph comprising n vertices arranged in a circle, with 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.

Proposition 3: For directed graphs, the minimum cost of a partial-clique cover may be lower than that of a clique cover. □

It should be noted that an ISCOD scenario corresponds to a directed graph; in specific cases, there may of course be an edge (v, u) whenever there is an edge (u, v) and *vice versa*.

V. ISCOD Algorithms

The problem of finding an optimal solution constrained to use XOR operations with each block used in at most one of them is that of finding a minimum clique cover, which is NP-complete [17]. (We conjecture that the unconstrained problem is also NP-complete.) The special case of finding a minimum cover by cliques of size two is simply the well known *maximum matching* problem, for which efficient algorithms do exist. (See [4] for useful bounds and a close approximation for the expected size of the maximum matching in a random graph.) We next explore efficient heuristic clique-cover algorithms.

One of the simplest heuristics is a greedy algorithm (picking blocks and then trying to match the rest of the blocks to those already in the forming clique), and in practice it yields reasonable results (see Fig. 5). Its computational complexity is $O(V^3)$. We next present LDG, our own heuristic clique-cover algorithm.

A. The Least Difference Greedy (LDG) Clique-Cover Algorithm

The ISCOD intuition underlying this algorithm is that a block x that is *present* in the caches of all members of a clique that is under construction (while not being requested by any of them) 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 LDG is to try and minimize the loss of degrees of freedom when considering candidates for enlarging a clique, as illustrated next.

Example 5: Consider four clients, A, B, C, D, requesting blocks 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 blocks. 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.

We now turn to a formal presentation of the LDG algorithm.

Definition 4 (LDG Terms):

1) LDG matrix (LDGM): A $(clients \times blocks)$ matrix over $\{0, 1, *\}$, corresponding to absent, requested and present blocks, respectively.
2) LDGM inter-entry distance (d): the distance between the values of two LDGM entries in the same column is defined as $d(0,0) = d(1,1) = d(*,*) = 0$, $d(0,*) = d(1,*) = 1$, and $d(0,1) = \infty$.
3) LDGM inter-row distance (dr):

$$dr(i,j) = \sum_{k \in columns} d(\text{LDGM}_{i,k}, \text{LDGM}_{j,k}). \tag{3}$$

4) LDGM 0–1 collision: Two rows i, j in the LDGM such that $dr(i,j) = \infty$, i.e, there is a column k in which one row contains a 0 and the other contains a 1.
5) *Row merging:* Given two rows i, j s.t. $dr(i,j) < \infty$, **replace** them by a row r

$$\text{LDGM}_{r,k} = \begin{cases} \text{LDGM}_{i,k} & \text{LDGM}_{i,k} = \text{LDGM}_{j,k} \\ \text{LDGM}_{i,k} & \text{LDGM}_{j,k} = * \\ \text{LDGM}_{j,k} & \text{otherwise.} \end{cases} \tag{4}$$

Example 6: Row i contains $\{0, 1, *, *\}$ if client i requests block 2 and has blocks 3, 4, with block 1 absent. If row $j = \{0, 0, 1, *\}$ then $dr(i,j) = 0 + \infty + 1 + 0 = \infty$ (there is a 0–1 collision).

TABLE I
LDG Matrix and Distance Matrix for the Graph in Fig. 4

	A B C D E
A	1 * * * 0
B	* 1 * 0 0
C	* * 1 0 0
D	* 0 0 1 *
E	0 0 0 * 1

	A	B	C	D	E
A	-	3	3	5	∞
B	-	-	2	∞	∞
C	-	-	-	∞	∞
D	-	-	-	-	3
E	-	-	-	-	-

Algorithm 1 (LDG Algorithm):
Given: M—The LDG matrix for the clients and requested blocks.
Produce: Transmitted blocks that jointly enable each client to derive the block that it requested.
1: **while** $\min\_dr = \min_{i \neq J \in \mathrm{M}}$'s rows $\{dr(i,j)\} < \infty$ **do**
2: Randomly pick two rows i, j of M s.t. $dr(i,j) = \min\_dr$
3: Merge rows i, j in M.
4: **end while**
5: **for all** row i of (the final) M **do**
6: Create a block by XORing all blocks in $\bigcup_k \{M_{i,k} = 1\}$.
7: **end for**

LDG Properties: The LDG matrix contains only blocks requested by some client. The row that corresponds to the client that requested block x will have an entry of 1 in the column for that block. In the column corresponding to block x there may entries of $*$ in other clients' rows, indicating that those clients have block x in their cache. An entry of $*$, as we shall soon see, expresses a degree of freedom.

Consider, for example, an (XOR-generated) block B from which a given client should derive its requested block. If block x is *present* in This Client's Cache, x may be used (XORed) in the generation of B because the client can easily remove x from B by xoring B with its own copy of x. If we check, for example, two clients that can be *ECC Matched* (see Section IV-A), we will find that in a column that one of them has an entry of 1, the other has an entry of $*$. The (finite) distance between two rows equals the number of $*$ marks lost when merging these rows. The intuition underlying the choice of the "nearest" rows for merging is to minimize the loss of degrees of freedom.

The computational complexity of the LDG algorithm is $O(V^3)$, so it is a practical option for a real system.

Example 7: Consider the graph depicted in Fig. 4. A simple *greedy* clique cover algorithm may very well start by merging nodes A and D, leading to a final cover by three cliques: $\{A, D\}\{B, C\}\{E\}$. Now examine the LDG run on the same graph. Table I describes the corresponding LDG matrix and distance matrix. [1] Looking at those matrices, LDG will start by merging B and C.[2] Moreover, when examining LDG's view of A, it is recognized that merging A with D is not a desirable step.

The LDG algorithm can be extended to support partial cliques, which makes it much more complex. We have done so, and experimented with several row-distance heuristics. Initial results for random graphs exhibit only a slight improvement over the regular LDG, but we do not

[1]The LDG distance is symmetric, so it suffices to keep half the distance matrix.

[2]The resulting row is $* \ 1100$, with $d(A, BC) = 4$ and $d(BC, D) = d(BC, E) = \infty$. In the following steps LDG will merge D and E and finally A with BC.

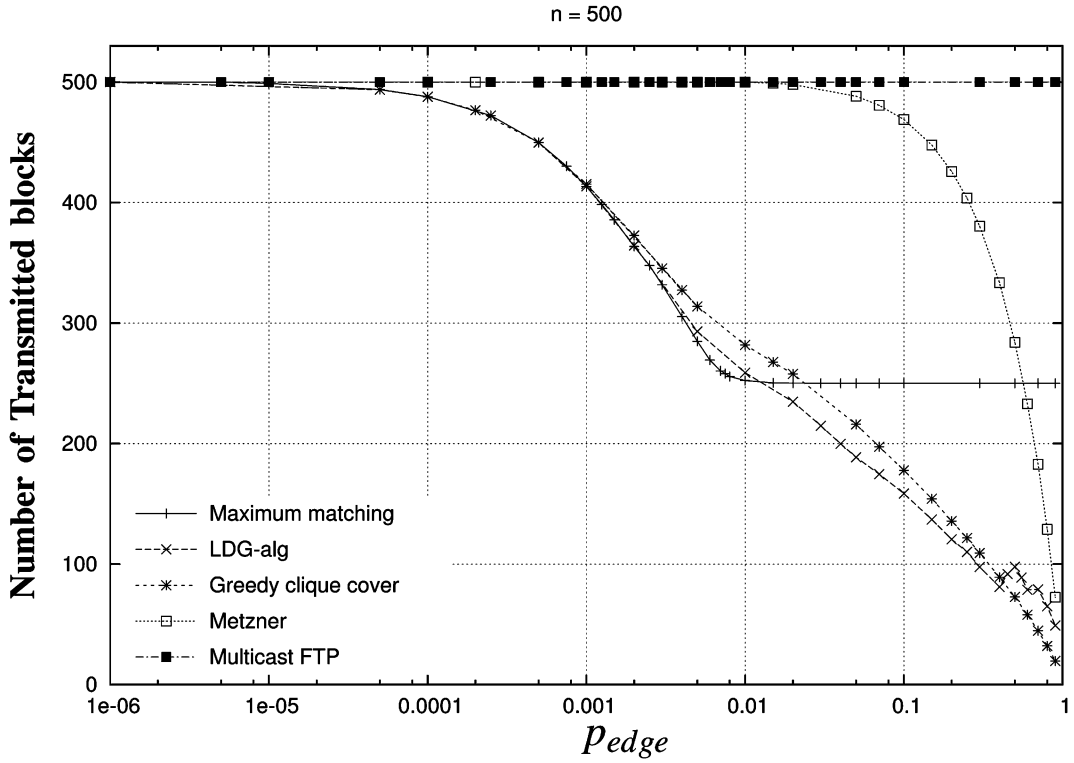

Fig. 5. Comparison of expected savings with various algorithms for a random undirected graph; $n = 500$.

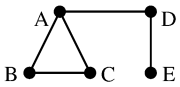

Fig. 4. Sample undirected graph.

know whether this is due to the heuristic algorithm or a property of the problem.

B. Improving Upon the Partial-Clique Cover Cost

The two-phase ISCOD approach refrains from enabling every client to derive the union of all requested blocks. However, it still does this within any given partial clique, thereby still providing more information than requested. Indeed, it can sometimes be improved upon.

Proposition 4: The partial-clique cover cost is not a lower bound on the amount of information that must be transmitted.

Proof: Consider the following case:

Client	requested block	cached blocks
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 blocks. However, there is a solution that requires the transmission of only two blocks: $M_1 = a \oplus b \oplus x$ and $M_2 = b \oplus c \oplus d$. (Client 5 will use $M_1 \oplus M_2$.) □

VI. A Comparison Among Different Schemes

In this section, we present a comparison among prior art and ISCOD variants for undirected random graphs. Fig. 5 presents simulation results for the mean number of blocks that must be transmitted with different schemes as a function of the edge probability p for $n = 500$ vertices. (The relative behavior of the schemes is similar at least up to $n = 5000$. Moreover, as n increases, the savings are already realized at lower p.)

In MFTP, following the initial attempt to transfer the file, the server resends every block 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 blocks, 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 ISCOD.

The curve for ISCOD using maximum matching exhibits a *"knee"* at the threshold value of p, and the saving approaches this technique's upper bound of 50%. 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.

With the LDG heuristic minimum clique cover algorithm, ISCOD performs as well as maximum matching for small p when the graph is sparse and cliques of cardinality greater than two (pairs) are rare. They perform better for large p, 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 so as to offset the suboptimal partitioning into cliques of size two by the clique-cover algorithms, hence the crossover. Of the two heuristic clique-cover algorithms presented, LDG is slightly 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 (see Section V). In a practice, the server can run both algorithms and pick the best solution.

The most important aspect of Fig. 5 is the substantial advantage of the algorithms in the ISCOD family over previous approaches when applied to systems that are the target of this correspondence: 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 Section VII.

VII. ISCOD-Based Protocols

A. Scalability Concerns and Mitigating Circumstances

Scalability of ISCOD may be hindered by the need for acquisition, storage and maintenance of *metadata* pertaining to client states, as well as by the processing of client requests and by congestion in the "thin", relatively inefficient upstream many-to-one contention channel. In practice, however, requests are generated asynchronously at a finite rate. Also, regardless of the benefit of jointly considering a large set of requests, application requirements usually limit response time and thus the number of requests that can be considered together. (There is moreover a diminishing return for considering request sets beyond a certain size.) Finally, we note that when the server handles a batch of requests, it actually needs only the metadata related to the blocks and clients in that batch.

The server's task may thus be divided into stages of collecting requests, calculating the ISCOD solution and transmitting the appropriate blocks. (These stages can moreover be pipelined for higher throughput.) We next present an efficient ISCOD protocol.

B. ISCOD Protocol

Algorithm 2: The protocol has four phases:

1) the server collects several client requests;
2) the server broadcasts the identities of the requests (names of the requested blocks) that it is about to fulfill;
3) a client whose request is about to be fulfilled, responds with a bitmap identifying which of these blocks it possesses;
4) the server collects the data and uses an ISCOD algorithm to decide what to transmit.

Algorithm 2 permits the central server to be nearly *stateless*—it only holds metadata for the current batch of requests. By limiting the number of requests handled as a group, the algorithm scales easily with the number of clients and blocks. The stateless server can also painlessly handle very large numbers of transient (i.e., mobile) clients as well as clients that alternate between periods of high and low activity.

VIII. Conclusion and Open Problems

We presented a new coding situation that is of practical interest, and 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 in the related ISCOD protocol. Graph-theoretic byproducts of this research include the definition of a k-partial clique in a directed 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 data that was transmitted in the past in response to requests or pushed to the clients during periods of low load.

Our ISCOD algorithms and even an optimal two-phase approach are sub-optimal. The problems of finding an optimal solution and lower bounds on the communication complexity (for a given instance or the expected value for a random graph) remain open. At the protocol level, the benefit of deferring a "tail" of requests that cannot be granted efficiently to the next batch, possibly subject to delay constraints, is unknown.

Acknowledgment

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. [Online]. Available: http://www.backweb.com
[2] Pointcast Homepage. [Online]. Available: http://www.pointcast.com
[3] Starburst Communications Homepage. [Online]. Available: http://www.starburstcom.com
[4] Y. Birk and T. Kol, "Informed-source coding-on-demand (ISCOD) over broadcast channels," in *Proc. IEEE INFOCOM*, San Francisco, CA, 1998, pp. 1257–1264.
[5] J. J. Metzner, "An improved broadcast retransmission protocol," *IEEE Trans. Commun.*, vol. 32, pp. 679–683, 1984.
[6] K. Sakakibara and M. Kasahara, "A multicast hybrid arq scheme using mds codes and gmd decoding," *IEEE Trans. Commun.*, vol. 43, pp. 2933–2940, Dec. 1995.
[7] A. Shiozaki, "Adaptive type-ii hybrid broadcast arq system," *IEEE Trans. Commun.*, vol. 44, pp. 420–422, Apr. 1996.
[8] A. A. El-Gamal and T. M. Cover, "Achievable rates for multiple descriptions," *IEEE Trans. Inf. Theory*, vol. IT-28, pp. 851–857, Nov. 1982.
[9] R. W. Yeung, "Multilevel diversity coding with distortion," *IEEE Trans. Inform. Theory*, vol. 41, pp. 412–422, Mar. 1995.
[10] N. Alon and A. Orlitsky, "Source coding and graph entropies," *IEEE Trans. Inf. Theory*, vol. 42, pp. 1329–1339, Sep. 1996.
[11] A. Orlitsky, "Worst-case interactive communication i: Two messages are almost optimal," *IEEE Trans. Inf. Theory*, vol. 36, pp. 1111–1126, Sep. 1990.
[12] (1996) Propogation, Replication and Caching. W3C. [Online]. Available: http://www.w3.org/pub/www/propogation/activity.html
[13] C. R. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of www Client-Based Traces," CS Dept. Boston University, Tech. Rep. BU-CS-95-010, 1995.
[14] V. Almeida and A. de Oliveira, "On the Fractal Nature of www and its Application to Cache Modeling," CS Dept. Boston University, Tech. Rep. BU-CS-96-004, 1996.
[15] D. Buck and M. Singhal, "An analytic study of caching in computer systems," *J. Par. and Distrib. Compt.*, vol. 32, pp. 205–214, Feb. 1996.
[16] R. Graham, M. Grötschel, and L. Lovász, Eds., *Handbook of Combinatorics*. New York: Elsevier Science B.V, 1995, vol. I.
[17] M. R. Garey and D. S. Johnson, *Computers and Intracability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.