Distributed Caching and Coding in VoD

Arezou Pourmir, Parameswaran Ramanathan
Electrical and Computer Engineering Department
University of Wisconsin, Madison
Madison, US

***Abstract*—Caching decreases content access time by keeping contents closer to the clients. In this paper we show that network coding chunks of different contents and storing them in cache, can be beneficial. Recent research considers caching network coded chunks of same content, but not different contents. This paper proposes three different methods, IP, layered-IP and Greedy algorithm, with different performance and complexity. Simulation results show that caching encoded chunks of different contents can significantly reduce the average data access time. Although we evaluate our ideas using Video on Demand (VoD) application on cable networks, they can be extended to broader contexts including content distribution in peer-to-peer networks and proxy web caches.**

Keywords— caching, network coding, Binary Integer Program

I. INTRODUCTION

In their pioneering work, Ahlswede et al. [1] first showed that an idea called *network coding* (NC) can be used to achieve optimal multicasting rate in multihop networks. In NC, intermediate relay nodes are required to strategically combine certain incoming packets using linear codes before forwarding them to all the adjacent nodes in the multicasting tree. Later Chou et al. showed that random linear coding is sufficient to reach the optimum NC performance [2]. Using this result, Gkantsidis et al. proposed a network-coding based method for distributing large files efficiently [3]. In this method, each file is divided into many small blocks so that when a client requests the file, the server/peer sends one or more random linear combination of the blocks (random linear coding) to the client. The client recovers the file upon receiving and decoding sufficient number of independent encoded blocks. Then Ma et al. suggested using sparse linear NC instead, to decrease the complexity and enhance the performance [4]. In [5] the authors show that NC can improve caching efficiency besides network performance. Reference [6] did an overview of the researches on applying NC on distributed storage systems.

There is also a lot of research on enhancing the performance of Video-on-Demand (VoD) networks. In [7], the authors propose a Mixed Integer Program (MIP) formulation to find the optimum content placement in a large scale VoD system. Their solution stores the whole video in a cache, or doesn't store it at all. In [8], fractional streaming based approach reduces the computational time required to solve the integer program. Maddah et. al propose a general solution to the caching problem and showed that their solution is within a factor of the optimum solution [9]. However, they do not consider any cooperation between the caches (each client is getting its requested data from its cache, or otherwise from the server) which distinguishes from the problem considered in this paper.

None of the existing work in VoD and content distribution networks consider storing network coded chunks of different contents. In this paper, we show the benefits of encoding chunks of different contents and storing the encoded chunks in the caches. The resulting gain is typically over and above the one obtained by network coding different chunks of the same content, which is shown in previous works, e.g. [5]-[8]. We are focusing on VoD in cable networks, although the idea is can be extended to content distribution networks, peer-to-peer networks, and proxy web caches. Our assumptions for cable networks are close to those in [10]. We consider the set-top boxes as peers, and these peers help the main VoD servers by caching and serving videos not only to their own clients, but possibly to any other client. The videos are divided to chunks of specific sizes, e.g. 2 minutes, depending upon the acceptable initial buffering delay.

Our problem is modeled as a binary integer program. We propose three methods to solve the problem, each with its own set of advantages and disadvantages. For evaluation, we compare the performance of these three methods and show the benefits of caching network-coded chunks in several different scenarios.

This paper is written in the following order: In section II we explain the motivation for caching encoded chunks of different files by giving some examples. In section III we propose three methods, IP, layered-IP and Greedy, to solve the data placement problem in caches. Simulation results are covered in section IV for different structures and different scenarios. At the end the paper is concluded and the future works are mentioned.

II. WHY CACHING ENCODED CONTENTS?

In the network shown in fig.1.a, assume that clients 1, 2 and 3 (R_1, R_2 and R_3) have the same demand statistics. In other words, if we sort the probability of the most frequently requested contents for these clients in three arrays, the arrays are the same. Assume that all three caches in this example, C_1, C_2 and C_3, have a storage capacity of 1 unit. So each client can get at most two chunks (units) of contents from the two caches it is connected to, at each moment. Without storing encoded contents, the best we can do is to provide two clients their two most requested data, by storing the most probable contents in two of the caches, and the second most probable one in the

other cache. For example if "a" and "b" are the most demanded data respectively, then storing "a" at C_1 and C_3, and storing "b" at C_2 gives R_1 and R_2 access to both "a" and "b", but R_3 would have access to only "a", and needs to get "b" from the original server, at a larger cost. But after considering the possibility of storing encoded data in the caches, by storing "a" at C_1 and "b" at C_2 and {a,b} at C_3, all three clients will have access to both "a" and "b". R_1 gets "a" and "b" from C_1 and C_2 respectively. R_2 gets "b" from C_2 and decodes "a" by getting {a,b} from C_3 and "b" from C_2. R_3 has a connection to C_1, so it can get "a" from it, and decodes "b" by geting {a,b} from C_3 and "a" from C_1. By {a,b} we mean any random linear combination of a and b, $k_1a + k_2b$, where k_1 and k_2 are some random coefficients from Galois field(t); the larger the t, the higher the probability of linear independency of different random linear combinations.

Fig.1.b shows another example. Assume that R_1 and R_2 are interested in the same set of data, e.g. "a" and "b" are the most frequently requested data for both of them, but the first two most common data for R_1 is a,b, while it is b,a for R_2. In other words, the most common data for R_1 is "a", but for R_2 it is "b" Also assume that all three caches, C_1 , C_2 and C_3, have storage capacity 1. In this scenario, the best way of storing data in the caches is to store "a" in C_2, "b" in C_3 and {a,b} in C_1. R_1 and R_2 will get "a" and "b" from C_2 and C_3, respectively. They both get {a,b} from C_1, and decode it given "a" and "b", to recover "b" and "a" respectively.

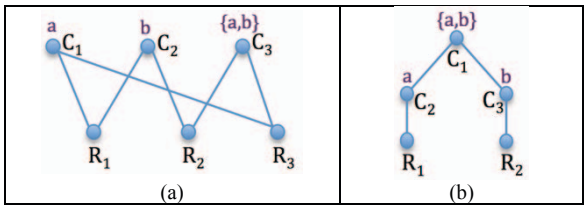

Fig. 1. Examples of structures in which storing coded data can be helpful

III. What to Store in Each Cache

We are considering caching in VoD in cable networks in this paper. A cable network is very likely to have a tree structure, which makes the analysis simpler. Our assumptions for cable networks are close to those in [10]. The set-top boxes are considered as peers, and these peers help the main VoD servers by caching and serving videos not only to their own clients, but possibly to any other client, if suitable. The videos are divided to chunks of specific sizes, e.g. 2 minutes, depending upon the acceptable initial buffering delay. These chunks might be stored in the caches originally and without encoding, or after encoding with some other chunks.

In this section different methods are suggested to solve the data placement problem in caches. In the first method, we formulate the problem in the form of an integer program and its output will tell us what to store in each cache, and how they should be stored, assuming that chunks of the original bit-streams of data or the encoded chunks of bit-streams from same or different files can be stored in each cache. This method will solve the optimum solution for the given condition. To handle the complexity of solving a large IP problem, layered-IP solution is suggested which divides the network to sub-nets, and solve the subnets in order instead of solving the whole network at the same time. Then we consider a greedy algorithm and how it works in our problem.

A. Problem Statement and Assumptions

We assume each set-top box C_j in cable network has a cache of size S_j. Each client, R_i, in our model is the user of a set-top box C_i. The cost of the links between the nodes, e.g. i and j, is C_{ij} and equal to the delay in sending a data unit from one node to another. So the average cost in our problem is equal to the average delay in receiving the data units. By data unit we mean bit-stream chunk of a specific size. The probability that client i, R_i, asks for item k is P_{ik} and we are assuming it is known; Calculating (predicting) the demand probability is out of the scope of this paper. The VoD servers in this paper are considered as a cache with all the files stored in each of them.

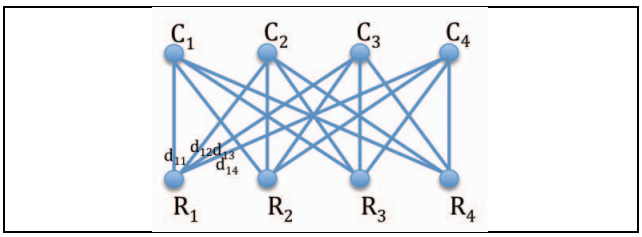

Fig. 2. By considering different costs for the links, we can put all caches in one layer. d_{ij} is the cost of sending a data unit from cache j to client i.

B. Formulating the problem as an Integer Program (IP)

To solve our problem using integer program, first we convert the original network structure to a two-layer network, with all caches (set-top boxes in the case of cable network) in one layer. Fig. 2 shows an example, with the cost of these new connections included. If the cost of a link from a client to a cache is larger than the cost of the link from that client to the closest server, we delete the link to that cache because it won't be used. The problem we are trying to answer in this case is what to store at each cache to minimize the average cost of reciving the requested data. The inputs of our formula are the costs of the links between the caches and the clients, and the request probability of different data for different clients.

Equation (1) shows our objective to minimize the average cost of data transfer, followed by the list of the constraints. Equation (1) is the binary integer program for the case when encoded chunks of data can be stored in caches, in addition to the original non-encoded chunks. The model has four sets of binary variables, x_{kj} , z_{kj} , $y_{kk'j}$ and u_{ikj}. x_{kj} is 1 if data k is stored in cache j, 0 otherwise. z_{kj} is 1 if data k is stored as an encoded chunk in cache j, 0 otherwise. If $y_{kk'j} = 1$ it means data k and k' are encoded together in cache j. u_{ikj} tells which cache(s) client i should receive data k from. Table I shows the complete list of the parameters and variables in our model, with a short explanation for each.

$$Minimize \sum_{i=1:I,k=1:K} p_{ik}\, C_{ij}\, u_{ikj} \qquad (1)$$

Constraints:

1. $\sum_{j=1:J} u_{ikj} \leq 2$
2. $z_{kj} \leq x_{kj}$
3. $y_{kk'j} \leq z_{kj}$; $y_{kk'j} \leq z_{k'j}$
4. $\sum_{j'=1:J+1} u_{ikj'} \leq 3 - u_{ikj} - x_{kj} + z_{kj}$
5. $z_{kj} + z_{kj'} \leq 3 - u_{ikj} - u_{ikj'}$
   $-z_{kj} - z_{kj'} \leq 1 - u_{ikj} - u_{ikj'}$
   $x_{kj} + x_{kj'} \leq 3 - u_{ikj} - u_{ikj'}$
   $-x_{kj} - x_{kj'} \leq 1 - u_{ikj} - u_{ikj'}$
6. $\sum_{k=1:K} x_{kj} - 0.5\, z_{kj} \leq S_j$
7. $C_{ij} \leq C_{ij'} + C_{max}(3 - u_{ikj} - x_{kj} - x_{kj'} + x_{kj} + x_{kj'})$
   $x_{kj} + x_{kj'})$
8. $C_{ij} \leq C_{ij'} + C_{ij"} + C_{max}(4 - u_{ikj} + z_{kj} + z_{k'j'} - x_{kj} - x_{k'j'} - y_{kk'j"})$
9. $C_{ij'} + C_{ij"} \leq C_{ij} + 2C_{max}(3 - u_{ikj'} - u_{ikj"} - x_{kj} + z_{kj})$
10. $C_{ij} + C_{ij'} \leq C_{ij1} + C_{ij1} + 2C_{max}(4 - u_{ikj} - u_{ikj'} - y_{kk'j2} - x_{k'j1} + z_{k'j1})$
11. $\sum_{j=1:J+1} u_{ikj} \geq 1$
12. $\sum_{i=1:I} u_{ikj} \geq x_{kj}$
13. $x_{kj} \geq 1 + u_{ikj} - \sum_{j'=1:J+1} u_{ikj'}$
14. $\sum_{k'=1:K} y_{kk'j} \geq z_{kj}$
15. $x_{k'j'} \geq y_{kk'j} + u_{ikj} + u_{ikj'} - 2$
16. $z_{k'j'} \leq 3 - y_{kk'j} - u_{ikj} - u_{ikj'}$
17. $\sum_{j'=1:J} u_{ikj'} \geq 2 - 2(2 - y_{kk'j} - u_{ikj})$
    $\sum_{j'=1:J} u_{ik'j'} \geq 2 - 2(2 - y_{kk'j} - u_{ik'j})$
18. $x_{kj} \in \{0,1\}$; $x_{k(J+1)} = 1$
    $z_{kj} \in \{0,1\}$; $z_{k(J+1)} = 0$
    $u_{ikj} \in \{0,1\}$
    $y_{kk'j} \in \{0,1\}$; $y_{kkj} = 0$; $y_{kk'(J+1)} = 0$; $y_{kk'j} = y_{k'kj}$

TABLE I. PARAMETERS AND VARIABLES IN THE IP FORMULATION

I:	Total number of the users
K:	Total number of the data packets
J:	Total number of caches
S_j:	Size of cache j
p_{ik}:	Probability of RX_i asks for item k
C_{ij}:	Cost for RX_i to get one data from cache j
x_{kj}:	Storage variable, 1 if data k is stored in cache j; 0 otherwise
z_{kj}:	Coding indicator variable, 1 if data k is stored coded with another data in cache j; 0 otherwise
u_{ikj}:	Delivery variable, 1 if RX_i gets data k from cache j; 0 otherwise.
$y_{kk'j}$:	Coding variable, 1 if data k is coded with k' and stored in cache j; 0 otherwise. $y_{kkj} = 0$

Note 1: The unit of cache size and data size are the same.
Note 2: VoD server is considered as $(J+1)^{th}$ cache to simplify the formula, and all data are stored in the source.

Now we briefly explain the constraints of our optimization formula. Constraint 1 limits encoding to at most 2 chunks of data. Constraint 2 ensures that data k is stored in cache j, if it is encoded there. Constraint 3 refers to the definition of $y_{kk'j}$ and checks that if data k and k' are encoded together in cache j, then the coding indicators for data k and k' in cache j are 1. Constraint 4 makes sure if data k is stored non-coded in cache j and user i gets k from cache j, it doesn't need any other cache to decode that data. The set of constraints in 5 ensure that if user i uses caches j and j' to decode data k, then data k is only in one cache, and it is encoded in that cache. Constraint 6 captures the capacity limit of each cache. Constraints 7, 8, 9 and 10 are cost optimality check. 7 and 8 confirm that if user i gets data k from cache j, non-coded, then the cost of getting data from any other cache non-coded, or any pair of caches encoded, is larger. 9 and 10 confirm that if user i gets data k from caches j' and j" encoded, then the cost of getting data from any other cache non-coded, or any other pair of caches encoded is larger. Constraint 11 ensures that for each user i and data k, content delivery is assigned to at least one cache. Constraint 12 checks that each stored data, encoded or non-coded, is used by at least one client. Constraint 13 assures if user i gets data k from cache j non-coded, then data k is in cache j. Constraint 14 confirms that if data k is encoded in cache j, then there is at least one data k' which is coded with k. Constraints 15 and 16 verify that if user i decodes data k from caches j and j', and k and k' are combined in cache j, then k' should be stored in cache j', non-coded. Finally constraint 17 ensures if k and k' are encoded in cache j, and user i asks for data k from cache j, it needs another cache to decode it as well.

Equation (2) is our binary integer program for the case when only original non-encoded chunks of data can be stored in the caches. We use the result of this optimum non-encoded case for comparison purpose in the simulations. The input parameters in this case are the same as those in equation (1), but here we have only two variables x_{kj} and u_{ikj}, with the same definition as before.

$$Minimize \sum_{i=1:I,k=1:K} p_{ik}\, C_{ij}\, u_{ikj} \qquad (2)$$

Const. 1. $\sum_{k=1:K} x_{kj} = S_j$
2. $\sum_{j=1:J+1} u_{ikj} = 1$
3. $C_{ij} \leq C_{ij1} + C_{max}(3 - u_{ikj} - x_{kj} - x_{kj1})$
4. $\sum_{i=1:I} u_{ikj} \geq x_{kj}$ for all $j=1:J$
5. $u_{ikj} \leq x_{kj}$ for all i, k, j
$x_{kj} \in \{0,1\}$; $x_{k(J+1)} = 1$; $u_{ikj} \in \{0,1\}$

Equations (1) and (2) are binary integer programs and Gurobi software [12] is used to solve them.

C. Layered-IP method

IP method gives us the optimum solution to our problem, but it is not possible to apply it on large networks because of its complexity. To solve this issue, we propose our layered-IP method. This new solution tries to divide the network to smaller networks and apply IP solution on these small networks separately. The order in which these subnets should be solved matters, because the answers to the lower layer ones

are the input for the higher layer ones. The exact details of the method are explained in the following steps:

1. First divide the network to some subnets with the following conditions:
   - Each subnet should be as large as possible to obtain the most gain from encoding, but small enough for IP algorithm.
   - Each subnet should be a connected network.
2. Solve the IP problem for the lowest layer subnet. Lowest layer subnets are those farthest from the server.
3. Calculate the remained probability array of the solved subnet, considering the stored data in its caches. To calculate the remained probability array, delete all the already stored data in the caches of the subnet from the probability arrays of that subnet. (A later example will make this step more clear.)
4. Consider each lower layer subnet as a single client for the higher layer cache(s) connected to it. Add the remained probability array of the lower layer subnet, to the probability array of the clients of the cache(s) connected to that lower subnet. Then normalize the result so the sum equals 1.
5. Repeat steps 2, 3 and 4 for the next lowest layer subnets, in order.

To make the procedure more clear, consider fig. 3. It shows a sample cable network in a way to show the layers easier to see, without loss of generality. In fig.3 caches are called C and clients are called R, with different indices. The Cache at the top layer of the network, C_{41} in this case, is the one closest to the VoD server. Caches in the lowest layers are those farthest from the server. The steps are explained for fig. 3 network:

1. First divide the network to subnets, considering the conditions mentioned before. An example of subnet selection is shown in fig. 4.
2. Solve the IP problem for subnets G_1, G_2 and G_3, the lowest layer subnets. None of these subnets need any information from any other subnet, because there is no lower subnet connected to them.
3. Consider subnet G_1. Assume that the probability arrays of the most requested contents for R_{11}, R_{12}, R_{13} and R_{21} are P_1, P_2, P_3 and P_4 respectively. To calculate the remained probability array of G_1, delete the data in these probability arrays which are already stored in a cache in G_1, and sum up the probabilities in all arrays of same contents. This is the remained probability array of G_1. Calculate the remained probability array of G_2 and G_3 in the same way. Notice that the sum of the elements in the remained probability array is not necessarily 1.
4. Add the remained probability of each client G_i, to the probability of the direct client of the cache G_i is connected to; then normalize the result so the sum equals 1.
5. Use the new calculated probability arrays and apply IP method on subnet G_4.

Although deciding how to divide the network might not be straightforward, but the selected subnets will be valid while the network structure remains the same. In addition, small changes to the network structure would affect only a few subnets close to the change point, and not the others. So the subnet decision complexity can be considered as a one-time complexity.

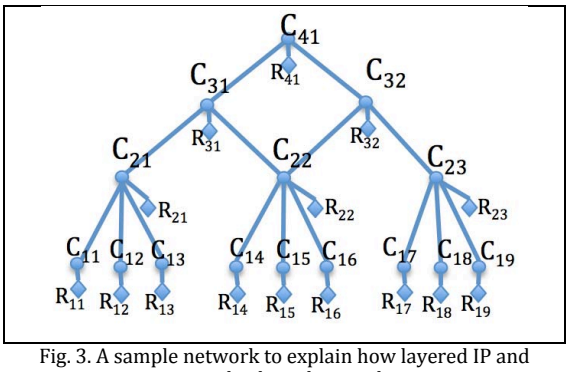

Fig. 3. A sample network to explain how layered IP and greedy algorithm work

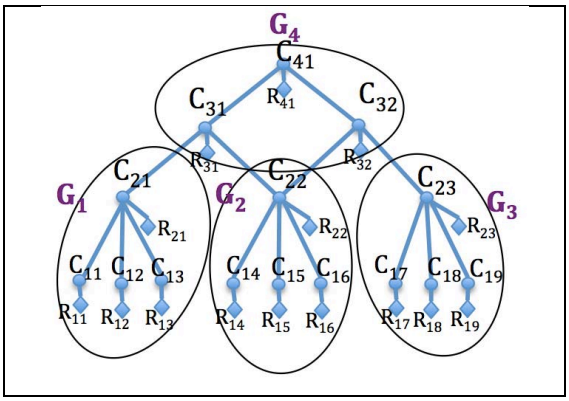

Fig. 4. Dividing the network in fig. 3 to subnets in layered-IP method

D. Greedy algorithm

In this section we propose a solution to our problem using a greedy method. Greedy methods find a locally optimum solution.

Our Greedy algorithm starts from the lowest layer caches (farthest from the server) and decide what to store in each of those caches to minimize the average access time, given the demand statistics of the clients connected to each. Then it goes one layer up (layer 2), and decides what to store in the upper layer caches to minimize the average access time, given the demand probability of each cache's client, and the remained probability arrays of lower clients, calculated by deleting the stored data in the lower layer caches connected to it. The cache's client probability array and the remained array should be summed and normalized to 1.

For all higher layers, the similar procedure explained for layer 2 caches will be applied, layer by layer, starting from lowest layers to highest layers. When a cache is connected to multiple caches from the layers above, its probability array will be considered in calculating the probability array of each of those caches. For example, nodes C_{31} and C_{32} in fig.3 which are connected to C_{22}, will both consider C_{22} remained probability when calculating their new probability array.

The Greedy method is much less complex compared to our IP method explained earlier, but may result in a bad performance. Fig. 5 shows an example of this case. Assume the following conditions:

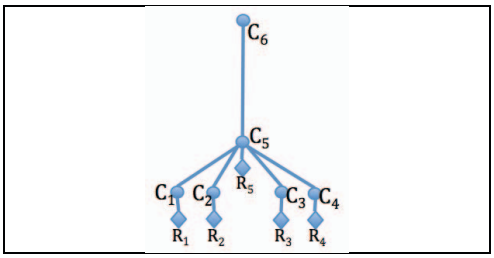

Fig. 5. An example case in which greedy algorithm results in a poor solution

1. Each cache has a 1 unit storage capacity.
2. All clients shown as R_i have the same probability array of [0.200, 0.199, 0.199, 0.199, 0.199] for their first five most requested contents which are a, b, c, d and e, respectively.
3. R_1, R_2, R_3, R_4 and R_5 can receive their file from any cache, but the cost of receiving it from C_6 is much higher than the other caches for all users.

By applying our Greedy algorithm, C_1, C_2, C_3, C_4 and C_5 all store a in their caches. But the better method is to keep a, b, c, d and e at the closer caches, C_1, C_2, C_3, C_4 and C_5 and they shouldn't all store the same data.

IV. Simulation Results

First, we want to examine how much storing encoded chunks of different contents can be helpful in caching. We applied the proposed IP solution on some randomly generated networks with the properties mentioned later in this section, and took average on all of them to see how much gain can be acheived by storing encoded data in the caches, whenever helpful. The results of this case, which we call "IP-Coding" and is the same as optimization model in equation (1), is compared to the optimum case when coding is not allowed, called "IP-noCoding" in this paper and same as optimization model in equation (2).

In table II, different network sizes are tested. If we show the size of a network as a pair of (# of clients, # of caches), the four examined networks have size (3,4), (4,5), (5,5) and (5,6). One sample network of size (4,5), already converted to a 2-layer network, is shown in fig. 6. All caches are assumed to have 1 unit capacity. The connections in each case are made randomly to generate different chances to show the real usefulness of network coding. The connection probability between each receiver and each cache is considered as 60% in these randomly generated networks. The minimum and maximum of the connection costs in the simulation are 0.05 and 1.00 respectively and the cost of requesting a content chunk from the source is assumed to be 1.00. The sample network in fig. 6 shows a random cost assignment as well. The demand probability arrays for all clients are assumed the same, with a distribution generated randomly on the data set. The size of the data set is equal to the sum of the size of all caches, shown by K in our IP formulations. Anything larger than that won't make any difference in the result.

TABLE II. Gain of Caching Encoded Contents in Four Different Network Sizes with Random Structure

Network size	(3,4)	(4,5)	(5,5)	(5,6)
Average Gain from Coding	1.6%	4.7%	5.5%	15.8%
Percentage of the networks with gain	8.0%	24.0%	34.0%	82.0%

As explained earlier in IP formulation, each networks is converted to a two-layer structure of clients and caches, by considering all caches in one layer, without loss of generality. This is because the larger cost of getting data from farther caches is considered in the cost of the connections.

The comparison in table II shows an average gain of 15.8% for the largest network can be achieved by storing encoded chunks in the caches, when beneficial. In general, we believe larger networks will have larger gain, because we will obtain the gain of small networks, plus having more opportunities to take advantage of encoding.

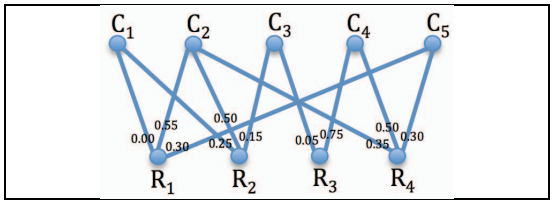

Fig. 6. One example of the networks considered in calculating the average in table I

In formulating the problem as an integer program as explained in section III.A, we assumed that the encoded chunks are the combination of 2 original chunks and not more to keep the analysis simple in this paper. It means the performance can be improved by allowing the encoding of more than two data chunks.

Another parameter that affects the performance of our algorithms is the demand probability array. The authors in [11] have shown that the popularity of WWW documents generally have a behavior similar to Zipf law, i.e., the relative access frequency for a document is inversely proportional to the rank of that document. This is a common assumption in the content distributions research and papers. Therefore we evaluate the performance of our solution with Zipf distribution assumption. In fig.7 we compare the gain from caching encoded contents in a network of size (4,5) with Zipf distribution of different exponents. The number of elements in Zipf function is 5 (total capacity of all caches.) Each point in the plot is the average of 50 different structures (different connections and link costs.) The simulation result shows the maximum gain of 7.82% for Zipf exponent of 2. By increasing or decreasing the exponent from 2, the gain decreases. For small exponents the distribution gets closer to a uniform distribution, and so the chance of taking advantage of coding is only from the structure, like the example in fig.1.a, and not the different interests of clients. For large exponents, only a few of the contents becomes important for all users and so coding different contents can't help much.

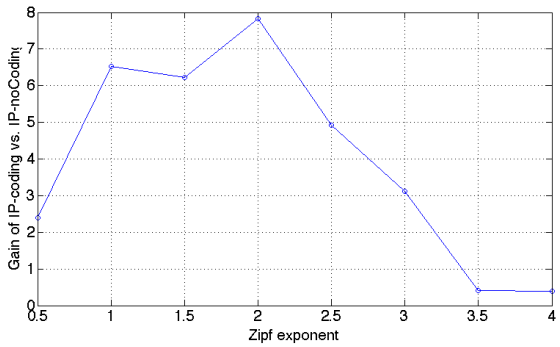

Fig. 7. Gain of IP-coding vs. IP-noCoding for different exponents of Zipf distribution function

We introduced two other solutions in this paper, the greedy and layered-IP algorithms, to handle large networks. To evaluate the performance of these two methods, we apply them to the network in fig.3, but we are assuming that only the caches in the lowest layer have a direct client attached to them, like a general case of VoD networks. For layered-IP case, the subnets are the same as those in fig.4. There are four layers of connections in the network; we assume that the cost of the lowest layer links (those connected directly to the clients), is 0.05, and the cost of those on higher layers is 0.25, 0.5, and 0.75, respectively. The cost of gettting a data chunk from the source is 1. All caches have storage capacity of 1 unit.

The probability distribution of clients demands is a Zipf function with exponent 2, but each client has a random assignment of the probabilities on the contents. In our example, the probability array of clients R_{11}, R_{12} and R_{13} are P_1= [0.7024, 0.1756, 0.0780, 0.0439], P_2 = [0.1756, 0.7024, 0.0780, 0.0439], P_3 = [0.7024, 0.1756, 0.0780, 0.0439] on data set {a, b, c, d}, which is a Zipf distribution with exponent 2. R_{14}, R_{15} and R_{16} have the same probability distribution, P_1, P_2 and P_3 respectively, but on data set {a, d, e, f}. The probability array of R_{17}, R_{18} and R_{19} are also P_1, P_2 and P_3 respectively, but on data set {g, h, i, j}.

We applied layered-IP (with encoding) and Greedy algorithm (no encoding) on this network. The average cost of content delivery in our example network with Greedy solution is 0.61 and with layered-IP solution is 0.27! Layered-IP improves the performance considerably compared to Greedy.

The IP algorithm (with encoding) has clearly the best performance among the proposed methods. But its complexity can be too much for large networks. Layered-IP seems to be a good performance-complexity tradeoff.

V. SUMMARY

In this paper, we introduced algorithms that effectively cache network-coded contents in order to reduce the content access time. The effectiveness of our approach was evaluated using simulations. We showed that performance improvements can be achieved by network coding chunks from different contents as compared only combining chunks from the same content. To deal with large networks, our approach involves a heuristic that considers the caches in a layered fashion. We are currently investigating other ways for dealing with large networks. The goal of these new approaches is to have a better complexity-performance tradeoff.

REFERENCES

[1] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," IEEE Transactions on Information Theory, vol. 46, no. 4, pp. 1204-1216, July 2000

[2] P. A. Chou, Y. Wu and K. Jain, "Practical Network Coding", Allerton Conference on Communication, Control and Computing , Monticello, IL, October 2003

[3] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," INFOCOM 2005

[4] G. Ma, Y. Xu, M. Lin, and Y. Xuan, "A content distribution system based on sparse linear network coding," in Proc. 3rd Workshop on Network Coding, Theory, and Applications, NETCOD 2007

[5] W. Liu, S. Yu, Y. Gao, W. Wu, "Caching efficiency of information-centric networking," IET Networks, Volume 2, Issue 2, June 2013, p. 53–62

[6] A. G. Dimakis, K. Ramchandran, Y. Wu, C. Suh, "A survey on network codes for distributed storage," Proceedings of the IEEE, March 2011, Vol 99, No 3

[7] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. Ramakrishnan, "Optimal content placement for a large-scale vod system," in Proceedings of the 6th International COnference, p. 4, ACM, 2010

[8] K. Leey, H. Zhang, Z. Shao, M. Chen, A. Parekhy and K. Ramchandran, ''An Optimized Distributed Video-on-Demand Streaming System: Theory and Design'', Allerton 2012, Oct 2012

[9] M. Maddah-Ali, U. Niesen, "Fundamental limits of caching," ISIT 2013, 1077-1081

[10] M. Allen, B. Zhao, R. Wolski, "Deploying Video-on-Demand Services on Cable Networks," ICDCS 2007

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," IEEE INFOCOM, vol. 1, March 1999

[12] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2013, http://www.gurobi.com