Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2010, Article ID 312714, 10 pages
doi:10.1155/2010/312714

Research Article

Cooperative Coding and Caching for Streaming Data in Multihop Wireless Networks

Dan Wang,[1] Jiangchuan Liu,[2] Qian Zhang,[3] and Fajun Chen[4]

[1] *Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*
[2] *School of Computing Science, Simon Fraser University Burnaby, British Columbia, Canada V5A 1S6*
[3] *Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong*
[4] *Key Laboratory of Science and Technology for National Defence of Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha, Hunan, China*

Correspondence should be addressed to Dan Wang, csdwang@comp.polyu.edu.hk

Received 25 February 2010; Accepted 15 May 2010

Academic Editor: Liang Zhou

This paper studies the distributed caching managements for the current flourish of the streaming applications in multihop wireless networks. Many caching managements to date use randomized network coding approach, which provides an elegant solution for ubiquitous data accesses in such systems. However, the encoding, essentially a combination operation, makes the coded data difficult to be changed. In particular, to accommodate new data, the system may have to first decode all the combined data segments, remove some unimportant ones, and then reencode the data segments again. This procedure is clearly expensive for continuously evolving data storage. As such, we introduce a novel Cooperative Coding and Caching (C^3) scheme, which allows decoding-free data removal through a triangle-like codeword organization. Its decoding performance is very close to the conventional network coding with only a sublinear overhead. Our scheme offers a promising solution to the caching management for streaming data.

1. Introduction

Multihop wireless networks as wideband Internet access solutions have been widely researched nowadays, and promote some real deployments for communities [1–4]. Since then, the requirement for supporting streaming applications in such infrastructures becomes more and more imperative [5, 6]. Path caching is a common used technology in wired networks for delivering media streaming efficiently, which can reduce the client-perceived access delays as well as server/network loads [7]. In the wireless paradigm, for the broadcast nature of wireless transmission, it is more direct and intuitive to cache streaming data on the way. Many works have been done to exploit the caching benefits in multihop wireless networks [8–10].

In the caching-enhanced multihop wireless networks, single wireless node usually has limited space for caching and can only save part of streaming data. It is common for a client to fetch data segments from multiple relay nodes. In this way, the caching-enhanced multihop wireless networks as a whole can be treated as distributed storage systems. However, previous works spend little attentions to the caching management, and data are unevenly distributed in the networks. Usually, a sophisticated caching searching algorithm is needed [10]. Recently, network coding, in particular, random linear coding, has been suggested as an elegant solution for distributed storage managements [11–13]. In a network-coding-based storage system, the original data segments (say N) are combined through linear operations with independent coefficient vectors. The combined data segments are of the same size as the original segments and are equivalent in decodability. Each relay node can therefore record a subset of the combined data segments, and a client is able to decode all the original data segments as long as N combined data segments are retrieved.

This combination process however makes the caching storage inflexible to change. More explicitly, since media data are usually coded as different important segments before transmission for providing scalable streaming abilities [14], if unimportant data segments are to be removed to accommodate new data, the system needs to first decode all the data segments, remove the unimportant ones, and then re-encode the data segments again. This operation is time- and resource consuming [15]. Even worse, given that a node can only store a partial set of the data segments, it is generally impossible for each single node to carry out the decoding operation.

To effectively solve this problem, we introduce a novel Cooperative Coding and Caching (C^3) scheme. C^3 extends the conventional random network coding [16], and enables decoding-free data removal through a triangle-like codeword organization. Its decoding performance matches that of the conventional network coding with only a sublinear overhead. It also enables retrieval of only a subset of the data. As such, it offers a promising solution to the caching management of streaming data in multihop wireless networks.

In this paper, we present the theoretical foundations of C^3 and a set of general rules for applying C^3 in the caching management. We then demonstrate the actual benefits of C^3 for streaming applications in multihop wireless networks. Again, we show that, while conventional network coding is capable to achieve high throughput, it cannot easily manage streaming data.

The remainder of the paper is organized as follows. We first present the system model, and demonstrate the superiority and problems when directly applying network coding to the system in Section 2. In Section 3, we offer the theoretical foundations of cooperative coding and caching. We discuss the design issues of C^3-based cache management in Section 4. Our preliminary simulation results are shown in Section 5. In Section 6, we discuss the related work. Finally, Section 7 concludes the paper and presents some future directions.

2. Preliminaries

2.1. Model and Notation. We now give the a formal description of the system. Our caching model is quite similar with the caching model of Ditto [9]. The main difference is that we apply our novel coding schema to manage cached data.

We consider a multihop wireless network of $\mathcal{N}$ nodes, each with a buffer of size B. Assume that we only want to cache totally N data segments for one session for the limited caching space. In general, we have $N > B$, that is, no single node can store all the data of a session. A media server located in the Internet which can be accessed by the clients of the multihop wireless network through gateway (GW). Media files are split into equally sized data segments. When a media file is first requested, there is no information about the media file in the network before; then the request will be sent to the Internet media server directly. A second or later requests will benefit from previous transmissions through caching. Requesting for the same media file in a community is a quite common user behavior as suggested by recent progresses in the traffic analysis and social networks. Wireless node in the network will cache all successfully received data segments. Data segment receiving can happen either when the node is on the routing path or when it is beside the path but can overhear the transmission. The system is continuously evolving with new streaming data coming and existing data being obsolete, though the total number of useful data segments is always N.

Since user requests are probably random from all parts of the wireless network, it is reasonable to assume that data segments are randomly distributed inside the network. We do not assume any indexing service (no matter centralized or distributed) in the network. For a client to retrieve the N data segments stored in the wireless network, it will simply broadcast the request to the nearest M neighbor nodes, which are not necessarily exactly nearest for performance tradeoff. It can be easily realized by sending out M requests and one request can be processed by only one node. Every node has a proxy module just like what they have in the study by Ditto. Every proxy can serve the data to its previous hop, either from its local cache or by requesting it from its next-hop proxy [9]. Without loss of generality, we assume that each node will provide one data segment from its storage space. Clearly, to obtain all the N data segments, we must have $M \geq N$, and even so, not all the data segments are necessarily obtained in such kind of retrieving scheme. Therefore, we define a success ratio, which serves as the major evaluation criterion in our study, as follows

Definition 1 (success ratio). The *success ratio* is the probability that a data retrieval scheme successfully retrieves all the N data segments. The default settings of M and B are $M = N$ and $B = 1$, which are their lower bounds for valid schemes.

2.2. Network-Coding-Based Caching: Superiority and Problems. We now show that network coding, in particular, random linear coding, can significantly improve the success ratio over a codingless caching system. With random linear coding, all data segments are stored in a combined fashion. More specifically, assume that the *original* data segments are c_j, $j = 1, 2, \ldots, N$; a coded data segment f_i (also referred to as a *combined* data segment) is generated as $\sum_{j=0}^{N-1} \beta_j \times c_j$, where $\beta = (\beta_0, \beta_1, \ldots, \beta_{N-1})$ is a coefficient vector, each item of which is randomly generated from a finite field F_q. It is worth noting that the size of each f_i remains equal to c_j. We define the *cardinality* of f_i to be the number of original data segments it contains, and the *full cardinality* of the system is the highest possible number, that is, N.

To obtain the N original data segments, one can simply collect any N combined data segments and then decode through solving a set of linear equations. Here, a necessary condition for successful decoding is that the coefficient vectors must be linearly independent. This is generally true if the coefficient is generated from a large enough field size q [12]. As shown in [17], the probability of linear independency is over 99.6% for $q = 2^8$, and this is almost independent of N. As such, for the network-coding-based data storage and collection scheme, the success ratio with $M = N$ and $B = 1$ is close to 100%.

Besides the combined data segments, the coefficient vectors also have to be collected by the client for decoding. Such overheads, generally of a few bytes, are much lower than the data volume, and the benefit of network coding thus well dominates these costs.

Unfortunately, while in conventional network coding it is easy to combine new data segments to existing data segments and increase the cardinality, the reverse operation is very difficult. Specifically, to remove an original data segment, we have to first decode the combined data segments, remove the unnecessary original data segments, and then recombine the remaining data segments. This is time and resource consuming. Even worse, it is often impossible to perform these operations in a single node given that $B < N$, as decoding requires N combined data segments.

As such, for the streaming application in multihop wireless networks, caching storage systems are continuously evolving; if we keep unimportant or obsolete data segments in the system, then the clients have to download more and more unnecessary data segments to successfully decode the expected useful data segments. Eventually, the buffer will overflow, and the system simply crashes. This becomes a key deficiency for applying conventional network coding in continuous data management. A related problem is that network coding has no flexibility in retrieving partial data sets only, for example, a set of the most important m original data segments that comprise the most important frames of a media file, where $m < N$.

3. Cooperative Coding and Caching: Basic Idea

In this section, we show a new coding scheme that conveniently solves the problem of data removal, thus facilitating caching management for streaming data. Our coding scheme enables the combination of only part of the original data segments, and we refer to it as *Cooperative Coding and Caching* (C^3); compre for example *Network Coding* (NC) and no coding at all (*Non-NC*).

3.1. Overview of Cooperative Coding and Caching

3.1.1. Code words. In C^3, instead of having full cardinality only, the combined data segments may have different cardinalities, from 1 to N. In addition, for each original data segment c^i, where i denotes the time sequence, there is a weight w^i associated to this data. The definition of weight in our streaming application combines the time stamp of the data segment and its relative importance to the media playback performance. Figure 1 shows an example of our weight assignment schema for a video clip. In the figure, data segments are classified into I, P, B three categories, which correspond to the I-frame, P-frame, and B-frame of the video to be streamed. Generally speaking, I-frame is the most important data for video playback, then the P-frame, and then the B-frame. So, in a batch, data segments corresponding to the I-frame are assigned with the highest weights.

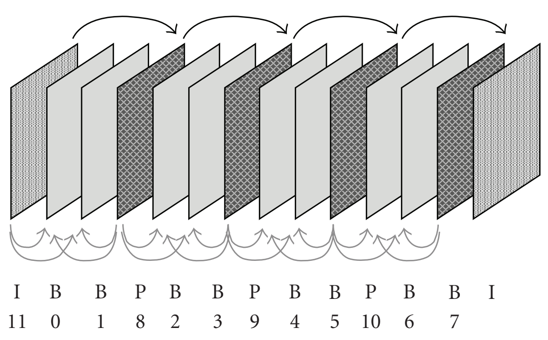

FIGURE 1: Demo of weight assignment for streaming data starting from w_x.

$$f^0 = [c_3, c_2, c_1, c_0],$$

$$f^1 = [c_3, c_2, c_1],$$

$$f^2 = [c_3, c_2],$$

$$f^3 = [c_3].$$

FIGURE 2: An example of C^3 for $N = 4$ and weight $w_3 \succ w_2 \succ w_1 \succ w_0$. The time index is not shown in this example.

It is worth noting that the weight assignment schema can be application specific, and our C^3 solution can be applied to all kinds of weight definitions which respect the following constraint. The only constraint is an operator $\succ$, which should follow the following. (1) (*Transitive*) for any w_i, w_j, w_k, if $w_i \succ w_j$ and $w_j \succ w_k$, then $w_i \succ w_k$; and (2) for every w_i, w_j, either $w_i \succ w_j$ or $w_i \prec w_j$. We have the following convention in this paper: $w_j \succ w_{j'}$ if $j > j'$ and we say that the weight of c_j is higher than $c_{j'}$ if $j > j'$, and c^j is a more recent data segment than that $c^{j'}$ if $j > j'$. For simplicity, we will use only the subscript or superscript for a data segment if the context is clear. Thus, for a data segment with weight index i and time index j, we use $c_i \doteq c^j$ to denote that they are the same data segment, that is, c_i^j. For original data segments $c_0, c_1, \ldots, c_{N-1}$, we have a coding base $\mathcal{B} = \{f^k \mid f^k = \sum_{j=k}^{N-1} \beta_j \times c_j,\ k \in [0, \ldots, N-1],\ \beta_j \in F_q\}$. We omit β_j in the following and use $f^k = [c_{N-1}, c_{N-2}, \ldots, c_k]$ for ease of exposition. The coding base for $N = 4$ is illustrated in Figure 2. The storage for each node is $\mathcal{S} = \{f_i^k \mid f_i^k \in \mathcal{B}, 0 \leq i \leq B-1\}$. Intuitively, each node stores a few combined data segments selected from the coding base $\mathcal{B}$.

Notice that, if $\hat{k}$ denotes the cardinality of a combined data segment, then the cardinality of f^k can be calculated by $\hat{k} = N - k$. The *cardinality difference* of f^{k_1} and f^{k_2} is $|k_1 - k_2|$. We may drop the superscript and use f_i provided that k is clear in the context to represent the ith combined data segment in this node.

A salient feature of this triangle-like coding scheme is the decoding-free data removal; that is, to remove the current least important original data segment c_0 in the system, we can simply drop the combined data segments containing c_0. Intuitively, c_0 is included in the combined data segments with the highest cardinality. The amount of these combined data segments consists of only a fraction of the combined data segments in the system and a removal of them will not adversely affect the success ratio of the system (recall that, with conventional network coding, all combined data segments have to be deleted in this case). Figure 3 illustrates a concrete example, where the new data segment c^4 is to replace data segment c_0. To simplify our example, we assume that $c_4 \doteq c^4$, that is, c^4 also has the highest weight. We see that, after data replacement, all $f = [c_3, c_2, c_1, c_0]$ are replaced by $f = [c_4]$, and the cardinalities of other combined data segments increase by one. This data replacement operation is decoding free and the system remains stable. A general observation can be drawn as follows.

Observation 1. For original data segments c_i and c_j, where $i > j$ (i.e., $w_i \succ w_j$), c_i will be deleted no earlier than c_j from the system. This is true irrespective to any data removing scheme.

Once we have data cached in the wireless subnet, two kinds of data request routing protocols can be applied depending on whether the data segments have been cached or not. First, when the requested segments can be retrieved inside the wireless network, then m data requests will be broadcasted to its neighbors. Every neighbor can only reply to one request, and has to decide whether to forward the request or just reply to it depending on its history record. Second, when the requested segments cannot be retrieved inside the wireless network, any unicast routing protocol in multihop wireless network can be used. We use the OSLR routing protocol in our model in this case. Though nodes on the route will cooperatively cache data segments when the traffic comes from outside of the wireless network, they will only respond to data requests on demand of the sink node. And packets will travel along the reverse route path of the request.

$$
\begin{aligned}
s_0 &: \{f_0 = [c_3, c_2, c_1, c_0], f_1 = [c_3, c_2]\},\\
s_1 &: \{f_0 = [c_3, c_2, c_1], f_1 = [c_3]\},\\
s_2 &: \{f_0 = [c_3, c_2, c_1], f_1 = [c_3, c_2]\},\\
s_3 &: \{f_0 = [c_3, c_2], f_1 = [c_3]\},\\
s_4 &: \{f_0 = [c_3, c_2, c_1, c_0], f_1 = [c_3]\},\\
s_5 &: \{f_0 = [c_3, c_2, c_1], f_1 = [c_3, c_2]\}.
\end{aligned}
$$

(a) C^3 before using c_4 to replace c_0

$$
\begin{aligned}
s_0 &: \{f_0 = [c_4], f_1 = [c_4, c_3, c_2]\},\\
s_1 &: \{f_0 = [c_4, c_3, c_2, c_1], f_1 = [c_4, c_3]\},\\
s_2 &: \{f_0 = [c_4, c_3, c_2, c_1], f_1 = [c_4, c_3, c_2]\},\\
s_3 &: \{f_0 = [c_4, c_3, c_2], f_1 = [c_4, c_3]\},\\
s_4 &: \{f_0 = [c_4], f_1 = [c_4, c_3]\},\\
s_5 &: \{f_0 = [c_4, c_3, c_2, c_1], f_1 = [c_4, c_3, c_2]\}.
\end{aligned}
$$

(b) C^3 after using c_4 to replace c_0

FIGURE 3: Data cached in 6 wireless nodes (s_0 through s_5) each with two buffer units. We omit the coefficients for each combined data segment for ease of exposition; however, it should be known that $f^0 \in s_0$ is not the same as $f^0 \in s_4$ as they are coded with different coefficient vectors. (a) shows C^3 before insertion of c_4, and (b) shows C^3 after the insertion of c_4 and removal of c_0. After this data replacement operation, the system is fairly stable in terms of the cardinalities of the combined data segments.

3.2. Distribution of Cardinality

3.2. Distribution of Cardinality. Cooperative coding and caching intrinsically manages the shape (i.e., cardinality) of the combined data segments. It is not difficult to see that, for a client retrieval, if the contacted nodes provide combined data segments with high cardinalities, then the success ratio will be higher. We summarize this in the following two observations.

Observation 2. The success ratio is maximized if every node provides the client the combined data segment with the highest cardinality from its buffer.

Observation 3. Consider time instances t_1 and t_2. If at t_1 the probability for each node to provide high-cardinality data segments is greater than at t_2, then success ratio for a data retrieval at t_1 is higher than that at t_2.

Generally speaking, in each particular time instance, it is ideal for the system to have combined data segments of cardinalities as high as possible. In an extreme case, if all the combined data segments in the system have cardinality N, the success ratio is 100% but the system is essentially reduced to the case of conventional network coding. Once a new data segment arrives, all the combined data segments will have to be deleted to make room for the new segment. For subsequent client retrievals, no data segment but the newest one can be answered.

As said, client requests could come from any part of the wireless networks, so it is reasonable to assume that cached data are uniform distributed inside the network storage. We now consider the performance of C^3 with a uniform cardinality distribution throughout the lifetime of the system. This distribution does not favor data arrivals or client retrieval at any particular time, and can serve as a baseline for further application-specific optimizations.

3.3. Performance Analysis of C^3. With the uniform distribution, we can focus on the success ratio of a single client data retrieval. Since the cardinality of each data segment is not necessarily N in C^3, it is possible that the success ratio is less than 100%. Let $B = 1$ for each node and the success ratio for obtaining i data segments by collecting i data segments using C^3 be $F(i)$. We quantify the success ratio in this scenario as follows.

Theorem 2. *Define $F(0) = 1$, then the success ratio*

$$F(N) = \left(\frac{1}{N}\right)^N \sum_{i=0}^{N-1} \binom{N}{N-i} F(i) i^i. \qquad (1)$$

Proof. The basic idea of our proof is to find the sets of combined data segments that are decodable, referred to as *valid sets*. For example, a set of combined data segments with cardinality 1 through N are decodable, and the set of combined data segments with all cardinalities being 1 is not decodable. The success ratio is given by the number of valid sets over the total number of possible sets of combined data segments; the latter is N^N.

A valid set can be constructed as follows: pick $N-i$ combined data segments of cardinality N and i combined data segments with cardinality less or equal to i. These i combined data segments should be a valid set (decode-able) in terms of i. For example, if $N=4$, a valid set may consist of two combined data segments of cardinality 4 and two combined data segments with cardinality less than or equal to 2. The number of the latter is $F(i)i^i$. Since the retrieval can be of any sequence,we need to fit all these combined data segments into N pickup sequences. For those $N-i$ combined data segments with cardinality N, there are $\binom{N}{N-i}$ locations to fit in. Notice that we do not need to shuffle the i combined data segments with smaller cardinalities as $F(i)i^i$ has already contained all possible shuffles. Therefore, the total number of valid sets is $\sum_{i=0}^{N-1}\binom{N}{N-i}F(i)i^i$ after summing up all $i \in [0, N-1]$, and the theorem follows. □

We further derive an upper bound and lower bound for C^3.

Theorem 3. *The upper and lower bounds of the success ratio are* $1-((N-i)/N)^N$ *and* $\prod_{i=0}^{N-1}((N-i)/N)$*, respectively.*

Proof. The success ratio is upper bounded by successfully obtaining the combined data segments with the highest cardinality. This probability is $1/N$ due to the uniform distribution. The probability of not getting it in N picks is $((N-i)/N)^N$, giving an upper bound $1-((N-i)/N)^N$.

The success ratio of getting a set of combined data segment with cardinality 1 through N is $(1/N)^N N!$, which is equal to $\prod_{i=0}^{N-1}((N-i)/N)$. This is clearly a lower bound. □

For comparison, we also calculate the success ratio of Non-NC, as follows.

Observation 4. If the distribution of the data segments is uniform, the success ratio for obtaining all N data segments by randomly collecting N data segments (Non-NC) is $\prod_{i=0}^{N-1}((N-i)/N)$.

A preliminary comparison of the success ratios can be found in Figure 4. Detailed comparisons as well as practical enhancements which substantially improve the success ratio will be presented in the following sections.

The probabilistic nature of success ratio only provides us a rough idea of how many combined data segments should be retrieved. If decoding cannot be carried out after retrieving a certain number of combined data segments, then the clients have to retrieve additional data segments. This may not be acceptable for delay-sensitive applications. An ideal case is thus to inform the client of exactly how many combined data segments should be retrieved so that they are guaranteed to decode out the necessary information.

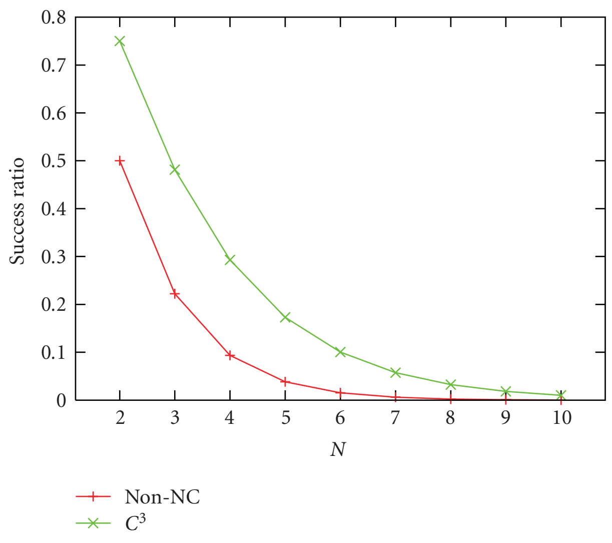

Figure 4: Success ratio as a function of N (in default value $M=N$ and $B=1$).

Before we give a concrete solution, we first illustrate the basic idea to achieve this. As shown in Observation 2, if each node sends a combined data segment with the highest cardinality in its buffer, the success ratio will be improved. In addition, if a node can always provide a combined data segment that has sufficiently high cardinality, then the success ratio will also be improved. We quantify these observations as follows. For each node, if it has a buffer size of $\sqrt{N}+1$, then it is able to store the data segments with cardinality starting from 1 to $N+\sqrt{N}$, that is, the cardinality difference of f_i and f_{i+1} is $\sqrt{N}$ for all $i \in [0, \sqrt{N}]$. Notice that we have extended the maximum cardinality of the system to $N+\sqrt{N}$. At anytime the node can provide a data segment with cardinality in $[N, N+\sqrt{N}]$. Consequently, if the client queries $N+\sqrt{N}$ nodes, and each node provides its combined data segment with cardinality no less than N, there will be $N+\sqrt{N}$ linear equations with the rank of the coefficient matrix being at least N and at most $N+\sqrt{N}$. Thus, the system can guarantee a successful decoding of all N original data segments. Formally speaking, we have the following.

Theorem 4. *The success ratio of* C^3 *with* $B=\sqrt{N}+1$ *and* $M=N+\sqrt{N}$ *is identical to the success ratio of NC with* $B=1$ *and* $M=N$*, that is, 100%.*

Proof. A rigorous proof can be found in [18]. □

This theorem shows that the success ratio of C^3 is identical with NC, with only a sublinear sacrifice of buffer size and retrieval cost. Yet it achieves continuous data management. This theorem also gives us important information; that is, to improve the success ratio of C^3, it is also important to have a balanced cardinality distribution within each single node, as each node is able to provide a combined data segment with fairly high cardinality at any time.

3.4. Maintaining Uniform Distribution of Cardinality. We have shown the performance of C^3 under a uniform cardinality distribution. It remains to show how a uniform distribution can be achieved and maintained in this dynamically evolving system.

At first, we need to initial the caching system to have uniform distributed cardinality. And this can be done easily in a centralized way. For example, we take the gateway node as the initialization control node. If there are P nodes in the network willing to cache the application data, then each of them will provide its reserved capacity to the gateway. Say that if we have $B * P$ units for caching, then the gateway will use a random generator to uniformly distribute cardinality vector to each node. And the overhead could be very low, since we only have to send P packets, each of which has the payload of $B * \log_2 N$ bits. The overhead can be further decreased, if we piggyback those cardinality initialization vectors to some real data packets. After initialization, we will apply a simple data replacement strategy to maintain this uniformity.

Assume that a new data segment $c_i \doteq c^j$ is generated to replace c_0. Our data replace procedure is as follows. (1) Remove $f^0 = [c_{N-1}, \dots, c_0]$. (2) For each node containing $f^{i+1} = [c_{N-1}, \dots, c_{i+1}]$, duplicate f^{i+1} and combine f^{i+1} with c_i to produce new $f^i = [c_{N-1}, \dots, c_i]$. (3) For each node containing $f^{i-k}, k = 1 \dots i-1$, combine c_i with f^{i-k}. (4) Send f^i to the nodes which just removed f^0. duplicate additional f^i to send if there are more nodes with f^0. And finally (5) delete unnecessary f^i if there are fewer nodes with f^0 to send.

Theorem 5. *If the cardinality is uniformly distributed, then after the above data replacement procedure, the distribution of the cardinality remains uniform, and the number of combined data segments stored in each node remains unchanged.*

Proof. The above procedure uses the new combined data segment to replace the deleted f^0. Therefore, the set of nodes which removed f^0 was refilled by f^i. The set of nodes which duplicated combined data segment f^i either sent this data segment out or discarded it. Thus, the number of combined data segments stored in each node is unchanged.

If the distribution of the cardinality is uniform, then the probability that a combined data segment has cardinality $\hat{k}$ is $1/N$ for all $\hat{k} = 1 \dots N$. After the data replacement with new data segment c_i, the probability of a combined data segment having cardinality $1 < \hat{k} < i$ remains unchanged. The probability of a combined data segment having cardinality i is the same as that having cardinality N before data replacement, which is $1/N$. The probability of a combined data segment having cardinality $i + k$, $1 \leq k \leq N - i$ is the same as that having cardinality $i + k + 1$ before data replacement, which again is $1/N$. Thus the distribution of cardinality remains uniform. □

Depending on applications, one may choose other, maybe simpler, data replacement schemes to maintain the uniformity. For example, if the new data segment is always the one with the highest weight, then new data segment can automatically replace f^0 and no transmission is necessary (see the example in Figure 3).

```
Algorithm Data Replacement ($c^j$)
  $c^j$: new data segment;
  estimate $w(c^j)$; $c_i \doteq c^j$
  if $\forall k w(c^j) < w(c^k)$, delete $c^j$
  for $l = 1 \dots B$
    if cardinality $(f_l) < N$,
      if $(f_l)$ contains $c_{i-1}$,
        $f_l = \beta_i c_i + f_l$;
      else if $f_l^{i+1} = [c_{N-1}, \dots, c_{i+1}]$,
        duplicate $f_l^i$ and save as $f_l^{i+1}$
        $f_{l+1}^i = f_l^i + \beta_i c_i$
  if $\exists f_i$ whose cardinality is greater than $N$
    delete $f_i$
```

Algorithm 1: Pseudocode for data replacement.

We have yet to show how the uniform/balanced cardinality in each node can be maintained. In a naive case, the nodes just exchange the combined data segments with others to make their cardinality balanced. This introduces high transmission overhead. Although in some applications the paramount objective is high success ratio for each client access [13], we will show some optimization schemes to substantially reduce this overhead in the next section.

4. C^3 for Caching Management

We now detail the caching management through C^3 in multihop wireless networks. The management behavior mainly contains two operations: one is the *data replacement*, and the other is the *data retrieval*.

4.1. Data Replacement. When a new data segment c^j is successfully received by a node, the node will perform the data replacement algorithm, as shown in Algorithm 1. This algorithm maintains the uniform distribution of cardinality in the entire network while replacing the less important original data segment by new data segment c^j if there is no available space.

As mentioned previously, we need to distribute some combined data segments to balance the distribution of cardinality in node level. Consider that we need a strictly balanced distribution; that is, the cardinality difference between two neighboring combined data segments f_i and f_{i+1} is strictly $\sqrt{N}$ in a node. Since combining the new data segment will make some of the data segments have a difference of $\sqrt{N} + 1$, a naive scheme will have to adjust the data segments in the entire system. This overhead is incurred each time a new data segment arrives. To avoid it, we propose the following alternative scheme.

In this new strategy, a valid cardinality difference of f_i and f_{i+1} is not strictly $\sqrt{N}$, but in between $(1/2)\sqrt{N}$ and $2\sqrt{N}$. Therefore, to balance the cardinality distribution in each node, we only need to transmit data segments in two scenarios given as follows. (1) If the cardinality difference between two combined data segments is greater than $2\sqrt{N}$, then the node just needs to obtain one combined data segment that can fit in the gap from other node. And (2) if the cardinality difference between two combined data segments is less than $(1/2)\sqrt{N}$, then the node just need to send out the data segment of cardinality in the middle. Since the cardinality difference of the data segments is doubled, if each node has a buffer size of $2\sqrt{N}$, it is easy to redefine the cardinality difference and still guarantee that the cardinality difference in each node is at most $\sqrt{N}$ using the new scheme. Formally speaking, we have the following.

Corollary 6. *Given a buffer space $2\sqrt{N}$, and the number of nodes contacted to be $N+\sqrt{N}$, C^3 with the new caching scheme in each node still achieves the same success ratio.*

The gaps of $(1/2)\sqrt{N}$ and $2\sqrt{N}$ are not unique. They can be generalized to $(1/k)\sqrt{N}$ and $k\sqrt{N}$ or other numbers depending on the buffer size on each node. In a practical heterogenous system, if $O(\sqrt{N})$ buffer size cannot be guaranteed, several nodes can collaborate as a cluster to form a combined buffer and provide data segments with reasonably high cardinality.

4.2. Partial Data Retrieval. While we have focused on retrieving N data segments, our C^3 is indeed flexible in retrieving a subset of the segments and with different quality of services. Specifically, for a client that wants to retrieve the most important m data segments ($m \leq N$), there are two possible services available. (1) *Guaranteed Service*. The client contacts $m+\sqrt{m}$ nodes and send a request message to each of them. This request message includes the type of the service (guaranteed service) and m. Each queried node will send back a data segment with the highest cardinality no greater than $m+\sqrt{m}$. (2) *Probabilistic Service*. The client sends retrieval request message to m nodes. This request message includes the type of service (probabilistic service). The nodes queried will send back combined data segment with cardinality no less than m. While downloading the combined data segments, the client simultaneously performs decoding operations. If it cannot decode out the original data segments, it will send additional requests to other nodes for additional combined data segments. This will help with optimizing the cost if there is no stringent delay requirement.

5. Simulation Results

In this section, we present our preliminary simulation results for C^3-based caching data management. We deploy 1000 static wireless nodes in the system. The default number of data segments to be cached for a source is $N = 50$ and the default buffer size B allocated for that source is 1. We examine other possible values in our simulation as well. The linear equations in network coding are solved using the Gaussian Elimination [19], and the coefficient field is $q = 2^8$, which can be efficiently implemented in a 8-bit or more advanced microprocessor [20]. To mitigate randomness, each data point in a figure is an average of 1000 independent experiments.

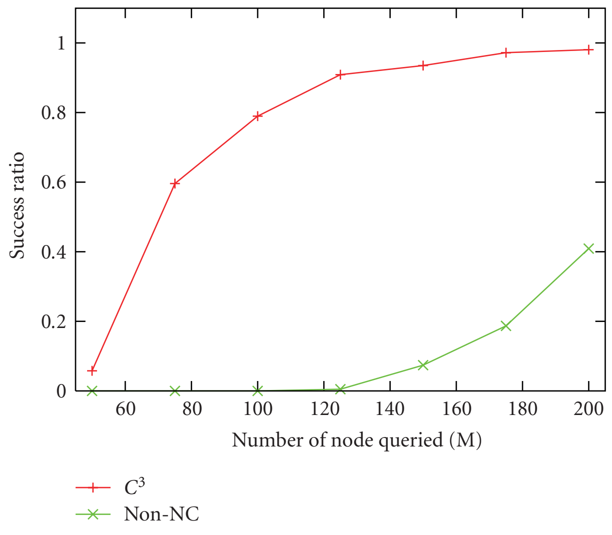

FIGURE 5: Success ratio as a function of M for C^3 and Non-NC.

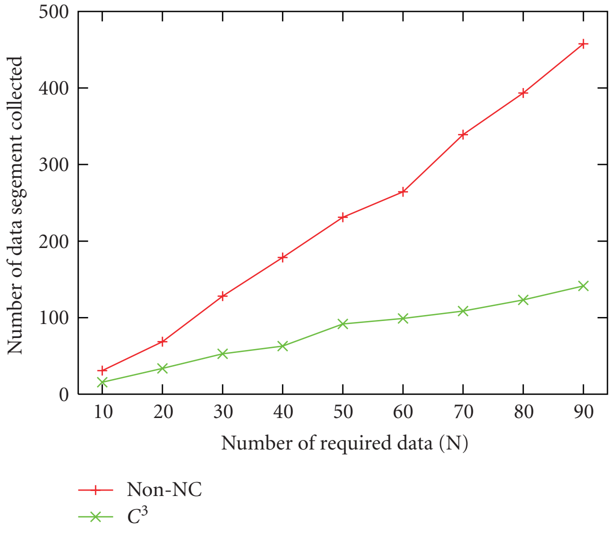

FIGURE 6: Number of combined data segment retrieved as a function of N.

Figure 5 shows the success ratio as a function of the number of data segments retrieved (N). Not surprisingly, the success ratio increases when N increases for both C^3 and Non-NC, but the improvement C^3 is more substantial. For example, if 100 data segments are to be retrieved, the success ratio is about 80% for C^3; for Non-NC, after retrieving 200 data segments, the success ratio is still 40% only.

In our next experiment, the client will first randomly retrieve 50 data segments, and if some original data segments are missing (for Non-NC) or the combined data segments cannot be decoded (for C^3), then the client will send additional requests one by one, until all the 50 data segments are obtained by successful decoding. The results are shown in Figure 6. It is clear that C^3 outperforms Non-NC for different

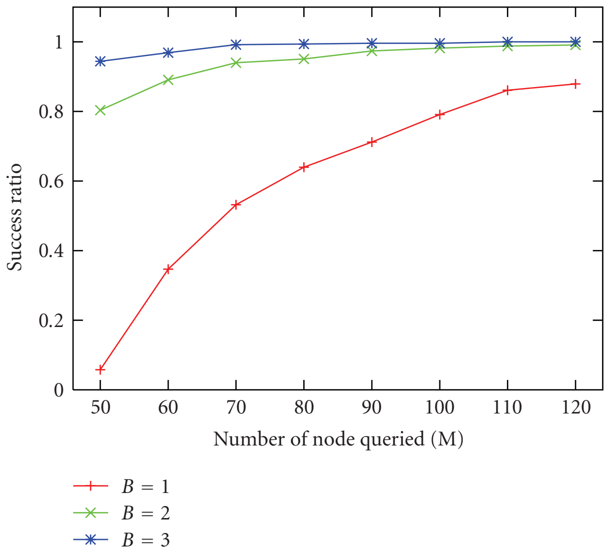

Figure 7: Success ratio as a function of M with different buffer size.

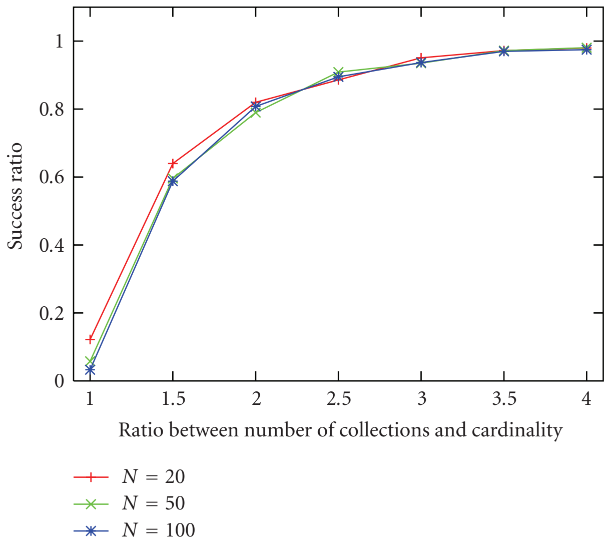

Figure 8: Success ratio as a function of $\lambda = M/N$.

N's. It can be seen that the number of data segment collected is linear with respect to the number of required data (N), but the slope for C^3 is much smaller. As a result, when N is greater than 50, the cost with Non-NC is 3 to 4 times higher than that with C^3.

We then increase the buffer size from $B = 1$ to 2 and 3 to investigate its impact. We require the nodes to provide the data segment of the highest cardinality for each client access. By carefully managing the buffer of each node, the cardinality of the combined data segments each node could provide is no less than $N/2$ and $2N/3$ for $B = 2$ and $B = 3$, respectively. The results are shown in Figure 7, where a buffer size expansion from 1 to 2 has a noticeable improvement in success ratio, and a buffer of 3 segments delivers almost optimal performance. This is not surprising because there is a higher degree of freedom for storing and uploading data in a larger buffer space. Notice that, on the contrary, the performance of Non-NC will not improve with the buffer size expansion as the nodes are unaware of which original data segments other nodes will provide to the client.

We further explore the impact of the cardinality N. In Figure 8, we depict the decoding ratio for different number of original data segments ($N = 20, 50$, and 100). The x-axis denotes the ratio between the number of data collected and the cardinality, that is, $\lambda = M/N$. We can see from Figure 8 that their differences are insignificant, and generally reduced when M increases. Recall that the performance of Non-NC decreases sharply when N increases, while NC is marginally affected by N only. These simulation results thus reaffirm that C^3 inherits the good scalability of NC.

6. Related Work

Proxy caching for media streaming over wired networks has been exploited for a long history [7]. Nowadays, as the rapid development of multihop wireless networks such as wireless mesh networks [1], proxy caching for media streaming in multihop wireless networks has also been discussed [7, 9, 10]. Different caching mechanisms have been proposed. However, previous works seldom pay attention to the caching data management. This paper takes advantages of coding to facilitate the caching management for streaming data in multihop wireless networks.

Network coding was first introduced in [21] to improve multicast throughput. To maximize the benefit of network coding, the linear codes should be constructed carefully such that the output at each destination is solvable. Randomized network coding was introduced in [16], which adopts randomly generated coefficient vectors, thus making the calculation of alphabet decentralized.

There are numerous recent studies applying conventional network coding and/or random linear coding in piratical systems. Examples include network diagnosis [13], router buffer management [22], energy improvement in wireless networks [23], data gossiping [24], as well as data dissemination in sensor networks [11] and in peer-to-peer networks [15].

For storage systems, various erasure codes have long been employed [25, 26]. Yet most of them require a central server for code block calculation; random linear coding is thus suggested for distributed storage [12]. A careful comparison between no coding, erasure codes, and random linear codes can be found in [12] as well.

While many of the studies have faced the problem of continuous data management, for example, in [20, 22, 27], their common solution is to cut the data flow in *generations*, that is, time periods, and combine all the original data segments in one generation. The length of a generation depends on the application and the choice is often experience based.

Our study on cooperative coding and caching extends the network coding design from a different aspect, namely, *decoding-free data removal*. We solve this problem by a novel triangle-like coding method, which largely inherits the power of network coding, and yet well matches the demands from continuous data management. While the cardinality-maximized combination process in network coding is the major source that improves its efficiency, our results show that the opposite direction, encoding only partial set of data segments, is worth consideration as well. Our initial study on C^3 with an application on data collection in sensor networks was presented in [17]. This paper extends [17] in the following aspects. First, we introduce the weight factor for the original data segments. This offers great flexibility in managing nonhomogeneous data, and in particular, data in a time series as shown in this paper. Second, this paper offers a more in-depth discussion on the effect of the cardinality distribution, which is an important factor for the system to evolve efficiently and effectively. Finally, we apply the C^3 in the caching management of streaming data in multihop wireless networks.

7. Conclusion and Future Work

In this paper, we introduced a novel solution for caching management of streaming data in multihop wireless networks, Cooperative Coding and Caching (C^3), which effectively solves the problem of removing obsolete information in coded data segments. The problem is a major deficiency of the conventional network coding. We provided general design guidelines for C^3 and presented a theoretical analysis of its performance. We then addressed a series of practical issues for applying C^3 in the streaming applications under multihop wireless networks.

Our C^3 offers a new look into the random linear combination process in conventional network coding. In network coding research, it is known that the higher the cardinality is, the more the benefits one may expect. Therefore, many existing schemes have focused on achieving a full cardinality in data combination. For example, the proposals in [11, 12, 15, 24] generally increase the cardinality by combining as much data as possible in intermediate nodes and then forward to others. Our work on cooperative coding and caching, however, shows that the opposite direction is worth consideration as well.

Nevertheless, there are still many unsolved issues for C^3. Theoretically, we suspect whether the overhead of $\sqrt{N}$ reaches the potential limit of C^3? Practically, we need more measurement works to examine the performance of C^3, such as estimating the expected delay and buffering time for node that is sending the request. At least, considering the recent flourish of data streaming in numerous fields, we believe that C^3 may be applied in many related applications beyond caching management for steaming in wireless networks.

Acknowledgments

D. Wang's work is supported by Grants Hong Kong PolyU/G-YG78, A-PB0R, A-PJ19, 1-ZV5W, and RGC/GRF PolyU 5305/08E.

References

[1] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Computer Networks*, vol. 47, no. 4, pp. 445–487, 2005.

[2] M. Conti and S. Giordano, "Multihop ad hoc networking: the reality," *IEEE Communications Magazine*, vol. 45, no. 4, pp. 88–95, 2007.

[3] M. Conti and S. Giordano, "Multihop ad hoc networking: the theory," *IEEE Communications Magazine*, vol. 45, no. 4, pp. 78–86, 2007.

[4] A. Aziz, A. El Fawal, J. Y. Le Boudec, and P. Thiran, "Aziala-net: deploying a scalable multi-hop wireless testbed platform for research purposes," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '09)*, pp. 33–36, New York, NY, USA, 2009.

[5] H.-P. Shiang and M. van der Schaar, "Multi-user video streaming over multi-hop wireless networks: a distributed, cross-layer approach based on priority queuing," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 4, pp. 770–785, 2007.

[6] D. Wu, S. Ci, H. Luo, H. Wang, and A. K. Katsaggelos, "Application-centric routing for video streaming over multi-hop wireless networks," in *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '09)*, pp. 1–9, June 2009.

[7] J. Liu and J. Xu, "Proxy caching for media streaming over the internet," *IEEE Communications Magazine*, vol. 42, no. 8, pp. 88–94, 2004.

[8] Y. Zhu, W. Zeng, H. Liu, Y. Guo, and S. Mathur, "Supporting video streaming services in infrastructure wireless mesh networks: architecture and protocols," in *Proceedings of the IEEE International Conference on Communications (ICC '08)*, pp. 1850–1855, May 2008.

[9] F. R. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. G. Andersen, "Ditto: a system for opportunistic caching in multi-hop wireless networks," in *Proceedings of the 14th Annual International Conference on Mobile Computing and Networking (MobiCom '08)*, pp. 279–290, ACM, New York, NY, USA, September 2008.

[10] F. Xie and K. A. Hua, "A caching-based video-on-demand service in wireless relay networks," in *Proceedings of International Conference on Wireless Communications and Signal Processing (WCSP '09)*, pp. 1–5, November 2009.

[11] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05)*, pp. 111–117, April 2005.

[12] S. Acedanski, S. Deb, M. Médard, and R. Koetter, "How good is random linear coding based distributed networked storage," in *Proceedings of the Workshop on Network Coding, Theory and Applications*, 2005.

[13] C. Wu and B. Li, "Echelon: peer-to-peer network diagnosis with network coding," in *Proceedings of the 14th IEEE International Workshop on Quality of Service (IWQoS '06)*, pp. 20–29, June 2006.

[14] M. Ouaret, F. Dufaux, and T. Ebrahimi, "Error-resilient scalable compression based on distributed video coding," *Signal Processing: Image Communication*, vol. 24, no. 6, pp. 437–451, 2009.

[15] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *Proceedings of the 24th*

IEEE Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '05), vol. 4, pp. 2235–2245, Miami, Fla, USA, March 2005.

[16] T. Ho, M. Medard, J. Shi, M. Effros, and D. Karger, "On randomized network coding," in *Proceedings of the 41st Annual Allerton Conference on Communication, Control, and Computing*, pp. 11–20, 2003.

[17] D. Wang, Q. Zhangt, and J. Liu, "Partial network coding: theory and application for continuous sensor data collection," in *Proceedings of the 14th IEEE International Workshop on Quality of Service (IWQoS '06)*, pp. 93–101, June 2006.

[18] D. Wang, J. Liu, and Q. Zhang, "Partial network coding for continuous data manage- ment in storage networks," Tech. Rep., School of Computing Science, Simon Fraser University, August 2006.

[19] J. Gentle, J. Chambers, W. Eddy, and S. Sheather, *Numerical Linear Algebra for Applications in Statistics*, Springer, New York, NY, USA, 1998.

[20] J. Widmer and J.-Y. Le Boudec, "Network coding for efficient communication in extreme networks," in *Proceedings of the ACM Workshop on Delay Tolerant Networking and Related Networks (WDTN '05)*, pp. 284–291, Philadelphia, Pa, USA, August 2005.

[21] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[22] S. Bhadra and S. Shakkottai, "Looking at large networks: coding vs. queueing," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM '06)*, pp. 1–12, April 2006.

[23] Y. Wu, P. A. Chou, and S.-Y. Kung, "Minimum-energy multicast in mobile ad hoc networks using network coding," *IEEE Transactions on Communications*, vol. 53, no. 11, pp. 1906–1918, 2005.

[24] S. Deb, M. Médard, and C. Choute, "Algebraic gossip: a network coding approach to optimal multiple rumor mongering," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2486–2507, 2006.

[25] R. Blauht, *Theory and Practice of Error Control Codes*, Addison Wesley, London, UK, 1983.

[26] D. Costello and S. Lin, *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, NJ, USA, 1982.

[27] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proceedings of the 41st Annual Allerton Conference on Communication, Control, and Computing*, pp. 40–49, 2003.