Video Streaming with Network Coding

Kien Nguyen, Thinh Nguyen, and Sen-Ching Cheung

Abstract

Recent years have witnessed an explosive growth in multimedia streaming applications over the Internet. Notably, Content Delivery Networks (CDN) and Peer-to-Peer (P2P) networks have emerged as two effective paradigms for delivering multimedia contents over the Internet. One salient feature shared between these two networks is the inherent support for path diversity streaming where a receiver receives multiple streams simultaneously on different network paths as a result of having multiple senders. In this paper, we propose a *network coding* framework for efficient video streaming in CDNs and P2P networks in which, multiple servers/peers are employed to simultaneously stream a video to a single receiver. We show that network coding techniques can (a) eliminate the need for tight synchronization between the senders, (b) be integrated easily with TCP, and (c) reduce server's storage in CDN settings. Importantly, we propose the Hierarchical Network Coding (HNC) technique to be used with scalable video bit stream to combat bandwidth fluctuation on the Internet. Simulations demonstrate that under certain scenarios, our proposed network coding techniques can result in bandwidth saving up to 60% over the traditional schemes.

I. Introduction

Multimedia streaming over the Internet is challenging due to packet loss, delay, and bandwidth fluctuation. Thus, many solutions have been proposed, ranging from source and channel coding to network protocols and architecture. For example, to combat the fluctuating and limited bandwidth, a scalable video bit stream is used to allow a sender to dynamically adapt its video bit rate to the available bandwidth at any point in time [1]. To reduce packet loss and the associated delay due to the retransmissions of the lost packets, Forward Error Correction (FEC) techniques have been proposed to increase reliability at the expense of bandwidth expansion [2]. Content Delivery Network (CDN) companies such as Akamai attempt to improve the throughput by pushing content to the servers strategically placed at the edge of the Internet. This allows a client to choose the server that results in shortest round-trip time and/or least amount of congestion.

Recently, the multi-sender streaming paradigm has been proposed as an alternative to edge streaming to provide smooth video delivery [3][4][5]. The main idea is to have each server storing an identical copy of the video. The video is partitioned into multiple disjoint parts, each part is then streamed from separate servers to a single receiver simultaneously. Having multiple senders is in essence a diversification scheme in that it combats unpredictability of congestion in the Internet. Specifically, smooth video delivery can be realized if we assume independent routes from various senders to the receiver, and argue that the chances of all routes experiencing congestion at the same time is quite small. If the route between a particular sender and the receiver experiences congestion during streaming, the receiver can re-distribute rates among the existing senders, or recruit new senders so as to provide the required throughput.

This multi-sender streaming framework is particularly well suited for CDN and P2P networks since multiple copies of a video are often present at these servers/peers either through a coordinated distribution of the video from an original CDN server, or through an uncoordinated propagation of contents in a P2P network such as KaZaa [6]. However, there are a number of drawbacks with the current multi-sender framework. First, many of the current multi-sender streaming schemes assume that identical copies of a video must be present at different servers/peers. This implies an increase in the overall storage. Second, a careful synchronization among the senders is needed to ensure that distinct partitions of a video are sent by different servers/peers in order to increase the effective throughput. In other words, an optimal partition algorithm must be able to dynamically assign chunks of different lengths to different servers based on their available bandwidths. This dynamic partition algorithm, however is often suboptimal due to the lack of accurate available bandwidth estimation. Third, for ease of controlling the sending rates as well as data partition, many multi-sender schemes assume a UDP-like transport protocol, which often cannot be used for computers behind a firewall in many networks. That said, we propose a multi-sender streaming framework using network coding technique that reduces the overall storage, the complexity of sender synchronization, and enables TCP streaming. Furthermore, we propose a Hierarchical Network Coding (HNC) technique that facilitates scalable video streaming.

The outline of the paper is as follows. In Section II, we discuss some background and motivation for video streaming via path diversity. Based on these discussions, we formulate a common abstract model for media streaming

in CDNs and P2P networks, which will be used to assess the performance of proposed network coding techniques for video streaming in Section III. Next, we introduce network coding concepts and the propose the hierarchical network coding (HNC) for scalable video streaming in Section IV. In Section V, we discuss the proposed joint network coding techniques and transmission protocol for video streaming. Next, simulation results for various coding techniques and protocols will be given in Section VI. Finally, we list some related work in Section VII and conclude in Section VIII.

II. Preliminaries

In this section, we discuss some background and motivation for video streaming via path diversity framework. Based on these discussions, we will highlight several important research issues associated with path diversity framework. The goal of these discussions is to bring about an abstract model for multi-sender streaming which is general enough, yet sufficient to characterize the performance in various settings.

A. CDN and P2P Networks

A typical Internet application sends packets that follow one and only route at any instance. An application has no control over which route its packets traverse, rather, the route is determined by the underlying Internet routing protocols. In recent years, overlay networks have been proposed as an alternative to enable an application to control its route to some extent [7]. The idea is, instead of sending the packets to the destination, an application sends its packets to an intermediate host belonged to an overlay network. This intermediate host then forwards the packets to the intended destination on the behalf of the sender. As a result, the packets will take a different route than the one determined by the underlying routing protocols. Path diversity framework takes one further step by allowing an application to send packets on multiple routes simultaneously. When packets are partitioned and/or coded properly, this path diversity framework has been shown to improve the visual quality of video streaming applications [8][5].

That said, P2P networks are overlay networks where two peers are connected together via a TCP connection. To send data from one peer to another, the data may go through a number of intermediate peers to get to the intended peer. This provides a natural framework for path diversity streaming via forcing the packets through intermediate peers. In addition, if a peer wants to view a video stream, and presumably a number of its neighbors (direct

connected peers) have either the complete or partial video, it can simultaneously request different parts of the video from different neighbors. Effectively, the video packets will traverse on different routes to the peer, thus congestion on one route will not have much effect on a peer's viewing experience when the remaining routes together, provide sufficient throughput.

Content Delivery Networks (CDNs) is also a natural framework for path diversity streaming. CDN aims to improve the application's performance by placing the servers near the customers in order to increase throughput and reduce latency. In a CDN, contents are distributed to a number of servers which are strategically placed around the edge of the Internet. When a customer requests a content, the nearest server with the desired content is chosen to serve that customer. This framework can be easily enhanced to allow multiple servers to deliver the content simultaneously to a customer, and thus obtaining the benefits of path diversity streaming, or more precisely, multi-sender streaming.

On the other hand, the advantages of multi-sender streaming framework come with many research issues to be resolved. In what follows, we will discuss network protocols to accommodate multi-sender streaming framework.

B. Network Protocols

TCP vs. UDP. Many interactive and live video streaming systems use UDP whenever possible as the basic building block for sending packets over the Internet. This is because UDP allows the sender to precisely control the sending rate, and if the network is not too much congested, a receiver would receive the data at approximately the same rate. This property is also desirable for live video streaming applications where minimal throughput often must be maintained for high quality viewing experience.

On the other hand, UDP is not a congestion aware protocol, in the sense that it does not reduce its sending rate in presence of heavy traffic load. As a result, when a large amount of UDP traffic is injected into a network, it can cause a global congestion collapse where majority of packets are dropped at the routers. For this reason, non-real time applications often use TCP that can adapt the sending rate to the network conditions automatically. This rate adaptivity prevents congestion collapse, and results in a fair and efficient throughput allocation for each application even when the network is congested. Furthermore, TCP-based applications are preferable since many networks actively filter out UDP packets which are often thought as a sign of possible flooding attacks from malicious automated software.

Based on these, it makes sense to use TCP when TCP's delay and throughput fluctuation can be minimized. As will be discussed shortly, our proposed network coding technique is designed for efficient TCP based transmission.

Push vs. Pull. In multi-sender streaming framework, the data come from multiple senders which leads to the question of how to coordinate the multiple transmissions to prevent or mitigate data duplication. One possible approach is for the receiver to request disjoint data partitions from multiple senders. The protocols based on this approach are called pull-based protocol, and they work well in many scenarios. In other scenarios, it may be better to use push-based approach where the senders simply send packets to a receiver without its request.

Majority of P2P systems use pull-based protocols [9][10][11] because of their robustness against peer joining and leaving the network. Pull-based protocols also use bandwidth efficiently, in the sense that a receiver does not receive any duplicate data from the senders. However, they have many drawbacks that might be unsuitable for some video streaming scenarios.

First, using pull-based protocols may result in lower throughput for a receiving peer due to lack of disjoint data from its neighboring peers. To illustrate this, consider a streaming a video from the source 0 to two receivers 1 and 2. Suppose these peers are connected to each other. Using the pull-based protocol, receiver 1 would request data from 0 and 2 while receiver 2 would request data from 0 and 1. Since these receivers are acting independently, both may request the same packets from the source 0. If they do, most of the time, the two receivers would have the same data, thus they cannot exchange new data with each other, resulting in lower throughput. Now, consider a simple push-based protocol in which, the source simply pushes the odd packets to receiver 1 and even packets to receiver 2. Each receiver then pushes the data it receives from one node to the other node. Effectively, the receiver 1 pushes the odd packets to receiver 2, and receiver 2 pushes even packets to receiver 1. Clearly, using this protocol, the throughput at each receiver is larger than that of using the pull-based protocol. Typically, when network topology is well-defined and relatively unchanged over time, the push-based protocols result in higher throughput than its pull-based counterparts. Also, the pull-based protocols often introduce high latency due to the requests, this may not be appropriate for media streaming applications.

Second, a careful coordination on which to be sent by which sender (pull-based protocol) is required to achieve optimal performance from the perspective of a particular receiver. As an example, assuming that two senders are

used for streaming, then sender 1 can stream the odd packets while the other streams the even packets, starting from the beginning of the file. As described, this approach is roughly optimal when the throughputs from the two senders are somewhat identical. On the other hand, when the throughput of server 1 is twice as large as that of server 2, then the pattern of packets received at the receiver will look like (0, 2, 1, 4, 6, 3, 8, 10, 5). Clearly, the gap between even and odd packets will grow with time. This is problematic for streaming applications where packets are played back in order, and the playback rate is larger than the throughput of the slow link. For example, if the playback rate is 2 packets/s, then even with the pre-buffering technique, the video player eventually has to stop to wait for odd packets since the arrival rate of odd packets is only 1 packet/s. We note that the total receiving rate at a receiver is 3 packets/s which, in principle, should be sufficient for a 2 packets/s stream. However, the suboptimal packet partition creates the scenario where the receiver receives many packets to be playbacked in the far future, but not enough packets in the near future for playback in time. A solution to this problem is to let the receiver dynamically requests the packets it needs. When there are many servers with different available bandwidths and are varied with time, complex dynamic coordination between the client and the servers is needed to achieve the optimal throughput.

Third, even when complex coordination is possible, this only works well if all the senders have the complete file or data segments of interest, so that a receiver can choose which packets from which senders based on the sender's available bandwidths which presumably can be observed by the receiver. In a P2P network, it is not always the case that the sending peers would have the complete file. In fact, previously discussed example showed that using the pull-based approach may result in lower throughput due to the duplication of data among the peers. In a CDN, it is possible to store duplicated versions of a video stream at different servers before a streaming session. However, this technique results in larger overall servers' storage.

As such, we believe that for certain scenarios, push-based protocols are better-suited for multimedia streaming since they are simple, and can provide high throughput and low delay. Although to be bandwidth effective, one must ensure that the data duplication at the receiver is minimal. As will be discussed shortly, our approach is to employ network coding to achieve this property.

C. Source Coding

Often, one must employ source coding techniques, in particular, scalable video coding, to mitigate the effect of Internet throughput variations on the viewing experience. Scalable video coding enables a sender to adapt its sending rate to the current network conditions while allowing a graceful degradation of the video quality. A typical scalable video bit stream consists of frames. Each frame consists of bits with different importance levels in terms of the visual quality. These bits are categorized into a hierarchy of layers with different importance levels. Thus, when the available bandwidth is small, sending bits from the most important layers and ignoring others would result in a smooth video playback, albeit slightly lower video quality. That said, we will discuss the hierarchical network coding technique designed for efficient multi-sender transmission of scalable video bit streams.

III. Streaming Model

To motivate the proposed abstract model for multi-sender streaming framework, let us first consider the distribution of a live or non-live video to the clients in a CDN. For a non-live setting, the origin server can distribute a video to a number of assisted servers prior to the start of a video streaming session. A client then randomly connects to a small number of these assisted servers in parallel to view the video. If each of the assisted server has the entire video, using a pull-based protocol, a client can request different parts of the video from different servers as discussed previously. However, requiring the video to be on every server implies much redundancy. Thus, an interesting question is how to distribute the video to the assisted servers such that, even when each server does not have the complete video, there is still high probability that a client can get the complete video from all the servers that it connects to. Intuitively, the key to a good distribution scheme is to ensure that the assisted servers share as little information as possible while allowing a client to obtain the complete video.

Another interesting question is how to best distribute a scalable video to the assisted servers. Intuitively, for a given redundancy, a good distribution scheme should provide a high chance for a client to obtain the base layer bits, perhaps at the expense of lower probability of its getting the enhancement layer bits.

That said, a simple distribution scheme could be as follows. At each time step, the origin server would pick a packet in playback order and randomly chooses a server to send the packet to. This process repeats until the a specified number of packets (redundancy level) has been sent. This scheme, however, tends to produce much

duplicated video parts among the servers chosen by a client for streaming, and thus reducing the chance of a client to obtain high throughput from multiple servers. On the other hand, from a client's viewpoint, when scalable video is used, having multiple servers storing duplicated base layer bits is beneficial. This is because a client now has a higher chance of obtaining the base layer bits from a random number of servers that it connects to. We note that because of the randomness in the way servers were chosen, the client may or may not have the packets it wants. *Thus the goal of the origin server is to code and distribute packets in such a way to result in high availability of packets needed by a client for smooth video playback.* Furthermore, when scalable video is used, the origin server may decide that it would distribute the important layer packets with high probability, i.e., more important layer bits end up at the assisted servers, thus increasing the probability of a client obtaining these bits.

In addition to CDN setting, let us consider a video broadcast session from a single source to multiple receivers (peers) in a P2P network. We assume a push-based protocol, in which the source pushes the packets to its neighboring peers who in turn push the data to other peers. Packets are pushed out by the source in some order. To reduce the playback delay, the source may want to send packets with earlier playback deadlines first. A peer then pushes the packets out to its peers in the order that these packets were received.

Since streaming is of concern, it is important to consider the set of packets available for a receiver at any point in time. To achieve smooth video playback, this set of packets must contain the packets that are used to playback the current video frame. From a receiver's viewpoint, this implies that its neighbors must have the packets it wants in a timely manner. Unfortunately, due many factors, e.g., variations in round trip time (due to topology), peer joins and leaves, bandwidth heterogeneity of peers, these packets arrive at the neighbors of a receiver in different order than the one they were sent by the source. Thus, within a small window of time, from a receiver's viewpoint, we assume these packets arrive at its neighbors in a somewhat random manner. The neighbors then randomly push the packets to the receiver. Clearly, the distribution of packets at these neighbors can be controlled to some extent by the source. For example, a source may push duplicated packets containing base layer bits to ensure their availability for the receiver. This scheme, however might take away the throughput used for transmissions of enhancement layer bits otherwise.

Based on these discussions, we are interested in the following abstract model. A source has a file. It allows to

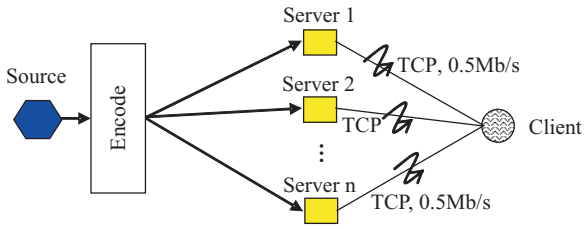


Fig. 1. An abstract model for path diversity streaming.

code and distribute the file in whatever way to a number of intermediate nodes (servers in CDNs and peers in P2P networks). A receiver then randomly connects to some of these intermediate nodes to obtain the file as shown in Figure 3. Thus, we model the process into two stages: the initial distribution of the packets to the intermediate nodes and the transmissions of packets from the intermediate nodes to a receiver. The arrival patterns of packets at the intermediate nodes are assumed to be somewhat random, and can be controlled to some extent by the source. In a CDN, these packet patterns are a direct result of how an origin server send packets to these assisted servers. On other other hand, in a P2P network, how the source send packets has an indirect effect on the distribution of packets at the intermediate nodes, i.e., neighboring peers of receiver. For scalability, we also assume that the intermediate nodes do not communicate with each other. Instead, these nodes simply push packets to a receiver in some random manner. Thus, one major concern is how to mitigate the data duplication when using push-based protocols. We note again that the push-based protocols can eliminate the packet partition problem that can reduce throughput while minimizing the coordination overhead as argued in Section II-B. That said, in this paper, we describe network coding approaches for the distribution of packets from a source to the intermediate nodes in order to minimize the storage redundancy (in CDNs) and bandwidth usage (in P2P networks). Furthermore, we describe a TCP-based streaming protocol that employs network coding technique to allow a receiver to achieve high throughput while minimizing the coordination overhead. We now introduce the necessary background on network coding techniques.

IV. NETWORK CODING

A. Random Network Coding

In their seminal network coding paper, Ahlswede *et al.*, showed that maximum capacity in a network can be achieved by appropriate mixing of data at the intermediate nodes [12][13]. The most elegant result of network coding is that the maximum network capacity is achievable using some *random network coding* [14][15] techniques, while this is not usually possible with the traditional store and forward routing.

Using random network coding (RNC), a peer encodes a new packet p_i by linearly combining n original packets as: $p_i = \sum_{j=1}^{n} f_{ij} c_j$ where f_{ij} are the random elements belonging to a finite field F_q having q elements. A node then includes the information about the f_{ij} in the header of the new packets and sends these new packets to its neighbors. If a receiver receives n encoded packets p_i's that form a set of n linearly independent equations, then it will be able to recover n original packets. The advantage of using this random network coding in CDN or P2P networks can be seen in the following simple CDN scenario.

Assuming that an origin server distributes a file to a number of assisted servers in a CDN. To increase the throughput, the origin server can first divide a file into n different chunks and randomly distributes these chunks to the assisted servers. A client then connects to these servers to get the file. Since each server randomly pushes pieces of the file simultaneously to a client, the time for a client to recover all n chunks is potentially much shorter than having the only origin server pushing the file. Note that this design scales well since no coordination among the servers is required. However, it is not optimal. Because of the random packet pushing, some of the packets received at a client may be duplicated, resulting in wasteful bandwidth. For example, an origin server may divide a file into 4 chunks c_1, c_2, c_3, and c_4, and randomly distributes to a number of assisted servers. As a result, assume that the chunks at server A are c_1, c_2, c_3, and at server B are c_2, c_3, and c_4. Now suppose that a receiver R connects to both servers A and B to stream the file from. Suppose further that A pushes out packets c_1, c_2, c_3 and B pushes out packets c_2, c_3, and c_4 in that order. After the first time slot, R obtains both c_1 and c_2. In the second time slot, R obtains c_2 and c_3, but since it already obtained c_2 from the previous time slot, it discards c_2. In the third time slot, it obtains c_3 and c_4, and discards c_3. As seen, R needs to download six chunks in three time slots to be able to receive the complete file.

Now let us consider the case where the origin server is allowed to use network coding. In particular, the origin server produces coded packets as a linear combination of the original packets, and distributes them to the servers A and B randomly. Formally, the coded packets are encoded as follows:

$$a_i = \sum_{j=1}^{3} f_{ij}^a c_j \ , \qquad b_i = \sum_{j=2}^{4} f_{ij}^b c_j,$$

where f_{ij}^a and f_{ij}^b are random elements belonging to a finite field F_q. Because of the randomness, each server is likely to have different packets, and thus R is also likely to receive different packets. For example, during the first two time slots, it is likely that R would receive different packets, two from each server. Suppose that R receives $a_1 = f_{11}^a c_1 + f_{12}^a c_2 + f_{13}^a c_3$ and $a_2 = f_{21}^a c_1 + f_{22}^a c_2 + f_{23}^a c_3$ from A, and $b_1 = f_{12}^b c_2 + f_{13}^b c_3 + f_{14}^b c_4$ and $b_2 = f_{22}^b c_2 + f_{23}^b c_3 + f_{24}^b c_4$ from B, then clearly, it will be able to recover c_1, c_2, c_3, c_4 if these four equations are linearly independent and f_{ij}^a and f_{ij}^b are known. It can be shown that if the field size is large enough, the probability of obtaining these independent equations is close to 1. For this scheme to work, the information about f_{ij}^a and f_{ij}^b must be included in the data packets. The number of bits required to specify f_{ij}^a and f_{ij}^b are $n \log(q)$ where n is the number of original packets while q is the size of the finite field. If $m >> n$ then these bits are negligible. Therefore, for most practical purposes, this network coding scheme can speed up the download time (4 packets as compared to 6 packets) without the overhead of coordination.

One important observation is that network coding incurs an additional delay before any of the original data can be recovered. Without network coding, R will be able to recover c_1 and c_2 during the first time slot. On the other hand, using network coding, c_1 and c_2 cannot be recovered until the second time slot, although after the second time slot, all c_1 through c_4 can be recovered simultaneously. In general, if a network coded packet is a combination of n packets, then a receiver will have to receive at least n coded packets in order for it to recover any one of the original packets. This potentially introduces unnecessary delay for video streaming applications. Therefore, we propose a network code structure that enables a receiver to recover the important data gracefully in the presence of limited bandwidth which causes an increase in decoding delay.

B. Hierarchical Network Coding

To increase the probability that the most important data (base layer bits) are available at the servers, and therefore can be pushed down to a receiver, a straightforward scheme is for a source to send more duplicates of the important data. For given bandwidth and storage requirements, this implies taking away some of the bandwidth and storage that might be used for the enhancement layer bits otherwise. For example, let us consider a two layer video bit stream, instead of sending every packet with equal chance (0.5), the source may want to first group the base layer bits and enhancement layer bits into to different types of packets: the base layer packets and enhancement layer packets. Next, it can push the base layer packets to the assisted servers with higher probability, e.g. 0.7 than those of an enhancement layer packets. For a limited redundancy, a receiver will likely to recover the base layer information. Also even when every server has the complete file (high redundancy), the receiver will be able to recover the base layer information faster since the assisted server pushes the packets randomly to a receiver. This method seems promising, however, as will be shown later, it is still far from optimal.

We now describe a hierarchical network coding scheme to overcome the decoding delay of the RNC and duplication of Uncoded schemes, while increasing the chance for a receiver to decode the important bits of the video in time [16]. Let us consider a r layers scalable video stream. We first divide the stream into a number of relatively large consecutive chunks. Each chunk consists of the bits from all the layers. Now, within each chunk, we group the all bits from the same layer i into a number of packets m_i. Denote these packets as b_1^i, b_2^i, ... $b_{m_i}^i$. Next, we code the packets within a chunk using one of the following r structures:

$$p_i = \sum_{j=1}^{m_1} f_j^1 b_j^1 + \sum_{j=1}^{m_2} f_j^2 b_j^2 + .. + \sum_{j=1}^{m_i} f_j^i b_j^i \tag{1}$$

where f_j^i are the non-zero random elements of a finite field F_q and b_j^i are the original packets of layer l_i. Assuming that l_1 and l_r are the most and least important layers, then a coded packet p_i would always contain the information from the base layer. In essence, the coded packets belongs to one of the r classes. Let us denote these classes as N_1 to N_r. The packets belonging to the most important class N_1 contain only information about the base layer. The packets belonging to second most important class contain the information about the base layer and the first enhancement layer. In general, the packets belonging to a k class contain information about layer 1 to k.

Using this encoding structure, given a random number of coded packets, the probability of recovering original

TABLE I

COMPARE CODING SCHEMES WITH 2 LAYERS DATA

Uncoded	WLNC	Hierarchical NC	RNC
a_1	a_1	a_1	a_1
a_2	a_2	a_2	a_2
	$a_1 + a_2$	$a_1 + a_2$	$a_1 + a_2$
b_1	b_1	$a_1 + b_1$	$a_1 + b_1$
b_2	b_2	$a_1 + b_2$	$a_1 + b_2$
	$b_1 + b_2$	$a_1 + b_1 + b_2$	$a_1 + b_1 + b_2$
		$a_2 + b_1$	$a_2 + b_1$
		$a_2 + b_2$	$a_2 + b_2$
		$a_2 + b_1 + b_2$	$a_2 + b_1 + b_2$
		$a_1 + a_2 + b_1$	$a_1 + a_2 + b_1$
		$a_1 + a_2 + b_2$	$a_1 + a_2 + b_2$
		$a_1 + a_2 + b_1 + b_2$	$a_1 + a_2 + b_1 + b_2$
			b_1
			b_2
			$b_1 + b_2$

packets from a base layer is always larger than those of other layers. In fact, the probability of recovering a packet from an important layer is always larger that of a less important layer.

To fine tune the probability of receiving a certain type of packets, one can also control the number of packets belonging to a certain types. For example, one can increase the probability of receiving base layer packets by generating more packets of N_1 type.

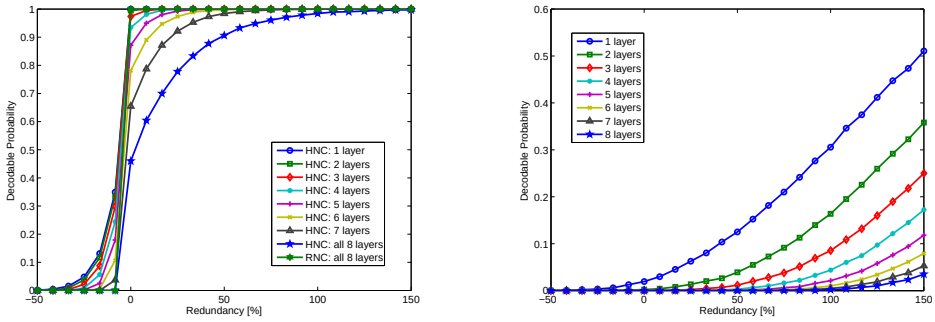
To illustrate our approach, let us consider a simple example involving only 4 packets belonging to one base and one enhancement layer. Let us denote the four packets as a_1, a_2, b_1, and b_2 with a_i's and b_i's belonging to the base and enhancement layers, respectively. Further suppose that the coefficients have binary values only. Table IV-B shows possible encoded packets for four coding schemes: Uncoded, Within Layer NC (WLNC), HNC, and RNC. The WLNC scheme produces coded packets which are linear combinations of the original packets belonging to the same layer.

For each scheme, assuming that the origin server randomly encodes a total of M packets which can be any of these packets as follows. With the exception of RNC, before coding any any packet, the origin server decides whether a packet should be coded as the base layer packet with probability P, or as the enhancement layer packet with probability with $1-P$. After the packet class has been chosen, a packet is randomly and uniformly generated. Equivalently, the packet is chosen uniformly from all the possible packets within a class. By choosing appropriate value of P, one can tune the probability of getting packets from certain classes. For the RNC, there is no class, thus a packet is randomly generated as a random linear combination of the original packets from the entire chunk.

Suppose that three encoded packets (M) are to be randomly generated by each scheme. It is clear to note that when using the non-network coding, exactly two of these packets have to be a_1 and a_2 in order to recover all the base layer packets (a_1 and a_2). For the WLNC scheme, to recover the base layer packets, two distinct packets have to come from the N_1 class. For the HNC scheme, the probability for recovering both a_1 and a_2 is larger than that of WLNC. This is because in addition to being able to recover the N_1 packets from two distinct packets from the N_1 class, this scheme is also able to recover the base layer packets with an appropriate combination of 1 N_1 packet and 2 N_2 packets, e.g., (a_1,a_1+b_1, a_2+b_1). Finally, for the RNC scheme , the probability of recovering base layer packets is approximately equal to that of HNC for this particular example. In more general scenario, RNC scheme would have lower probability of obtaining important layers when small number of packets are chosen.

We note that if the origin server only generates packets from N_1 class, then one has largest probability of recovering the base layer packets. However by doing so, one will never be able to recover packets from the enhancement layer. HNC enables one to recover both base and enhancement layer packets with different probabilities. For RNC in a general setting, as argued in Section IV-A, it may take longer time (more packets) to be able to recover any of the original packets. But when it does so, it can recover all the packets simultaneously.

As a simple example to show the benefits of HNC, we use a scalable stream with 8 layers. The base layer contains 8 packets while other 7 enhancement layers contain 4 packets each. P is set to 1/8, i.e., the probability of generating a packet from any layer is the same. We compare the layer recoverability of non-network coding and HNC schemes as a function of the total number of random packets generated. The more packets are generated, the higher recoverability at the expense of larger redundancy. Redundancy is the number of additional packets needed



(a) (b)

Fig. 2. Layer decodable probabilities as a function of redundancy using (a) HNC and (b) non-network coding.

for decoding all the packets in a layer. Figure 2(a) shows this decodable probability for every layer as a function of redundancy when using HNC and RNC. Similarly, Figure 2(b) shows the decodable probability when no coding is used.

As seen, when the redundancy is less than 0, using HNC, the receiver is able to decode the most important layer with higher probability than that of using RNC. Redundancy less than zero represents the case where a receiver does not receive all $1 \times 8 + 7 \times 4$ packets. This could be due to the servers do not have enough packets or simply the throughput from all the servers to a receiver is not sufficient for the receiver to receive all the packets in time. In this case, a receiver is still able to recover packets from the important layers with some high probability. However, HNC incurs additional bandwidth when the redundancy ranges from 0 to 150%. After 150%, both HNC and RNC results in approximate performance.

On the other hand, when Uncoded scheme is used, the decodable probability is substantially smaller for a same specified redundancy level. Even with the redundancy of 150%, a receiver still fails to decode the layers with high probability. In section VI, we will show more detail performances of network coding schemes for realistic scenarios and compare their performances with traditional Reed-Solomon codes.

V. Joint Network Protocols and Coding Schemes

We now propose network coding schemes for multi-sender streaming framework that reduces the coordination among servers in CDN or peers in P2P networks. We first discuss the RNC-based protocol.

In this scheme, a video stream F is randomly network coded and dispersed to a number of servers/peers in the

network. As described before, a file is partitioned into N chunks $c_1, c_2, ..., c_N$. Each chunk c_i is further divided into n small packets $p_{i1}, p_{i2}, ..., p_{in}$. Now, for each chunk c_i, the origin sender will randomly network code the packets within it, to produce a number of coded packets. These packets will be randomly sent to the assisted servers in a CDN or peers in a P2P network. Note that each server/peer does not need to keep all n coded packets. They may keep only a fraction of the coded packets, but each server/peer will have some coded packets from every chunk c_i. Therefore, the total amount of storage of this scheme is small than that of the traditional CDN.

Using this approach, the receiver first requests all the senders to send their packets p_{1i}'s from the first chunk c_1. The sender then pushes the packets within a chunk to a receiver in a random manner. After the receiver receives roughly n coded packets, it will be able to recover n original packets. It then immediately sends a request to all the senders to start streaming the packets from the second chunk c_2. In the meanwhile, the receiver can start playback the video. The process continues until the end of the stream is reached. Clearly, there is a delay at the beginning due to the time for the receiver to receive n independent packets. The attractive feature of this scheme is that no dynamic packet partition is required. All senders are sending at their available time-varying bandwidth until the receiver sends an *end of chunk* request to move to the next chunk. Therefore, TCP can be employed for streaming. The effective throughput at the receiver is roughly equal to the total throughputs from all the senders. At any point in time, one sender may have a slow connection, but as long as the total throughput is larger than the playback rate, the receiver will be able to playback the video smoothly.

We emphasize that this scheme achieves maximum throughput without the complex coordination for allocating the packets. However, it may not work well when the aggregate throughput of all the senders is smaller than the video bit rate. Thus, one cannot playback the video smoothly.

We now describe a HNC-based protocol that employs scalable video bit stream to solve this problem. Similar to the RNC-based scheme, the video file is partitioned into chunks. However, instead of using random network coding, we employ HNC technique. As discussed previously, HNC packets are coded based on the importance levels of the bits in a scalable bit stream.

The advantage of HNC is that, given a smaller number of received coded packets due to smaller throughput during some period of time, the probability of decoding the base layer using HNC is higher than that of RNC

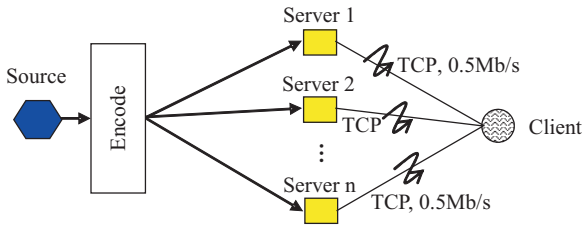


Fig. 3. Simulation setup.

scheme. Thus, when a receiver determines that it may not have enough time to wait for all the coded packets within a chunk, it can readily decode the packets from the important layers, and ignore other undecodable packets. It then signals the senders to send packets from the next chunk. This technique allows a receiver to playback a low quality but smooth video.

VI. Simulation Results

In this section, we investigate the performances of the proposed schemes. To be concrete, our simulations assume a CDN scenario in which, there is an origin server with the original video stream. This server distributes either uncoded or coded packets to a number of assisted servers which are then responsible for streaming the video to a client as shown in Figure 3. In this simulation, the origin server has a 3 layers scalable video bit stream with a rate of 432kbps. The base layer rate is 192kbps, while the rates for the two enhancement layers are 120kbps each. The original stream is divided into a number of chunks c_i of length 1 second. Thus each chunk consists of 36 packets of size 1500 bytes. As a result, the base layer and each of two enhancement layers has 16 and 10 packets, respectively. The origin server distributes the packets to 3 servers using some schemes. Next, TCP is employed to transmit data from these three servers to a single receiver simultaneously.

We consider the following push-based transmission protocol with different coding schemes. In particular, the schemes of interest are: Uncoded , Reed Solomon coding, RNC, WLNC, and HNC. Except for the non-network coding techniques, the protocol used in these schemes are identical to the one described in Section V.

Uncoded. Packets are not coded, however, they are divided into three classes corresponding to the number of video layers. The origin server randomly pushes packets to the assisted servers with different probabilities P_1, P_2, and P_3, based on the classes that the packets belong to.

Reed Solomon (RS). Using this scheme, m original packets within a chunk are coded into $m(1+b)$ coded packets and distributed randomly to the servers. In this case, $m = 16, 10, 10$ for the base layer, first enhancement layer, and second enhancement layer, respectively. Similar to the Uncoded scheme, the packets of three different classes are generated and pushed to the servers with different probabilities.

RNC. The origin server randomly generates a number of packets as linear combinations of 36 packets for each chunk, and distributes these packets to the assisted servers. As a result, each assisted server keeps a fraction of coded packets which are pushed to the receiver randomly. Note that packets are not grouped into class, thus all coded packets are randomly generated with equal probability.

WLNC. The origin server applies network coding to the packets belonging to the same layer. The coded packets are then generated and pushed to the assisted servers with different probabilities according to their classes.

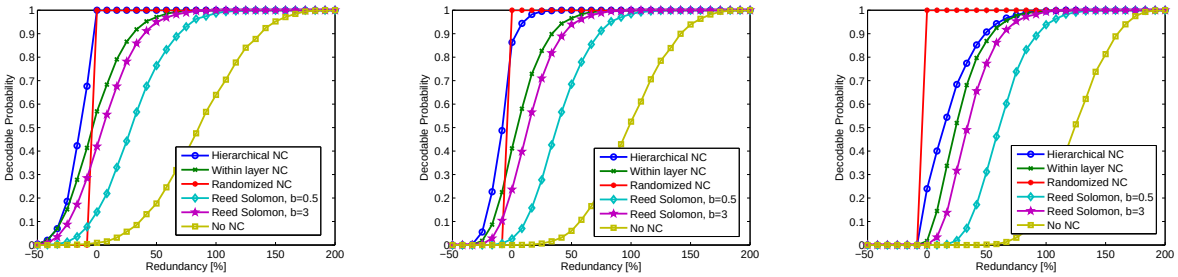
HNC. The origin server employs HNC which results in three classes of packets as described in Section IV-B. These coded packets are generated and pushed to the assisted servers with different probabilities.

First, we characterize the probability of a receiver being able to decode certain layer as a function of storage for different coding schemes. A layer is decodable if all of its packets are recoverable. This implies that there are enough distinct coded packets for this layer on the servers. With the exception of RNC, one important parameter for these schemes are the probabilities for which the packets from certain classes are generated and sent to a receiver. Intuitively, higher probability of sending packets from a class results in higher probability for a receiver being able to decode all the packets from that class. That said, we present the simulation results on the decodability for different transmission (equivalently, generation) probabilities for different classes.

Figure 4 shows the decodable probabilities for different layers when using different schemes. The transmission probabilities for each layer are set to equal to each other, i.e., $P_1 = P_2 = P_3$. As seen, when the redundancy level is less than zero using HNC has the largest probabilities of recovering layers 1 and 2, followed by WLNC, RS, RNC, and Uncoded schemes. On the other hand, when the redundancy level is greater than zero, RNC scheme has the largest probability of recovering layers 1 and 2, followed by HNC, WLNC, RS, and Uncoded schemes. When the redundancy level is 150%, HNC, WLNC, RS, and RNC schemes can recover all three layers with a probability close to 1, but the Uncoded scheme requires redundancy of almost 200% to accomplish the same goal.

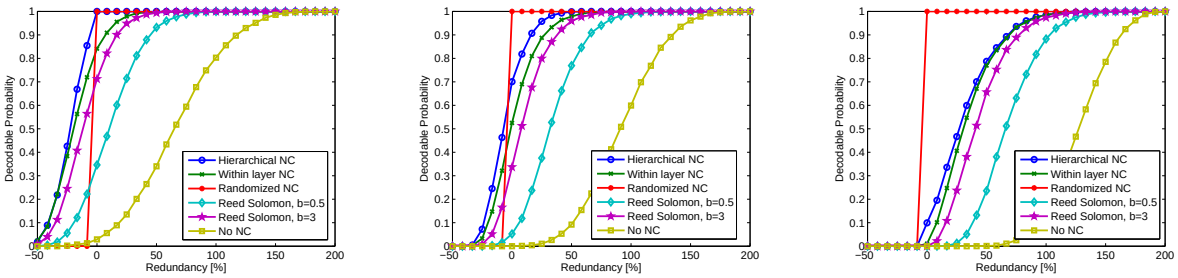
Similarly, Figures 5 and 6 show the decodable probabilities of different schemes when the transmission probability for layer 1 increases. In particular, the transmission probabilities for the layers are now set as $(P_1, P_2, P_3) = (0.4, 0.3, 0.3)$ and $(P_1, P_2, P_3) = (0.5, 0.25, 0.25)$, respectively. As seen, the decodable probability for layer 1 increases for all the schemes. This is intuitively plausible as more packets of layer 1 are likely to be sent to the assisted servers, and thus a receiver is likely to be able to decode all the packets from this layer. On the other hand, this comes at the expense of not getting packets from other layers as shown in Figure 5 and 6.

From the simulation results, it is best to employ RNC and HNC when the redundancy is greater or smaller than zero, respectively.



(a) (b) (c)

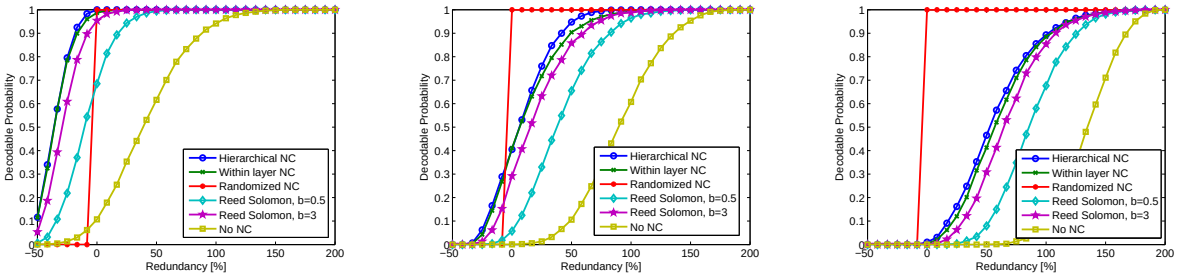
Fig. 4. The probability of a receiver being able to decode (a) first layer; (b) first and second layers; (c) All 3 layers; $P_1 = P_2 = P_3 = 1/3$.



(a) (b) (c)

Fig. 5. The probability of a receiver to be able to decode (a) first layer; (b) first and second layers; (c) All 3 layers; $P_1 = 0.4$, $P_2 = 0.3$, $P_3 = 0.3$.

We now consider a scenario where each server has much redundancy, thus a receiver will be able to recover the packets if there is sufficient throughput between itself and the servers. One problem arises, however, when the total throughput at a receiver is less than the video playback rate due to network congestion. In that case, using HNC



(a) (b) (c)

Fig. 6. The probability of a receiver to be able to decode (a) first layer; (b) first and second layers; (c) All 3 layers; $P_1 = 0.5$, $P_2 = 0.25$, $P_3 = 0.25$.

may allow a receiver to receive and decode the most important layers early and in time for playback. A receiver then can request the next chunk in a video stream from the servers when it determines that there is not enough throughput to receive the enhancement packets.

To simulate this scenario, we use network simulator NS [17]. We use a number of servers to transmit data to a receiver as shown in Figure 3. Heavy traffic between the servers and the receiver are generated using on-off exponential distribution with mean of 300kbps. The on and off periods are set to 50 ms each. The physical bandwidth for the link between the servers and the receiver are set to 500 kbps. Since TCP is used for transmission the data, the available throughputs of different connections vary with time, resulting in different number of packets received per some unit time. Figure 7 (a) and Figure 7(b) show the average time before a client can decode different layers within a chunk for different schemes when using 3 and 6 servers, respectively. As seen, for the Uncoded scheme, the time to decode any layer is largest due to the high probability of getting duplicated packets. The performances of RS schemes are better than Uncoded, but worse than those of RNC and HNC schemes. For the RNC scheme, the time to decode all 3 layers is smallest. However, the time to decode 1 and 2 layers are longer than those of the HNC. This is due to the fact that RNC scheme mixes all the packets together; thus it requires a larger number coded packets to decode any packets. However, when enough number of packets are received, it can decode all the packets simultaneously. On the other hand, HNC allows a receiver to recover the important packets early, but pays extra overhead to recover all the packets. This is suitable for scalable video streaming since if there is not enough bandwidth as automatically dictated by TCP congestion control, the receiver can instruct the servers to start

sending packets from the next chunk. In the meanwhile, the receiver can playback the important layers that it has received. Similar results are obtained when the physical bandwidth of the links are reduced to 70 kbps and 60 kbps

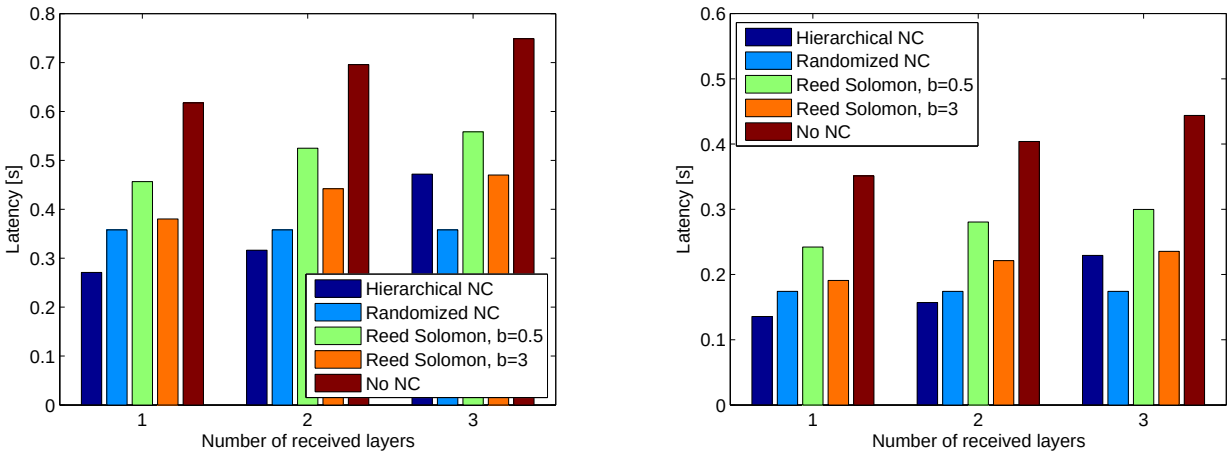


Fig. 7. Latencies to receive one chunk using non-network coding, RS, RNC, and HNC for (a) 3 servers; (b) 6 servers; Link bandwidth = 500 kbps.

as shown in Figure 8, respectively.

We now compare HNC technique with the coordinated technique in which the receiver instructs the servers 1, 2, 3 to send its packets with equal rates. Packets from more important layer are sent first, followed by the less important ones. It is easy to see that if the available bandwidth of all the servers are equal to each other, then this coordinated technique is close to optimal. However, when the available throughputs of these servers are not equal and varied with time, then the HNC technique can outperform this coordinated technique significantly. In particular, we assume that relatively large disjoint partitions of data among the senders are used in this coordinated technique. Furthermore, a receiver can dynamically request a new data allocation from the senders due to the change in the estimated throughput. However, the new allocation request is only sent after the receiver has received all the data from the current allocation. Therefore, if the partition size is large, when the available throughputs change, a receiver may have to wait some time before requesting a new data allocation from the servers. This may result in suboptimal throughput.

We simulate the unequal throughputs by injecting on-off exponential traffic with the mean of 450 kbps for one link, and 250 kbps for each of the other two links. The physical bandwidth for each link is set to 500 kbps. As

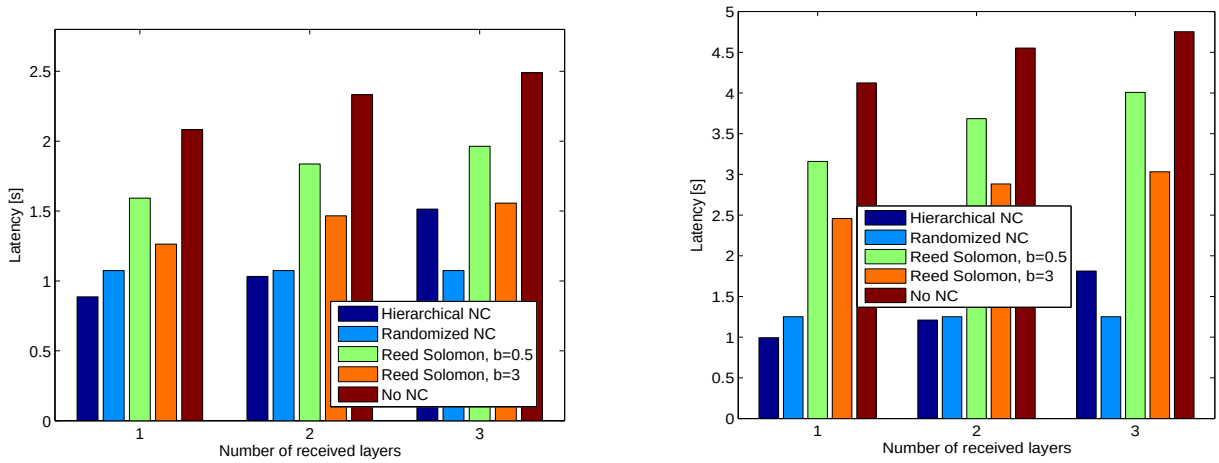


Fig. 8. Latencies to receive one chunk using non-network coding, RS, RNC, and HNC using six servers (a) physical bandwidth per link = 70 kbps ; (b) physical bandwidth per link = 60 kbps.

seen in Figure 9, both schemes are able to obtain the base layer packets in reasonable short time. However, the coordinated scheme take along time to receive the enhancement packets. This is due to the congestion at one link. During this congestion period, for the coordinated scheme, two other servers are idle since they already sent all of their packets. However, the partition sent by the server with the congested link takes a long time to arrive at the receiver. This happens because the receiver cannot dynamically repartition the packets fast enough to accommodate the change in the available throughput. As a result, the coordinated approach takes up to 60% more time to obtain a complete chunk.

VII. Related Work

Many P2P file sharing systems such as BitTorrents or KaZaa can be viewed as multi-sender systems [18]. This is because a BitTorrent file is partitioned into multiple distinct pieces, and these pieces are then exchanged among the peers to increase the receiving throughput [1]. Thus, at any point in time, a peer can receive multiple pieces from different peers. BitTorrents, however, is not designed for streaming since the pieces of data received at a peer, can be significantly out-of-order. CoolStreaming, on the other hand, is designed for streaming [10]. However, it does not use any sophisticated coding. Thus the performance can be shown *theoretically* lower than those of

[1]As compared to using a single server to send the pieces to multiple receivers

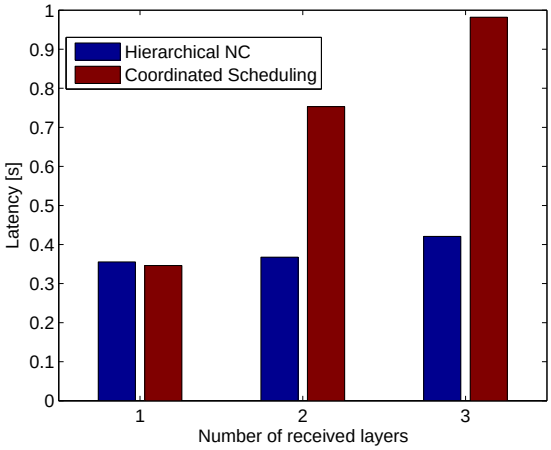


Fig. 9. Latencies for coordinated transmission vs. non-coordinated HNC based transmission.

P2P systems that employ coding. Systems that employ coding techniques include the work of Byers *et al.* [19] on informed overlay network. In this work, Byers *et al.* proposed to partition data and make use of peers to increase the throughput of the system. In this approach, each node randomly sends different coded partitions on different links. Data reconciliation techniques are then used to reduce data redundancy sent between nodes. Random network coding technique has also been previously used in Avalanche system for file distribution in P2P network [20]. Avalanche employs random network coding with large block size, and has been shown to substantially improve the overall throughput compared to other P2P systems. However, because of the large block size, Avalanche is not designed for streaming. Regarding storage, our CDN model is similar to the work of Acendanski *et al.* on the distributed server storage using random network coding [21].

In other work, Padmanabhan *et al.* used multiple overlay multicast trees to stream multiple descriptions of the video to the clients [22]. Each multicast tree transmits a description of the video. When a large number of descriptions are received, higher video quality can be achieved. Recently, Li *et al.* proposed MutualCast [23] which focuses on throughput improvement for data dissemination in P2P network. MutualCast employed partitioning techniques and a fully connected topology to ensure that the upload bandwidth of all the nodes is fully utilized.

VIII. Conclusions

We have proposed a *network coding* framework for efficient media streaming in CDNs or P2P networks in which, multiple servers/peers are employed to simultaneously stream a video to a single receiver. Our framework reduces the redundancy storage and simplifies the tight synchronization between the senders and receivers. Furthermore, we proposed an HNC technique to be used with scalable video bit stream to enable a receiver to adapt to the available bandwidth. Simulation results demonstrate that under certain scenarios, our proposed schemes can result in bandwidth saving up to 60% over the traditional schemes.

References

[1] W. Tan and A. Zakhor, "Real-time internet video using error resilient scalable compression and tcp-friendly transport protocol," *IEEE Transactions on Multimedia*, vol. 1, pp. 172–186, june 1999.

[2] H. Ma and M. El Zarki, "Broadcast/multicast mpeg-2 video over wireless channels using header redundancy fec strategies," in *Proceedings of The International Society for Optical Engineering (SPIE)*, November 1998, vol. 3528, pp. 69–80.

[3] R. Rejaie and A. Ortega, "Pals:peer to peer adaptive layered streaming," in *NOSSDAV*, June 2003.

[4] T. Nguyen and A. Zakhor, "Distributed video streaming," in *Proceedings of the SPIE - The International Society for Optical Engineering, Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2002, vol. 4673, pp. 186–95.

[5] J. Apostolopoulos, "On multiple description streaming with content delivery networks," in *InfoComm*, June 2002, vol. 4310.

[6] http://www.kazaa.com.

[7] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris, "The case for resilient overlay networks," in *Proceeding of HotOS VIII*, May 2001.

[8] T. Nguyen and A. Zakhor, "Multiple sender distributed video streaming," *IEEE Transactions on Multimedia*, vol. 6, no. 2, pp. 315–326, April 2004.

[9] R. Rejaie and S. Stafford, "A framework for architecting peer-2-peer receiver-driven overlays," in *NOSSDAV*, June 2004.

[10] X. Zhang, J.C. Liu, B. Li, and T.P. Yum, "Coolstreaming/donet: A data-driven overlay network for efficient live media streaming," in *INFOCOM*, March 2005.

[11] D.A. Tran, K.A. Hua, and T. Do, "Zigzag: An efficient peer-2-peer scheme for media streaming," in *INFOCOM*, April 2003.

[12] R. Ahlswede, N. Cai, R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inform. Theory*, vol. 46, pp. 1204–1216, July 2000.

[13] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Trans. Networking*, vol. 11, no. 3, Oct 2003.

[14] T. Ho, R. Koetter, M. Mdard, M. Effors, J. Shi, and D. Karger, "A random linear network coding approach to multicast," *IEEE/ACM Transtions on Information Theory*, vol. 10, Oct 2006.

[15] T. Ho, R. Koetter, M. Mdard, D. R. Karger, and M. Effros, "Benefits of coding over routing in a randomized setting," in *IEEE International Symposium on Information Theory*, 2003.

[16] K. Nguyen, T. Nguyen, and S. Cheung, "Peer-to-peer streaming with hierarchical network coding," in *IEEE International Conference on Multmedia and Expo, 2007*, July 2007.

[17] Information Sciences Institute, http://www.isi.edu/nsnam/ns, *Network simulator*.

[18] http://www.bittorrents.com.

[19] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, October 2004.

[20] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *IEEE Infocom*, march 2005.

[21] S. Acendanski, S. Deb, M. Medard, and R. Koetter, "How good is random linear coding based distributed networked storage?," in *NetCod*, 2005.

[22] V.N. Padmanabhan, H.J. Wang, P.A. Chou, and K. Sripanidkulchai, "Distributed streaming media content using cooperative networking," in *ACM NOSSDAV*, Miami, FL, May 2002.

[23] J. Li, P. Chou, and C. Zhang, "Mutualcast: An efficient mechanism for one-to-many content distribution," in *ACM Sigcomm Asia Workshop*, April 2005.