

Peer-to-Peer Streaming with Hierarchical Network Coding

Kien Nguyen and Tinh Nguyen
School of Electrical and Computer Engineering
Oregon State University
Email: {nguyenki, tinhhq}@eecs.orst.edu

Sen-ching Cheung
Electrical and Computer Engineering Department
University of Kentucky
Email: cheung@engr.uky.edu

Abstract— In recent years, Content Delivery Networks (CDN) and Peer-to-Peer (P2P) networks have emerged as two effective paradigms for delivering multimedia contents over the Internet. An important feature in CDN and P2P networks is the data redundancy across multiple servers/peers which enables efficient media delivery. In this paper, we propose a *network coding* framework for efficient media streaming in either content delivery networks or P2P networks in which, multiple servers/peers are employed to simultaneously stream a video to a single receiver. Unlike previous multi-sender schemes, we show that network coding technique can (a) reduce the redundancy storage, (b) eliminate the need for tight synchronization between the senders, and (c) be integrated easily with TCP. Furthermore, we propose the Hierarchical Network Coding (HNC) technique to be used with scalable video bit stream to combat bandwidth fluctuation on the Internet. Simulation results demonstrate that our proposed scheme can result in bandwidth saving up to 40% for many cases 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 the 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]. From the architecture perspective, Content Delivery Network (CDN) companies such as Akamai employ the edge architecture by pushing content to the edge of the network, and by strategically placing servers at the edge of the Internet in such a way that each client can choose the server resulting in shortest round-trip time and least amount of congestion.

Recently, the multi-sender streaming paradigm has been proposed as an alternative to edge streaming for providing smooth video delivery [3][4][5]. The main idea is to have each server storing an identical copy of the video. The video is then 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 a original CDN server, or through an uncoordinated propagation of contents in a P2P networks 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, many schemes assume a UDP-like transport protocol, which 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. We first list a few related work in Section II. Next, we provide a short introduction of network coding in Section III, then propose the HNC techniques to be used for P2P scalable video streaming. In Section IV, we describe a protocol for multi-sender streaming using network coding. Simulation results to demonstrate our approach is presented in Section V.

II. RELATED WORK

Many P2P file sharing systems such as BitTorrents can be viewed as multi-sender systems [7]. 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 throughput¹. 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. BitTorrents also does not use any sophisticated coding, thus the performance can be shown theoretically lower than P2P systems that employ coding. For example, Byers *et al.* [8] 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. In other work, Padmanabhan *et al.* used multiple overlay multicast trees to stream multiple descriptions of the video to the clients [9]. 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 [10] 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.

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

III. NETWORK CODING

The network coding field started with the pioneering paper by Ahlswede *et al.*, who showed that maximum capacity in a network can be achieved by appropriate mixing of data at the intermediate nodes [11]. The most elegant result of network coding is that the maximum network capacity is achievable using some *random network coding* technique, while this is not usually possible with the traditional store and forward routing.

A. Random Network Coding for P2P Networks

Using random network coding, a peer encodes a new packet p_i by linearly combining n original packets as:

$$p_i = \sum_{j=1}^n f_{ij} c_j \quad (1)$$

where f_{ij} are the random elements belonged 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 P2P networks can be seen in the following simple scenario.

Assuming that a server wants to distribute a file to a number of peers in a P2P network. To increase the throughput, the server first divides a file into n different chunks and randomly distributes these chunks to the peers. The peers then can exchange their chunks among each other in a random manner. Since each peer downloads pieces of the file from other peers and the server simultaneously, the time for a peer to recover all n chunks of the packets is potentially much shorter than that of downloading the file from the server only. Note that this design scales well since no coordination among the peers is required. However, this design is not optimal. Because of the random exchange of packets (for scalability of not tracking the packets), some of the packets received at a peer may be duplicate, and thus results in wasteful bandwidth. For example, suppose a peer C randomly retrieves packets of a file from other peers A and B simultaneously. The file is divided into 4 chunks c_1, c_2, c_3 , and c_4 . Assume also that the chunks at peer A are c_1, c_2, c_3 , and at peer B are c_2, c_3 , and c_4 . Now suppose C starts to download the chunks c_1, c_2, c_3 and from A and c_2, c_3 , and c_4 from B in that order. After the first time slot, C obtains both c_1 and c_2 . In the second time slot, C downloads 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 downloads c_3 and c_4 , and discards c_3 . As seen, C needs to download six chunks or three time slots to be able to receive the complete file.

Suppose each peer is allowed to use network coding in which, it encodes an output chunk as a linear combination of the input chunks. In particular, peers A and B may randomly encode their packets as:

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

where f_{ij}^a and f_{ij}^b are random elements belonged to a finite field F_q . If C downloads $a_1 = f_{11}^a c_1 + f_{12}^a c_2 + f_{13}^a c_3$ 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. Also, information about f_{ij}^a and f_{ij}^b can 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 $m \gg n$ 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, C 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 are also 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 presence of limited bandwidth which causes an increase in decoding delay.

B. Hierarchical Network Codes

We design HNC such that when a small number of coded packets are received, with high probability, the most important data can be recovered. To illustrate our approach, suppose a file is divided into a number of packets, and each packet belongs to one of the 3 classes: A, B , and C with A and C being the most and least important, respectively. We further assume that there are six packets in the file with a_1, a_2 belonged to A , b_1, b_2 belonged to B , and c_1, c_2 belonged to C . We then randomly code the packets using one of the following structures:

$$\begin{aligned} N_1 &= f_1^1 a_1 + f_2^1 a_2 \\ N_2 &= f_1^2 a_1 + f_2^2 a_2 + f_3^2 b_1 + f_4^2 b_2 \\ N_3 &= f_1^3 a_1 + f_2^3 a_2 + f_3^3 b_1 + f_4^3 b_2 + f_6^3 c_1 + f_6^3 c_2 \end{aligned} \quad (2)$$

where f_i^j are the random non-zero elements of a finite field F_q . Using this structure, it is easy to see that the probability of recovering a_i 's is always larger than that of b_i 's and the probability of recovering b_i 's is always larger than that of c_i 's. This is because if one assumes that all packets arrive at a peer in a random manner, then only two packets of N_1 type are needed to recover a 's while 4 packets of either N_1 or N_2 types are needed to recover a 's and b 's. To fine tune the probability of receiving a certain type of packets, one can control the number of packets belonged to a certain types. For example, one can increase the probability of receiving packets of type N_1 by generating more packets of N_1 type.

IV. P2P STREAMING WITH NETWORK CODING

In a traditional video streaming application, a video is streamed from a server to a client. However, if the path between the server and the client experiences heavy congestion, the quality of the video can degrade significantly. To overcome congestion, many researchers have proposed the path-diversity streaming technique in which, the different parts of the video are simultaneously streamed from multiple servers to the client on multiple distinct routes [4][12][3][5]. With appropriate channel and source coding techniques and rate allocation among the servers, the video quality at the receiver can be improved significantly. In addition, video streaming using multiple servers also allows fine-grained load balancing to improve network performance. Unfortunately, current approaches to multi-sender video streaming requires a careful coordination between a client and server to achieve optimal performance. In particular, assuming that two servers are used for streaming, then server 1 can stream the odd packets while

the other streams the even packets, starting from the beginning of the file. This approach works when there are only two servers, and that their average bandwidth are equal and constant throughout the streaming session. When there are many servers with different available bandwidth, and these bandwidths are varied with time (e.g. TCP is used for streaming), then obtaining the optimal packet partitions for each server requires a complex dynamic coordination between the client and the servers. Even when complex coordination is possible, the inaccurate estimation of available bandwidth is often not possible which results in suboptimality.

We now propose two network coding schemes for multi-sender streaming framework that reduces the coordination among servers. In the first scheme, a video file F is randomly network coded and dispersed to a number of servers/peers in the network. In particular, our model is similar to the networked storage architecture as described in [13]. In this model, a file is partitioned into N chunks c_1, c_2, \dots, c_N . Each chunk c_i is further divided into n small packets $p_{i1}, p_{i2}, \dots, p_{in}$. Now, for each of the chunk c_i , the origin server will randomly network code the packets within it, to produce a number of coded packets. These packets will be randomly sent to the other servers/peers. 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 other approaches.

Using this approach, the client first requests all the servers to send their packets p_{i1} 's from the first chunk c_1 . After the client receives roughly n coded packets, it will be able to recover n original packets. It then immediately sends a request to all the servers to signal them to start streaming the packets from the second chunk c_2 . In the meanwhile, the client can start video playback. 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 client to receive n independent packets. The attractive feature of this scheme is that no dynamic packet partition is required. All servers are sending at their available time-varying bandwidth until the client sends an *end of chunk* request to move to the next chunk. Therefore, TCP can be employed for streaming. We emphasize that this scheme achieves maximum throughput without the complex coordination. This scheme, however may not work well when the aggregate bandwidth of all the servers is smaller than that of the video bit rate. Thus, one cannot playback the video smoothly.

We propose a second scheme to solve this problem using HNC together with scalable video bit streams. A scalable video bit stream is composed of a base layer and several enhancement layers. The base layer is the most important layer and must be present in order to have a reasonable video quality. The enhancement layers are organized in a hierarchical fashion such that the first enhancement layer must be present for the second enhancement layer to be useful, and the second enhancement layer must be present for the third enhancement layer to be useful, and so on. As the client receives more layers, the higher quality video can be obtained. In a single server, single client, and non network coding scenario, a scalable video bit stream allows the server to adapt its bandwidth to the network conditions by varying the number of enhancement layers. On the other hand, a direct application of scalable bit stream to the scenario involving multiple servers with network coding technique is non-trivial due to the mixing of packets resulted from network coding.

Our approach is to apply HNC on the existing hierarchy of a scalable bit stream. In particular, the origin server would randomly code packets using HNC technique based on the importance levels

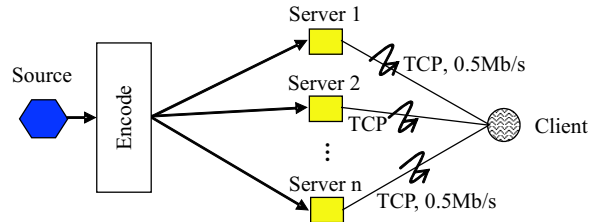


Fig. 1. Simulation setup.

of the bits in a scalable bit stream. HNC technique allows a client to likely receive the most important packets first. Thus, if the current available bandwidth is low, the client can decide to playback the video using the most important packets, e.g. important layers, and requests the servers to send the packets from the next chunk, depending on the current available bandwidth. This technique allows a client to have a low quality but smooth video playback.

V. SIMULATION RESULTS

In this section, we investigate the performances for different multi-sender schemes. We first consider the following uncoordinated schemes: non-network coding, random network coding, and HNC schemes. For simplicity, we assume that there is an origin server with the original file. This server distributes either uncoded or network coded packets to a number of non-origin (streaming) servers which are then responsible for streaming the video to a client as shown in Figure 1. In this simulation, we use a file of F which is divided into a number of chunks c_i of 1Mbyte, each chunk consists of 10 packets of size 100K bytes. Note that 100 Kbytes is not the size of the packet sent to the network. This is simply the size of the packets that are used in network coding operation. We use TCP to transmit data from multiple servers to a single client.

Non-network coding scheme. The origin server distributes the entire video file to all the streaming servers. Each streaming server randomly sends packets belong to a chunk c_i . When the client signals an “end” signal, the server then moves on to the next chunk, and randomly sends packets of this new chunk.

Network coding scheme. The origin server has previously generated network coded packets (linear combination of 10 packets for each chunks) and distributed these packets to the streaming servers. Thus, for each chunk c_i , each streaming server keeps a fraction of random network code packets. Each streaming server then sends these packets at random to a client.

HNC scheme. The origin server has previously encoded the packets using HNC. In particular, the packets of the video are classified into 3 layers A , B and C , and are randomly coded by the origin server using the following structures:

$$\begin{aligned} N_1 &= f_1^1 a_1 + f_2^1 a_2 & (3) \\ N_2 &= f_1^2 a_1 + f_2^2 a_2 + f_3^2 b_1 + f_4^2 b_2 \\ N_3 &= f_1^3 a_1 + f_2^3 a_2 + f_3^3 b_1 + f_4^3 b_2 + f_5^3 c_1 + f_6^3 c_2 \end{aligned}$$

where f_i^j 's are random coefficients of a finite field F_q . The origin server generates packets such that a packet has equal probability of being in either N_1 , N_2 , or N_3 types. For simulation purpose, the number of packets in layers A , B , and C are equal to each other. These packets are randomly distributed to the streaming servers. Note that non-origin servers are oblivious to the packet types. They just randomly send their packets to the client.

Heavy background traffic between the servers and the client are generated using on-off exponential distribution with mean of 300kbps

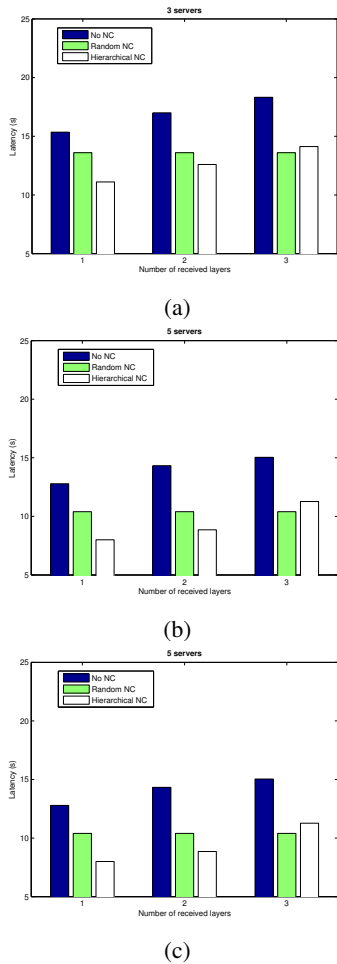


Fig. 2. Latencies for non-network coding, random-network coding, and HNC coding for (a) 3 servers; (b) 5 servers; (c) 7 servers.

and the on and off periods of 50 ms each. Figure 2 shows the average time before a client can decode different layers within a chunk for different schemes. As seen, for the uncoded scheme, the time to decode any layer is largest due to high probability of getting duplicate packets. For the random network coding scheme, the time to decode all 3 layers is shortest. However, the time to decode 1 and 2 layers are longer than those of the HNC. This is due to the fact that random network coding 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, 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.

We now compare HNC technique with the coordinated technique in which, the receiver instructs the servers 1, 2, 3 to send distinct packets with equal rates. It is easy to see that if the available bandwidth of all the servers is equal to each other, then this coordinated technique is optimal. However, when the available bandwidth of these servers are not equal and varied with time, then the HNC technique can outperform this coordinated technique significantly. In particular, we simulate heavy congestion of a link in a three server scenario by

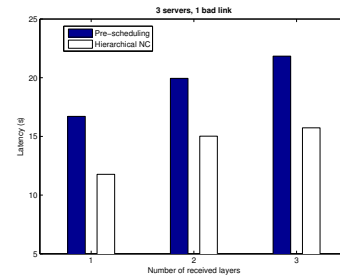


Fig. 3. Latencies for coordinated transmission vs. non-coordinated HNC based transmission.

injecting into link one the same on-off exponential traffic but with the mean of 500kbps. As seen in Figure 3, the receiver cannot dynamically repartition the packets based on the available bandwidth in time, thus the coordinated approach takes up 40% more time to download a chunk.

VI. CONCLUSIONS

We have proposed a *network coding* framework for efficient media streaming in either content delivery networks 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 tight synchronization between the senders. Furthermore, the proposed HNC technique to be used with scalable video bit stream enable a receiver to adapt to the available bandwidth. Simulation results demonstrate that our proposed scheme can result in bandwidth saving up to 40% for many cases 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] <http://www.bittorrents.com>.
- [8] 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.
- [9] 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.
- [10] J. Li, P. Chou, and C. Zhang, "Mutualcast: An efficient mechanism for one-to-many content distribution," in *ACM Sigcomm Asia Workshop*, April 2005.
- [11] R. Ahlswede, N. Cai, R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inform. Theory*, vol. 46, pp. 1204–1216, July 2000.
- [12] T. Nguyen and A. Zakhor, "Distributed video streaming with forward error correction," in *Packet Video Workshop*, Pittsburg, PA, April 2002.
- [13] S. Acemanski, S. Deb, M. Medard, and R. Koetter, "How good is random linear coding based distributed networked storage?," in *NetCod*, 2005.