A P2P VIDEO DELIVERY NETWORK (P2P-VDN)

Kien Nguyen, Thinh Nguyen
School of EECS
Oregon State University
{nguyenki, thinhq}@eecs.oregonstate.edu

Yevgeniy Kovchegov
Department of Mathematics
Oregon State University
kovchegy@science.oregonstate.edu

ABSTRACT

Current video streaming and storage systems such as YouTube, are based on the client-server model, thus do not scale well in terms of bandwidth and computation. This paper describes a Peer-to-Peer Video Delivery Network (P2P-VDN) that provides both performance improvement and scalability based on three architectural elements. First, the proposed P2P-VDN employs a Random Network Coding (RNC) scheme that breaks a video stream into multiple smaller pieces, codes, and disperses them throughout peers in the network, in such a way to maximize the probability of recovering the original video under peer departures and failures. Second, the proposed P2P-VDN employs a scalable mechanism for automating the data replenishment process using RNC that is necessary to maintain a sufficient level of redundancy for video stored in the network. Third, the proposed P2P-VDN employs a path-diversity protocol for a client to simultaneously stream a video from multiple peers in the P2P-VDN. Simulations demonstrate that under certain scenarios, our proposed P2P-VDN can result in bandwidth saving up to 60% over the traditional architecture.

Index Terms— Network coding, Data replenishment, Media storage and streaming

1. INTRODUCTION

The most popular video storage and streaming system on the Internet is YouTube. Its users can upload, view and share their video clips with others. In January 2008 alone, nearly 79 million users had viewed over 3 billion videos. Although the numbers are impressive, majority of YouTube videos are rather short and of low-quality, resulting in manageable amounts of storage and streaming bandwidth. This allows YouTube to centrally control and coordinate their servers, but its bandwidth cost is estimated at 1 million dollars a day. Furthermore, YouTube, like many existing streaming services, still employs the traditional server-client model for streaming. This does not scale since a server computational capability or the bandwidth ultimately limits the number of clients that can be served at any given time.

This work is supported by NSF under grants: CAREER CNS-0845476 and CNS 0834775.

The recent development of Peer-to-Peer (P2P) networks opens a new possibility of building completely distributed systems that have the potential to eliminate the computational and bandwidth bottlenecks existed in the traditional client-server architecture. To that end, a fair amount of distributed system research has recently been focused on using P2P platforms to build reliable, large scale distributed systems for Internet services [1] [2]. Notably, in Chord, Stoica *et al.* employ a Distributed Hash Table (DHT) structure in order to provide large scale Internet services, e.g., DNS in a scalable manner. In this paper, we describe a Peer-to-Peer Video Delivery Network (P2P-VDN) for providing video streaming services over the Internet using Chord as its indexing architecture. The proposed P2P-VDN has the potential to achieve both high scalability and performance based on three key designs.

First, in our proposed P2P-VDN, a video is broken into multiple pieces, coded properly, then dispersed to a number of peers in the network. This is in contrast with the current video storage and streaming systems such as YouTube or Akamai whose individual videos are stored in their entireties at a video server, or at multiple servers if video replication is employed. As will be described subsequently, using a proper indexing architecture such as Chord, a video publisher will be able to publish his video, i.e. to disperse his coded video pieces to the P2P-VDN. This is done in such a way to allow a client to efficiently locate the pieces of a previously published video, and use them to recover the original video. From a user perspective, this is much like uploading one's video to one of the YouTube server. However, unknown to the uploader, his video will be broken up into parts, coded properly, and dispersed to multiple peers in the P2P-VDN. And unknown to a client, his requested video is being retrieved not from a single server, but from multiple peers.

Second, in the P2P-VDN, videos are stored at peers who, unlike servers, are unreliable due to their frequent departures and failures. As a result, video availability is a major concern. Therefore, the P2P-VDN employs a data replenishment mechanism to either proactively or passively fill in the missing data due to peer departures or failures. It does so by recruiting the incoming peers to take part in sharing the burden of storage and streaming resources. Specifically, data replenishment mechanism is designed to automatically maintain a constant level of redundancy for videos in the network over

978-1-4244-4581-3/09/$25.00 ©2009 IEEE

time, by allowing incoming peers to store appropriate *generated* data. This compensates for the data loss due to a departing peer. Its aim is to increase the probability that, at any moment, peers in the network, collectively store a sufficient number of coded pieces of a particular video that allows a perfect reconstruction of that video. Furthermore, the data replenishment is designed to be distributed, scalable, and does not require the presence of the original videos. Peers take part in the data replenishment process in a random and independent manner, but yet collectively, the video in the network is robust against peer dynamics. As will be discussed subsequently, the key to such a design is to employ Random Network Coding (RNC).

Third, to watch a video, a client requests simultaneous transmissions from multiple peers in the P2P-VDN that collectively have all the pieces to reconstruct the requested video. This approach creates multiple Internet routes for transmitting a video to a client, resulting in larger throughput and higher video quality. To accomplish this, we describe a path-diversity streaming protocol using network coding technique that reduces the complexity of sender synchronization while enabling TCP streaming.

Our paper is organized as follows. First, we provide some background and related work on P2P streaming systems and coding techniques in Section 2. In Section 3, we describe three key elements of the proposed P2P-VDN system: data dispersion, data replenishment, and path diversity streaming protocol. In Section 4, we present some simulation results, showing the performance improvement of the proposed P2P-VDN.

2. BACKGROUND AND RELATED WORK

In recent years, the number of P2P video streaming systems such as PPLive and PPStream has grown significantly. The main advantage of the current P2P video streaming systems is the bandwidth saving for a live video broadcaster. In a broadcast session, there are multiple peers (participants) who desire to watch the same video. Therefore, the broadcaster may need only to send a single video stream to one or a few peers who then relay the same video stream to other peers. The process repeats itself until every peer receives the video stream. Effectively, the P2P streaming technique above only requires one or few video streams from the broadcaster, rather than N streams for N users as required in the client-server model. Thus, P2P streaming eliminates the bandwidth and computational bottlenecks.

On the other hand, for video-on-demand applications, it is likely that at any moment, there are only one or a few users who want to watch the same video. In this case, the bandwidth advantage of P2P systems seems to be significantly diminished. For this reason, the video-on-demand market has been dominated by Content Delivery Networks (CDN). Because of the on-demand characteristic, many more videos must be stored on CDN servers, in order to satisfy a potentially large number of different client's requests at any moment. Therefore, from both network bandwidth and storage perspectives, using a single video streaming server is not possible. As such, many CDNs such as Akamai use multiple servers to optimize the streaming performance for a large number of users at different locations in the Internet. Specifically, Akamai servers are strategically placed at the edge of the Internet such that the nearest server to a client is chosen for streaming, thus improving the client's viewing experience.

That said, an Akamai-like approach might not be an optimal approach since the overall storage amount, bandwidth capacity, and computational capability still scale linearly with the number of clients. We advocate a new P2P approach to designing video-on-demand systems in which participated peers contribute not only bandwidth but also storage resources. However, by nature, peers join and leave the network frequently. As a result, videos might be lost temporarily or permanently. Thus, there is a need to develop techniques and policies for providing sufficient redundancy to ensure that, when a video is requested, it is available in the network.

A simple strategy is to replicate a video at some number of peers. This approach, however, is storage inefficient. A more effective approach is to use FEC. A FEC code with rate n/k ensures that if k or more distinct packets are received out of a FEC block of n packets, then the original k data packets can be recovered [3]. Thus, for every k packets belonging to a video, one can generate additional $n-k$ parity packets, and distribute all n packets to a number of peers in the network. For example, using Reed Solomon code $RS(8,4)$, one may distribute 1 packet to each of the 8 peers. As long as 4 of these peers are in operation, a client will be able to recover the entire data block.

If not designed carefully, this approach is suboptimal. Consider the same example above for a very unstable network. As such, we want to distribute those packets to 16 peers rather than just 8 peers. The problem is, for each FEC block, we have only 8 data packets, thus it is necessary that some peers must share identical packets. Because of this, if at some given time, only four peers are in the network, they might collectively have fewer than 4 *distinct* packets. As a result, a client cannot recover the original block from these peers.

The work by Acendanski *et al.* [4] and Nguyen *et al.* [5, 6] use the Random Network Coding (RNC) technique to overcome this problem. Using RNC, a coded packet is a random linear combination N original packets. Each peer then can keep a fraction of the total number of coded packets. If at any given time, the peers in the network collectively have a set of coded packets that form a set of N linearly independent equations, then a client connecting to these peers will be able to recover N original packets. Using the random coefficients from a large finite field, the probability of getting linearly independent packets can be shown close to 1. This effectively eliminates the duplicate packets problem posed by using RS code.

In a real-world scenario, without any active injection of

redundancy into the network, a content will eventually disappear since peers will depart or the file will be deleted with some non-zero probability. Dimakis *et al.* [7, 8] proposed a way of restoring the missing redundancy from the remaining peers. Specifically, their work show that, using RNC for storage, when a new peer joins the network, it can download minimal amount of data from other peers to restore the missing redundancy. We will describe a similar data replenishment process in our proposed system.

3. THE P2P-VDN ARCHITECTURE

In this section, we describe the overall architecture of the proposed P2P-VDN to be used as a platform for video-on-demand applications. As mentioned in the Introduction, the P2P-VDN are built on three key designs: data dispersion, data replenishment, and path-diversity streaming protocol. These three designs correspond to the three main operational phases in the P2P-VDN. The data dispersion is used when a user publishes his video to the P2P-VDN, similar to the video uploading done by a YouTube user. The difference is that the video is broken in multiple pieces, coded and dispersed to many peers, rather than being stored in its entirety at a single YouTube server. The data replenishment is a continual data maintenance process that ensures high probability of a video being recoverable by a client when it is requested. This data replenishment is needed for the P2P-CDN since videos are stored at unreliable peers which enter and depart the network frequently. On the other hand, no such mechanism is necessary for YouTube since by assumption, its servers are reliable. The path diversity streaming protocol is used when a client requests a video from the P2P-VDN. The client might or might not belong to the P2P-VDN. In this case, multiple peers in the P2P-VDN will cooperate among each other via the path diversity streaming protocol, to simultaneously stream the requested video to the client. For YouTube, no special protocol is needed. Rather, TCP is sufficient to stream a video from a single YouTube server to the client. We now describe and motivate each of the three key designs.

3.1. Data Dispersion

One of the key differences between the P2P-VDN and the existing CDNs is not only the P2P-VDN employs participating peers rather than servers to store the videos, but in the way how a video is being stored at the peers. Akamai, for example, replicates videos in their entireties at multiple edge servers. This allows for proximity optimization, i.e., a nearest server to the client with the requested video in term of round trip time, will be chosen to stream that video to the client. Thus, it is necessary that the videos are replicated sufficiently at different servers. However, as discussed in Section 2 replicate videos is not a storage efficient method. Therefore, in the P2P-VDN, a video is broken up into multiple pieces. These pieces are coded using the RNC technique. The coded pieces are then dispersed randomly to multiple peers. We emphasize that data dispersion is good not only because it avoids the central point of failure, but also results in better load balancing.

Using the RNC technique, a video publisher first breaks a video stream into a number of segments. Each segment is further broken up into a number of packets. Each packet $\mathbf{p_j}$ can be viewed as a vector of elements belonging to a finite field. For example, if $G(2^8)$ is used, then each element of the finite field can be represented by an 8-bit pattern. Therefore, a packet of length $8L$ bits can be viewed as a vector of L elements from $G(2^8)$. A coded packet or equivalently, a vector of finite elements, $\mathbf{c_i}$ is produced by linearly combining k original packets as:

$$\mathbf{c_i} = \sum_{j=1}^{k} f_j \mathbf{p_j} \tag{1}$$

where f_j are the elements taken at random from the finite field F_q having q elements.

For every k original packets, the publisher generates $n > k$ coded packets. The value of n depends on the desired redundancy, the larger n, the more redundancy. The publisher also includes the information about the f_{ij} in the header of each of coded packets. Therefore, one should note that if a client has access to any of the k encoded packets $\mathbf{p_i}$'s that form a set of k linearly independent equations, then it will be able to recover the k original packets by solving a set of linearly independent equations. We can show that if the finite field is sufficiently large, the probability of having linearly independent equations is essentially 1. Furthermore, if the packet size is large, the overhead in storing the extra bits in the packet header for f_j is negligible.

Now, after generating these coded packets, the publisher then sends these packets at random to a number of peers. It is important to note that each peer does not need to store all k coded packets. Rather, to save space a peer may store only a fraction of a coded video packets. We also assume an existing infrastructure that enables the publisher to determine the IP addresses of a number of active peers in the network, so as to randomly select a number of these peers.

Next, in order for a client to be able to recover these pieces, we use a modified version of Chord for indexing these pieces. In particular, we use a mapping that hashes a searchable video title into a value, e.g. 64-bit value. Using Chord, the peer whose IP address is closest to this value, is used to store the set of IP addresses of the peers that were randomly picked for storing the coded video packets. Effectively, the client would be able to determine which of the peers that store the desired video by searching through the DHT in $\log_(N)$ where N is the number of participating peers. Since peers join and leave frequently, the DHT structure is updated accordingly as described in [2].

We note that, it is the case that, for some period of time, there may not be sufficient number of peers with the requested video in the network that enable the client to recover

the desired video. Thus, an automatic data replenishment in the P2P-VDN to counter the effect of peer departures is described below.

3.2. Data Replenishment

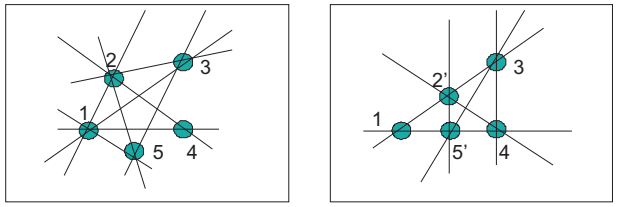
When a peer leaves the network, so does its data. This effectively reduces the robustness of the system temporarily or permanently if the peer never rejoins or rejoins without its data. To avoid this, a peer can transfer its data to some other peers before its departure. However, media data such as a video tends to be large, making a peer less willing to wait until the transfer completes. This process implies that, without any proactive data replenishment, the redundancy level of a video in the network is continuously reduced. At some point, the video is not likely to be recoverable.

Theoretically, if one is to replace the exact missing data in the network, the redundancy level would remain the same. However, a typical peer may not have the complete video that allows it to reproduce an arbitrary missing portion, nor does it know what the leaving peer has. Instead, we propose the following data replenishment scheme. Assume that the network knows the peer departure or failure rate. This can be estimated empirically, e.g., using some kind of periodic sampling. Thus, we can determine an appropriate replenishment rate to counter the information deletion rate. Because of limited space, in this paper, we do not focus on the replenishment rate. Rather, we describe how replenishment is performed.

Assuming that on average, when a peer leaves the network, there is at least one peer join the network within a short period, or there is at least one existing peer with spare storage capacity. This allows for another peer to take over the storage responsibility of the departed peer, to maintain the same redundancy of a piece of data. The new peer will randomly connect to a number of peers and download some fraction of their data. It then generates its new packets as some random linear combinations of the packets it obtains from other peers, in an attempt to maintain the same level of redundancy for the video.

This method is attractive for its distributed nature and scalability. Also, the original data is not needed to be present in the network. While there are other issues that need to be considered, in this paper, we focus on the analysis of data robustness in a network given such data replenishment scheme. In other words, given piece of data, what is the probability of being able to recover this piece of data as a function of the number of replenishments?

Clearly, the replenished data is linearly dependent on the data that are used to generate it. So if the number of packets used to generate a new packet is fewer than the number of packets in the original piece of the data and replenishments are repeated over time, then there may be a non-negligible chance that all peers contain relatively few independent data, making the network less robust. To illustrate this, consider using RNC on a video and dispersing the coded packets to five peers, each peer stores one coded packet. Assuming



(a) (b)

Fig. 1. a) Original robust state of the network; (b) Less robust state after peer departures and data replenishments.

that a client is able to recover the video if it obtains three or more independent packets. Furthermore, assuming that every time a peer leaves, two randomly chosen peers will send their packets to the replacement peer. The replacement peer then randomly combines the two packets to generate its own packet. Fig. 1(a) shows the geometric representation of the original robust state of the network where the circles represent packets. If there are three or more points lay on one line, they are co-linear. In other work, there are only two independent points on any line. In this state, if a client connects to any three peers, it will be able to recover the video, or geometrically, recover the entire plane. It is possible because none of the 3 points are co-linear. Now, suppose peers 2 and 5 depart the network. By chance, peers 1 and 3 are chosen to replenish data for peer 2, while 1 and 4 are chosen to replenish data for peer 5. As a result, the generated data are now 2' and 5'. 2' are linearly dependent on 1 and 3 while 5' are linearly dependent on 1 and 4 as shown in Fig.1(b). Now, if a client connects to peers 1, 2', and 3, it will not be able to recover the video since these points are co-linear. A similar situation arises when it connects to 1, 5', and 4. Effectively, the robustness on data recoverability in the network has been reduced. Eventually, this piece of data become unavailable when all the points lay on the same line.

This robustness, i.e., the recoverable probability of a video depends on how many peers are chosen to replenish the data and the amount of data each peer has. Figure 2 shows the simulation results on the recoverable probability as the network evolves under repeated replenishments. This simulation uses two peers for replenishment and the given parameters (a, b, c) is (total number of network coded packets, number of independent packets needed to recover a video, number of packets stored at each peers). As shown, the higher number of peers, the longer it takes before a file cannot be recovered. It is interesting to note that the robustness decreases quickly if the initial data redundancy is not sufficient. We have the following Propositions:

Proposition 3.1 *If the data replenishment is performed by having a new peer simply copy the data from two random peers, then the average number of replenishments before the data cannot be recovered is $O(n^2)$, where n is the number of participating peers during the replenishment process.*

Proposition 3.2 *If the data replenishment is performed fol-*

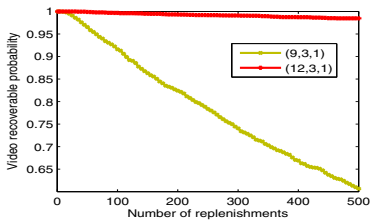


Fig. 2. Data recoverable probability as a function of replenishments.

lowing the RNC technique above using two peers to generate data, then the average number of replenishments before the data cannot be recovered is at least $O(2^n)$, where n is the number of participating peers during the replenishment process.

The proofs for the Propositions above can be found in the Appendix. As seen in the Propositions, the proposed replenishment mechanism is theoretically much better than the naive random duplication technique in terms of maintaining data redundancy for recovery.

3.3. Path-diversity Streaming Protocol

In this section, we discuss how distributed storage using RNC can help to facilitate path diversity streaming. 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 [9]. 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 describe the network coding scheme for multi-sender streaming framework that reduces the coordination among servers. In this scheme, a video stream F is randomly network coded and dispersed to a number of peers in the network. In this model, a stream is partitioned into N chunks. Each chunk is further divided into k small packets $\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_k}$. Now, for each chunk, the video publisher will randomly network code the packets within it, to produce a number of coded packets. These packets will be distributed to a number of peers at random as described in Section 3.1. Note that each peer does not need to keep all k coded packets. They may keep only a fraction of the coded packets, but each peer will have some coded packets from every chunk. Therefore, the total amount of storage of this scheme is small than other approaches.

To stream a video, the client connects to a number of peers with the fraction of the desired video. It first requests these peers to send their packets p_i's from the first chunk. After the client receives roughly k coded packets, it will be able to recover k original packets. It then immediately sends a request to the senders to signal them to start streaming the packets from the second chunk. 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 k independent packets. The attractive feature of this scheme is that no dynamic packet partition is required. All sending peers send packets 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. 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.

4. SIMULATION RESULTS

In this section, we investigate the performance of the proposed architecture by comparing the non-network coding (Non-NC) and random network coding (RNC) scheme. We assume that a video publisher distributes either uncoded or coded packets to a number of peers which are then responsible for streaming the video to a client. In this simulation, the video publisher has a video bit stream with a rate of 432kbps. The original stream is divided into a number of chunks of length 1 second. Thus each chunk consists of 36 packets of size 1500 bytes. The publisher distributes the packets to a number of peers using non-network coding and random network coding scheme. As the network evolves, periodic data-replenishment is performed to keep the P2P-VDN in the robust state. Next, TCP is employed to transmit data from these peers to a single client simultaneously. We use the finite field size of 2^8 for all the network coding operations.

We consider the following transmission protocol with Non-NC and RNC schemes. The protocol used in these

schemes are identical to the one described in Section 3.3.

Non-NC scheme. The packets are not coded. The publisher randomly pushes the uncoded packets to the peers. Each storage peer has some packets from every chunk.

RNC scheme. The publisher randomly generates a number of packets as linear combinations of 36 packets for each chunk, and distributes these packets to the peers. As a result, each storage peer keeps a fraction of coded packets which are pushed to the receiver randomly.

First, we characterize the probability of a receiver being able to decode a chunk as a function of storage. A chunk is decodable if all of its packets are recoverable. This implies that there are enough distinct coded packets for this chunk on the senders. Figure 3 shows the decodable probabilities when using different schemes. As seen, RNC scheme has the larger probability of recovering the video than that of the Non-NC scheme for the same redundancy level. RNC schemes can recover the chunk with a probability close to 1 with 0% redundancy, but the Non-NC scheme requires redundancy of almost 200% to accomplish the same goal. We now consider

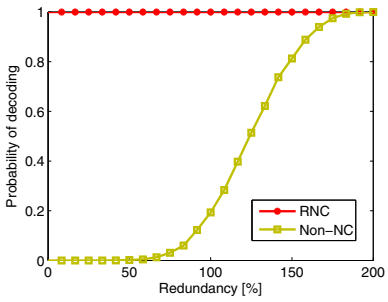


Fig. 3. The probability of a receiver being able to decode a chunk.

a scenario in which the sending peers collectively have total of 200% storage redundancy, thus a client will be able to recover the data if there is sufficient throughput between itself and the sending peers. To simulate this scenario, we use network simulator NS. The receiver connects to a number of peers to download the video. Heavy traffic between the senders 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 links between the senders 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 unit time. Figure 4 shows the average time before a client can decode a chunk when connecting to different number of peers. As seen, for the Non-NC scheme, the time to decode is higher due to the high probability of getting duplicated packets. It takes up to 60% more time to download a chunk with larger fluctuation.

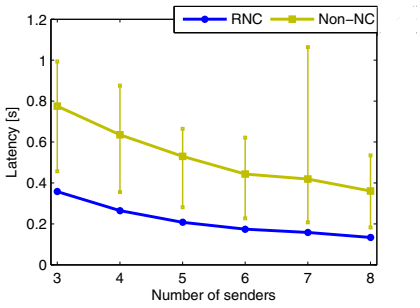


Fig. 4. Latencies to receive one chunk.

5. CONCLUSION

We have proposed a Peer-to-Peer Video Delivery Network that provides both performance improvement and scalability. It does so by employing data dispersion, data replenishment, a path diversity streaming protocol. Simulations demonstrate that under certain scenarios, the proposed P2P-VDN can result in bandwidth saving up to 60% over the traditional scheme.

6. REFERENCES

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *Proceedings of ACM SIGCOMM*, August 2001.

[2] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, September 2001.

[3] S. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1994.

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

[5] 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.

[6] K. Nguyen, T. Nguyen, and S. Cheung, "Video streaming with network coding," *Submitted to Springer Journal of Signal Processing*, January 2008.

[7] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized erasure codes for distributed networked storage," *IEEE Transactions on Information Theory*, June 2006.

[8] A. G. Dimakis, "Codes on graphs for distributed storage in wireless networks," M.S. thesis, University of California at Berkeley, 2005.

[9] T. Nguyen and A. Zakhor, "Distributed video streaming with forward error correction," in *Packet Video Workshop*, Pittsburg, PA, April 2002.

Proof of Proposition 3.1

Suppose a video is split into two input vectors (packets) $\mathbf{p}_1$ and $\mathbf{p}_2$, and there are n peers, each containing either $\mathbf{p}_1$ or $\mathbf{p}_2$. Whenever a peer leaves, new peer is picked at random out of $n-1$ existing peers, and the input vector (either $\mathbf{p}_1$ or $\mathbf{p}_2$) is copied to the

new peer. The process is of birth-and-death type on $\{0, 1, \ldots, n\}$ with two absorbing states, 0 and n. We would like to estimate the mean absorption time.

In the above birth-and-death process the forward probabilities are given by

$$p_k = \frac{k(n-k)}{n(n-1)} \quad \text{for } k = 1, 2, \ldots n, \quad \text{and } p_0 = 0$$

and the backward probabilities are

$$q_k = \frac{k(n-k)}{n(n-1)} \quad \text{for } k = 0, 1, \ldots n-1, \quad \text{and } q_n = 0\ .$$

The expected hitting time $h(k) = E_k[T_0 \wedge T_n]$ solves the following recurrence equation: For $k = 1, 2, \ldots, n-1$,

$$\begin{aligned} h(k) &= 1 + \frac{k(n-k)}{n(n-1)} h(k-1) + \frac{k(n-k)}{n(n-1)} h(k+1) \\ &+ \left(1 - \frac{2k(n-k)}{n(n-1)}\right) h(k) \\ h(0) &= h(n) = 0 \end{aligned} \tag{2}$$

The above equation can be rewritten as follows

$$y_k = -(n-1)\left(\frac{1}{k} + \frac{1}{n-k}\right) + y_{k-1},$$

where $y_k = h(k+1) - h(k)$.

Thus

$$\begin{aligned} y_k &= -(n-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right) \\ &- (n-1)\left(\frac{1}{n-k} + \cdots + \frac{1}{n-1}\right) + y_0. \end{aligned} \tag{3}$$

Now, by symmetry,

$$-y_0 = y_{n-1} = -2(n-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{n-1}\right) + y_0$$

and therefore

$$y_0 = (n-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{n-1}\right). \tag{4}$$

Plugging y_0 into (3), and after some algebraic manipulations, we obtain:

$$\begin{aligned} h(k+1) = (n-1)k &\left(1 + \frac{1}{2} + \cdots + \frac{1}{n-1}\right) \\ -(n-1)(k+1) &\left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right) \\ +(n-1)(n-k-1) &\left(\frac{1}{n-k} + \cdots + \frac{1}{n-1}\right). \end{aligned} \tag{5}$$

Taking $k + 1 = \frac{n}{2}$, we obtain

$$h\left(\frac{n}{2}\right) \approx \ln 2 \cdot n^2 \tag{6}$$

Proof of Proposition 3.2

A video is split into three input vectors $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$. Each peer contains a different linear combination of the three input vectors. Each time a peer is replaced, it picks two random peers and takes a random linear combination of the two vectors. The question is, how long will it take before the rank of $n \times 3$ matrix of coefficients falls below three?

We model the process by splitting the peers into cliques, each having rank two. There are originally $\binom{n}{2}$ cliques. Observe that a clique can be absorbed by another clique only when it has two peers in it. It might reappear later. The state space is therefore $\{absorbed,\ 2, 3, \ldots, n\}$. In fact a clique is a birth and death process on the above state space with the following forward and backward probabilities, given there are k peers in a clique:

$$p_k = \frac{n-k}{n} \cdot \frac{\binom{k}{2}}{\binom{n-1}{2}} = \frac{k(n-k)(k-1)}{n(n-1)(n-2)} \tag{7}$$

and

$$q_k = \frac{k}{n} \cdot \left[1 - \frac{\binom{k-1}{2}}{\binom{n-1}{2}}\right] = \frac{k(n-k)(n+k-3)}{n(n-1)(n-2)} \tag{8}$$

Let $\phi(k)$ be the associated probability harmonic function:

$$\phi(k) = 1 + \sum_{m=2}^{k-1} \frac{q_2 \cdots q_m}{p_2 \cdots p_m} \quad k = 2, 3, \ldots, n \tag{9}$$

Plugging in, we obtain

$$\begin{aligned} \phi(k) &= 1 + \sum_{m=2}^{k-1} \binom{n+m-3}{n-2} \\ &= \sum_{j=0}^{k-2} \binom{n-2+j}{n-2} = \binom{n+k-3}{n-1} \end{aligned} \tag{10}$$

via basic combinatorics.

If we denote k_t is the size of a given clique at time t, then $\phi(k_t)$ is a martingale, and by the Stopping Theorem, the probability that the clique, starting with three peers expands to all n peers while never shrinking to two is

$$\frac{\phi(3) - \phi(2)}{\phi(n) - \phi(2)} = \frac{n-1}{\binom{2n-3}{n-1} - 1}$$

Thus, it will take an average of

$$\frac{\binom{2n-3}{n-1} - 1}{n-1}$$

trials for a clique to start growing, and reach the size of n before shrinking to 2. Here, by Stirling's formula,

$$\frac{\binom{2n-3}{n-1} - 1}{n-1} \sim \frac{1}{\sqrt{\pi} n^{3/2}} 2^{2n}$$

Since there are no more than $\binom{n}{2}$ cliques at one time, it will take an average of at least

$$\binom{n}{2}^{-1} \cdot \frac{\binom{2n-3}{n-1} - 1}{n-1} \sim \frac{1}{\sqrt{\pi} n^{7/2}} 2^{2n-2} \tag{11}$$

replenishments before the video is no longer recoverable.