Streaming Stored Playback Video Over A Peer-to-Peer Network

K. Kalapriya
CAD Lab, SERC
Indian Institute of Science
Bangalore, India-560 012
Email: kalapriya@cadl.iisc.ernet.in

R. Venkatesh Babu
Centre for Quantifiable QoS in Communication Systems
Norwegian University of Science and Technology
Trondheim, Norway
Email: venkat@Q2S.ntnu.no

S. K. Nandy
CAD Lab, SERC
Indian Institute of Science
Bangalore, India-560 012
Email: nandy@cadl.iisc.ernet.in

Abstract— **Streaming live video over peers in the Internet is gaining popularity since it has the advantage of reducing the load on the server and enable the server to perform other specialized services more effectively. Further it reduces the server bandwidth and is therefore no longer a limiting factor for the number of clients served. In this paper we propose a architecture for streaming stored video over peer-to-peer network. While a given peer continues to receive segments of the stream, simultaneously caching these segments locally renders this peer to in turn act as a source for other peers. These peers are highly transient in nature and the exit of a peer from the network results in loss of video segments. We overcome this in our architecture by exploiting the inherent redundancy of FEC channel coding of video streams. Through simulations we establish the novelty of our architecture and show that our proposed solution incurs minimum overhead on the peers and does not increase the playback latency much more than the jitter buffer latency.**

I. Introduction

Applications like Internet video broadcasts, corporate telecast, distance learning etc., requires transmission of streaming video to multiple users over the Internet simultaneously. Streaming video uses a small buffer at the end-system that allows viewing of video while simultaneously downloading segments of the video into the buffer. Streaming of video helps to reduce the startup time of playback video as opposed to the traditional download and play system. Streaming video can be further differentiated as 'Streaming Live Video' and 'Streaming Stored Playback video'. In Live streaming video, the clients receive the streams from the time of request whereas in stored playback streaming the server starts a new stream for every request.

Moreover, video streaming is a bandwidth intensive task and is likely to exhaust the available bandwidth at the source while satisfying a meager set of clients. Such systems can benefit much from highly available network resources at the end system/clients. These entities called **peers** co-operate to share storage and bandwidth. End Systems while receiving these streams act as routable components and distribute streams to other end systems.

Streaming playback video, which aims at minimizing the startup delay has been addressed in the past [1][2][3]. Padmanabhan *et al.,* [4] uses a scheme of co-operative networking for downloading video. This is the traditional 'download and play' in which the startup delay is equal to the time taken to download the video.

Significant work has been done to extend application layer multicast to streaming live video [5][6][7]. The proposed solutions use the peer-to- peer architecture to stream video. The end-systems form a network of arbitrary topology referred to as ad-hoc networks and contribute resources such as storage and bandwidth to the community. Extending application layer multicast to streaming stored playback video has not been addressed in the past. We address the problem of extending application layer multicast using peer-to-peer architecture for streaming stored video for flash crowds. We provide architecture considerations for storing the initial segments among the peers to mimic a request of stored streaming video.

One of the main issues of such systems is the transient nature of the peers that cache the initial segments of video stream. This introduces additional constraint from the traditional streaming video systems that use peer-to-peer architecture to distribute media. Transience of peers is specifically considered in [8] for live streaming video. Proposed solution consists of a peering layer to handle the transience of peers. They indicate that under homogeneous unicast edge and node characteristics, an almost-complete spanning tree is the optimal overlay tree for packet loss, packet delay and time to first packet metrics. In this paper we propose a technique to reduce the packet loss during the transience of peers by exploiting the inherent redundancy of channel coding.

In this paper we propose a distributed and temporal caching scheme of the initial segments of the stream among the peers. This helps to maintain the initial segments of a playback video after which the stream can be treated as near live stream. Transience of peers can lead to loss of cached segments leading to non-availability of continuous video segments and hence form a 'gap'. It also contributes to break in connectivity to the stream due to loss of segments. A new stream will be requested whenever a gap is formed as opposed to live video streaming. This problem can be overcome by exploiting redundancy introduced in the video segments. Our proposed scheme imposes minimum overhead on end-systems in terms of buffer requirements, caching and protocol overhead.

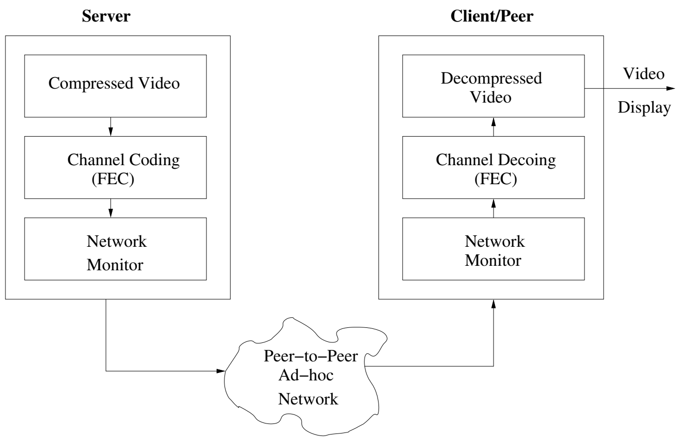


Fig. 1. Architecture for Streaming Stored Video

II. Architecture for Streaming Stored Video System on Peer-to-Peer network

In streaming stored video, the server will serve every individual request as a new stream. In such systems the number of streams served is directly proportional to the bandwidth available at the server. Network resources like bandwidth and storage available in abundance at the end systems can be used to overcome the problem of bandwidth saturation at the server. These end-systems or peers while receiving the stream also help in distributing the stream to other peers. Transmitting video over the peers is challenging due to the following reasons, firstly peers do not exhibit server like characteristics. Secondly they are highly transient in nature. The transience of peers will lead to loss of video packets.

Figure 1 shows a typical architecture for streaming stored video over peers. For Internet applications, channel coding is typically used in terms of block codes. Here a compressed video stream is first chopped into segments, each of which is packetized into k packets to generate an n-packet block, where $n > k$. To perfectly recover a segment, a user only needs to receive any k packets in n-packet block. On a client request, the packetized video segments are transported over the network formed by the peers. Such a network formed by the peers is called as overlay network. For the rest of the paper, all references to network is to be viewed in terms of overlay network. Three components are identified that needs significant attention for streaming stored video (i) caching (initial segments) by the peers in the networks (ii) Error resilient encoder (iii) Buffer requirement at the end-system/peer. Caching of initial segments helps to mimic a new request to the stream. Error resilience encoding enhances robustness of compressed video to packet loss.

A. Distributed caching among Peer

Transmission of video segments is modeled as a discrete time model (T) at the granularity of video packets. Every peer receiving the stream contributes buffer of size C_{psize} in accordance with the availability of its memory. It can also serve the stream it receives to other peers. We define two sets of peers, those that participate as transit node for video stream called as serving peer and those that contribute buffer for temporary caching called as helper peers. When a helper peer exits it leads to formation of 'gap' due to the non-availability of continuous cache segments. When a serving peer exits it leads to loss of connectivity to the streams for a receiving peer due to loss of video segments. A serving peer can be helping peer. We adhere to the following notation in our analytical study of streaming stored video using peer-to-peer architecture.

- C_{tot}: Total amount of cache buffer/blocks available
- C_{curr} : Total amount of cache buffer/blocks currently available
- C_{psize} : cache size contributed by each peer
- S_{tot} : Total number of individual streams originating from the server
- S_{cache}: Total number of streams requested when caching scheme is used
- T_{req} : Request /Arrival Time of a request, or joining time of a peer
- T_{cmax} : Total time units equivalent to cached blocks.
- T_{dur} : Total duration of the stream.

On arrival of a peer, the total cache (C_{tot}) increases by the size of the cache contributed by the peer and is expressed as

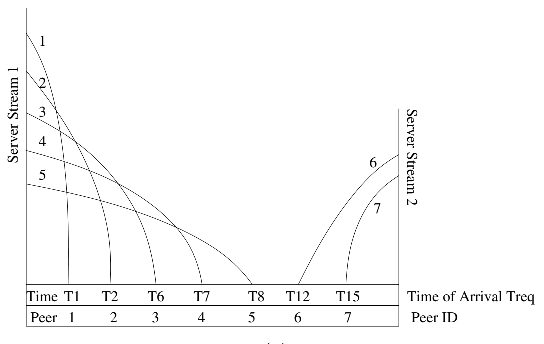
$$C_{tot} = C_{curr} + C_{psize}$$

Without loss of generality C_{tot} can be mapped to the discrete time scale T as T_{cmax}. We use T_{cmax} and C_{tot} interchangeable and its meaning will be implicit in the context it is being used.

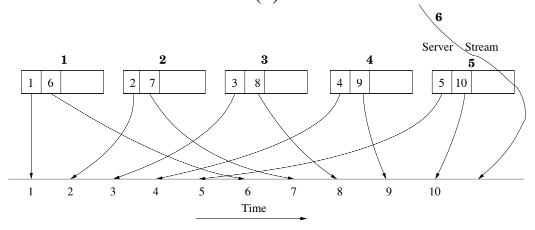
These cache blocks (C_{tot}) are used to store the initial segments of video up to time given by C_{tot}. Request arriving at time T_{req} prior to T_{cmax} can be served by same stream after obtaining the initial segments up to T_{req} from the peers that had cached them. This leads to the formation of cluster consisting of a set of peers that had cached the initial segments and a stream serving the peers.

A new stream is requested when either $T_{req} > T_{cmax}$ or exit of one of the caching peer that contributed significantly. Request for a new stream creates a new cluster. Our intention is to find how many such clusters can be formed expressed as S_{cache}/S_{tot} and the average size of the cluster. It should be noted that the cached blocks remain until all the peers in that cluster have downloaded the segments and do not stay throughout the duration of the stream.

Figure 2(a) gives a logical view of peers being served by a stream. The total available cache blocks (C_{tot}) is 8. Peers 1,2,3 and 4 at intervals T1, T2, T6, T7, T8 are served by a single server stream (server stream 1) forming a cluster. A 'gap' is formed between the intervals 8 and 12. Peers 6,7 arriving at time T12 and T15 is served by a new server stream (server stream 2) and forms a new cluster. Figure 2(b) shows detailed view of fetching the initial segments from the peers and being served by the stream. Peers 1,2,3,4 and 5, has cached the segments up to T10. Peer 6 arriving at T10 could fetch the initial segments from the peers and receive the stream continuously. This requires extra buffer on the peer to receive



(a)



(b)

Fig. 2. (a) Video stream from the server serving multiple peers (b) Initial segments of video being served by peers to a single request at time T10

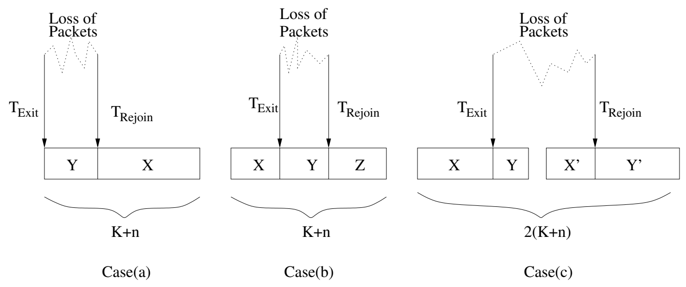


Fig. 3. Transience of peers and loss of video segment

the continuous stream while downloading initial segments of the stream. A cluster is said to be in stabilized state when a gap is formed *i.e.,* the same stream can serve no more peers.

A helper peer or a server peer can exit abruptly and may form a 'gap'. For a helper peer, C_{psize}/C_{tot} gives the contribution of the peer in terms of cached segments. This leads to the formation of gap with probability.

$$P_{gap} = \frac{C_{psize}}{C_{tot}} \tag{1}$$

where P_{gap} is the probability of formation of the gap. This implies that probability of formation of 'gap' is proportional to the contribution of peer to the total cache size. This will lead to a new 'stream request' to the server for fetching the subsequent peers who request the same stream.

On exit of a serving peer, the peers that were receiving the stream from that serving peer find another equivalent peer and continue to receive the stream. It takes time for the receiving peer to find an equivalent peer. During this transience there will be loss of video segments. In the next section we propose a redundancy technique to overcome this problem.

B. Redundancy Requirement on exit of a serving peer

We exploit the FEC (Forward Error Code) channel coding technique to introduce redundancy of video packet in peers. During transience of peers this redundancy can be used to reconstruct the lost frames. Let n denote the number of redundant segments required, n_{pexit} the number of helper peers that exit, k the number of segments required to decode a video segment, n_{peer} the number of helper peers

In case of exit of serving peer, n is bounded by the time taken to repair the partition created by the transient peer.

$$n = C(T_{exit} - T_{rejoin}) \tag{2}$$

where $(T_{exit} - T_{rejoin})$ gives the time taken to repair the partition and $C(T_{exit} - T_{rejoin})$ is simply the mapping of discrete time intervals to video segments.

Figure 3 shows the various cases for loss of segments during the transience of peers.

Case (a) : Consider a peer exiting while transmitting the start of video segments containing $(k+n)$ packets. Let Y be the number of packets lost due to the transience of peers and X packets received after repair of the partition. We know that the total number of packets transmitted in a group is $k+n$.

Therefore, $X+Y = k+n$. To decode the group it has to receive at least k packets. This implies that $X = k; Y = n$.

Case (b) : Consider a peer exiting during the middle of transmitting the $k+n$ packets. Let X and Z packets be the number of segments received before and after the repair of partition. We know that the total number of segments transmitted in a group is $k+n$.

Therefore, $X+Y+Z = k+n$. To decode the group it has to receive at least k segments. This implies that $X+Z = k; Y = n$

Case (c) : Consider a peer exiting during the transition between two contiguous segments. Let X and Y represent the received and lost packets of the first group and X' and Y' denote the lost and received packets of the second group respectively.

$X+Y = k+n$
$X'+Y' = k+n$
$X+Y+X'+Y' = 2(k+n)$

Using case (a) and case (b) it can be shown that $X'+Y = n$

In all the three cases it can be seen that the total number of packets lost during the process of repairing the partition is n. In other words n should capture the time taken to repair the partition and is given by equation (2). This implies peer discovery protocols have to be designed so as to minimize the time taken to find an equivalent peer.

C. Redundancy Requirement on exit of a helper peer

In case of exit of helping peer, n is bounded by the

$$n <= n_{pexit}.(k/n_{peer}) \tag{3}$$

to avoid forming a gap.

In (3) when $k = n_{peer}, n = n_{pexit}$
when $k < n_{peer}, n < n_{pexit}$
when $k > n_{peer}, n$ is bounded by (k/n_{peer}).

This implies that k plays an important role in determining the cluster size (number of peers that form the cluster). Large value of k increases the probability of forming a gap and reduces the size of the cluster.

Combining (2) and (3) we obtain,

$$n = max\left[C(T_{exit} - T_{rejoin}), (n_{pexit} * (k/n_{peer}))\right] \qquad (4)$$

Thus adding redundancy equivalent to n given by the above expression decreases the formation of gaps, thereby reducing the number of new stream request to the server. Consequently in (1) if

$$C_{psize} < n, P_{gap} \approx 0. \qquad (5)$$

This implies that when a peer has contributed cache blocks less than the size of n, the probability of formation of gap is nearly zero. Intuitively it can be seen that larger the value of n, i.e., more redundancy is added to the stream, less is the probability of formation of gap. But increasing 'n' increases the bandwidth requirement by a factor of n/k limiting the value of n to the bandwidth available.

To maintain connectivity while downloading the initial segments of the stream, we provide extra buffer at the peer to receive the continuously flowing stream. This is discussed in detail in the next subsection.

D. *Buffer feasibility on the Peer*

This scheme requires additional buffer to receive the already flowing stream while downloading the initial segments cached at the peers. This buffer is in addition to the jitter buffer maintained by the peers to reduce the variation in the available bandwidth. This additional buffer acts as a repository for video segments till all the initial segments are downloaded.

Below we give minimum and maximum bounds on this buffer size. Let B_s be the buffer of size s required to store the flowing stream. Let T_{start} be the time the stream started. A request at time $T_{req} < T_{cmax}$ fetches the initial segments from the peers. The amount of initial segments is given by $T_{req} - T_{start}$. Buffer size B_s is required for the duration of the time it takes to download the initial segments.

$$B_s = Buffer(T_{req} - T_{start}). \qquad (6)$$

Total peer buffer to receive and display video is $B_s + B_j = Buffer(T_{req} - T_{start}) + B_j$.

In (6) when $T_{req} = T_{start}$, $B_s = 0$ gives the lower bound on the buffer size B_s and when $T_{req} = T_{cmax}$ gives the upper bound on the buffer size B_s.

Jitter buffer can be used to download the initial frames while simultaneously receiving the flowing stream in this additional buffer. Without loss of generality it can be assumed that it takes the same time to download an initial segment and a flowing segment. Once the jitter buffer is full the decoding process can be started and the jitter buffer can be reused to download further cached segments if any. After all the initial segments have been downloaded, the data can be taken from the additional buffer. The startup delay is always bounded by the size of the jitter buffer. The additional buffer complements the jitter buffer in reducing the variations of available bandwidth.

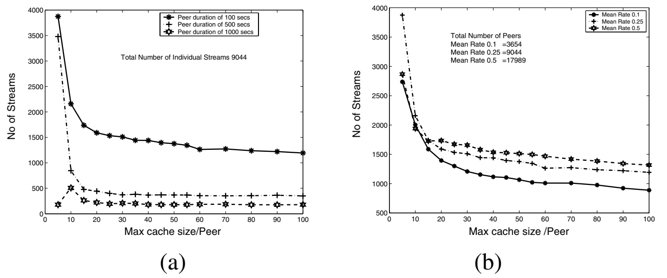


Fig. 4. (a) Streams requested by the peers for different request rates (b) streams from the server for different duration of peers

III. Simulation Results

We use discrete time based simulation for studying effects of varying cache sizes and duration of the peers. Unless otherwise stated we assume that the server has infinite resources for the purpose of simulation. We measure the following performance parameters: i) The number of satisfied request ii) The load of the server iii) Buffer (B_s)at the peer to maintain connectivity.

To simulate a flash crowd we had considered high request rates about 9 requests/minute, 15 requests/minute and 30 requests/minutes. To consider the transient nature of peers we simulated peer groups of different durations or (100secs, 500secs and 1000secs), lifetime of the peers. Data sets were collected for varying cache size contributed by the peers.

Figures 4(a) and 4(b) show the number of streams requested from the server for different duration of peers and mean rates respectively. It can be observed that even when contribution of client is small (orders of 5 to 20 units) the number of streams requested is reduced by 65 % approximately. Also the number of requests to the server falls steeply when the client cache size is increased from 5 to 20 after which the number of requests remains approximately constant. This is due to the fact that high contribution of cache by the peer penalizes the scheme on exit of such peers by forming gaps. It can be seen that longer the number the duration of peers, less is the number of streams requested (Fig. 4(a)) and when the inter-arrival time is small the number of streams requested is less (Fig. 4(b)).

Simulations were carried out to find the contribution of insufficient cache blocks to cache initial segments and penalty incurred by the exit of a peer that has contributed to caching. Both the situations lead to formation of gap , thereby increasing the number of requests to the server.

Figures 5(a) and 5(b) show the number of streams requested due to insufficient blocks to cache the initial segments. A small increase in cache size from 5 to 10 (≤ 20) reduces formation of gaps significantly (about 33% reduction in the number of streams requested). It can be seen that increase in cache size leads (more availability of cache to store initial segments) to less number of requests to the server.

Graphs6(a) and 6(b)plots the penalty incurred due to exit of a peer that has contributed significantly for caching. It is

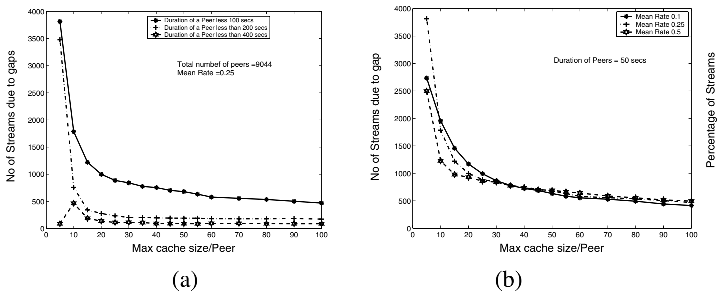


Fig. 5. (a) Total number of streams requested due to insufficient cache size for all durations (b) Total number of streams requested due to insufficient cache size for all arrivals.

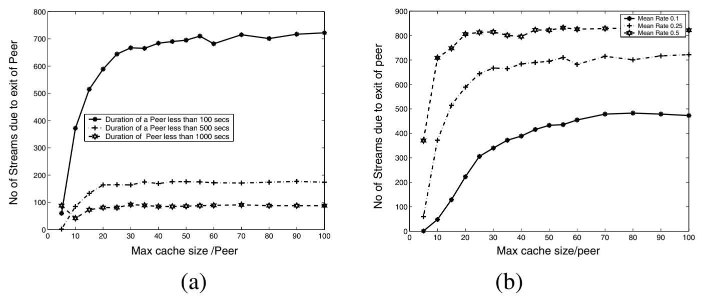


Fig. 6. Cache size per client Versus No. of streams due to exit of peer (a) for different duration of peers (b) for different arrival rates

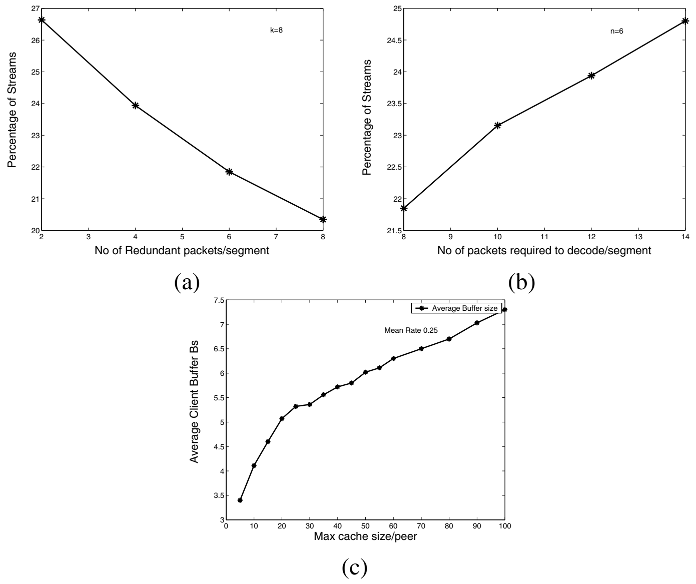


Fig. 7. No. of streams due to (a) the effect of k and (b) effect of n (c) Buffer Requirement for the peer to maintain connectivity

observed that higher the contribution of the peer, greater is the penalty incurred on exit of such peer(more requests to the server).

Figures 7(a) and 7(b) show the effect of increasing values of $'k$ and $'n'$. It can be seen that increasing k increases the number of requests to the server, since any loss in k packets would lead to a new request. This implies that k gives a bound on the cluster size *i.e.,* number of peers in the cluster. Increase in n (Fig. 7(b)) reduces the number of streams requested from the server.

Figure 7(c) shows the buffer requirement at the peer to capture the flowing stream while downloading the initial segments for various cache sizes. It can be seen that the average size of the buffer is about 5 units (of about few KBs).

From these results presented above, it can be concluded that contribution of small size ($\leq$ 20 units) of cache by peers leads to reduction in number of streams from the server while serving large number peers. This leads to conservation of bandwidth at the server where it is limited. Our scheme also incurs minimum overhead in terms of buffer requirements (about 5 units) on the peers to maintain connectivity .

IV. Conclusions and future Work

We have considered the issue of streaming stored playback video and differentiate it from live streaming video by the requirement of storing the initial segments. Peer-to-peer architecture helps to effectively use the end system resources. Initial segments are temporarily cached among a set of peers who share their storage and form a network. It can be shown that with minimum overhead (about 5 units of buffer) on the end-system the number of individual stream originating from the server can be significantly reduced. This helps to reduce the bandwidth over-utilization near the server and increasing the overall throughput of the system (*i.e.,* increased number of peers served). Further work can be continued with formation of clusters of similar requirements in terms of rate and quality video. More over techniques like embedded coding can be looked upon to satisfy each cluster of varying requirements with a single stream.

References

[1] D. Wu, Y. Hou, Y. Zhang, "Transporting real-time video over the Internet: Challenges and approaches," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1855-1875, December 2000.

[2] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M. Peha. "Streaming Video over the Internet: Approaches and Directions", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 282-300, March 2001.

[3] R. Rejaie, M. Handley, and D. Estrin, "Architectural Considerations for Playback of Quality Adaptive Video over the Internet," *Tech. Rep.*, USC, Nov, 1998.

[4] V. N. Padmanabhan, H. J. Wang , P. A. Chou, and K. Sripanikulchai, "Distributing streaming media content using cooperative networking", in *ACM NOSSDAV-2002*, pp. 177-186, May, 2002.

[5] H. Deshpande, M. Bawa, and H. Gracia-Molina, "Streaming Live Media over a Peer-to-Peer Network", *Tech.Rep. CS-2001-26*, CS Dept., Stanford University, 2001.

[6] Duc A. Tran, Kien Hua and Tai Do, "ZIGZAG: an efficient peer-to-peer scheme for media streaming", *IEEE INFOCOM*, vol. 2, pp. 1283 - 1292, 2003.

[7] D. Xu, M. Hefeeda, S. Hambrush and B. Bhargava, "On Peer-to-Peer Media Streaming", in *Proc. of International Conference on Distributed Computing Systems (ICDCS'02)*, pp. 363-371, July 2002.

[8] Mayank Bawa, Hrishikesh Deshpande, and Hector Garcia-Molina, "Transience of Peers and Streaming Media," in *Proc. of HotNets-I*, October 2002.