COCONET: Co-Operative Cache driven Overlay NETwork for p2p vod streaming

Abhishek Bhattacharya, Zhenyu Yang, and Deng Pan

Florida International University, Miami FL 33199, USA,
{abhat002,yangz,pand}@cs.fiu.edu

Abstract. Peer-to-Peer (P2P) approaches are gaining increasing popularity for video streaming applications due to their potential for Internet-level scalability. P2P VoD (Video On-Demand) applications pose more technical challenges than P2P live streaming since the peers are less synchronized over time as the playing position varies widely across the total video length along with the requirement to support VCR operations such as random seek, Fast-Forward and Backward (FF/FB). We propose *COCONET* in this paper, which uses a distributed cache partly contributed by each participant thereby achieving a random content distribution pattern independent of the playing position. It also achieves an $O(1)$ search efficiency for any random seek and FF/FB operation to any video segment across the total video stream with very low maintenance cost through any streaming overlay size. Performance evaluation by simulation indicates the effectiveness of *COCONET* to support our claim.

Key words: peer-to-peer, Video On-Demand, streaming, overlay network, co-operative cache

1 Introduction

Today's Internet provides a powerful substrate for real-time media distribution due to the widespread proliferation of inexpensive broadband connections which makes live streaming and on-demand media applications more important and challenging. As mentioned in [1], YouTube has about 20 million views a day with a total viewing time of over 10,000 years till date which clearly makes VoD streaming to be one of the most compelling Internet applications. P2P approach of content distribution has already being proved to be useful and popular for file sharing and live streaming systems with a plethora of applications found in the Internet. P2P based design can achieve significant savings in server bandwidth for VoD systems also, as stated in [1]. But, unlike live streaming applications, very few P2P VoD systems have being implemented and successfully deployed over the Internet. In order to alleviate server load, the state-of-art P2P VoD systems allow peers exchange video blocks among each other having overlapped playing positions. In a VoD session, the users watching the same video may well be playing different parts of the stream, and may issue VCR commands at will to jump to a new playback position leading to fundamentally lower levels of content overlap among the peers and higher need for frequently searching new supplying peers. It is observed that efficient neighbor lookup is important for supporting VCR operations. [9] presents a detailed analysis on a large scale P2P VoD system and enumerates the major design challenges. Among these, the fundamental challenge in designing a P2P VoD system lies in offering VCR operations such as random seek/FF/FB which require greater control over the coordination of peers. This is very unlike live streaming where the users are always in the same playing position and have no VCR related operations.

We propose *COCONET*, a novel and efficient way of organizing peers to form an overlay network for supporting efficient streaming and neighbor lookup for continuous playback or FF/FB VCR operations. *COCONET* utilizes a co-operative cache based technique where each peer contributes a certain amount of storage to the system in return for receiving video blocks. *COCONET* uses this co-operative cache to organize the overlay network and serve peer requests, thereby reducing the server bottleneck supporting VCR related operations. In current P2P VoD systems, peers share video segments only with nearby (forward/backward) neighbors based on its playing position [1]. We envision a problem with this type of content distribution scheme. In highly skewed viewing patterns, most of the peers are clustered around a particular playing position and very few peers are scattered at different positions throughout the video length, thereby the peers may not find any or very few neighbors to satisfy their demand. *COCONET* avoids this situation and is able to serve peers that are at random playing positions which are not at all related to the sender's playing position.

In order to find new supplier peers at different parts of the movie length, P2P VoD systems need to maintain an updated index of the live peers with their available video segments. Currently most deployed P2P VoD systems rely on centralized trackers for maintaining the index. This mechanism imposes a huge query load on the tracker with the expansion of the system. *COCONET* does not use indexing at the tracker. Instead, the tracker only maintains a small subset of live peers which is queried only once as a rendezvous point when a new peer joins the system. Each *COCONET* peer builds an index based on the co-operative cache contents which helps to find any supplier peer for any video segment throghout the enitre video length.

Our main contributions in this paper are: (1) We are able to achieve a search efficiency of $O(1)$ during continuous playback or random seek to any position across the entire video stream with a high probability; (2) The control overhead of *COCONET* is also low to maintain the overlay structure upon any peer dynamics when the system is being subjected to heavy churn and moreover it has better load balancing and fault tolerance properties; and (3) One of the attractive features of *COCONET* is a distributed contributory storage caching scheme which helps to spread the query load uniformly through the overlay and organizes the overlay in a uniform and randomized fashion which makes the content distribution independent from playing position.

The remainder of this paper is organized as follows. In Section 2 we survey related work from the literature. Section 3 presents the preliminary design structure and key ideas. In Section 4 we discuss the detailed protocols in *COCONET* and analyze their performance. Section 5 presents performance evaluation of *COCONET* using simulations and we conclude in Section 6.

2 Related Work

The most studied overlay design for video streaming in the literature is tree and mesh structure. Over the last decade, many proposals have been put forward such as P2Cast [13] and oStream [11] where the basic stream is provided by an application layer multicast tree which searches for appropriate patching streams from the root peer. Similar to the tree-based schemes, P2VoD [3] organizes nodes into multi-level clusters according to their joining times. The major problem with these kind of overlays is its difficulty to maintain a consistent structure which is vulnerable in a highly dynamic environment typical of P2P based VoD systems. Multi-tree approach was proposed by SplitStream [12] where the video stream is divided into multiple sub-streams using coding techniques. One stream is sent through each tree which achieves better resilience to tolerate failures since even if one tree fails, the peers will continue to receive video blocks through the other trees with a possibly degraded quality.

PROMISE [4] uses mesh-based overlays for streaming. Although they improve bandwidth utilization by fetching data from multiple peers, supporting VCR operations in VoD service such as random seek is not easy since it is difficult to have a neighbor lookup mechanism to locate supplier nodes. Mesh-based overlays are useful for distributing the content but is not so effective for searching which is one important criterion for supporting VCR operations in a VoD system as indicated in [2]..

A dynamic skip list based overlay network is proposed in [2], where all the peers are connected sequentially according to their playback progress at the base layer of the skip list and each peer may also randomly connect to a few adjacent peers on the higher layers. The lookup efficiency is shown to be $O(\log N)$ where N is the total number of peers. A ring assisted overlay is proposed in [5], where each peer maintains a set of concentric rings with different radii and places neighbors on the ring based on the similarity with their cached content with a search complexity of $O(\log(T/w)))$ where T is the video size and w is the buffer size. Many DHT based approaches have also been proposed such as [6], but a DHT lookup takes logarithmic messaging complexity with respect to the number of peers in the system. With playback progress, cached blocks are frequently flushed off from the buffer which will cost a DHT update and this will incur a lot of overhead in the long run. InstantLeap [10] proposes a hierarchical overlay network which is based on the playing position of the peer. Peers are divided into a number of groups where each peer belongs to one group at a time and maintains limited membership information of all the other groups by exchanging random messages which helps to perform any random seek operation in $O(1)$ messaging overhead. *COCONET* also achieves a $O(1)$ lookup complexity but with reduced protocol overhead than [10]. This can be observed from the fact that [10] builds its index based on each peer's playing position i.e. available segments in the playing buffer. So whenever any peer changes its playing position due to continuous playback/leap and moves from one group to another, the index needs to be updated which involves a lot of messaging overhead. In contrast, *COCONET* is completely independent of playing position and builds its index based on the stable storage buffer which is kept unchanged as long as the peer is in the system. Thus, *COCONET* completely avoids the costly and frequent index update operation as the peers change their playing position and also helps for better segment availability of the entire system.

3 Design Principle

We present the basic system model in this section. We have N peers in the system and a Video Server S which stores the entire video with an upload capacity of S_u Mbps. A centralized tracker T maintains s_T, a set of t live peers in the system and is updated with a periodicity of t_T. The video is divided into M segments and the play time for each segment is t_M with a data rate of D. Each peer contributes a part of storage space to the system which is defined as the *storage buffer* (*aka. storage cache* or *co-operative cache*) with a size of b segments. For the sake of simplicity, we assume segment level granularity at all the levels of *COCONET*. Each peer also has a *playing buffer* of size k segments which contains the current playing segment and a few consecutive segments for supporting continuous playback and pre-fetch. The entire system parameters are listed in Table 1 for easy reference, with some of them explained in later sections. The system architecture diagram of a *COCONET* node is shown in Figure: 1.

3.1 Membership Management

Gossip-based algorithms have recently become popular solutions for message dissemination in large-scale P2P systems. In a typical gossip-based scheme, each peer sends a message to a set of randomly selected nodes which in turn repeat the process in the next round. The gossiping continues until the message has reached everyone. The inherent random property of gossip helps it to achieve resilience to failures and enable decentralized operations but at the cost of redundancy. The two most important knobs of gossip are: *fan-out* and *ttl*. Fan-out is the number of gossip partners maintained by each peer and ttl is the number of gossip rounds before the message is discarded. As stated in [7], for gossip to be successful the partner list should be a randomized subset of the entire population and should be refreshed before each gossip round. In *COCONET*, the membership is managed by disseminating join messages randomly to a set of peers which in turn forward to some other peers or keep it with a certain probability, thereby keeping the overlay connected. We employ this mechanism for membership management

Table 1. List of System Parameters

Definition	Notation
Number of Peers	N
Video Server	S
Tracker	T
Tracker Size	s_T
Number of total Video Segments	M
Play time of one video segment	t_M
Size of storage buffer	b
Size of playing buffer	k
Failure tolerance	c
TTL for Join Message	ttl_j
TTL for Gossip Message	ttl_g
SegmentMap	S_M
TimeOut for SegmentMap	t_S
Intial Buffering delay	t_b
Gossip Periodicity	t_g
Tracker Update Periodicity	t_T
Peer Upload Capacity	P_u
Peer Download Capacity	P_d
Server Upload Capacity	S_u
Data Rate	D
Storage Buffer Request Timeout	t_s
Playing Buffer Request Timeout	t_p

which is similar to SCAMP [7] with some modification which will be described later. Based on the join messages, each peer p maintains the *InView* (peers

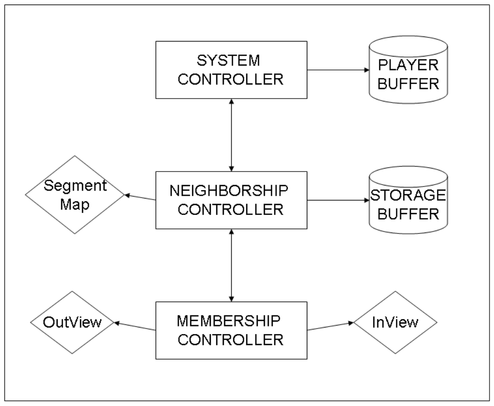

Fig. 1. System view of a *COCONET* node

which know the existence of p) and *OutView* (peers that p knows to exist) set as a partial view of the entire system which helps to facilitate node join/leave operations.

3.2 Co-Operative Caching

As already mentioned, each peer in the system has to contribute a part of its storage as *storage cache* to serve other peers. Contribution awareness is already very popular in P2P file sharing applications which helps in increasing system resource and is also employed in VoD systems such as [9]. But to efficiently manage this distributed storage for better performance in alleviating server load is still a significant challenge. *COCONET* tries to solve this problem where each peer on joining the system randomly caches b segments in its buffer in the hope of serving other peers when it is required. The segments in the storage cache remain unchanged as long as the peer remains in the system. This randomly distributed cache helps to increase system stability as there is very little chance that any video segment will be unavailable in any of the participating peers. This in turn translates to server load alleviation to a large extent which is another advantage for *COCONET*. The major chance for segment unavailability in *COCONET* is due to insufficient bandwidth resources for which the peer will be forced to query the Video Server, S.

3.3 Neighborship Management

As mentioned, efficient neighbor lookup is one of the key requirements in VoD systems for supporting VCR related operations. Each *COCONET* peer achieves this by maintaining a *SegmentMap*, S_M which is basically a list of M entries with the i-th entry, S_M^i representing the set of peers that have cached the i-th segment in their storage buffer. The underneath mechanism is a gossip-based algorithm which helps to disseminate S_M information through the entire overlay. The gossiping is done through the peer list of OutView which is constructed during the join operation. The gossip-based scheme will help to fill up S_M for each peer. So, essentially S_M serves as an index for the entire set of M segments and is utilized for neighbor lookup. It is trivial to observe that VCR operations can easily be satisfied by looking up S_M for the corresponding peers containing the required segments and downloading from them. Given, S_M is correctly maintained, any lookup operation can be satisfied in $O(1)$ messaging complexity by *COCONET*. We also maintain an additional failure tolerance factor c, which means that we keep c distinct peer entries for each entry, S_M^i in SegmentMap. Since there are a total of M segments in S_M, so the total number of entries in the SegmentMap of a *COCONET* peer comes to cM. This tolerance factor, c is a design choice and can be tuned according to application demand which will essentially help to tolerate peer departure/failure during churn conditions. We discuss the detailed protocol in Section 4.2.

3.4 Content Distribution Pattern

Existing P2P VoD systems distribute content from the playing buffer which is highly synchronized according to the paying position and requires any peer to download a video segment from another peer within the same playing segment or the next few consecutive segments. In highly skewed viewing patterns, if a peer issues a request for a segment with no nearby peer then the system fails to answer and has to resort for server resource. As a contrast, *COCONET* utilizes the storage buffer for content distribution and so is independent of playing position making any peer to download from a random peer with a completely different playing position. Thus, the previous situation due to highly skewed viewing patterns will have less severe impact in *COCONET* since the content distribution pattern is completely randomized.

4 Detailed Protocol

One of the important design goals of *COCONET* is to populate S_M as quickly as possible with a tolerance factor of c. This means that there should be c neighbors on average for each SegmentMapID, S_M^i. The fill-up size of S_M should be cM which is the target value for each peer. The importance of S_M in *COCONET* is obvious since all the neighbor lookup is performed through it and efficient maintenance of S_M is critical for system performance. To achieve a total size of cM, it needs to contact at least cM/b neighbors since each neighbor has b segments in storage buffer. An important theorem from SCAMP [7] states that, given a group size as N and the partial view size maintained at each peer as $O(\log N)$, the probability for a gossip to reach every member in the group converges to $e^{-e^{-c}}$ provided the link/node failure probability is not greater than $c/(c+1)$. *COCONET* exploits this theorem to reach a group size of cM/b by maintaining a partial view (i.e., OutView) size of $\log(cM/b)$. The partial view represents a randomized subset of the total number of peers in the system and thus, we use the partial view as gossip partners which will help to effectively disseminate the information among the participants. For gossip to be successful, logarithmic number of neighbors are required for information dissemination and within logarithmic rounds there is a high probability that the information reaches every member in the group, as pointed out in [7]. So, *COCONET* sets information dissemination gossip fan-out to be $\log(cM/b)$ and within $\log(cM/b)$ rounds of gossiping there is a high probability that the storage buffer information of the source peer has reached the required number of peers. Thus, there is a high probability that total list size for S_M to approach cM in logarithmic gossip rounds only.

4.1 Protocol for Join/Leave Operation

Each *COCONET* node is provided with a unique identifier. Tracker, T maintains a partial list of t live peers (t is maintained through periodic updating by the peers). Each joining peer initially contacts the tracker to acquire a *contact peer*. Then the joining peer sends a *join* message to the contact peer. The join protocol is very similar to SCAMP [7] with a little tuning. The join message is a 3-tuple of $\langle$*sender peer ID, join peer ID, ttl*$_j\rangle$, where sender peer is the one that sends the join message, join peer is the one that have initiated the join protocol for entering the system and ttl_j refers to the number of hops by the join message before it is killed. ttl_j is set for limiting the number of join rounds so that it may not move for an infinite number of times and is generally killed whenever any peer receives the same message for more than 10 times by simply discarding the received join message. Any peer other than the contact node, receiving a join message either keeps it or forwards it to a random neighbor from its OutView with a probability proportional to $\log(cM/b)/OutView.size$. This helps to keep the OutView size close to $\log(cM/b)$ with a high probability and will be used later for gossiping to disseminate information of S_M. The pseudo-code of *join* is listed in Table: 2.

Table 2. Join Protocol

```
At Join Node:
  contact_node ← Tracker
  OutView ← OutView ∪ {contact_node}
  send_join(contact_node, join_node, _)

At Contact Node:
  InView ← InView ∪ {join_node}
  forall n ∈ OutView
    send_join(n, join_node, $ttl_j$)

At Other Nodes:
  if ($ttl_j == 0$)
    return /* drop the join msg if ttl expired */
  with probability $\log(cM/b)/OutView.size$ do
    if (join_node ∉ OutView)
      OutView ← OutView ∪ {join_node}
    else
      choose randomly n ∈ OutView
      send_join(n, join_node, $ttl_j - 1$)
```

The protocol for leave operation involves the modification of InView which is essentially a list consisting of nodes which contains its nodeID in their partial views. The leaving node simply informs $c + 1$ random neighbors in InView to replace its nodeID with a random neighbor selected from the partial view of the node that invoked the leave operation. Then, it informs the rest of the neighbors in InView to simply remove its nodeID entry without replacing it. This protocol is entirely local and requires no global information. The protocol is simple and we skip its pseudo-code for brevity of space.

4.2 Protocol for SegmentMap Exchange

The SegmentMap exchange is the essential component of *COCONET* which helps to populate S_M by gossiping with neighbors in OutView. To achieve reliability, gossip sends a lot of redundant messages across the communication channel which is not suitable for bandwidth hungry streaming applications. So we need to tune the gossip protocol to avoid sending excessive messages after a certain system criterion is met. The gossip messages are sent according to some probability proportional to $(\frac{c}{avg.\ entry\ size\ of\ S_M})$. This ensures that gossiping will be switched off when the average entry size of S_M comes close to c with a high probability. During peer churns, *COCONET* detects and removes the dead entries from S_M and if the average entry size of S_M goes below c, gossiping will be switched on automatically in the next cycle resulting in repopulating S_M. The gossip message is a 4-tuple $\langle$*sender node ID, this*$\rightarrow$*node ID,* ttl_g*, SegmentMap information*$\rangle$ where ttl_g is set to be $O(\log(cM/b))$ for effective information dissemination as discussed above. Any peer receiving a gossip message employs a push-pull based dissemination mechanism wherein the receiver peer sends its S_M information to the sender and also updates its S_M from the sender. The pseudo-code of S_M exchange is listed in Table: 3.

Table 3. SegmentMap Exchange Protocol

```
At Sender Node:
  avg_entry_size ← $\sum_{i=1}^{M} S_M[i].size/M$
  with probability $\propto \frac{c}{avg\_entry\_size}$ do
    choose randomly n ∈ OutView
    send_gossip(n, node_ID, $ttl_g$, $S_M$)

At Receiver Node of gossiped $S_M^s$:
  if ($ttl_g == 0$)
    return /* drop since ttl expired */
  for $i \leftarrow 1$ to $M$ do
    $S_M[i] \leftarrow S_M[i] \cup S_M^s[i]$
  choose one random n ∈ OutView
  send_gossip(n, node_ID, $ttl_g - 1$, $S_M^s$)
```

4.3 Protocol for Caching

This protocol is very simple where each *COCONET* peer randomly selects b segments for caching. After joining and the SegmentMap established, it sends download request to other peers if any entry is found in S_M. If it receives a positive reply within timeout from any peer p then it downloads from p. Otherwise, it requests the server S for the segment.

4.4 Protocol for Retrieving Segments

One of the most frequent operations in *COCONET* is the lookup operation for supporting continuous playback or random seek. Usually the query will be for a particular segment i and the system is required to return a neighbor list where each of the neighbors contain the segment i in its storage buffer. It is trivial to observe that for lookup operation to be successful in *COCONET* it is essential that S_M is maintained correctly with sufficient number of entries to tolerate failures. As mentioned, *COCONET* tries to keep c neighbor entries for each segment so that a maximum of $c - 1$ failures can be tolerated without disrupting system performance. *COCONET* maintains the overlay network ordered on the basis of storage buffer blocks and does not consider the playing buffer for message dissemination. The pseudo-code of *look-up* is listed in Table: 4.

Table 4. Lookup Protocol

```
Input: Query for segment $q$
At Node n:
  for $i \leftarrow 1$ to $b$ do
    if (storage_buffer[$i$] == $q$)
      return storage_buffer[$i$]
  multicast_download_request($S_M[q]$, $q$)
  wait for availability reply with timeout
  if (timeout == $false$)
    select peer $p$ with earliest reply timestamp
    send_download_request($p$, $q$)
    receive segment from $p$
    return
  else
    send_download_request($S$, $q$)
    receive segment from $S$
```

5 Experimental Evaluation

In this section we present our simulation results for *COCONET*. We have implemented a discrete event simulator in C++ supporting an overlay size of 10,000 or more simultaneous peers. We have used GT-ITM [8] to generate the underlying physical network for our simulations based on transit-stub model. The network consist of 15 transit domains, each with 25 transit nodes and a transit node is connected to 10 stub domains, each with 15 stub nodes. We randomly choose peer from the stub nodes and place the video server on a transit node. The delay along each link was selected proportional to the Euclidean distance between the peers. We have set our simulation settings to be $P_u = 1$ Mbps, $P_d = 4$ Mbps and $S_u = 80$ Mbps with $D = 500$ kbps and the total viewing length of the video to be 128 minutes. Each segment size is set to be 3.7MB which corresponds to one minute video length. Each experiment was run for a length of 7,500 seconds. The peers join the overlay following a Poisson arrival model with arrival rate, $\lambda = 0.1$. The peer departure pattern follows an exponential distribution with an expected life time of 20 minutes. The other static parameters of our simulation are: $M = 128$, $t_j = 25$, $t_M = 60$ sec, $t = 250$, $k = 2$ segments, $b = 4, 8, 16$ segments, $c = 4$, $t_S = 5$ sec, $t_g = 20$ sec, $t_T = 50$ sec, $t_s = t_p = 25$ sec. We do not assume any transmission error in channels. For designing a VoD system it is important to efficiently utilize the upload bandwidth of all the peers in the system since it is the most scarce system resource and so we perform our discussion of experimental analysis to its usage efficiency. We avoid peer download analysis since download bandwidth is less likely to be the system bottleneck as we assumed it to be four times compared to upload bandwidth for each peer in our simulation settings. We also assume streaming a single video in our simulation scenario which is more simple to analyse.

5.1 Server Load

One of the most important objectives of P2P VoD system is to reduce server load. Figure 2 shows the server load with varying overlay sizes in *COCONET*. Server load is measured on a per streaming session basis, where it is measured by the percentage ratio of the total number of downloaded segments from the server to the total number of downloaded segments by all peers in the entire session. As we can observe from Figure 2, there is a slight increase of server load with increase in overlay size which is around 2% rise for every increase of 2,000 in overlay size. This is a very narrow increase rate and so we can conclude that *COCONET* scales well to large overlay sizes. Another interesting fact to observe is that, for similar overlay sizes, server load is greatly reduced on increasing the size of storage buffer. This can be intuitively justified by noticing that, the overall system demand remains same but system availability increases since there will be more number of available segments with higher value of b. Analyzing more specifically, we observe that there is a significant amount of server load alleviation when b goes up from 4 to 8 with around 15% reduction for 4,000 and 6,000. This effect is not so prominent when b goes from 8 to 16 which is around 4% in average.

5.2 Storage Buffer Usage Efficiency

In this section we study the buffer usage efficiency as this will be an important indicator for *COCONET* system performance. For each session, we measure the total number of storage buffers available in the system which is a constant, $b \times N$ and the total number of storage buffers that have been downloaded one or more times. We calculate the buffer usage efficiency by taking the ratio of the previous two parameters and plot the results in Figure 3. With a higher value of b, there are more number of available segments in the system which in

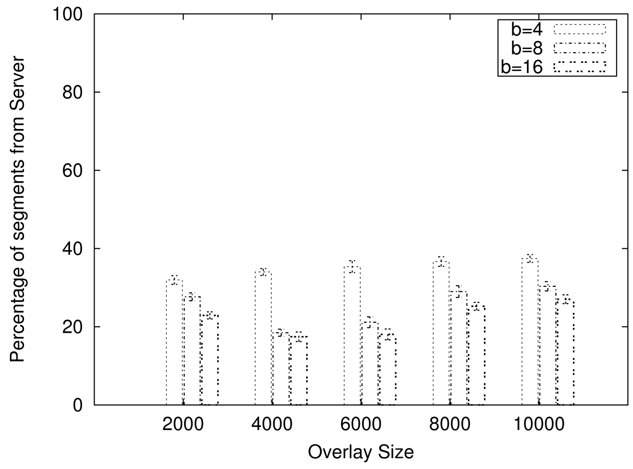

Fig. 2. Server load for varying overlay sizes

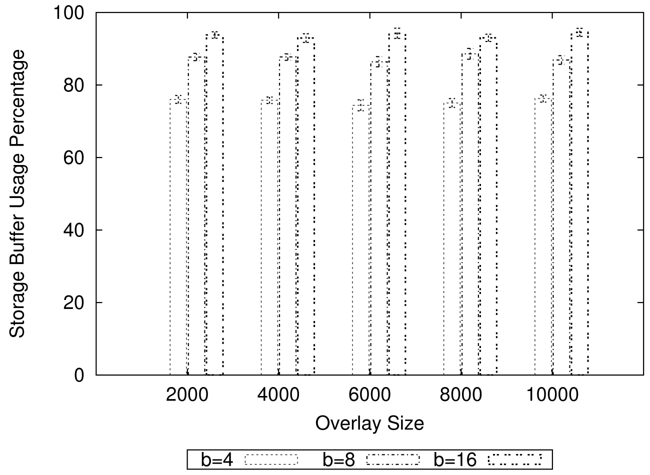

Fig. 3. Storage buffer efficiency for varying overlay sizes

turn helps to distribute the segments more efficiently among the peers. We also plot the probability density function of storage buffer usage for one streaming session in Figure 4 with an overlay size of 10,000 for $b = 8$ and we observe that the majority usage pattern is uniform excepting a somewhat higher usage at one point.

5.3 Load Balancing

In this section we experiment on the available upload bandwidth usage for each peer. We plot the results in Figure 5 which corresponds to the aggregated bandwidth utilization efficiency for all peers per streaming session. The plots show

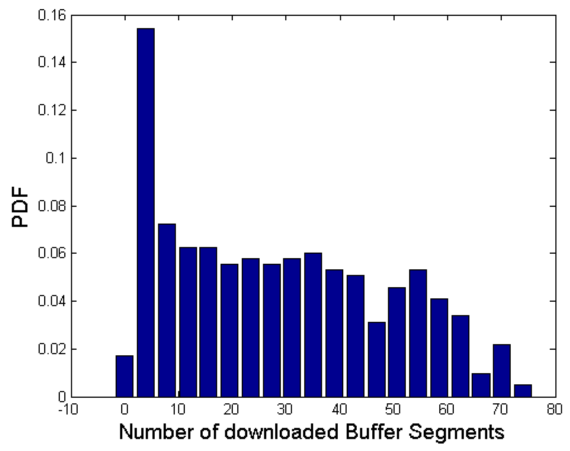

Fig. 4. Probability density plot for a function of number of downloaded buffer segments in a 10,000 node network

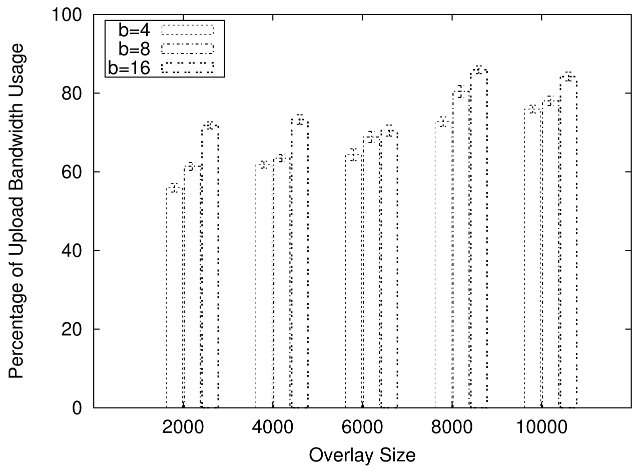

Fig. 5. Upload bandwidth usage efficiency

the average efficiency calculated over all the peers for a streaming session. We observe an increase of efficiency with increase of both overlay size and storage buffer size. For all the cases, the total segment availability of the system rises which translates to a better upload bandwidth efficiency. More optimizations for bandwidth efficiency can be achieved by using lower level granularity for data transmission since we use segment level transmission in our simulator. We also plot the cumulative probability distribution as a function of percentage of upload bandwidth usage in Figure 6 for $b = 4$ and an overlay size of 6,000. The same pattern follows with various overlay sizes. We can notice from Figure 6, that the upload bandwidth usage is efficiently utilized and distributed among the participants with a maximum usage of 60% for $b = 4$ for an overlay size of 6,000. We feel that this is an area for futher improvement by carefully analyzing the situation and optimizing the bandwidth usage to a greater degree which will help to improve the overall system performance.

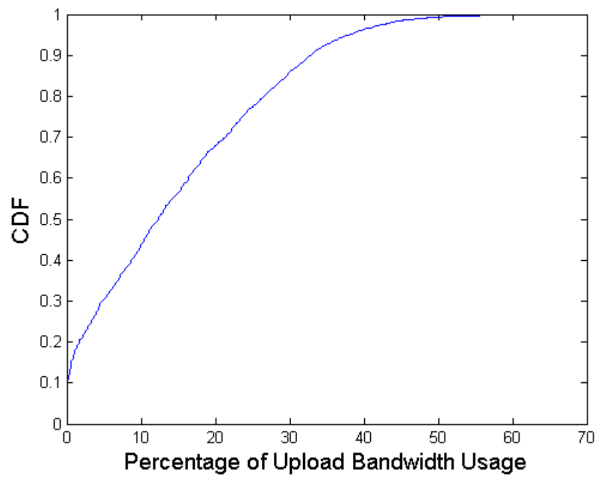

Fig. 6. Cumulative Distribution Plot for a function of % Upload Bandwidth Usage for b=4, N=6,000 peers

5.4 Peer Churn/Departure

In this section we experiment the performance of *COCONET* in churn/failure conditions. We evaluate streaming performance during peer failure/departure. We adopt a metric known as Segment Miss Ratio(SMR) which is basically the number of segments that have not reached the playing buffer within playback deadline divided by the total number of segments that should have been played till that time. Initially we start with 10,000 peers and after a while when all the peers have started playing, we randomly kill a peer every 10 seconds till we have failed 30% of the initial population. Figure 7 plots the SMR value as an average of the whole system at specific time intervals for different values of b. We can observe that initially the failure impact is more but as time progresses, more peers join and more storage buffers come into the system which increase segment availability. We analyze peer specific performance in Figure 8 where we plot the percentage of missing segments for each node. It can be seen that most of the peers have a SMR less than 10% and very few peers with high SMR. Each *COCONET* peer employs a failure recovery scheme by removing dead entries from S_M during the download request process if the neighbor fails to reply within a certain period of time. This will help to avoid wastage of messages to dead

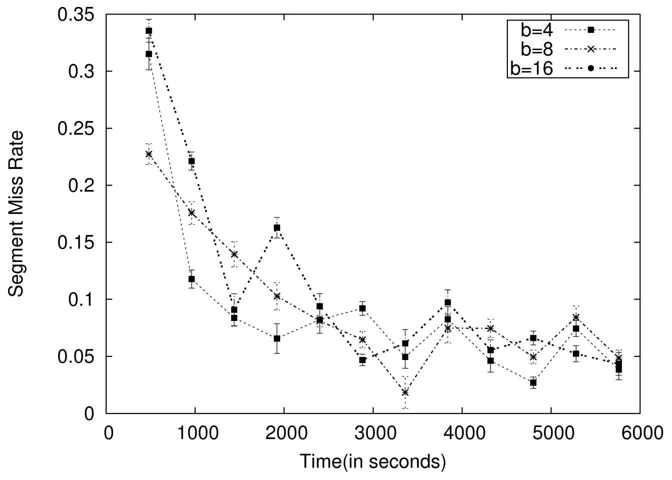

Fig. 7. System specific streaming performance at continuous time interval

peers. We do not employ separate heartbeat process for failure recovery since this process works well in our scheme with the added advantage of lesser overhead of control messages.

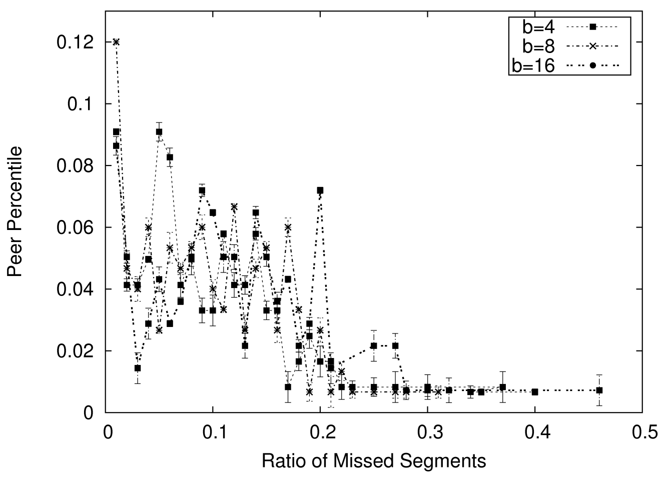

Fig. 8. Node specific streaming performance for each node in the overlay

5.5 VCR Operations

In this section we study the effect of VCR operations such as random seek/FF/FB in *COCONET*. To simulate random-seek, we employ the same model from Section 5.4 where the peer failure events are replaced by random seek. Figure 9

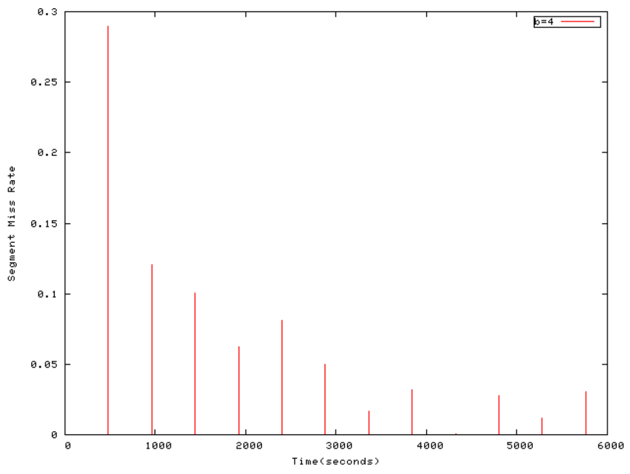

Fig. 9. Streaming performance for random seek operation

plots the result which indicates that the SMR is initially very high but after a certain period of time is almost averaging around 5%. We also simulated fast forward VCR operation and plotted in Figure: 10 to study its effects. Again we used the same model from Section 5.4 with peer failure events replaced by FF. We experimented FF operation for different speeds such as 2X, 4X and 8X where essentially for 2X we play the same content quantity with double the speed and likewise. We can observe from the graph that 8X has the highest missing segments whereas 2X and 4X are within comparable ranges.

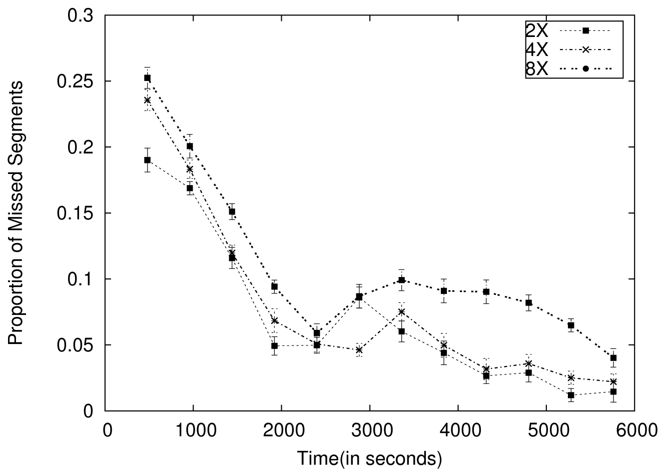

Fig. 10. Quality of streaming for fast-forward operation

6 Conclusion

This paper proposes a novel way of organizing peers based on storage cache content where each peer contributes a part of storage to form a large distributed cache which helps in alleviating server load to a greater extent by co-operative caching. Some of the most notable features of *COCONET* are high segment availabilty for any viewing patterns, uniform query load distribution, randomized content distribution pattern with uniform storage cache access pattern. As future work, we would like to: (a) deploy *COCONET* in PlanetLab for exercising its performance under real network dynamics, (b) employ certain predictive pre-fetching schemes to intelligently recover segments based on user viewing pattern, and (c) extend for multi-video scenario where the storage buffer can also be utilized to serve other peers watching different movies.

References

1. C. Huang, J. Li, K. W. Ross: Can Internet Video-on-Demand be Profitable? Proceedings of ACM SIGCOMM, pp. 133–144 (2007).
2. D. Wang, J. Liu: A Dynamic Skip List Based Overlay for On-Demand Media Streaming with VCR Interactions. IEEE Trans. Parallel and Distributed Systems, pp. 503–514. (2007).
3. T. Do, K. A. Hua, M. Tantaoui: P2VoD: Providing fault tolerant video on-demand streaming in peer-to-peer environment. Proc of ICC, pp. 1467–1472 (2004).
4. M. Hefeeda, A. Habib, B. Botev, D. Xu, B. Bhargava: Promise: Peer-to-Peer Media Streaming using CollectCast. Proccedings of ACM Multimedia, pp. 45–54 (2003).
5. B. Cheng, H. Jin, X. Liao: Supporting VCR Functions in P2P VoD Services using Ring-Assisted Overlays. Proc of the IEEE Intl Conf on Communications, pp. 1698–1703 (2007).
6. N. Vratonjic, P. Gupta, N. Knezevic, D. Kostic, A. Rowstron: Enabling DVD-like Features in P2P Video-on-Demand Systems. Proc of SIGCOMM Peer-to-Peer Streaming and IPTV Workshop, (2007).
7. A. M. Kermarrec, L. Massoulie, A. J. Ganesh: Probabilistic Reliable Dissemination in Large-Scale Systems. IEEE Transactions on Parallel and Distributed Systems, pp. 248–258 (2003).
8. E. Zegura, K. Calvert, S. Bhattacharjee: How to model an internetwork. Proceedings of IEEE INFOCOM, pp. 594–602 (1996).
9. Y. Huang, T. Z. J. Fu, D. M. Chiu, J. C. S. Lui, C. Huang: Challenges, Design and Analysis of a Large-scale P2P-VoD System. Proceedings of SIGCOMM, pp. 375–388 (2008).
10. X. Qiu, C. Wu, X. Lin, F. C. M. Lau: InstantLeap: Fast Neighbor Discovery in P2P VoD Streaming.Proceedings of ACM NOSSDAV , pp. 19–24 (2009).
11. Y. Cui, B. C. Li, K. Nahrstedt: oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks. IEEE Journal on Selected Areas in Communication, pp. 91–106 (2004).
12. M. Castro, P. Druschel, A. M. Kermarrec: SplitStream: High-bandwidth content distribution in a cooperative environment. Proceedings of SOSP, pp. 298–313 (2003).
13. Y. Gou, K. Suh, J. Kurose, D. Towsley: P2Cast: Peer-to-Peer patching scheme for VoD service. Proceedings of WWW, pp. 301–309 (2003).