A Dynamic Skip List-based Overlay for On-Demand Media Streaming with VCR Interactions

Dan Wang and Jiangchuan Liu
School of Computing Science
Simon Fraser University, British Columbia, Canada
{danw, jcliu}@cs.sfu.ca

***Abstract*— Media distribution through application-layer overlay networks has received considerable attention recently, owing to its flexible and readily deployable nature. On-demand streaming with asynchronous requests, and in general, with VCR-like interactions, nevertheless remains a challenging task in overlay networks. In this paper, we introduce the Dynamic Skip List (DSL), a novel randomized and distributed structure with inherent ability to accommodate dynamic and asynchronous clients. We establish the theoretical foundations of the DSL and demonstrate a practical DSL-based streaming overlay. In this overlay, the costs for typical operations, including join, leave, fast-forward, rewind, and random-seek are all sub-linear to the client population. The model also seamlessly integrates a fast data scheduling algorithm using linear network coding, yielding fast and robust downloading from multiple suppliers. Our simulation results show that the DSL-based overlay delivers reasonably smooth playback with diverse client interactivities while keeping the computation and bandwidth overheads low.**

I. Introduction

The widespread penetration of broadband access has made the Internet of today a popular vehicle for real-time media distribution. Scalable streaming to a large client population remains a challenging task due to limited server capacity and, to date, weak IP multicast support. Recently, however, overlay streaming has emerged as a promising solution to this problem [3][5]. In a peer-to-peer overlay network, each node receives media data from certain neighboring nodes. The data are cached in its local buffer, and then relayed to other neighbors, eventually realizing a multicast distribution. All these operations are implemented in the application layer, which means the system is highly flexible and readily deployable.

A key challenge in an overlay streaming system is to construct a data distribution structure among the collaborative nodes. This structure should be capable of accommodating autonomous nodes that join or leave the overlay at will and can even crash without notification. The problem is further complicated when introducing on-demand playback requests, and more general VCR interactions such as pause/resume, random-seek, fast-forward, and rewind. These services, though attractive to clients and content providers alike, call for an additional indexing structure to locate the expected data segments with asynchronous playback offsets. As a result, VCR operations have seldom been incorporated in existing overlay streaming systems.

In this paper, we propose the Dynamic Skip List (DSL), a novel data structure that effectively facilitates the above operations, as a solution to these problems. A DSL is a randomized structure consisting of a set of layers. Each new node, with its playback offset as a key, first joins a base layer, and then randomly and independently promotes itself to upper layers. Logical links to its neighbors in each layer are set up during this promotion process, which can then be used to quickly locate nodes with the expected keys through a fast skipping operation. We stress the following salient features of a DSL: 1) Its probabilistic nature eliminates costly rebalancing operations after nodes join or leave, making it a highly efficient and adaptive structure; 2) Its parallel logical links have inherent power to support multi-peer collaboration in an overlay network; and 3) Both the search cost and the state information kept at a node are sub-linear (constant or logarithmic) to the DSL size, suggesting good scalability.

We go on to demonstrate a practical overlay network that seamlessly integrates indexing and data distribution through a DSL, then discuss the key issues involved in realizing this DSL-based overlay. The system also integrates a linear network coding based algorithm for data scheduling, which yields optimal data downloading from multiple partners.

The performance of a DSL-based streaming overlay is then examined under diverse network and client configurations. The results show that it achieves a reasonably stable streaming rate and a low control overhead. It also scales well and accommodates highly dynamic client

behaviors with limited buffer spaces. More importantly, it supports VCR operations effectively at a reasonably low cost while achieving satisfactory playback quality. The latter is difficult to achieve using existing systems.

The remainder of this paper is organized as follows. Section II presents the background to the model and reviews related work. The model is described in Section III, together with an overview of DSL. In Section IV, we discuss the construction and maintenance of the DSL-based overlay, as well as its support for VCR operations. We further explore the multi-supplier data scheduling in Section V. The performance of the DSL-based streaming overlay is examined in Section VI. Finally, Section VII presents our conclusions and offers some future directions for research.

II. Background and Related Work

Over the last decade, many proposals for multi-channel on-demand broadcasting [15] have been put forward. These have explored the temporal locality of client requests, allowing a media server to accommodate concurrent client requests through batching, patching, or periodic broadcasting. These protocols rely on underlying broadcast or multicast channels, preferably in the network layer. The deployment of IP multicast over the Internet, however, remains restricted. This is due to a range of practical and political issues, such as the lack of incentives to carry multicast traffic. In addition, VCR interactions would consume significant server or network resources [2][18][19].

Recently, application-layer solutions have received much attention, owing to the massive buffering capacity of personal computers and the readily deployable nature of these protocols. Our work is partially motivated by the studies on live media streaming through application-layer overlay networks. Originating from IP multicast, the proponents of such systems often advocate a tree structure out of the overlay nodes; e.g. NICE [3], SpreadIt [8], and ZIGZAG [26]. Constructing and maintaining an efficient distribution tree among the overlay nodes is a key issue for these systems. Enhancements have also been proposed which address the unbalanced load or vulnerability of the tree structure. Examples of this include building a mesh-based tree (Narada and its extensions [5], and Bullet [16]), maintaining multiple distribution trees (Split-Stream [4]) or gossip partners (CoolStreaming [31]), and leveraging layered coding (PALS [25]) or multiple description coding (CoopNet [22]). Our DSL is a relatively light-weight structure, but also supports multi-peer data delivering, which is more robust. In addition, our target is on-demand streaming, which calls for different solutions for asynchronous requests and more complex VCR operations.

Several pioneering works on peer-to-peer on-demand streaming (e.g. [7], [9], [11], and [13]) are closely related to our proposal. In this scenario, the video data provided by some seeding nodes are spread among nodes of asynchronous demands, and one or more nodes can collectively supply the buffered data to a new demand. These systems do not rely on dedicated IP multicast channels, and many of them do not maintain an explicit structure among the autonomous nodes. The asynchronous requests are accommodated through a search in the initial joining process, which can be time consuming. CollectCast [13] and oStream [7] suggest using a centralized server to maintain the playback progress of all the nodes. Although the server is no longer required to distribute the video stream to every client, the demand for it to maintain the indexing information can still be high in the presence of frequent node joins and leaves. P2VoD [9] and TAG [32] logically organize the nodes into a linear or tree structure. With the playback offset being a *key*, these structures assist random search and some other, complicated VCR operations in a distributed manner. However, a linear structure is not capable of providing asynchronous access in sub-linear time, while a tree structure requires complicated rebalancing operations. In addition, fast-forward and rewind are not well-supported by either. These issues are addressed in our proposed DSL structure.

III. Dynamic Skip List (DSL)

We consider an overlay streaming system consisting of a content server and a set of clients. Each client is an application-layer overlay node with a size-limited buffer. The media file in the server is partitioned into equal-length segments, and distributed to the clients upon demand. A client node will cache the received data, and may supply it to other nodes from its local buffer.

The clients can join or leave the overlay at will; they can also crash without notification. Their playback requests are generally asynchronous with different starting offsets. In addition, various VCR operations are to be supported in the system, including pause and resume, random-seek, fast-forward, and rewind.

Given the data asynchronicity in the overlay, communications cannot be set up directly between any node pairs, particularly given the limited buffer size. A key challenge is thus to construct a data distribution structure that enables the clients to collaborate asynchronously. The VCR interactions also call for an indexing structure so that a client can efficiently locate suppliers of the expected data segments.

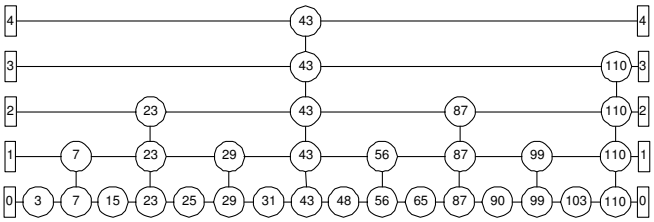


Fig. 1. A Regular Skip List of 16 nodes.

We now introduce the Dynamic Skip List (DSL), which enables both types of function, and effectively balances operation speeds and cost. In this section, we first establish the theoretical foundation of the DSL; its use in the asynchronous streaming overlay will be detailed in the two sections which follow.

A. Overview of Skip List

We first give an overview of a regular skip list. A skip list is an ordered list with additional, parallel links. Assume there are N keys in the list, indexed from 1 to N. The $i \times 2^l$-th key, $i = 1, 2, \cdots, l = 0, 1, \cdots$ will have links to the $(i-1) \times 2^l$-th and the $(i+1) \times 2^l$-th keys respectively. This translates to a layered structure, as shown in Fig. 1, where the link distance between two neighboring nodes in layer l is 2^l.

In this layered representation, a single key in the list is mapped into multiple logical nodes along the same column. Since the parallel links in higher layers skip geometrically more than those in lower layers, a key search can be started from the highest layer, so as to quickly *skip* unnecessary parts, and then progressively move to lower layers until it achieves a hit. The complexity of this top-down search is bounded by $O(\log N)$.

A skip list can also be constructed in a random fashion: each key is first inserted into the *base layer* (layer 0), and then randomly *promotes* itself to the upper layer with probability $\frac{1}{2}$. If successful, the key will leave a node copy in the previous layer, and try to promote itself again in the new layer until it fails or a *MaxLayer* is met. Assuming it stops at layer l, it will then connect to all the neighbors from layer 0 through layer l. This randomized version achieves the same search performance as the deterministic version when MaxLayer is set to $\log(N)$ [23].

B. Dynamic Skip List

Compared to other typical indexing structures, such as an AVL tree or a B+ tree, a randomized skip list is significantly easier to implement and generally faster. More importantly, its probabilistic nature eliminates the need for costly rebalancing operations after each key insertion, making it an attractive solution for distributed applications. Applying the skip list in our system, however, is not straightforward. Firstly, the number of keys in a skip list has to be predefined, and so is the value of MaxLayer; Secondly, higher layer nodes in a skip list often encounter significantly more hits than others; for example, the highest-layer node can be accessed for each search operation, and is thus vulnerable. Thirdly, the random promotion could generate unbalanced layers, which is to say, layers with far more or less logical nodes than might be expected, which greatly reduces the maintenance and search efficiency.

Dynamic Skip List (DSL) addresses these limitations by allowing an adaptive setting for the list size, and effectively compressing unbalanced layers. A DSL is built in a similar way to a regular skip list, but there is no a MaxLayer limit; a newly inserted key will stop promoting itself only when it fails.

Lemma 1: Let Y_l denote the number of logical nodes in layer l. The expected number of logical nodes in layer l, $E(Y_l)$, is $\frac{N}{2^l}$.

Proof: Let indicator random variable Y_l^i denote whether the ith key has a corresponding logical node in layer l. According to the promotion process, we have $Pr[Y_l^i = 1] = \frac{1}{2^l}$ and $Pr[Y_l^i = 0] = 1 - \frac{1}{2^l}$. It follows that $E(Y_l^i) = \frac{1}{2^l} \times 1 + (1 - \frac{1}{2^l}) \times 0 = \frac{1}{2^l}$, and hence $E(Y_l) = \sum_{i=1}^{N} E(Y_l^i) = \frac{N}{2^l}$. ■

Lemma 2: For all layers $l < L$, the probability for $Y_l < E(Y_l) \times (1 - \frac{1}{C})$ is $O(\frac{1}{N})^{\epsilon^-}$, and $Y_l > E(Y_l) \times (1 + \frac{1}{C})$ is $O(\frac{1}{N})^{\epsilon^+}$ where $L = \log\left(\frac{N}{\log N}\right)$, C is a positive constant, $\epsilon^- = (\frac{1}{C})^2 \log_e 2$, and $\epsilon^+ = (\frac{1}{C})^2 \log_e 4$.

Proof: From Chernoff Inequality [21], we have $Pr[Y_l < E(Y_l) \times (1 - \frac{1}{C})] < e^{-E(Y_l)(\frac{1}{C})^2}$. In layer L, since $E(Y_L) = \frac{N}{2^L} = \log N$, we have $e^{-E(Y_L)(\frac{1}{C})^2} = e^{-\log N(\frac{1}{C})^2} = (\frac{1}{N})^{(\frac{1}{C})^2 \log_e 2} = O(\frac{1}{N})^{\epsilon^-}$. The case for $Y_l > E(Y_l) \times (1 + \frac{1}{C})$ can be proved in a similar manner. ■

Intuitively, lemma 2 suggests that the number of nodes in a low layer is generally closer to the expected value, and we refer to such a layer as a *balanced layer*. The layers above $L = \log\left(\frac{N}{\log N}\right)$ might be unbalanced owing to excessive promotions. Fig. 2 shows an example. Here, starting from layer 5, there is only one logical node in each layer. This means that unnecessary neighbor information has to be maintained, making the search efficiency potentially quite low across these layers.

To solve this problem, we compress all the layers above L into a single top layer. The following lemma gives the node distribution.

Lemma 3: The number of layers in a DSL after

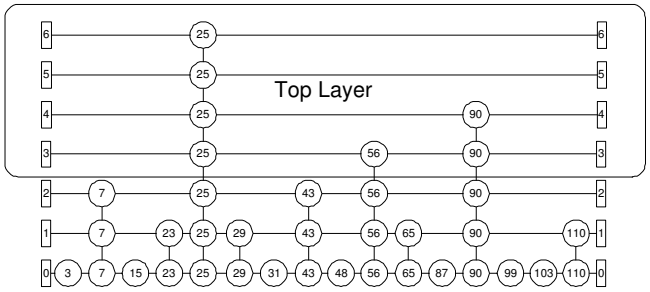


Fig. 2. A Dynamic Skip List (DSL) of 16 nodes.

compression is $\log\left(\frac{N}{\log N}\right)$, and the expected number of logical nodes in the top layer is $2\log N$.

Proof: In DSL, all the nodes above $L = \log\left(\frac{N}{\log N}\right)$ are merged into the top layer, and their expected number is $\sum_{l=L}^{\infty} E(Y_l) = 2\log N$. ■

Since the number of logical nodes in the top layer is bounded by $O(\log N)$, this small set of nodes can be easily monitored by a single entity, such as the content server in an overlay network. It is worth noting, however, that N is not fixed in this dynamic list; nor is it known *a priori*. Hence, we should keep track of the number of the keys to determine which of the nodes belong to the top layer. Assume $\hat{Y}_l$ is the number of logical nodes we observed from layer l, N can be estimated as follows:

Lemma 4: $\hat{Y}_l 2^l$ is an unbiased estimator of N, the total number of keys in a DSL.

Proof: From Lemma 1, we know that $E(Y_l) = \frac{N}{2^l}$ or $E(2^l \times Y_l) = N$. Hence, $\hat{Y}_l 2^l$ is unbiased in estimating N [12]. ■

IV. DSL-Based Overlay Construction, Maintenance, and VCR Operations

The mapping between a DSL and an asynchronous overlay is straightforward: The playback offset of a client serves as its key in the DSL, and the set of logical nodes associated with this key all map to the client node in the overlay. The key is updated over time according to the playback progress. Since the playing speed is identical for all the normal clients, their relative distances will not change, unless VCR operations are invoked.

The client also maintains the logical links in the DSL, and the content server needs to keep track of all the logical nodes in top layer. To this end, the server periodically checks the size of a randomly selected layer, estimates the overlay size N from Lemma 4, and then accordingly releases or merges layers to ensure that there are $O(\log N)$ logical nodes in the top layer. From the previous discussions, it is easy to show that each overlay node, including the server, maintains at most $O(\log N)$ extra information.

A. Join Operation

When a new client is to join the overlay, it first contacts the content server, which redirects the client to a top-layer node with the closest key. The new client then performs a top-down search to insert itself into the base layer, and chooses the left neighbor (with earlier playback time) as its supplier in the overlay. It goes on to conduct the bottom-up random promotion, and set up its links to the corresponding neighbors in each layer. Note that the top-down search takes $O\left(\log\left(\frac{N}{\log N}\right)\right)$ time, and the potential neighbors across all the layers can be recorded in this process. As such, we can make the following observation on the cost for a joining operation.

Theorem 5: The expected message cost for a client join is $O\left(\log\left(\frac{N}{\log N}\right)\right)$.

Proof: In the join algorithm, the expected number of logical nodes that the newly joined client should contact is constant in each layer. Since the total number of layers is $O\left(\log\left(\frac{N}{\log N}\right)\right)$, the expected message cost is also bounded by $O\left(\log\left(\frac{N}{\log N}\right)\right)$. ■

In practice, there could be multiple nodes that are eligible to supply data to a newly joined client, and the suppliers may also have bandwidth constraints. We will discuss this general scenario in the next section, and present an efficient multi-supplier searching and scheduling algorithm.

B. Leave Operation and Failure Recovery

A client that is scheduled to leave the overlay should first notify its neighbors in the DSL such that that they can re-connect with each other to form new neighboring relations.

Theorem 6: The expected message cost for a client departure is $O(1)$.

Proof: For a client where the highest corresponding logical node is in layer l, a graceful departure requires $O(l)$ messages to notify all neighbors in layer 0 through l. Since we expect there to be $\frac{N}{2^l}$ nodes in layer l, the average number of messages for a departure is $\frac{1}{N}\left(\sum_{l=0}^{\log\left(\frac{N}{\log N}\right)} l\frac{N}{2^l}\right) = \frac{1-\left(\frac{1}{2}\right)^{\log\left(\frac{N}{\log N}\right)}}{\frac{1}{2}} = 2 - \frac{\log N}{N}$, which is $O(1)$. ■

Intuitively, this constant amortized cost holds because the number of nodes with $O(\log N)$ neighbors is quite small. Most of the nodes have far fewer neighbors in the DSL, and hence lower costs. A similar argument suggests that while the maximum extra information kept at each client node is $O(\log N)$, the average is $O(1)$ only.

Every client also periodically exchanges echo messages with its neighbors in the DSL, enabling an abrupt client failure to be easily detected. The parallel links in

the DSL then enable the affected neighbors to perform local repairs. As an example, in Fig. 2, when the node with key 65 fails, all its neighbors, 56, 87, and 90, will detect the failure. Starting from the link between 56 and 90 in layer 2, the three nodes will detect each other's existence in layers 1 and 0, and form new neighborships. This is a variation of the search operation and the cost is, again, at most $O(\log N)$.

C. VCR-like Interactions

Since the cost for a leave and then re-join with a new playback offset is only $O(\log N)$, this combination can be used to implement most typical VCR operations, including fast-forward and rewind at speed v which can be realized by playing one segment out of v segments [29]. However, a fast-forward or rewind movement generally consists of a series of such jump operations, necessitating more efficient solutions.

The key issue to a jump operation is to locate the suppliers with expected segments in time. We note that a DSL provides effective support toward this operation through its horizontal links, which enable a client to skip unnecessary nodes at a fairly stable speed and hence segments. We now present a simple analysis of the cost of the jump operations for fast-forward, and determine the layers that the node should follow. The analysis and operations also apply to rewind movements after applying a symmetric transform.

Given that an overlay node has a size-limited buffer, the client initiating the fast-forward operation will either stay with its current supplier or move to a new one after a jump operation. Assuming that the playback offset between two consecutive nodes in the DSL base layer differs d segments on average, and the client can retrieve n data segments from its supplier with VCR speed v, we have the following lemma describing the nodes to be skipped.

Lemma 7: The number of logical nodes to be skipped ahead is $\frac{n(v-1)}{d}$ for each jump operation.

Proof: Assume the current playback offset of the client is j; after playing n segments at speed v, the next segment to be retrieved will be $j+nv$. Meanwhile, the playback offsets of all normal-speed (1x) nodes are advanced by n segments. Since the average difference between two consecutive nodes is d, we need to skip $\frac{n(v-1)}{d}$ nodes in the jump operation. ■

Theorem 8: Each jump operation for fast-forwarding needs $O(1)$ time.

Proof: The average distance between two nodes in a DSL layer is $\log\left(\frac{n(v-1)}{d}\right)$ is $\frac{n(v-1)}{d}$. Hence, following the horizontal links in this layer, each jump operation needs only $O(1)$ time to skip $\frac{n(v-1)}{d}$ nodes. ■

The above theorem suggests that the cost for a jump operation is independent of the overlay size, nor the fast-forwarding/rewinding speed. Such performance can be difficult to achieve with linear or tree structures, because there are no logical links for efficient skipping nodes with regular distances.

In practice, when $\log\left(\frac{n(v-1)}{d}\right)$ is not an integer, the client can adaptively set n; i.e. stay for a longer or shorter period with the current supplier and move along higher or lower layer links to achieve an average speed of v. We discuss the range of n in Appendix and more details can be found in [27]. The maximum speed that DSL can provide is bounded by its highest layer. Nevertheless, a speed of no more than 32x can be easily achieved even in a small DSL overlay, and, as investigated in [29], a higher speed is rarely perceived as useful by users, nor is it supported in most commercial VHS or DVD players.

V. Multi-Supplier Data Scheduling

In a DSL overlay, iteratively searching its neighbors along the base layer links, a client can easily locate others with overlapped buffers. All of them can be potential suppliers. In this section, we generalize the model of data delivery to this multi-supplier scenario. This is clearly more efficient and robust, particularly when we consider network heterogeneity and dynamics. However, a scheduling algorithm is necessary in order to retrieve the expected segments without duplication.

A. Network Coding based Data Scheduling

The scheduling problem can be formalized as follows. Given an expected set of data segments to be retrieved in a window of size w, and S suppliers each with a subset of the segments and available bandwidth b_j, $j = 1, 2, \cdots, S$, we need to find an *assignment* $\mathcal{A} = \{(i, j)\}$ for retrieving segment i from supplier j, such that the finishing time for the last segment is minimized.

The problem closely resembles the parallel machine scheduling problem, which is known to be NP-hard [6]. It is particularly complicated given the highly heterogeneous bandwidth from different suppliers. On the other hand, we are aware that an overlay node, unlike conventional network nodes, can actively manipulate the data in its buffer and naturally realize network coding [1][10][14][17][33]. Theoretically, this enables a node to retrieve fractionized segments or jobs in the parallel machine scheduling context. Given such a relaxation, we now show that provided the node finds a set of suppliers whose total available bandwidth is no less than the streaming rate, an optimal scheduling algorithm exists.

We focus here on linear coding for its simplicity and efficiency. We denote the *original* data segments as $c_1, c_2, \cdots, c_w$. A random linear coding on $\{c_i\}$ is a vector $f_i = \sum \beta_i \times c_i$, for $i \in (1 \cdots w)$, where the co-efficient vector β_i is randomly generated in a finite field F_q of size q. We refer to f_i as *combined* data segments, which also span a range of w. If all f_i are linearly independent, once a node has a subset of f_i that spans range w, it can recover all the w original segments by solving a set of linear equations.

Applying network coding in the overlay nodes, we can make the following observation.

Lemma 9: For any assignment without using network coding, there is a corresponding assignment using network coding with a shorter or equal finishing time.

Proof: Let $\mathcal{A}$ be an arbitrary assignment without using network coding. Let m_s be the number of data segments retrieved from supplier s. We construct a corresponding assignment $\mathcal{A}'$ using network coding, where exactly m_s combined segments are retrieved from supplier s. The downloading times for the two assignments are thus identical, and after decomposition, assignment $\mathcal{A}'$ provides all the segments as did $\mathcal{A}$. ■

This leads to a network coding-based greedy scheduling algorithm, as shown in Fig. 3.

Algorithm Greedy Scheduling
  $m =$ the number of segments received;
  $m_d =$ the number of segments decoded;
do
  if (any parts of $f_1 \cdots f_m$ can be decoded)
    Decode and update m_d;
  if (supplier s is idle) **and**
    (s has data not included in $c_1 \cdots c_{m_d}$)
    Request f_{m+1} from s;
while $m < w$.

Fig. 3. A network coding based greedy scheduling algorithm.

The greedy algorithm continuously examines whether the retrieved data segments can be decoded during the downloading process. Whenever the subset of segments is decomposable, they are decoded immediately. When a supplier idles, the node will check whether it has data of interest, such as data segments that have not been downloaded or useful for decoding, and, if so, it will retrieve them accordingly.

Theorem 10: The greedy scheduling algorithm with linear network coding is optimal in downloading time.

Proof: It is easy to show that the greedy algorithm downloads w combined data segments in $\frac{wc}{b}$ time, where $b = \sum_{i=1}^{s} b_i$ and c is the size of a segment. Assume the coefficients are linearly independent (we discuss this later in this section), the w original data segments can be fully recovered from these combined data segments.

On the other hand, the total downloading bandwidth across all the suppliers is at most $b = \sum_{i=1}^{s} b_i$. The minimum downloading time for w data segments is thus $\frac{wc}{b}$ for any scheduling algorithm. It follows that the greedy algorithm with linear network coding is optimal. ■

B. Practical Considerations

Clearly, the scheduling algorithm guarantees playback continuity only if the total bandwidth from all the suppliers is greater than the streaming rate, which is a basic requirement for any streaming application. The clients also need to exchange such extra information as coefficients, segment availability, and segment request. We note that their volume is two orders of magnitude lower than that of the media data, and also that they can be piggybacked by the data segments. Our experimental results show that the extra bandwidth required for such data is far less than 1%, which is negligible in practice. The remaining major practical concerns are the computation overhead of network coding and the linear dependency of coefficients.

1) Computation Overhead: This is an important concern when realizing any active operations in a network node. Compared to existing channel coding schemes, the network coding in our system is simply a set of linear operations with well-known fast algorithms. In addition, coding and decoding are on a segment basis, and, for k segments, the operation required is the inversion of a $k \times k$ matrix, which requires $O(k^3)$ time only. Especially, we only need to decode a very small part of file each time. As an example, for a buffer of 15MB and a segment of 128KB, even if all the data segments are combined, we have $k \cong 100$. In our experience, the computation time is less than 1ms in a typical Pentium IV 2.8GHz PC.

2) Linear Dependency: For a small set of suppliers, if the coefficients are independently and randomly generated at each node as in our implementation, their dependency can be extremely low [20]. This is verified in Tab. I, which shows the probability of linear independency as a function of the finite field size for four suppliers. In our experiments, we chose $q = 2^8$, and performance of the scheduling algorithm was rarely affected by linear dependency.

VI. Performance Evaluation

We now evaluate the performance of the DSL-based overlay, and compare it with existing systems.

TABLE I
PROBABILITY OF LINEAR INDEPENDENCY AS A FUNCTION OF FINITE FIELD SIZE (q).

q	Probability	q	Probability	q	Probability
2^1	0.288788	2^5	0.967773	2^9	0.998043
2^2	0.688538	2^6	0.984131	2^{10}	0.999022
2^3	0.859406	2^7	0.992126	2^{11}	0.999511
2^4	0.933595	2^8	0.996078	2^{12}	0.999756

A. System Configuration

We use the GT-ITM [30] topology generator to produce networks for simulation, and have performed experiments using diverse network and client configurations. In this section, we present representative results based on the following settings: The network consisted of 4 transit domains, each with 5 transit nodes, and a transit node was connected to 6 stub domains, each with 8 stub nodes. We then added 16 stub nodes to produce a 1000-node network. The default bandwidth settings between two stub nodes, a transit and a stub node, and two transit nodes were 4Mbps, 9Mbps, and 12Mbps respectively. We also added cross traffic to emulate dynamic link bandwidths.

The overlay streaming system was built on top of this underlying network topology. We placed the content server and clients on randomly selected network nodes. For each sample result presented in this section, we repeated the placement and simulation ten times to mitigate the effect of randomness. The streaming rate was 128Kbps and the length of the stream 150 minutes, both of which are typical for Internet-based video delivery. The default size of the client-side buffer was 15Mbytes, which can be easily accommodated in state-of-the-art personal computers. We also investigated the impact of different buffer sizes in the experiments.

B. Control Overhead

We first investigated the control overhead for the DSL overlay construction and maintenance. We were interested in both global and local effects, that is, the average cost per operation and the maximum cost that a client could incur. The costs are measured in the number of messages exchanged, reflecting both the bandwidth consumption and the execution time of an operation.

1) Overhead for Join Operation: Fig. 4 depicts the average message cost per join operation for different DSL overlay sizes. To better understand the benefit of DSL, we have also plotted the results obtained using a regular skip list (RSL). It can be seen that for the join operation, the overheads for DSL and the regular skip list are reasonably close, both following a logarithmic function of the overlay size. DSL is slightly better, owing to the top layer compression.

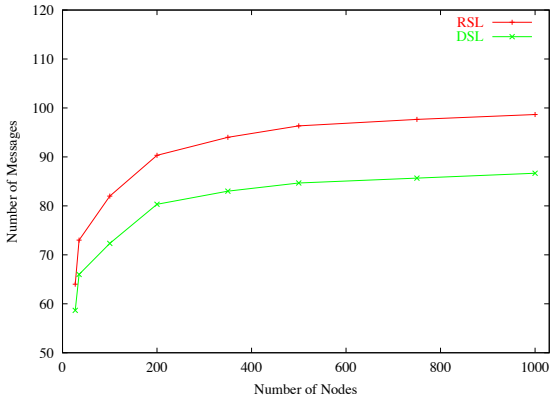


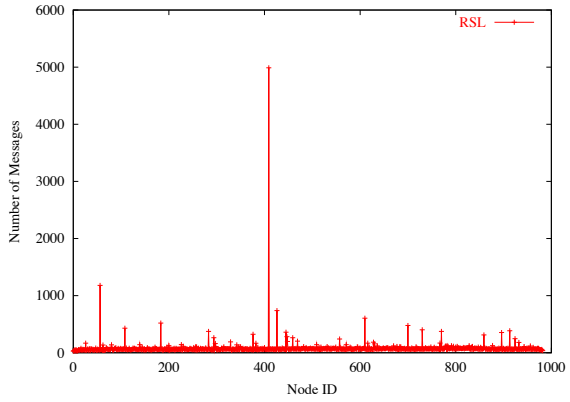
Fig. 4. The maximum number of messages per join operation for Regular Skip List (RSL) and Dynamic Skip List (DSL).

However, the message distribution among the overlay nodes can be quite different for the two structures. Fig. 5(a) depicts the total message cost per node, summed over 1000 join operations. For the regular skip list, the message distribution is highly skewed: about 90% of the messages are at 8 nodes, which corresponds to the high-layer logical nodes in the regular skip list. In particular, over 5000 messages are at a single node (ID: 408). Such overloaded nodes can be quite vulnerable. VCR operations will further aggravate this problem. On the contrary, as shown in Fig. 5(b), the distribution in the DSL is much more uniform, and better scalability and stability can thus be expected.

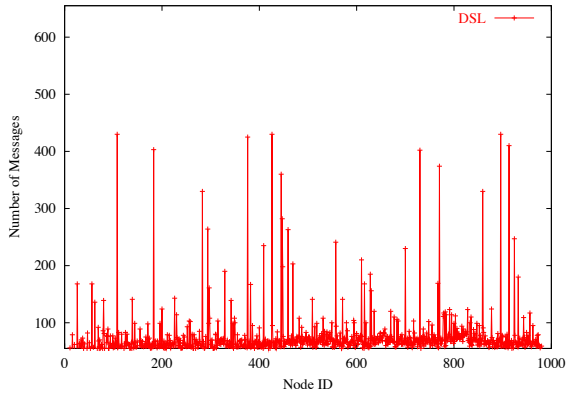
There are other deficiencies of the regular skip list, such as weak support to dynamic overlays. Hence, in the following experiments, we focus on the DSL-based overlay only.

2) Overhead for Leave and Failure Recovery: Fig. 6 shows the message cost per leave operation for different overlay sizes. It can be seen that the average cost is almost flat, which is consistent with our analysis in Section IV. We also recorded the maximum cost, which is logarithmically related to overlay size. Intuitively, the maximum cost occurs when a node in the high layer of DSL leaves. Such a node maintains $O(\log N)$ neighboring links, and all of them have to be re-connected afterwards.

To investigate the cost for failure recovery, we randomly failed 10%, 20%, and 30% of nodes in the overlay. Fig. 7 presents the average message cost per failure recovery for different overlay sizes. Again, the costs are reasonably constant across different failure rates and



(a) Regular Skip List



(b) Dynamic Skip List

Fig. 5. Message distribution among the overlay nodes for 1000 join operations. (a) Regular skip list (RSL); (b) Dynamic skip list (DSL).

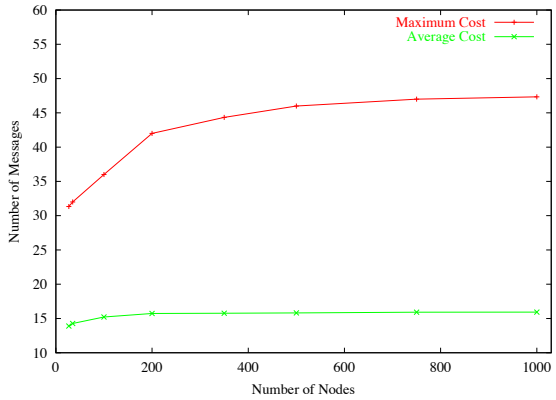


Fig. 6. Message cost (max and average) per leave operation for different DSL overlay size.

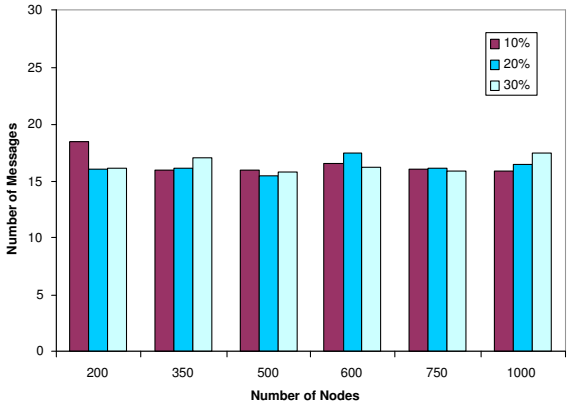


Fig. 7. Message cost per failure recovery for different DSL overlay sizes.

overlay sizes. Such results suggest that a DSL overlay is generally stable and scalable in terms of control overhead.

C. Streaming Quality

In the second set of experiments, we examined the streaming quality of the DSL overlay. Given that playback continuity is critical for real-time media streaming, we adopted a Segment Missing Rate (SMR) as the major criterion for evaluating the streaming quality. A data segment is considered missing if it is not available to client till the play-out time, and the SMR for the whole system is the average ratio of the missed segments across all the participating clients during the simulation period.

For comparison, we also simulated an existing on-demand overlay streaming system, *oStream* [7], under the same network and buffer settings. *oStream* employs a tree structure, in which each client node caches played-out data and relays to its children, which may have asynchronous playback offsets. A centralized directory server is used to maintain global information on the overlay. This also facilitates client join or failure recovery. Details of *oStream* can be found in [7].

We focused first of all on streaming quality for asynchronous playback requests only. The initial playback offsets of the requests were uniformly distributed between 0 and 150 minutes.

To emulate local bandwidth fluctuations, we randomly injected traffic to the network links such that the available bandwidth at each link varied over time yet the total available bandwidth of the network remained at 80% of the base setting (with no cross traffic).

Fig. 8 plots the segment loss rates (SMRs) for 1000-node DSL and *oStream* overlays during a 4500-sec simulation. It can be seen that the loss rate of DSL is generally less than 0.1, which is not only lower than *oStream*, but also more stable. From a video decoding point of view, such a loss can be effectively masked by interleaving or error-concealment techniques [28]. On the other hand, the loss rate of *oStream* greatly fluctuates over time, with a peak value as high as 0.65, resulting in poor video quality. This is mainly because *oStream* relies

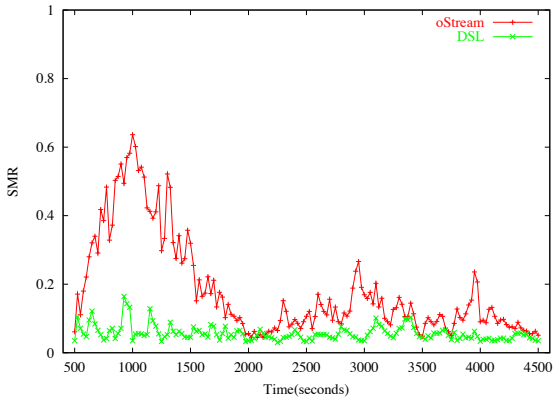


Fig. 8. Segment missing rate (SMR) for DSL and oStream with local bandwidth fluctuation.

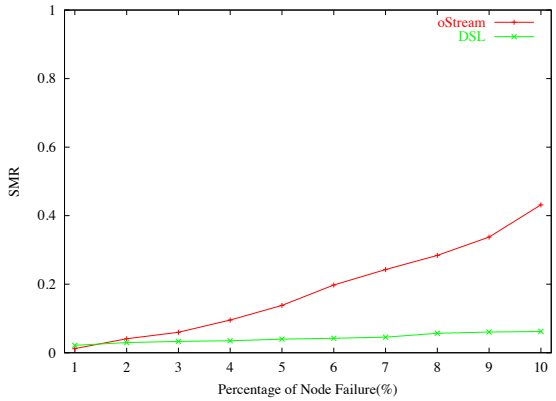


Fig. 10. Segment missing rate (SMR) for DSL and oStream under different node failure rates.

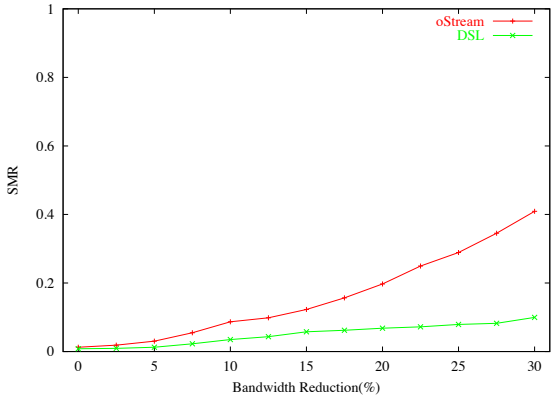


Fig. 9. Segment missing rate (SMR) for DSL and oStream under global bandwidth reductions.

on a specific tree structure for streaming, so bandwidth reduction at an internal link of the tree, and particularly at those close to the root, could result in severe loss across many descendants.

It is known that not only do the available bandwidths of local links vary over time, but also the overall available bandwidth of a network changes hourly and daily due to the rhythms of working and sleeping hours, and working days and weekends. Hence, in the next experiment, we compared the performance of DSL and *oStream* under different global network bandwidths. Their segment loss rates are depicted in Fig. 9, where the overall available bandwidth of the network was gradually reduced from 100% to 70% of the base setting.

Not surprisingly, for both DSL and *oStream*, SMR increases as the overall bandwidth decreases. However, the DSL overlay is much less vulnerable to bandwidth reductions, and the rate of increase of SMR is generally lower than that of *oStream*, especially when the reduction is over 20%. For example, for a reduction of 30%, the SMR of *oStream* has reached 0.4, or 40% of the segments have been lost or missed the playback deadline; yet the SMR of DSL is still no more than 0.1. We suggest that this is because *oStream* explores the available bandwidth at a small subset of network links only, so bandwidth reduction at an internal link of the tree could result in loss multiplicity in all downstream nodes. On the other hand, a node in DSL has multiple potential suppliers and the network coding makes effective use of their available bandwidths. In our experiments, we have observed that over 95% of nodes in DSL have more than two suppliers and the loads are fairly shared among them. Fig. 10 depicts the streaming quality as a function of node failure percentage for DSL and *oStream*, reaffirming that multi-supplier scheduling with network coding greatly enhances the robustness of the system.

D. Impact of VCR Interactions and Buffer Size

We also examined streaming quality with VCR interactions, in particular, fast-forward and rewind. The current version of *oStream* does not support these operations. To enable comparison, we implemented an enhanced *oStream*, in which a jump operation is realized through moving through the links in the tree structure either in the direction of the parent (for forwarding) or the descendant (for rewind). This distributed search avoids repeated contacts with the server. The latter is clearly non-scalable for frequent VCR operations in large overlays.

We first randomly picked up 10% of clients to perform fast-forward or rewind operations with 2x speed. The duration of each VCR operation varied from 5% of the stream length to 40%. Fig. 11 shows streaming quality for the DSL and the enhanced *oStream* overlays. Clearly, DSL outperforms *oStream*, and its quality is almost

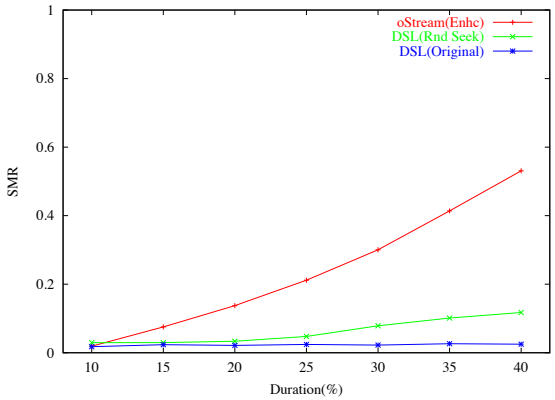


Fig. 11. Segment missing rate as a function of VCR duration for DSL and oStream.

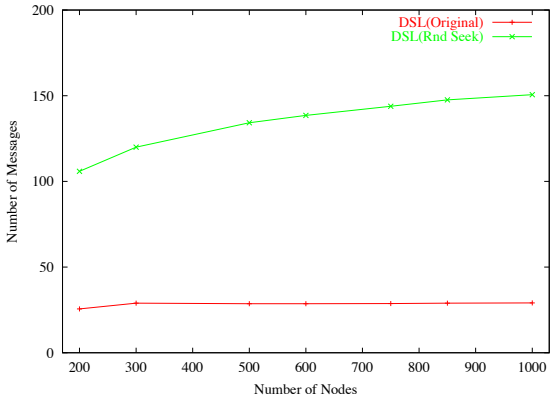


Fig. 12. Message cost per jump operation.

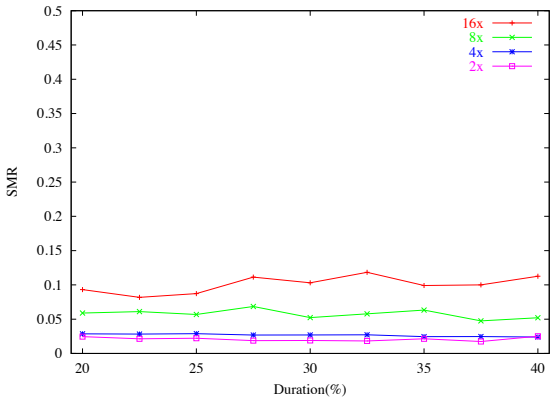


Fig. 13. Segment missing rate as a function of VCR speeds for a 1000-node DSL overlay.

independent of the duration of the VCR operations. On the other hand, the quality of the enhanced *oStream* quickly becomes worse as duration increases, and is generally unacceptable for a duration of greater than 20% of the stream length. Intuitively, for a tree overlay like *oStream*, the node initiating fast-forwarding or rewinding may still find the expected data segments in the buffers of its parent or close ancestors, but such an inefficient linear search will soon cause it to suffer from buffer outage.

As mentioned before, each jump can be implemented through a random-seek operation as well, i.e., leave and re-join with the updated offset. To understand its (in)effectiveness, we also plotted streaming quality in this implementation. As shown in Fig. 11, its performance is generally acceptable with short VCR durations, but becomes worse with increased duration. This is mainly because random-seek needs a much higher cost and hence longer time to identify subsequent suppliers, and the lag accumulates over time. Such costs are compared in Fig. 12. Random-seek also involves contact with the server, which, as discussed before, does not scale well.

Fig. 13 further demonstrates the impact of VCR speed. Again, we can see that streaming quality for the DSL overlay is reasonably good at low and medium VCR speeds (2x to 8x). There are more losses with larger variances for 16x speed. As discussed in Section III, at this speed, more links in the higher layers could be accessed; some of these layers are not well-balanced, which leads to inaccurate jumps and hence more segment losses. Nonetheless, a speed over 16x is rarely useful in the opinion of a viewer [29], and, under high speeds, the quality of other schemes is indeed much worse. Our experience is that when the speed goes beyond 4x, the quality of the enhanced *oStream* is already unacceptable.

The buffer size of each client is another key parameter in application-layer overlay streaming. Though it would be desirable if every overlay node were to cache the whole video stream, it is often impractical given the large data volume. The results of the previous experiments have suggested that the default buffer size, 10% of the video stream, is sufficient to achieve low segment loss rates when only asynchronous requests are being made. In Fig. 14, we depict streaming quality with different buffer sizes in the presence of VCR operations. In this 1000-node overlay, the default buffer size again worked reasonably well, and the improvement with larger buffer sizes is marginal, particularly at low and medium speeds. It is also worth noting that, with the default setting, the computation time for the scheduling algorithm is less than 10 ms, which is acceptable for real-time adaptation.

VII. Conclusion and Future Work

In this paper, we have introduced a novel data structure, Dynamic Skip List (DSL), for on-demand overlay media streaming. A DSL is a randomized and distributed structure, which inherently accommodates node

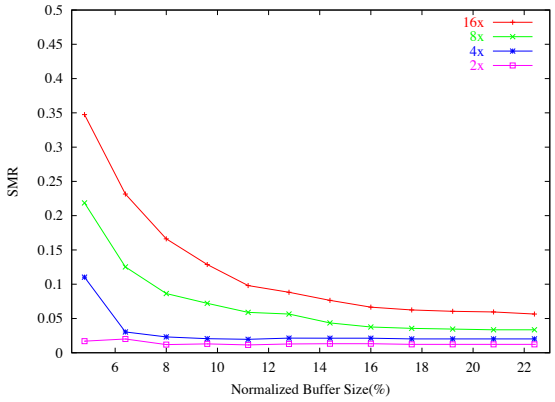


Fig. 14. Segment missing rate as a function of buffer size for different fast-forwarding/rewinding speeds.

dynamics and asynchronous requests. We have discussed the practical issues involved in realizing a DSL-based streaming overlay, and presented an optimal algorithm for multi-peer data scheduling with linear network coding. Furthermore, we have shown that all typical VCR operations can be implemented in this overlay with $O(1)$ or $O(\log N)$ message costs.

The performance of the DSL-based overlay has been examined under different network and client configurations, and the results re-affirm its excellent scalability and robustness. Its streaming quality is reasonably good with asynchronous requests and frequent VCR operations, while the latter has seldom been supported in existing overlay systems.

We are building a DSL prototype, and expect that this will enable further practical issues to be identified and solved. We are also interested in incorporating advanced network coding or video coding algorithms; an example is the Multiple Description Coding (MDC) [28], which is a good match to the DSL structure and could further improve its robustness.

REFERENCES

[1] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network Information Flow", *IEEE Transaction on Information Theory*, vol. 46, pp. 1204-1216, Jul. 2000.

[2] K. Almeroth and M. H. Ammar, "The Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 6, pp. 1110-1122, Aug. 1996.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. ACM SIGCOMM'02*, Pittsburgh, PA, Aug. 2002.

[4] M. Castro, P. Drushel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *Proc. ACM SOSP'03*, New York, NY, Oct. 2003.

[5] Y. Chu, S. Rao, and H. Zhang, "A Case for End System Multicast," in *Proc. ACM SIGMETRICS'00*, Santa Clara, CA, Jun. 2000.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, Second Edition, MIT Press, Cambridge, MA, 2001.

[7] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 91-106, Jan. 2004.

[8] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming Live Media over Peer-to-Peer Network," *Technical Report*, Stanford University, 2001.

[9] T. Do, K. Hua, and M. Tantaoui, "P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment," in *Proc. IEEE ICC'04*, France, Jun. 2004.

[10] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proc. IEEE INFOCOM'05*, Miami, FL, Mar. 2005.

[11] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-Peer Patching Scheme for VoD Service," in *Proc. ACM WWW'03*, Hungary, May, 2003.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Predication*, Springer, New York, NY, 2001.

[13] M. Hefeeda and B. Bhargava, "On-Demand Media Streaming over the Internet," in *Proc. IEEE FTDCS'03*, Puerto Rico, May. 2003.

[14] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *Proc. IEEE ISIT'03*, Japan, Jun. 2003.

[15] A. Hu, "Video-on-Demand Broadcasting Protocols: A Comprehensive Study," in *Proc. IEEE INFOCOM'01*, Anchorage, AK, Apr. 2001.

[16] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. ACM SOSP'03*, New York, NY, Oct. 2003.

[17] Z. Li, B. Li, D. Jiang, and L. Lau, "On Achieving Optimal Throughput with Network Coding," in *Proc. IEEE INFOCOM'05*, Miami, FL, Mar. 2005.

[18] W. Liao and V. Li, "The Split and Merge Protocol for Interactive Video-on-Demand," *IEEE Multimedia*, vol. 4, no. 4, pp. 51-62, Oct. 1997.

[19] H. Ma, K. Shin, and W. Wu, "Best-Effort Patching for Multicast True VoD Service," *Kluwer Multimedia Tools and Applications*, vol. 26, no. 1, pp. 101-122, 2005.

[20] M. Medard, S. Acedanski, S. Deb, and R. Koetter, "How Good is Random Linear Coding Based Distributed Networked Storage?" in *Proc. NETCOD'05*, Italy, Apr. 2005.

[21] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, NY, 1995.

[22] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proc. ACM NOSSDAV'02*, Miami, FL, May, 2002.

[23] W. Pugh, "Skip Lists: A Probabilistic Alternative to Balanced Trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668-676, Jun. 1990.

[24] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," in *Proc. ACM SIGCOMM'04*, Portland, OR, Aug. 2004.

[25] R. Rajaie and A. Ortega, "PALS: Peer to Peer Adaptive Layered Streaming," in *Proc. ACM NOSSDAV'03*, Monterey, CA, Jun. 2003.

[26] D. Tran, K. Hua, and T. Do, "A Peer-to-Peer Architecture for Media Streaming," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 121-133, Jan. 2004.

[27] D. Wang and J. Liu, "A Dynamic Skip List Based Overlay for Asynchronous Media Streaming," *Technical Report*, Simon Fraser University, May, 2005.

[28] Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video Processing and Communications*, Prentice Hall, 2001.
[29] B. Wildemuth, G. Marchionini, M. Yang, G. Geisler, T. Wilkens, A. Hughes, and R. Gruss, "How Fast is too Fast? Evaluating Fast Forward Surrogates for Digital Video," in *Proc. JCDL'03*, Houston, TX, May, 2003.
[30] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *Proc. IEEE INFOCOM'96*, San Francisco, CA, Mar. 1996.
[31] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DoNet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming," in *Proc. IEEE INFOCOM'05*, Miami, FL, Mar. 2005.
[32] M. Zhou and J. Liu, "Tree-Assisted Gossiping for Overlay Video Distribution," to appear in *Kluwer Multimedia Tools and Applications*, 2005.
[33] Y. Zhu, B. Li, and J. Guo, "Multicast with Network Coding in Application Layer Overlay Networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 107-120, Jan. 2004.

APPENDIX

We now provide a detailed analysis of the jump operation for fast-forward; in particular, we will derive the range for n, the number of segments to be retrieved from a supplier at a fast-forward speed v. We consider the following physical constraints, most of which are adapted from [24]:

B: The buffer size of each client.
T: The connection time from one client to the other.
t: The playing time for each data segment.

We also assume that the downloading time of each segment is smaller than t, which is the basic requirement for continuous streaming.

Lemma 11: For fast-forwarding speed v, the maximum number of data segments that a supplier can provide is bounded by $n^+ = \frac{B(B-v)}{vB-B}$.

Proof: n^+ consists of two parts: the first is $n_1 = \frac{B}{v}$, which are the segments already in the buffer, and the second are the data segments that are downloaded while playing the n_1 data segments. We now focus on the second part. Since the supplier advances its playback at a normal 1x speed, once the n_1 segments have been played, another n_1 segments will be downloaded as long as the downloading speed is faster than the playing speed. Inductively, the total number of segments that a supplier can provide becomes $n^+ = B\left(\frac{1}{v} + (\frac{1}{v})^2 + (\frac{1}{v})^3 + \cdots + (\frac{1}{v})^I\right)$, where I denotes the number of iterations, and $\frac{B}{v^I} = 1$, or $I = \lfloor \log_v B \rfloor$. Therefore, we have $n^+ = \frac{B(B-v)}{vB-B}$. A special case is $v = 1$, where the number of data segments that can be retrieved from the supplier is unlimited; i.e. the client can always stay with their current supplier. ■

On the other hand, since after playing n data segments, the client must connect to the next supplier to ensure continuous playback, we have $nt \geq T$, and the range of n is thus $\left[\frac{T}{t}, \frac{B(B-v)}{vB-B}\right]$.

Theorem 12: The maximum speed that the system can provide is no higher than $\frac{tB^2-TB}{TB-tB}$.

Proof: Directly from $\frac{T}{t} \leq n \leq \frac{B(B-v)}{vB-B}$ for a valid n. ■