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

Dan Wang, *Student Member*, *IEEE*, and Jiangchuan Liu, *Member*, *IEEE*

Abstract—Media distribution through application-layer overlay networks has received considerable attention recently, owing to its flexibility 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 that inherently accommodates 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 sublinear to the client population. The model also seamlessly integrates a smart data scheduling algorithm using linear network coding, yielding fast and robust downloading from multiple suppliers. Our simulation results show that the DSL-based overlay is highly scalable. It delivers reasonably smooth playback with diverse client interactivities while keeping the computation and bandwidth overheads low.

Index Terms—Distributed systems, peer-to-peer networks, video on-demand streaming.

1 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, however, remains a challenging task due to limited server capacity and, to date, weak IP multicast support. Recently, overlay streaming has emerged as a promising solution to this problem [4], [6]. 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 that 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 the autonomous nodes that join or leave the overlay at will or even crash without notification. The problem is further complicated when introducing on-demand playback requests and, more general, such VCR interactions 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. This is difficult to achieve through a centralized entity, because VCR operations are more frequently invoked than node joining or leaving and often persist for a long duration. As a result, the VCR operations have seldom been incorporated in existing overlay streaming systems.

In summary, the on-demand streaming overlay is expecting a *distributed*, *scalable*, and *robust* structure for asynchronous client requests and VCR interactions. To this end, we propose the Dynamic Skip List (DSL), a novel data structure that effectively realizes the above demands. 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) both the search cost and the state information kept at a node are sublinear (constant or logarithmic) to the DSL size, suggesting good scalability, and 3) its parallel logical links have inherent power to support multipeer collaboration in an overlay network. We also present a linear network-coding-based algorithm for data scheduling, which yields optimal data downloading from the multiple peers in the DSL.

We go on to demonstrate a practical overlay network that seamlessly integrates indexing and data distribution through a DSL and discuss the key issues involved in realizing this DSL-based overlay. The performance of the DSL-based streaming overlay has been examined under different network and client configurations. The preliminary results show that it achieves a reasonably stable streaming rate with a low control overhead. It scales well

- D. Wang is with the School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada, V5A 1S6 and the Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. E-mail: danw@cs.sfu.ca.
- J. Liu is with the School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada, V5A 1S6. E-mail: jcliu@cs.sfu.ca.

Manuscript received 21 June 2006; revised 12 Apr. 2007; accepted 20 June 2007; published online 25 July 2008.
Recommended for acceptance by P. Mohapatra.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0163-0606.
Digital Object Identifier no. 10.1109/TPDS.2007.70748.

and accommodates highly dynamic client behaviors with limited buffer spaces. More importantly, it effectively supports diverse VCR operations at reasonably low costs, which is difficult to achieve in existing systems.

The remainder of this paper is organized as follows: Section 2 presents the background of our work and reviews the related work. The system model is described in Section 3, together with an overview of the DSL. In Section 4, we discuss the construction and maintenance of the DSL-based overlay, as well as its support for VCR operations. We further explore the multisupplier data scheduling in Section 5. The performance of the DSL-based streaming overlay is examined in Section 6. Section 7 concludes the paper and offers some future research directions.

2 Background and Related Work

Over the last decade, many proposals for multichannel on-demand broadcasting have been put forward [17]. They have explored the temporal locality of client requests, allowing a media server to accommodate concurrent client requests through batching, patching, or periodic broadcasting. Although VCR interactions could be implemented in some of these protocols, they will consume significant server or network resources [2], [20], [21]. Moreover, 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, due to a range of practical and political issues such as the lack of incentives to carry multicast traffic.

Recently, application-layer solutions have received much attention, owing to their readily deployable nature and the massive buffering capacity at state-of-the-art personal computers (PCs). 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; for example, NICE [4], SpreadIt [9], and ZIGZAG [30]. Constructing and maintaining an efficient distribution tree among the overlay nodes is a key issue for these systems. To address the unbalanced load or vulnerability of a tree, enhancements have also been proposed. Examples of this include building a mesh-based tree (Narada and its extensions [6] and Bullet [18]), maintaining multiple distribution trees (Split-Stream [5]) or gossip partners (CoolStreaming [36]), and leveraging layered coding (PALS [27]) or multiple-description coding (CoopNet [24]). Our DSL-based overlay also supports multipeer data downloading, yet 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 (for example, [8], [10], [12], and [15]) 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 [15] and oStream [8] suggest using a centralized server to record 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 a large overlay. It becomes a bottleneck when frequent VCR operations are introduced. To solve this problem, P2VoD [10] and TAG [37] logically organize the nodes into a linear or tree structure. With the playback offset being an indexing key, these structures assist random seek in a distributed manner. However, a linear structure is not capable of assisting asynchronous accesses in sublinear time; on the other hand, a tree structure requires complicated rebalancing operations after client joining, leaving, or seeking. Moreover, neither of them well supports fast-forward and rewind. These issues are addressed in our proposed DSL structure.

Yin et al. [34] proposed TrustStream, a novel architecture for secured media streaming distribution. Their main focus is on preventing unauthorized clients from receiving the media content. To address the scalability issue, they also adopted a multilayered architecture, with cluster headers being elected for a different layer. Content delivery for cluster headers is based on a multicast tree, whereas peer-to-peer content sharing is used within clusters. Our DSL differs from TrustStream in that we focus on VCR interactions in peer-to-peer media distribution, and the layers in the DSL enable efficient jumping for such interactions.

Finally, it is worth noting that the basic skip list and its extensions have recently been applied to practical network applications [3], [13]. Specifically, Harvey et al. [13] have shown a scalable peer-to-peer network using SkipNet, a generalization of the skip list. Their focus is mainly on file indexing service, as well as content and path locality. The objective and, hence, solutions are different from the on-demand streaming applications targeted in our study.

3 Dynamic Skip List

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 could even crash without notification. Their playback requests are generally asynchronous with different starting offsets. In addition, various VCR operations are to be supported, 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 considering 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.

TABLE 1
List of Notations

Notation	Definition
N	Overlay size (number of clients)
L	Top layer boundary
v	Fast-forward or rewind speed
B	Client buffer size
t	Playback time of a data segment
T	Connection time to another client
n	Number of segments to be retrieve from a supplier
d	Average difference of playback offsets between two clients
S	Number of suppliers in multi-supplier scenario
b_j	Available bandwidth of supplier j
c_i	Original data segments
f_i	Combined data segments using network coding
w	The cardinality of f_i (the number of different original data segments it includes)
m_s	Number of segments retrieved from a supplier s
β_i	Linear coding coefficient
q	Finite field size

We now introduce the DSL, which enables both types of function and effectively balances the speeds and costs. In this section, we first establish the theoretical foundation of the DSL; its use in the asynchronous streaming overlay will be detailed in the following two sections. Some major notations used in this paper are summarized in Table 1.

3.1 Overview of Skip List

We first give an overview of a basic skip list. A skip list is an ordered list of *keys* with additional parallel links. The key in a skip list can be any sortable property of an object, for example, its time, size, or weight. Given that here, we are interested in supporting VCR interactions in a VoD overlay, the playback offset of each node is a natural choice for its key. Assume that 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 hits a logical node having the key. The complexity of this top-down search is $O(\log N)$ [25].

A skip list can also be constructed in a random fashion [25]: 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 logical 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. It is known that in expectation, this randomized version has the same search performance as the deterministic version when the MaxLayer is set to $\log(N)$.

3.2 Dynamic Skip List

Compared to other typical indexing structures such as an AVL tree or a B+ tree, a randomized skip list is significantly

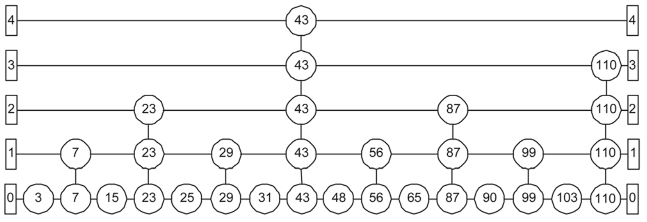

Fig. 1. A regular skip list of 16 nodes. The number in each node represents the key of this node.

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 basic skip list in a streaming overlay with dynamic and interactive clients, however, is not straightforward. First, the number of keys in a skip list has to be predefined and so does the value of the MaxLayer. Second, 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. Third, the random promotion could generate unbalanced layers (especially in the higher layers), which is to say, layers with far more or less logical nodes than might be expected. This greatly reduces the maintenance and search efficiency.

Our 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 randomized skip list (RSL), but there is no MaxLayer limit; a newly inserted key will stop promoting itself only when it fails. As such, the size of the DSL does not have to be predefined.

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

$$E(Y_l^i) = \frac{1}{2^l} \times 1 + \left(1 - \frac{1}{2^l}\right) \times 0 = \frac{1}{2^l}$$

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 $L = \log(\frac{N}{\log N})$, 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's Inequality [23], we have

$$Pr\left[Y_l < E(Y_l) \times \left(1 - \frac{1}{C}\right)\right] < e^{-E(Y_l)(\frac{1}{C})^2}.$$

In layer L, since $E(Y_L) = \frac{N}{2^L} = \log N$, we have

$$e^{-E(Y_L)(\frac{1}{C})^2} = e^{-\log N(\frac{1}{C})^2} = \left(\frac{1}{N}\right)^{(\frac{1}{C})^2 \log_e 2} = O\left(\frac{1}{N}\right)^{\epsilon^-}.$$

The case for $Y_l > E(Y_l) \times (1 + \frac{1}{C})$ can be proved similarly. □

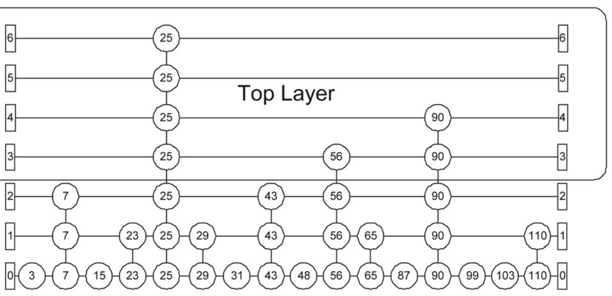

Fig. 2. A DSL of 16 nodes. The number in each node represents the key of this node.

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 $L = \log(\frac{N}{\log N})$ 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, which reduces the search efficiency.

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

Lemma 3. *The number of layers in a DSL after compression is* $\log(\frac{N}{\log N})$*, and the expected number of logical nodes in the top layer is* $2 \log N$.

Proof. In DSL, all the nodes above $L = \log(\frac{N}{\log N})$ 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 has it to be known a priori. Assuming that $\hat{Y}_l$ is the number of logical nodes 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 [14]. □

3.3 Mapping between DSL and Asynchronous Overlay

The mapping between a DSL and a streaming overlay with asynchronous clients is straightforward: The playback offset of a client serves as its key in the DSL, and all logical nodes associated with this key 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 playback distances and, hence, the DSL structure will not change over time, unless a client joins, leaves, or crashes or a VCR operation is 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 the 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 in the system, including the server, maintains at most $O(\log N)$ extra information.

4 DSL-Based Overlay Construction, Maintenance, and VCR Operations

In this section, we detail the construction and maintenance of a DSL-based overlay. We also discuss the practical issues toward supporting all the typical VCR operations.

4.1 Join Operation

We assume that the content server persists all the time, serving as an anchor node in the overlay. 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 (which has an 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 $O(\log(\frac{N}{\log N}))$ 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* $O(\log(\frac{N}{\log N}))$.

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 $O(\log(\frac{N}{\log N}))$, the expected message cost is also bounded by $O(\log(\frac{N}{\log N}))$. □

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

4.2 Leave Operation and Failure Recovery

A client that is scheduled to leave the overlay should first notify its neighbors in the DSL such that they can reconnect with each other to form new neighborships.

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

Proof. For a client whose 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 the expected number of nodes in layer l is $\frac{N}{2^l}$, the average number of messages for a departure is

$$\frac{1}{N}\left(\sum_{l=0}^{\log\left(\frac{N}{\log N}\right)} l\frac{N}{2^l}\right) = \frac{1-(\frac{1}{2})^{\log\left(\frac{N}{\log N}\right)}}{\frac{1}{2}} = 2 - \frac{\log N}{N},$$

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 although the maximum extra information kept at a 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 help the affected neighbors 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 $O(\log N)$.

4.3 VCR-Like Interactions

Since the cost for a leave and then rejoin with a new playback offset is only $O(1) + O(\log(\frac{N}{\log N})) = O(\log N)$, this combination can be used to implement most typical VCR operations.

4.3.1 Pause and Resume

The client can simply stop playback but may still stay in the overlay, accepting and supplying data at the normal speed. If the pause time is very long and its buffer overflows, it can temporarily leave the overlay. Once the resume command is given, it can rejoin with the original offset.

4.3.2 Random Seek

The client can simply leave and then rejoin the overlay with the new offset after seeking.

4.3.3 Fast-Forward and Rewind

Assuming that the fast-forward or rewind speed is v, these two operations can be realized by playing one segment out of v segments [33]. This can be implemented through leave and then rejoin as well. However, since a fast-forward or rewind movement generally consists of a long series of such jump operations, more efficient solutions are expected.

Given that an overlay node has a size-limited buffer, after jumping $v - 1$ segments, the fast-forwarding or rewinding client will either stay with its current supplier or move to a new one if the expected segment is not within the current supplier's buffer. The key issue of a jump operation is thus to locate the suppliers with the expected segments in time. We note that the DSL structure provides effective support toward this operation through its horizontal links, which enable a client to skip unnecessary nodes (and, hence, segments) at a fairly stable speed. 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 a symmetric transform.

We assume that the playback offsets of two consecutive nodes in the overlay differ by d segments on the average, and the client can retrieve n data segments from its supplier with VCR speed v. Then, following lemma gives 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 that 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-forward needs $O(1)$ time.*

Proof. The average distance between two nodes in a DSL layer $\log(\frac{n(v-1)}{d})$ 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 or 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 at regular distances.

In practice, $\log(\frac{n(v-1)}{d})$ might not be an integer. In this case, the client can temporarily move along higher or lower layer links or adaptively set n, that is, stay for a longer or shorter period with the current supplier, to achieve an average speed of v. We now derive the range for n, the number of segments to be retrieved from a supplier at a fast-forwarding speed v, and the maximum fast-forwarding speed that the overlay can support.

We consider the following physical constraints, which are adapted from [26]: B, the buffer size of each client; T, the connection time from one client to the other; and t, the playing time for each data segment. We also assume that the downloading time of each segment is smaller than t, which is a basic requirement for continuous streaming.

Lemma 9. *For fast-forwarding speed v, the number of data segments that a supplier can provide is at most $n^+ = \frac{B(B-v)}{vB-B}$.*

Proof. n^+ consists of two parts: the first is $n_1 = \frac{B}{v}$, which are the segments already in the supplier's 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. Therefore, the total number of segments that a supplier can provide is $n^+ = B(\frac{1}{v} + (\frac{1}{v})^2 + (\frac{1}{v})^3 + \cdots + (\frac{1}{v})^I)$, where I denotes the number of iterations, $\frac{B}{v^I} = 1$, or $I = \lfloor \log_v B \rfloor$. It follows that $n^+ = \frac{B(B-v)}{vB-B}$. A special case is $v = 1$ (1x speed or normal playback), where the number of data segments that can be retrieved from the supplier is unlimited; that is, the client can always stay with their current supplier(s). □

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 $[\frac{T}{t}, \frac{B(B-v)}{vB-B}]$.

Theorem 10. *The maximum speed that the system can provide is no higher than* $\frac{tB^2-TB}{TB-tB}$.

Proof. The proof can be seen directly from $\frac{T}{t} \leq n \leq \frac{B(B-v)}{vB-B}$ for a valid n. □

Nevertheless, a speed of no more than 32x can be easily achieved even in a small to medium DSL overlay (say, less than 500 nodes), and, as investigated in [33], a higher speed is rarely perceived as useful by users, nor is it supported in most commercial VHS or DVD players.

5 Multisupplier Data Scheduling

In a DSL overlay, iteratively searching its neighbors along the base layer links, a client can easily locate others with overlapped buffers, and all of them can be potential suppliers. In this section, we generalize the model of data delivery to this multisupplier scenario, which is clearly more efficient and robust for heterogeneous and dynamic clients. However, a scheduling algorithm is necessary in order to retrieve the expected segments without duplication.

5.1 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 [7]. 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*, which was first proposed in some theoretical studies [1], [16], [19], [38]. Following these works, many practical systems based on network coding have been built and evaluated recently [11], [28]. In particular, it has been shown [29] that network coding can be supported in hardware with negligible overhead. Intuitively, network coding allows a node to retrieve fractionized segments or jobs in the parallel machine scheduling context. Given such a relaxation, we now show that provided that 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 network 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 coefficient 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:

```
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.

Lemma 11. *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.

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, that is, useful data segments that have not been downloaded yet, and if so, it will retrieve them accordingly.

Theorem 12. *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. Assuming that 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. □

5.2 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 requests. We note that their volume is two or more orders of magnitude lower than that of the media data and also that they can be piggybacked by the data segments. Our experimental

TABLE 2
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

results show that the extra bandwidth required for such data is far less than 1 percent, which can be negligible in practice. The remaining major practical concerns are the computation overhead of network coding and the linear dependency of coefficients.

5.2.1 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. More importantly, we only need to decode a very small part of the media stream each time [28], [32]. As an example, for a buffer of 15 Mbytes and a segment of 128 Kbytes, even if all the data segments are combined, we have $k \cong 100$. In our experiments, the computation time is less than 1 ms in a typical Pentium IV 2.8-GHz PC.

5.2.2 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 [22]. This is verified in Table 2, 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 the performance of the scheduling algorithm was rarely affected by linear dependency.

6 Performance Evaluation

In this section, we present our performance evaluation for the DSL-based overlay. Although the results remain preliminary, they have reaffirmed the salient features of the DSL-based overlay: good scalability and low overhead with VCR operations.

6.1 Network Configuration

We use the GT-ITM [35] topology generator to produce networks for our simulation. In this section, we present representative results based on the following settings: The network consisted of four transit domains, each with five transit nodes, and a transit node is connected to six stub domains, each with eight stub nodes. We then add 16 stub nodes to produce a 1,000-node network. The default bandwidth settings between two stub nodes, a transit and a stub node, and two transit nodes are 512 Kbps, 1 Mbps, and 1.5 Mbps, respectively. Such bandwidths are only upper bounds, and the available bandwidth might be lower and vary over time, as discussed in the following section.

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 10 times to mitigate the effect of randomness. The streaming rate was 256 Kbps, and the length of the stream was 150 minutes, both of which are typical for Internet-based video delivery. The default size of the client-side buffer was 15 Mbytes, which can be easily accommodated in state-of-the-art PCs. We also investigated the impact of different buffer sizes in the experiments.

6.2 Cross Traffic and Client Dynamics

To emulate a dynamic network environment, we injected cross traffic to the network. The flows of the cross traffic were randomly placed between node pairs, using the shortest path routing. Each flow followed an ON/OFF model, and the traffic intensity during the ON period was distributed from 20 percent to 80 percent of the bottleneck-link bandwidth between the two nodes. The total volume of the cross traffic varied in the experiments, and its impact will be discussed in the following sections. Each client also changed its status following an ON/OFF model: it actively participated in the overlay during an ON period and left during an OFF period. Both ON and OFF periods were exponentially distributed with an average of 900 seconds, which was $\frac{1}{10}$ of the total stream length.

6.3 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 were measured in the number of messages exchanged, reflecting both the bandwidth consumption and the execution time of an operation.

6.3.1 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 an RSL for overlay construction. It can be seen that for the join operation, the overheads for DSL and the RSL are reasonably close, both following a logarithmic function of the overlay size. DSL is slightly better, owing to the top layer compression.

However, the message distribution among the overlay nodes can be quite different for the two structures. Fig. 5a depicts the total message cost per node, summed over 1,000 join operations. For the RSL, the message distribution is highly skewed: about 90 percent of the messages are at eight nodes, which corresponds to the high-layer logical nodes in the RSL. In particular, over 5,000 messages are at a single node (ID: 408). Such overloaded nodes can be quite vulnerable, and VCR operations will further aggravate this problem. On the contrary, as shown in Fig. 5b, the distribution in the DSL is much more uniform, which suggests better scalability and stability. There are other deficiencies of the RSL, such as the weak support to overlays of dynamic sizes. Hence, in the

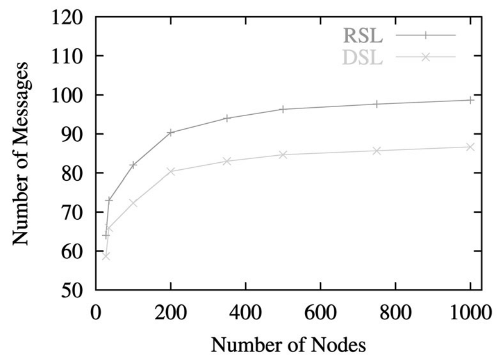

Fig. 4. The maximum number of messages per join operation for RSL and DSL.

following experiments, we focus on the DSL-based overlay only.

6.3.2 *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 4. We also recorded the maximum cost, which is logarithmically related to the overlay size. Intuitively, the maximum cost occurs when a node in the highest layer of DSL leaves; such a node maintains $O(\log N)$ neighboring links, and all of them have to be reconnected afterward.

To investigate the cost for failure recovery, we randomly failed 10 percent, 20 percent, and 30 percent 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 overlay sizes. These results suggest that a DSL overlay is stable and scalable in terms of control overhead.

6.4 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* [8], 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 the global information of the overlay, which facilitates client join or failure recovery. Details of *oStream* can be found in [8].

We focused first of all on streaming quality with asynchronous playback requests. 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

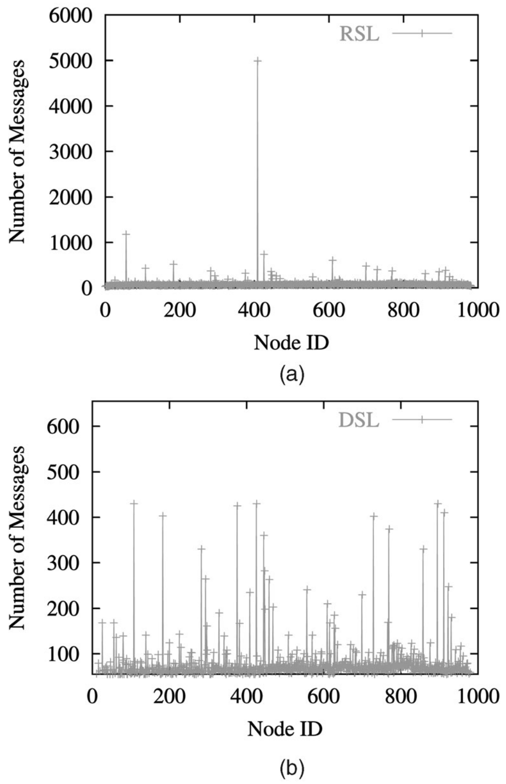

Fig. 5. Message distribution among the overlay nodes for 1,000 join operations. (a) RSL. (b) DSL.

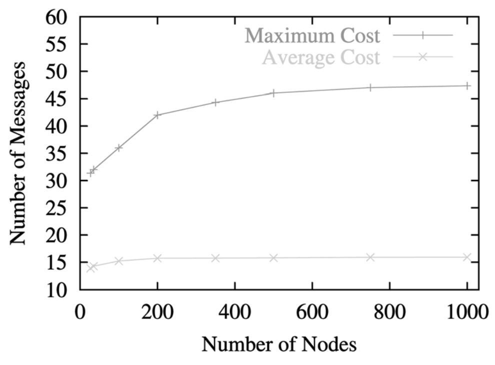

Fig. 6. Message cost (max and average) per leave operation for different DSL overlay sizes.

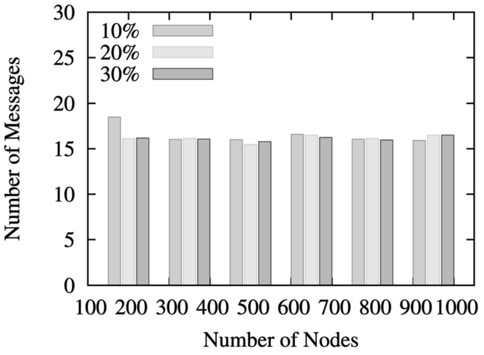

Fig. 7. Message cost per failure recovery for different DSL overlay sizes.

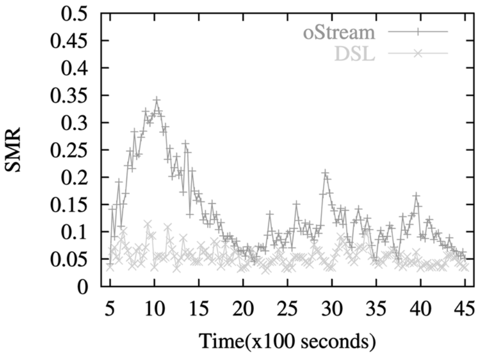

Fig. 8. SMR for DSL and oStream with local bandwidth fluctuation.

the total available bandwidth of the network remained at 80 percent of the base setting (with no cross traffic).

Fig. 8 plots the segment loss rates (SMRs) for 1,000-node DSL and *oStream* overlays during a 4,500-second 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 [31]. On the other hand, the loss rate of *oStream* greatly fluctuates over time, with a peak value as high as 0.35, resulting in poor video quality. This is mainly because *oStream* relies on a specific tree structure for streaming, so bandwidth reduction at an internal link of the tree, particularly at those close to the root, could result in a severe loss across many descendants.

We are also interested in the performance of individual nodes. In Fig. 9, we plot the percentage of missing segments for each node. We can see that 98 percent of nodes have an SMR of less than 5 percent. The variance across 99 percent of nodes is 14 percent of the total segments, which is reasonably low. We do notice that there are a few nodes suffering from higher average loss rates. By carefully investigating our log files, we find that most of these nodes join the overlay in its initial stage, and a large number of losses occur during that period. In other words, in the initial stage when the overlay is small, there can be higher quality fluctuation. We notice that this is an intrinsic problem of peer-to-peer communication and happens in many other systems. A possible solution is to use a hybrid architecture,

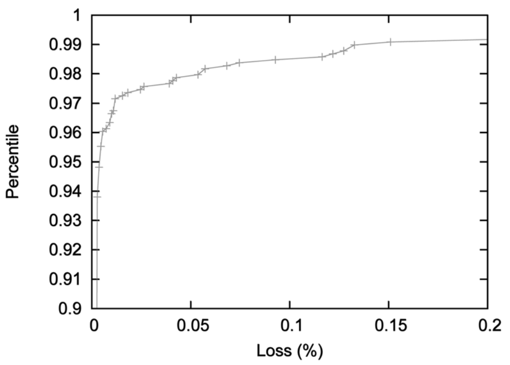

Fig. 9. Segment missing for each node in the DSL overlay.

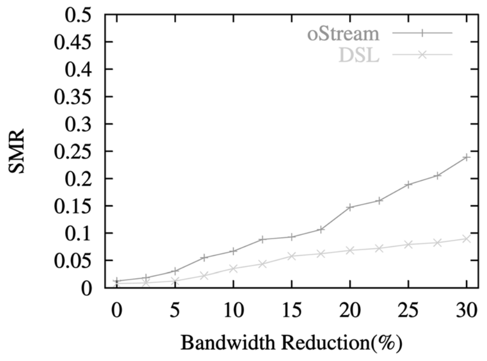

Fig. 10. SMR for DSL and oStream under global bandwidth reductions.

which provides stronger support from the server during the initialization procedure.

We are also interested in the behavior of each node. In Fig. 9, we plot the percentage of missing segments for each node. We see that 98 percent of nodes have less than 5 percent of missing segments. The number of nodes who suffered a larger loss is very small. By carefully investigating our log file, we find that these were the nodes that joined the overlay earlier. We suggest that at the initial stage where the overlay is small, there could be higher fluctuation. We believe further improvement is possible and can be an interesting future work.

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. 10, where the overall available bandwidth of the network was gradually reduced from 100 percent to 70 percent 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 percent. For example, for a reduction of 30 percent, the SMR of *oStream* has reached 0.25, or 25 percent of the segments have been lost or missed the playback deadline, yet the SMR of DSL is still less than 0.1. We conjecture 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 more than 95 percent of nodes in DSL have more than two suppliers, and the loads are fairly shared among them. Fig. 11 depicts the streaming quality as a function of node failure percentage for DSL and *oStream*, reaffirming that multisupplier scheduling with network coding greatly enhances the robustness of the system.

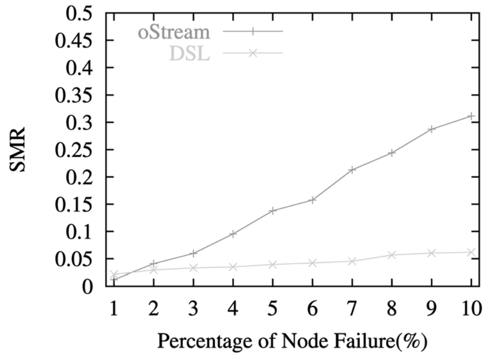

Fig. 11. SMR for DSL and oStream under different node failure rates.

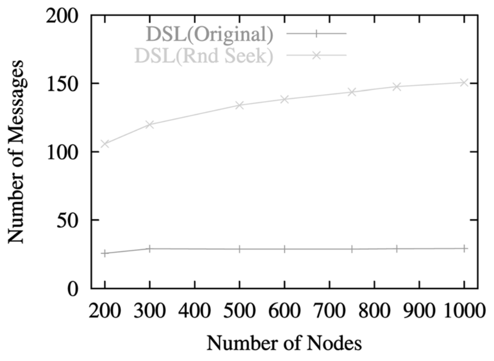

Fig. 13. Message cost per jump operation.

6.5 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 by 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 nonscalable for frequent VCR operations in large overlays.

We first randomly picked up 10 percent of the clients to perform fast-forward or rewind operations with 2x speed. The duration of each VCR operation varied from 5 percent of the stream length to 40 percent. Fig. 12 shows streaming quality for the DSL and the enhanced *oStream* overlays. Clearly, DSL outperforms *oStream*, and its quality is almost 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 greater than 20 percent 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, that is, leave and then rejoin with the updated offset. To understand its (in)effectiveness, we also plotted streaming quality in this implementation. As shown in Fig. 12, 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. 13.

Fig. 14 further demonstrates the impact of VCR speed. Again, we can see that the streaming quality for the DSL overlay is reasonably good at low and medium VCR speeds (2x to 8x). There are more losses and larger variances for the 16x speed. As discussed in Section 3, 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. Nevertheless, a speed greater than 16x is rarely useful in the opinion of a viewer [33], 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. Although 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 percent of the video stream, is sufficient to achieve low segment loss rates when only asynchronous requests are being made. In Fig. 15, we further depict streaming quality with different buffer sizes in the presence of VCR operations. In this 1,000-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

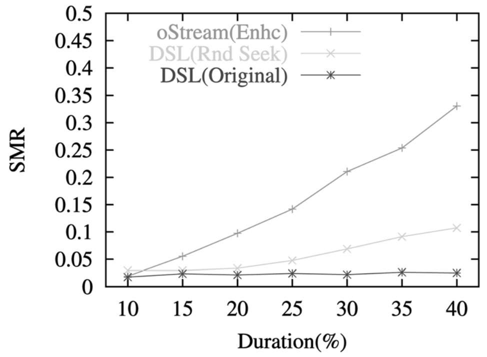

Fig. 12. SMR as a function of VCR duration for DSL and oStream.

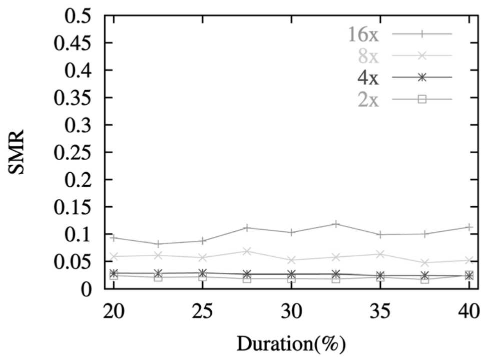

Fig. 14. SMR as a function of VCR speeds for a 1,000-node DSL overlay.

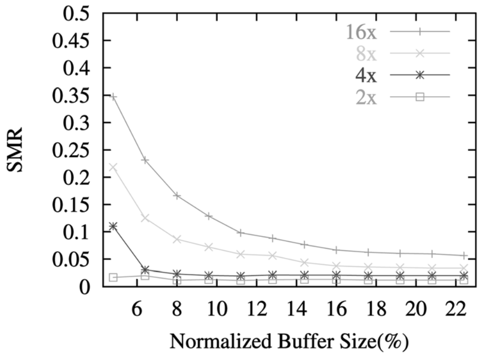

Fig. 15. SMR as a function of the buffer size for different fast-forwarding/rewinding speeds.

worth noting that with the default setting, the computation time for the network coding is less than 1 ms, which is acceptable for real-time adaptation.

7 Conclusion and Future Work

In this paper, we introduced a novel data structure, DSL, for on-demand overlay media streaming. A DSL is a randomized and distributed structure, which inherently accommodates node dynamics and asynchronous requests. We have shown that all typical VCR operations can be implemented in this overlay with $O(1)$ or $O(\log N)$ message costs. We also discussed the practical issues involved in realizing a DSL-based streaming overlay and presented an optimal algorithm for multipeer data scheduling with linear network coding.

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

As a future work, we are interested in investigating the performance of the DSL-based streaming overlay with more realistic network configurations and heterogeneous clients, possibly using the PlanetLab testbed, and comparing it with those overlay systems using advanced structures. We are also interested in incorporating advanced video coding algorithms; an example is the Multiple Description Coding (MDC) [31], which is a good match to the DSL structure and could further improve its robustness.

Acknowledgments

J. Liu's work was supported in part by the Canadian NSERC Discovery Grant 288325, an NSERC Research Tools and Instruments Grant, a Canada Foundation for Innovation (CFI) New Opportunities Grant, and an SFU President's Research Grant.

References

[1] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network Information Flow," *IEEE Trans. Information Theory,* vol. 46, pp. 1204-1216, July 2000.
[2] K. Almeroth and M.H. Ammar, "The Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service," *IEEE J. Selected Areas in Comm.*, vol. 14, no. 6, pp. 1110-1122, Aug. 1996.
[3] J. Aspnes and G. Shah, "Skip Graphs," *Proc. 14th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '03),* Jan. 2003.
[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. ACM SIGCOMM '02,* Aug. 2002.
[5] M. Castro, P. Drushel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03),* Oct. 2003.
[6] Y. Chu, S. Rao, and H. Zhang, "A Case for End System Multicast," *Proc. ACM SIGMETRICS '00,* June 2000.
[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms,* second ed. MIT Press, 2001.
[8] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," *IEEE J. Selected Areas in Comm.*, vol. 22, no. 1, pp. 91-106, Jan. 2004.
[9] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming Live Media over Peer-to-Peer Network," technical report, Stanford Univ., 2001.
[10] T. Do, K. Hua, and M. Tantaoui, "P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment," *Proc. IEEE Int'l Conf. Comm. (ICC '04),* June 2004.
[11] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," *Proc. IEEE INFOCOM '05,* Mar. 2005.
[12] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-Peer Patching Scheme for VoD Service," *Proc. 12th Int'l World Wide Web Conf. (WWW '03),* May 2003.
[13] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties," *Proc. Fourth Usenix Symp. Internet Technologies and Systems (USITS '03),* Mar. 2003.
[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Predication.* Springer, 2001.
[15] M. Hefeeda and B. Bhargava, "On-Demand Media Streaming over the Internet," *Proc. Ninth IEEE Int'l Workshop Future Trends of Distributed Computing Systems (FTDCS '03),* May 2003.
[16] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," *Proc. IEEE Int'l Symp. Information Theory (ISIT '03),* June 2003.
[17] A. Hu, "Video-on-Demand Broadcasting Protocols: A Comprehensive Study," *Proc. IEEE INFOCOM '01,* Apr. 2001.
[18] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03),* Oct. 2003.
[19] Z. Li, B. Li, D. Jiang, and L. Lau, "On Achieving Optimal Throughput with Network Coding," *Proc. IEEE INFOCOM '05,* Mar. 2005.
[20] 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.
[21] 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.
[22] M. Medard, S. Acedanski, S. Deb, and R. Koetter, "How Good Is Random Linear Coding Based Distributed Networked Storage," *Proc. First Workshop Network Coding, Theory, and Applications (NETCOD '05),* Apr. 2005.
[23] R. Motwani and P. Raghavan, *Randomized Algorithms.* Cambridge Univ. Press, 1995.
[24] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," *Proc. 12th Int'l Workshop Network and Operating System Support for Digital Audio and Video (NOSSDAV '02),* May 2002.
[25] W. Pugh, "Skip Lists: A Probabilistic Alternative to Balanced Trees," *Comm. ACM,* vol. 33, no. 6, pp. 668-676, June 1990.
[26] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," *Proc. ACM SIGCOMM '04,* Aug. 2004.
[27] R. Rejaie and A. Ortega, "PALS: Peer-to-Peer Adaptive Layered Streaming," *Proc. 13th Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '03),* June 2003.

[28] P. Rodriguez, C. Gkantsidis, and J. Miller, "Comprehensive View of a Live Network Coding P2P System," *Proc. ACM/Usenix Internet Measurement Conf. (IMC '06),* Oct. 2006.

[29] H. Shojania and B. Li, "Parallelized Progressive Network Coding with Hardware Acceleration," *Proc. 15th IEEE Int'l Workshop Quality of Service (IWQoS '07),* June 2007.

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

[31] Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video Processing and Communications.* Prentice Hall, 2001.

[32] M. Wang and B. Li, "How Practical Is Network Coding," *Proc. 14th IEEE Int'l Workshop Quality of Service (IWQoS '06),* June 2006.

[33] 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," *Proc. Joint Conf. Digital Libraries (JCDL '03),* May 2003.

[34] H. Yin, C. Lin, F. Qiu, and D. Wu, "TrustStream: A Novel Secure and Scalable Media Streaming Architecture," *Proc. 13th ACM Int'l Conf. Multimedia (Multimedia '05),* Nov. 2005.

[35] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," *Proc. IEEE INFOCOM '96,* Mar. 1996.

[36] 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," *Proc. IEEE INFOCOM '05,* Mar. 2005.

[37] M. Zhou and J. Liu, "Tree-Assisted Gossiping for Overlay Video Distribution," *Kluwer Multimedia Tools and Applications,* 2005.

[38] Y. Zhu, B. Li, and J. Guo, "Multicast with Network Coding in Application Layer Overlay Networks," *IEEE J. Selected Areas in Comm.,* vol. 22, no. 1, pp. 107-120, Jan. 2004.

Dan Wang received the BSc degree in computer science from Peking University, Beijing, in 2000, the MSc degree in computer science from Case Western Reserve University, Cleveland, Ohio, in 2004, and the PhD degree in computer science from Simon Fraser University, Burnaby, B.C., Canada, in 2007. He will join the Department of Computing, Hong Kong Polytechnic University, as an assistant professor. His research interests include peer-to-peer networks, wireless sensor networks, and QoS routing. He is a student member of the IEEE.

Jiangchuan Liu received the BEng degree (cum laude) in computer science from Tsinghua University, Beijing, in 1999, and the PhD degree in computer science from the Hong Kong University of Science and Technology in 2003. He is currently an assistant professor in the School of Computing Science, Simon Fraser University, B.C., Canada, and was an assistant professor at the Chinese University of Hong Kong from 2003 to 2004. His research interests include media streaming, wireless sensor networks, and peer-to-peer overlay networks. He serves as a TPC member for various international conferences, including IEEE INFOCOM and IWQoS. He was an information system cochair for IEEE INFOCOM 2004 and a guest editor of *ACM/Kluwer Journal of Mobile Networks and Applications*, special issue on wireless sensor networks and wireless mesh networks. He is an editor of *IEEE Communications Surveys and Tutorials*. He is a member of the IEEE, the ACM, and Sigma Xi.