Scalable and Adaptive Streaming for Non-Linear Media

David Gotz
gotz@cs.unc.edu

Ketan Mayer-Patel
kmp@cs.unc.edu

University of North Carolina at Chapel Hill
CB #3175, Sitterson Hall
Chapel Hill, NC 27599 USA

ABSTRACT

Streaming of linear media objects, such as audio and video, has become ubiquitous on today's internet. Large groups of users regularly tune in to a wide variety of online programming, including radio shows, sports events, and news coverage. However, non-linear media objects, such as large 3D computer graphics models and visualization databases, have proven more difficult to stream due to their interactive nature. In this paper, we present Channel Set Adaptation (CSA), a framework that allows for the efficient streaming of non-linear datasets to large user groups. CSA allows individual clients to request custom data flows for interactive applications using standard multicast join and leave operations. CSA scales to support very large user groups while continuing to provide interactive data access to independently operating clients. We discuss a motivating sample application for digital museums and present results from an experimental evaluation of CSA's performance.

1. INTRODUCTION

Digital media streaming, made possible by the proliferation of both digital media and broadband data networking, has become nearly ubiquitous. For example, radio program streams are available online from a variety of sources ranging from major market music broadcasts to small college radio stations.

To date, media streaming has largely been limited to *linear media*. Linear media objects, such as audio and video, consist of data arranged in a fixed and linear ordering. For example, video consists of a linear sequence of frames arranged along the time dimension. Every user that accesses a linear media stream receives the same flow of information.

The dominance of linear media in the context of online streaming matches the long time dominance of linearity in more traditional media, including books, film, and television. The linear nature of theater, for example, is what allows entire audiences to be satisfied by observing a common stage performance.

However, recent advances in computing and interactive technology have led to the growing importance of *non-linear media*. Non-linear media objects, such as video games, interactive visualizations, and virtual environments, provide individual data orderings to each user in response to their local requirements and interactions.

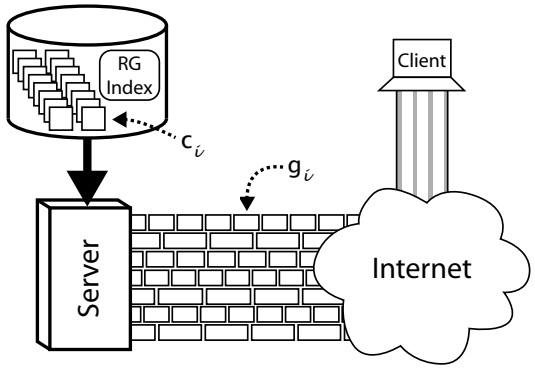

Figure 1: Channel Set Adaptation (CSA) enables efficient streaming of non-linear media to large groups of independently operating users. We partition the media object into semantically meaningful clusters, labeled c_i. These clusters are then mapped to a large set of broadcast or multicast channels, labeled g_i. Clients compose custom data flows that match their local application requirements without ever contacting the central server by managing their Active Channel Set through scalable subscription operations.

Linear and non-linear media are fundamentally different in the experience they provide to consumers. Non-linear media experiences require unique presentations to each participating user. For example, every user playing a video game is presented with a different flow of information in response to their individual interactions.

The differences between linear and non-linear media pose new challenges to media streaming techniques. In particular, the need to deliver a custom data flow to each member of a large group of independent users can not be solved using traditional media streaming techniques which typically rely on common interests across the entire receiver population.

In this paper, we propose *Channel Set Adaptation* (CSA), a novel approach that provides scalable and adaptive streaming for non-linear media. CSA allows for the distribution of non-linear media datasets to large groups of independent users. Each user is able to independently express data requirements and preferences. In response, custom data flows designed to match those individual needs are delivered to each client in a a way that scales well to support very large user groups.

1.1 Main Results

We present a complete solution for scalable and adaptive streaming of non-linear media. Our work is motivated by our driving de-

UNC Technical Report TR05-022
October, 2005

sign philosophy for a "simple server" streaming solution that scales to support large groups of users by pushing all per-client work away from the server and toward individual clients. We describe the three primary components of our work: a data representation abstraction, a channel-based media communication model, and a client-driven adaptation algorithm.

Taken together, these components form Channel Set Adaptation (CSA), a framework for scalable and adaptive streaming of non-linear media. We have implemented an experimental prototype that uses CSA for streaming a large image-based rendering dataset to a group of independently operating clients. We include results from an experimental evaluation of our prototype that show CSA to be an effective technology for supporting large scale non-linear streaming under a wide range of operating conditions.

1.2 Organization

The remainder of this paper is organized as follows. We review background information and related work Section 2. We discuss a number of design considerations for achieving scalable delivery for non-linear streaming in Section 3. In Section 4, we present Channel Set Adaptation, our solution for scalable and adaptive non-linear media streaming. We describe our experimental prototype in Section 5 and present results from our experiments in Section 6. Finally, in Section 7, we conclude and explore areas for future work.

2. BACKGROUND AND RELATED WORK

In this section, we discuss a selection of previous work most related to our research. There is a large body of research in the area of linear media streaming for audio and video. Much of this work explores efficient streaming techniques for large user groups. We provide an overview of these research efforts, as well as a review of early work in non-linear streaming for interactive datasets. We also review two efforts in scalable database access that explore issues similar to those faced in our work.

2.1 Linear Streaming Techniques

Streaming technologies for linear media objects have received a large amount of attention in recent years as media streaming has matured into a fixture on today's internet. Several commercial technologies, including Real Network's RealAudio and Microsoft's Windows Media are now readily available and used to stream both audio and video content.

These technologies are based on several fundamental research efforts, including application-level framing, [9], forward-error correction [18], and early research initiatives in developing successful streaming protocols [17]. This has led to several protocol standards for supporting real-time streaming, including RTP [20] and RTSP [21].

2.2 Linear Streaming to Large User Groups

The high bandwidth requirements for streaming audio and video have motivated several efforts to more efficiently support the distribution of linear media data to large groups of users. The multicast network model [10], where data streams are efficiently distributed to groups of interested users, was developed as an efficient alternative to unicast.

Multicast allows a user to join a group of receivers, all of whom receive an identical flow of data. A multicast server is then able to transmit a single stream to the entire group, rather than send individual streams to each user. The single stream is replicated as needed within the network and delivered to the interested participants.

Problems with deployment of IP Multicast, the standardized version of infrastructure multicast, have led to significant effort in developing application level multicast (ALM) [3, 4, 6, 7, 8]. Rather than relying upon core network resources to perform group management and data replication, ALM performs these tasks at the application level using the very hosts that are participating in the multicast session.

Both IP Multicast and ALM techniques deliver identical flows to all receivers, making them ideal solutions for scalable linear media delivery. However, even for linear data, more flexibility is often required. Several researchers have explored novel uses of multicast protocols to provide limited flexibility in the time of access to linear media. For example, scalable video-on-demand can be accomplished through pyramid broadcasting [23] and its many derivatives [1, 14, 15]. Similarly, other work, such as Receiver-Driven Layered Multicast [16], has explored using layered media delivery via multicast to improve flexibility in the rate of data delivery.

Despite this large body of work, scalable solutions have been largely limited to linear media objects. These techniques depend upon the predictable access patterns associated with linear media applications. In our work, we explore techniques that exploit multicast delivery for scalable and adaptive *non-linear* media streaming, where data access patterns are not known *a priori*.

2.3 Non-Linear Media Streaming

Several researchers have explored techniques for *single-user* streaming of nonlinear datasets, particularly in the area of computer graphics. For example, streaming for complex 3D geometric models can be accomplished by selectively transmitting multi-resolution models of geometric objects based on the user's navigation of the scene [22]. This work introduces a benefit function that evaluates the relative utility of various models to drive the selective transmission. Our work uses a similar utility-driven approach that employs a more generic utility metric [11].

Progressive mesh representations, which prioritize geometric information based on their importance to overall shape, have been used to develop geometric data streams that are resilient to lost packets during transmission [2]. The data encoding includes redundant copies of the low resolution geometric information to speed loss recovery.

Other researchers have explored single-user streaming for alternative computer graphics techniques. For example, selective transmission techniques have been applied to image-based rendering with concentric mosaics [25]. Similar work has addressed the streaming techniques for point-based models [19].

These techniques, while supporting streaming access to non-linear media, are all based on individual user requests where the streaming server performs per-client work. As a result, the server workload and outgoing bandwidth requirements typically limit these solutions to very small user populations.

Recognizing the need for more scalable solutions, some researchers have explored support for broadcasting geometric data [5] for scalable access. However, this work is limited to broadcast environments and does not allow any per-client control over the received data flow. All users receive the exact same flow of information, making it most applicable to small datasets where last-mile bandwidth efficiency is not a concern. Unfortunately, the internet is not a broadcast medium and the last-mile links are often the primary communication bottleneck link for individual clients.

2.4 Scalable Database Access

The database community has explored scalable access frameworks that attempt to scalably support large numbers of simulta-

neous queries. Two of these efforts, the Datacylce Architecture and Broadcast Disks, use solutions that draw on concepts that are closely related to our work.

The Datacycle Architecture [13] has been proposed for very high throughput database systems. In this architecture, the entire database is broadcast repeatedly over a local high-bandwidth communication network. Data filters attached to the network then work in parallel to search the stream of data and satisfy complex queries.

In more recent work, Broadcast Disks [1] were developed for asymmetric communication environments where bandwidth is abundant for downstream transmission but expensive for upstream queries. Data is repeatedly broadcast over a single broadcast channel, and rates for repeating the broadcast of individual data elements are chosen to control their expected access times.

3. ACHIEVING SCALABILITY

A scalable solution for non-linear media streaming requires a carefully balanced design that can manage the tradeoff between (1) the requirement of delivering custom flows to each client and (2) the need to remove per-client work from the server for scalable performance. In this section, we present a number of design considerations that address this tradeoff. We first discuss our simple server design philosophy. We then describe the spectrum of possible delivery solutions.

3.1 Simple Server Design Philosophy

We have embraced a simple server design philosophy in our solution for non-linear media streaming. Our design goal is to push all computation away from the server and toward the participating clients. This client-driven approach is motivated by two factors.

First, we are striving for a constant server load model. If the server itself is responsible for any per-client tasks, the goal of a constant server load is impossible to reach. Second, adaptation is performed independently on each client and must reflect local system and application conditions. Therefore, the logical location for per-client adaptive decisions is on the individual clients themselves.

These two factors led us to adopt a simple server design philosophy where the centralized server is tasked with constant level work loads that are equally useful for all participants and independent from the needs of any individual clients. The simple server design leads directly to a bounded server load that is independent of the number of participating clients.

3.2 Spectrum of Delivery Solutions

There is a wide spectrum of possible solutions for delivering non-linear media to large audiences. In this section, we first present a sample application to give our discussion a concrete context. We then outline the two extremes of the solution spectrum. Finally, we describe the compromise approach which we adopt in our work.

3.2.1 Example Non-Linear Application

We will use a digital museum application as an illustrative example throughout the remainder of this section. Consider a digital museum that aims to digitize and share a famous space (e.g., the Palace of Versailles) with a group of virtual visitors from around the globe. This could be accomplished by capturing a large set of digital pictures from the scene, storing them in an image-based rendering (IBR) dataset, and making them available online.

IBR is a computer graphics technique that uses real world pictures from a scene as input, and renders novel photo-realistic views of the scene in response to a user moving a virtual viewpoint. The novel views are generated by interpolating between the captured samples. Users can navigate through the virtual space interactively, exploring the scene with the same freedom that video game players have while exploring a game's virtual setting.

IBR datasets are typically very large in size. A digital museum would therefore want to stream the dataset to each user to avoid long download delays. In addition, because users will be navigating the scene independently, they will each require a unique flow of image data. We therefore need to support non-linear streaming that scales to support a large group of museum visitors.

3.2.2 The Adaptive Extreme

There are a variety of possible solutions for supporting the digital museum streaming application. At one extreme, the most adaptive architecture for streaming IBR data is a unicast client-server model. Under this model, each client would first obtain a list of all available images and their semantic (e.g., position in space) and syntactic (e.g., encoding dependencies) relationships.

Armed with this list, a client would iteratively determine which images are most important using a benefit function and request those images from the server. For example, images located closer to the virtual viewpoint would be considered more useful than images located further away. As the user's viewpoint moves through the virtual space, new image requests would be generated and passed to the server.

Allowing the client to make individual image requests provides the highest degree of adaptive behavior to each client. They can custom compose the incoming stream of images by specifically requesting each photograph.

However, this approach does not take advantage of any similarity in interests across users. The server must respond individually to each client's requests and the server's outbound bandwidth requirement grows linearly with respect to the number of clients. This design does not scale well and violates our simple server design philosophy.

3.2.3 The Scalable Extreme

At the other extreme, the most scalable architecture for the digital museum application is to cluster all of the data into one large unit and transmit the data over a single multicast channel. The server would repeatedly transmit the information on a carousel. Individual clients would then tune in to the channel, continuing to receive data until the entire dataset has been downloaded.

This architecture is infinitely scalable because the server does not perform any per-client work and the design adheres strictly to the simple server design philosophy.

However, clients have no options for adapting the flow of images and must settle for the predefined linear ordering chosen when grouping the images into a monolithic cluster of data. This is the reason that multicast works so well for linear media distribution where all users have identical interests. It fails, however, to allow the non-linear data access that is required for digital museum application.

3.2.4 A Scalable and Adaptive Solution

Ideally, we would like to retain both the adaptive nature of the first extreme and the scalable properties of the second extreme. These goals can be reached through a middle-ground approach which achieves both scalable and adaptive distribution of non-linear media.

Under this approach, we would group related data elements into clusters. This would partition a media object into several several larger blocks, each of which has a semantic meaning. For example, in our sample application, we might group all the low resolution pictures from one corner of a room into a single cluster of images.

We could then distribute each cluster using multicast to scalably deliver them to all interested clients. For example, if five users were exploring the same corner of a room, they could all subscribe to the multicast stream that contained the associated cluster of images. Users in a different room would choose instead to subscribe to an alternate multicast stream with an image cluster that more closely matched their requirements.

If clients were aware of the available multicast channels and their associated semantic meanings (e.g., which images are in which channel), they could intelligently subscribe to the multicast streams that contain information most most relevant to their needs. As those needs changed over time, clients could quickly choose to subscribe to whichever multicast streams had become most appropriate.

Clients could compose a custom flow of images based upon the order of their subscriptions. At the same time, the server would perform no per-client work and would only be responsible for transmitting a fixed number of multicast streams. This channel-based approach to providing scalable and adaptive access to non-linear media forms the conceptual foundation for our research.

4. CHANNEL SET ADAPTATION

In this section, we present *Channel Set Adaptation* (CSA), a framework that enables efficient streaming of non-linear datasets to large user groups. CSA allows individual clients to request custom data flows for interactive applications using standard multicast join and leave operations. CSA scales to support very large user groups while continuing to provide interactive data access to independently operating clients.

There are three major portions of the CSA framework. First, we discuss the data representation formalisms used to express data relationships and dependencies. Second, we detail our communication model which is designed to provide scalable service to large user groups. Finally, we describe the client-driven adaptation algorithms that perform both congestion and content control.

4.1 Data Representation

A critical task in the CSA framework is the expression of relationships between individual data elements. We need formal structures for expressing both syntactic relationships (e.g., encoding dependencies) and semantic relationships (e.g., similarity in meaning or utility). To satisfy this requirement, we use the Representation Graph (RG) abstraction [11], first proposed as a generic representation model for multidimensional adaptation.

The RG model is a flexible representation abstraction designed specifically for expressing both syntactic and semantic data relationships in multimedia databases. The RG framework also provides mechanisms for evaluating the relative utility of individual elements of information in the database based on dynamic system conditions.

An RG is composed of a graph-based structure embedded within a multidimensional utility space. Individual elements of information are modeled as *nodes*. Syntactic dependencies are expressed via a set of *edges* that connect sets of dependent nodes. Semantic relationships between nodes are expressed by the nodes' positions within the utility space. Furthermore, the RG model defines *clusters* as groups of edges which are accessed atomically. Each cluster is considered a semantically consistent unit of data. The underlying structure of an RG, including the list of nodes and their connectivity, is stored explicitly as an *RG Index*.

There are two parts of the RG abstraction that are particularly important within the context of CSA: (1) Clusters and (2) the RG Index.

```
<gal>
  <utilityspace dimensionality=5>
    <subspace id=0 dimensionality=3>
      <dimension type=navigable name=x> </dimension>
      . . .
    </subspace>
      . . .
  </utilityspace>
  <nodelist>
    <node id=1 subspace=0 pos=12.332,23.32,1.1> </node>
    . . .
  </nodelist>
  <clusterlist>
    <cluster id=1 resource=30012 cost=308>
    . . .
  </clusterlist>
  <edgelist>
    <edge id=1 src=12 dest=14 cluster=1> </edge>
    . . .
  </edgelist>
</gal>
```

Figure 2: A brief sample the XML index format used by GAL, the Generic Adaptation Library. The index describes the inherent structure of the data representation (i.e. nodes, edges, and clusters), while omitting the actual encoded data.

4.1.1 Clusters

A cluster is a block of data, corresponding to one or more edges in the RG model, which is semantically consistent and accessed atomically. For example, clusters in Section 3.2.4 represented groups of images captured from similar locations. When modeled using an RG, a dataset is essentially partitioned into a set of clusters, $C = \{c_1, \cdots, c_n\}$.

4.1.2 RG Index

The RG Index is a specification of the underlying structure of the RG. The index is a concise enumeration of the nodes, edges, and clusters that make up the RG, as well as the definition of the utility space in which the graph. A sample of the XML-based index format is shown in Figure 2. This format is part of the Generic Adaptation Library, an implementation of the work described in [11]. The index does not include any actual media data and is therefore very small in size in comparison to the overall dataset.

4.2 Media Communication Model

Our media communication model is designed to meet two goals. First, the model must allow individual users to access the non-linear media interactively and independently. Second, the model must scale to support large groups of independent users.

In this section, we present our solution for meeting the two competing requirements of interactivity and scalability. Our approach delivers custom data flows to each user while maintaining a constant and bounded server load that is independent of the number of users. There are three primary components of our design: (1) channel-based transmission, (2) session initiation, and (3) client behavior. Finally, we discuss the implications of our communication model with respect to our two design goals.

4.2.1 Channel-Based Transmission

CSA requires the central server to maintain a large set of communication channels, noted as $G = \{g_1, \cdots, g_n\}$. In this context, a channel is an individual data flow to which users can subscribe and unsubscribe. Upon subscription, users have no control over the data contained in an individual channel. They must either ac-

cept the data flow assigned to the active channel, or unsubscribe to terminate the flow of information. The subscription model of our channel-based transmission scheme can be easily supported in both broadcast and multicast networks.

The number of channels in G is equal to the number of clusters in the RG model used to represent a media object. A one-to-one mapping $M : C \mapsto G$ maps each cluster $c_i \in C$ to a corresponding channel $g_i \in G$. Because clusters are semantically consistent blocks of data (e.g., sets of images from the same location), the mapping M assigns a semantic meaning to each channel g_i. The mapping information in M is appended to the RG Index.

At runtime, the server simultaneously transmits all channels in G. Each channel g_i is transmitted at a constant bit rate. It is important to note that the data assigned to each cluster is typically finite in size. In this case, the server transmits the data on a carousel transmission schedule, repeatedly sending out the entire cluster of data with a cyclical schedule. The server is not responsible for any other tasks. The overall architecture is shown in Figure 1.

4.2.2 Session Initiation

Clients are responsible for initiating a new sessions. The first step for a new client is to obtain a copy of the RG Index. This transaction must be supported through some out-of-band mechanism. For example, the RG Index could be made available through a well-known HTTP or FTP host.

The RG Index contains the cluster-to-channel mapping, M, as well as the traditional RG Index contents describing the semantic and syntactic data relationships of the associated non-linear media dataset. The RG index is essentially a menu describing which communication channels are available as well as each channel's assigned semantic meaning. For example, in the digital museum application, the $RGIndex$ would specify which channels contained images from each part of the Palace of Versailles.

4.2.3 Client Behavior

Following session initiation, a client has all the information it needs to begin receiving the non-linear data stream. Using the client-driven adaptation algorithm we will describe in Section 4.3, the client begins to manage its *Active Channel Set* (ACS).

The ACS is a list of all channels to which the client is currently subscribed. By choosing which channels are in the ACS as well has how many channels are active any any point in time, a client can compose a unique stream that delivers a custom flow of non-linear media data that is individually tailored to meet the needs of the client.

4.2.4 Satisfying Design Goals

Our media communication model meets our two primary design goals. First, individual users can access the non-linear media stream interactively and independently through management of the ACS. Second, our model easily supports large groups of independent users because of the channel-based transmission design. We defer our coverage of the algorithms for managing the ACS until Section 4.3. In this remainder of this section, we concentrate on the scalable properties of our communication model.

As outlined in our Simple Server Design Philosophy, a key requirement for any scalable solution is the removal of all per-client work from the server. We achieve this requirement by utilizing a channel-oriented network infrastructure, which can be supported by a broadcast or multicast network. This leads to a highly scalable server-side solution whose performance is independent of the number of participating clients.

The independence of server performance from the number of clients is a critical property in CSA. It allows us to determine a constant upper bound on computation and bandwidth requirements. As a result, servers can be properly provisioned with a finite and static level of resources to support, in the ideal case, an infinite number of simultaneous users.

4.3 Client-Driven Adaptation

Individual clients are responsible for adapting their incoming data flows to match their own application preferences and resource requirements. Adaptation is accomplished independently by each client as they manage their ACS.

ACS management is performed through two fundamental operations. The first operation, $\mathbf{Sub}\{g_i, ACS\}$, is used to subscribe to a new channel. Upon subscription, the new channel is added to the ACS. The second operation, $\mathbf{Unsub}\{g_j, ACS\}$, is used to unsubscribe from an already active channel. This operation removes channel g_j from the ACS assuming it is a member. The two operations are defined below.

$$\mathbf{Sub}\{g_i, ACS\} = ACS \cup \{g_i\} \quad (1)$$

$$\mathbf{Unsub}\{g_j, ACS\} = ACS \setminus g_j \quad (2)$$

Both the subscribe and unsubscribe operations can be performed in broadcast or multicast networks without any direct contact with the server. By defining adaptation in terms of these two operations, we can ensure adaptive data flows as well as scalable performance.

The client-driven adaptation algorithm must accomplish two tasks. First, it must perform *congestion control* to manage the speed at which data arrives. Second, it must perform *content control* to achieve the individualized data flows required by non-linear media applications. We define both of these adaptive tasks in terms of the subscribe and unsubscribe operations in the following sections.

4.3.1 Congestion Control

A client participating in a non-linear media stream using CSA must manage the speed at which data arrives over the network through a process known as congestion control. This is done by managing the size of the ACS.

Under our channel-based transmission scheme, the server offers a large set of constant bitrate channels, G. Clients subscribe to a subset of this offering, so that $ACS \subset G$. Because each channel $g_i \in ACS$ is offered at a constant bitrate, the overall bitrate of the arriving ACS is determined by the size of the set, or $|ACS|$. The congestion control problem for CSA is analogous to the problem faced in Receiver-Driven Layered Multicast [16], and we apply a similar solution.

At runtime, the client adjusts the size of the ACS through subscribe and unsubscribe operations At signs of network congestion, such as the detection of lost packets, the client decreases $|ACS|$ through an unsubscribe operation. In order to maintain the most useful data flow after the decrease in subscription level, a client will choose to unsubscribe from the least useful active channel. We utilize the Utility-Cost Ratio (UCR) metric described in [11] for this evaluation. The UCR metric combines application-specific utility and cost functions to determine how best to adapt a multimedia dataset.

In times of exceptionally strong network performance, the client probes for additional bandwidth by increasing $|ACS|$ through a subscribe operation. Once again, we use the UCR metric to determine which channel should be added to the ACS.

A series of timers are used for each level of subscription to improve stability and to allow the system to converge more quickly to an appropriate subscription level. A simplified version of the congestion control algorithm is shown in lines 2-9 of Figure 3.

```
1   repeat forever:
2     if ((experiencingNetworkLoss) and (timerExpired))
3       c_active = GetLeastUsefulActiveCluster(RG, ACS)
4       g_active = M(c_active)
5       Unsub(g_active, ACS)
6     else if ((notExperiencingNetworkLoss) and (timerExpired))
7       c_inactive = GetMostUsefulInactiveCluster(RG, ACS)
8       g_inactive = M(c_inactive)
9       Sub(g_inactive, ACS)
10    else
11      c_inactive = GetMostUsefulInactiveCluster(RG, ACS)
12      c_active = GetLeastUsefulActiveCluster(RG, ACS)
13      if Utility{c_inactive} > Utility{c_active}
14        g_inactive = M(c_inactive)
15        g_active = M(c_active)
16        Unsub(g_active, ACS)
17        Sub(g_inactive, ACS)
18      endif
19    endif
```

Figure 3: A simplified version of the client-driven adaptation algorithm.

4.3.2 Content Control

In parallel to congestion control, each client must also perform content control. This task is unique to the problem of non-linear streaming. In traditional linear media applications, data is delivered in a fixed order and there is no freedom to change the order to meet application needs. However, individualized control over the contents of an arriving data stream is a primary requirement for non-linear media access.

Content control is performed by aggressively changing channels over time, managing the ACS to ensure that the active channels match the current application requirements. Recall that the data representation abstraction builds clusters that are semantically consistent. As a result, each channel has an associated semantic meaning. This allows us to use channel subscription operations to express an application's needs for specific semantically meaningful units of data.

At runtime, a client iteratively compares the least useful active channel, g_{active}, with the most useful inactive channel, $g_{inactive}$. Whenever it is discovered that the utility of $g_{inactive}$ is greater than that of g_{active}, the two swap positions and $g_{inactive}$ becomes a member of the ACS. Lines 11-18 of Figure 3 show a simplified version of the algorithm.

The channel subscription pattern is driven by the evaluation of utility that is performed on each iteration. We use the same UCR metric as the congestion control algorithm for determining the relative utility of each channel.

The UCR metric is a spatial measure of utility defined on the representation graph structure included in the RG Index. Most importantly, it evaluates utility with respect to the current application conditions and preferences. As a result, the sequence of subscription operations performed in the adaptation process is determined uniquely on each client in response to user interactions and locally changing system conditions.

Each client will exhibit their own pattern of subscription requests based upon their own local needs. For example, Figure 4 illustrates a possible sequence of subscription operations over a small window of time. In the figure, the ACS starts at size two and grows to size four with the subscriptions at times t_1, and t_2. At time t_3, the congestion control algorithm determines that the ACS is too large and the ACS is contracted back down to size three. The content control algorithm initiates a channel swap at time t_4. This

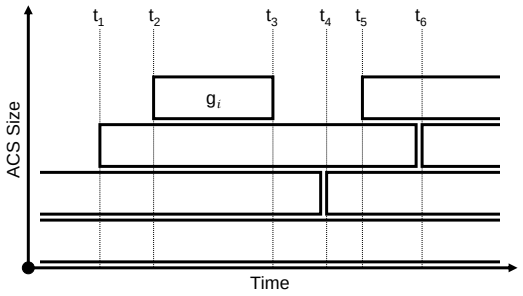

Figure 4: A hypothetical plot that shows the evolution of the ACS over time.

is followed by another subscription to enlarge in the ACS (at t_5) and another channel swap (at t_6).

The concatenation of data flows, following a series of subscribe and unsubscribe operations, produces a unique flow of data that is delivered to each individual client. In addition, the unique flow is composed without any direct communication between clients and the server. As a result, CSA can deliver unique, customized data flows to individual clients in fully scalable manner.

5. EXPERIMENTAL PROTOTYPE

This section describes the experimental prototype used to evaluate the performance of Channel Set Adaptation. We first present the target application for the prototype: a image-based rendering tool for digital museums. We then describe the testbed and methodologies used in our experiments.

5.1 Prototype Application

We have developed a experimental prototype to evaluate Channel Set Adaptation as a solution for non-linear media streaming. The prototype application is an image-based rendering (IBR) application for digital museums. IBR environments allow free viewpoint exploration of digitized spaces, immersing users in a photorealistic recreation of a virtual place that can be navigated interactively. When combined with digital museums, these environments promise to enable large populations from around the globe to explore remote artifacts and locations [12].

IBR applications require non-linear access to possibly massive sets of images as input to their reconstruction algorithms. This domain is therefore a strong match for our work in supporting non-linear media streaming that can scale to support a large crowd of digital museum visitors.

Our prototype is designed as a client-server system with a single image server that uses Channel Set Adaptation to scalably transmit streams of images to a set of interested clients. Each client is able to navigate through the digitized space independently along their own unique paths. For this reason, each client must be able to adapt their own data flow based on their particular needs.

The sample dataset used in our experiments consists of approximately 8,000 color images captured from within a library environment. Each image has a resolution of 512×512 pixels. The pictures are distributed across a $2D$ plane at eye level.

We use an RG composed of nearly 16,000 nodes. These nodes correspond to each of the 8,000 images stored at two resolutions. The number of clusters in the RG was varied across experiments. The overall RG was embedded within a five dimensional utility space, defined by three spatial dimensions, image resolution, and spatial density.

5.2 Experiment Testbed

We performed a series of experiments to evaluate the performance of Channel Set Adaptation in meeting the demands of non-linear media streaming for large user groups. Rather than rely on network simulation, we chose to execute our experiments on the Emulab network testbed [24] which uses network emulation to achieve more realistic operating conditions.

In all of our experiments, we employed network topologies that were provisioned with a single server with a 100Mbps network connection. Our network model assumed that all bandwidth bottlenecks occur within the "last mile" for each client. We therefore modeled all core links within our topology with the same 100Mbps bandwidth as the server. Links connecting clients to the core network were given a fixed bandwidth of 5Mbps for all of the experiments reported in Section 6.

5.3 Experiment Methodology

In this section, we describe our methodology in evaluating the performance for our experimental prototype. We describe both our approach to client emulation and the formulation of the SUM performance metric.

5.3.1 Client Emulation

Because our experiments required that we simulate large groups of users, we were forced to emulate user behavior through an automated process. For all experiments, participating clients navigated a ten minute path through the IBR dataset. The path included a variety of movement types including both fast and slow movements and changes of direction.

During the ten minute execution time, we compute a performance metric once per second. We describe the performance metric in more detail in Section 5.3.2. When presenting average performance values, we consider only the second five minutes of performance data from each ten minute session. This allows us to avoid any transient events that may occur during the early and less stable moments of a session when analyzing average behavior.

5.3.2 The SUM Performance Metric

Throughout our evaluation, we measure performance using the *Summed Utility Metric* (SUM), an application-independent performance metric for evaluating the behavior of our prototype that is based upon the abstract adaptation framework proposed in [11]. The SUM measures system performance as a function of the current state of the representation graph.

The SUM metric requires no domain-specific knowledge because it is defined as a function on the abstract RG data structure. Application knowledge is incorporated into the metric through the application-defined utility metric. The SUM metric measures how well a system delivers data to a client in response to a specific utility metric. Therefore, we can compare the performance of various delivery mechanisms so long as the underlying RG and utility metrics remain the same. This is the methodology used in our evaluations in Section 6 where we compare the effectiveness of various approaches to the digital museum streaming application.

The SUM is derived from the notion that the system's adaptive performance can be measured by the utility of the data it has obtained at any given point in time. The RG structure allows us to mark obtained data elements by placing the elements in the *resolved* state. We can then apply the utility metric to the each resolved node. The SUM is the sum of all of resolved node utility values. A full discussion on possible node states and state transitions is beyond the scope of this paper and we refer you to [11] for more information. We formally define the SUM metric in Equation 3, where R is the set of all resolved nodes and $UtilMetric$ is the application-specific utility metric.

$$\mathbf{SUM} = \sum_{n_i \in R} UtilMetric(n_i) \tag{3}$$

It is important to note that the SUM is a measure of performance at a single point in time. To capture a reliable measure of system behavior, the SUM metric must be evaluated repeatedly over a period of time.

6. RESULTS

We performed a series of experiments using our prototype and the Emulab network testbed. These experiments highlight the ability of Channel Set Adaptation (CSA) to support large user groups and to provide the custom data flows required for non-linear media streaming. We also explore the impact that certain engineering parameters have on overall performance.

6.1 Scalable Delivery

One of the primary motivations for the CSA framework is the ability to support large groups of independent users. We performed a series of experiments to evaluate performance at a range of group sizes. Our evaluations compared three delivery mechanisms.

First, we configured the prototype to use the traditional unicast request-response model for non-linear media distribution. The remaining two configurations were CSA-based: one using a multicast network and the other a broadcast network.

In the two CSA configurations, the broadcast-based experiment serves as a benchmark for ideal multicast performance. Broadcast supports the same channel-based subscription model as multicast, but without the overhead of group management. However, broadcast solutions are only deployable over dedicated networks (such as cable television). For this reason, we also include results from experiments using multicast-based CSA.

In each of the three configurations, we experimented with group sizes ranging from 1 to 65. Because we are using emulation with actual hardware resources, the upper limit in this range was determined by the size of the testbed. We utilized the GAL adaptation library and identical utility metrics across all experiments, allowing us to compare performance using the SUM metric.

Under both CSA variants, we used identical RG representations with 160 clusters, and therefore 160 channels. For unicast, we used the same RG with one modification. We placed every edge in its own cluster, resulting in a cluster count of 15,568. Because clusters define the granularity of data access, this change provided complete freedom of access to the unicast experiments.

The results of these experiments are shown in Figure 5. For all experiments, we provisioned the server with 100Mbps of bandwidth and each client with 5Mbps. As a result, the unicast server was able to fully support nearly 20 clients ($n = 20$) before saturating it's network resources. At small group sizes of $n < 20$, the unicast configuration outperforms the broadcast-based CSA variant by roughly 14%. The increase in performance for unicast is directly attributable to the two orders of magnitude increase in the number of clusters. The additional clusters provide greater flexibility in data access and allow the clients to make data requests to more tightly match their requirements. We call the drop in performance due to clustering in CSA the *cluster penalty*.

Once the group size reaches $n = 20$, the unicast server's bandwidth reaches the point of saturation. Past this point, the performance drops as the group size increases. Performance will asymptotically approach zero as the server's bandwidth is divided in ser-

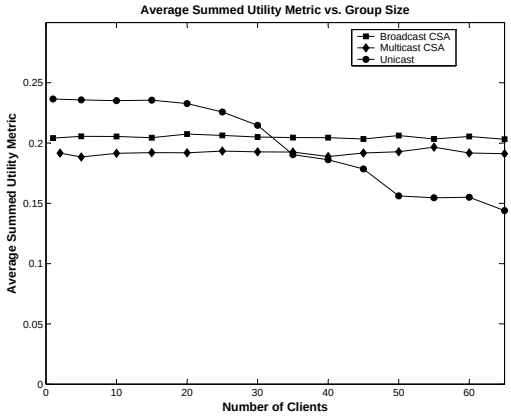

Figure 5: System performance for group sizes up to 65.

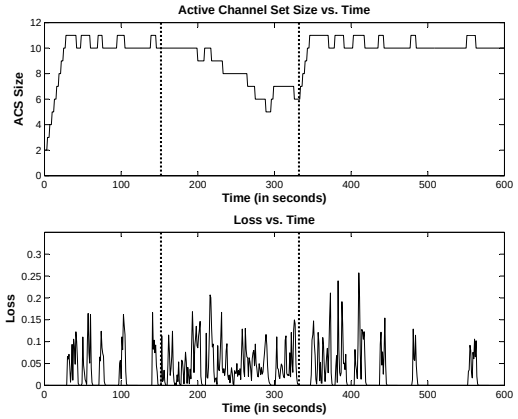

Figure 6: Rate adaptation in response HTTP cross traffic.

vice of more and more users.

Under broadcast-based CSA, performance is independent of group size. This important result shows that our solution can deliver independent non-linear media streams to very large user groups without saturating the central server. This is highlighted by the flat plot for broadcast performance in Figure 5. While CSA pays a penalty in performance for small group sizes, it dramatically outperforms unicast for large groups of users. After a crossover point at $n \approx 32$, the drop in unicast performance due to congestion is higher than the cluster penalty. As n grows, the performance gap continues to increase.

The exact location of the crossover point is determined by several engineering parameters including the amount of bandwidth provisioned to the server, the average bottleneck bandwidth for each client, and the degree of clustering used in the CSA solution. However, the general shape of the plots in Figure 5 will hold regardless of the specific parameter values. These results show that a fine-grained unicast architecture remains the appropriate solution for small user groups. The CSA solution will perform far better with larger group sizes.

The third plot in Figure 5 shows that the performance for multicast CSA, similar to broadcast CSA, exhibits immunity to group size. Performance remains flat as group size climbs toward 65. However, the multicast configuration performs slightly worse than broadcast. The drop in performance is due largely to leave latencies that delay the effect of subscription operations which are the principle adaptive mechanism. Slower adaptation leads to a drop in performance. We further explore the impact of these overheads in Section 6.3.

The absence of join and leave latencies for broadcast-based CSA makes its performance a benchmark for ideal CSA performance. It corresponds to the best possible performance for any multicast-based CSA implementation.

6.2 Adaptation for Congestion Control

A key component of the client-driven adaptation algorithm is congestion control. As described in Section 4.3.1, we adjust the size of the ACS in response to changes in network loss rates. When network conditions remain positive, the size of the ACS is increased and data arrives at the client at a faster rate. When significant loss rates are detected, the client shrinks its ACS and the data rate decreases.

We evaluated the performance of our congestion control algorithm by observing its behavior in the face competing TCP traffic. In one experiment, we began a new CSA session for a client that initially had no competing traffic over its bottleneck link. After two and a half minutes ($t = 150$), we introduced a 180 second load of simulated HTTP traffic over the congested link. At $t = 330$, the HTTP traffic ceased. The results are shown in Figure 6.

In the first 30 seconds, the size of the ACS quickly increases as the client performs its initial probe for available bandwidth. At $t \approx 30$, the ACS grows to size 11 and congests the bottleneck link. The increase in ACS size is matched by a spike in the loss rate estimate. As a result, the congestion control algorithm backs the ACS down to size 10. From $t = 30$ to $t = 150$, the client continues to probe for additional bandwidth, but at growing intervals as the timer duration increases.

At $t = 150$, the competing HTTP traffic begins flowing over the bottleneck link and the measured loss rate begins to climb. Typically, the client would back down extremely fast in response to the increased loss rate. However, in this case, as shown in Figure 6, the client initially hesitates to back down from $|ACS| = 10$. The delayed response is due to the fact that the onset of competing traffic occurred nearly simultaneously with a decrease in ACS size.

After detecting that the loss rates remained steady, the client continued to back down, with the ACS size falling to as low as five. At $t = 330$, the HTTP traffic was removed from the bottleneck link and the client detected an improvement in network conditions. Very rapidly, the CSA size was increased to 10 following the same probing pattern as seen at the start of the session.

The timers used to govern the rate of increase and decrease in ACS size are tunable and can be configured to yield faster back-off times at the expense of lower stability. The specific settings for the timer parameters should be chosen to best match a particular application. For example, stability is less critical for the IBR prototype application than it is for typical video streaming. We can therefore set the timer parameters to adapt more quickly to changes in network congestion.

6.3 Leave Latency and Content Control

The CSA adaptation algorithm uses subscription operations to perform both content and congestion control. Any significant latency between the issuance of a subscription operation and the actual effect on transmission can have dramatic impact on overall performance.

In particular, various multicast implementations (e.g., IP multicast, application layer multicast (ALM) protocols, etc.) exhibit a wide range in *leave latency*: the time it takes between an unsub-

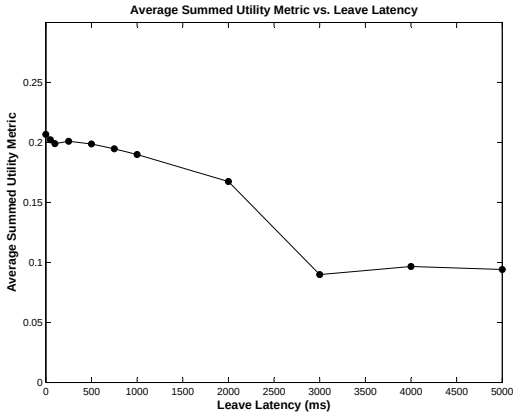

Figure 7: Impact of leave latency on performance.

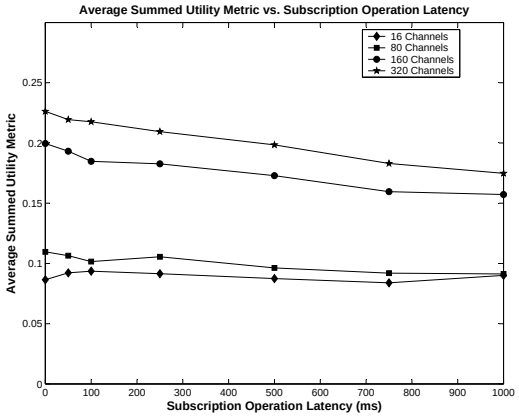

Figure 8: Relationship between performance, latency, and the number of available channels.

scribe request and the actual termination of the data flow. For example, our experiments found that IP multicast showed an average leave latency of about three seconds. Depending on their design, ALM protocols can be significantly better or worse.

We designed an experiment to evaluate the impact of leave latency on CSA performance by introducing artificial leave latencies from 0 to 5000 milliseconds. The results are shown in Figure 7. The experiment shows that longer latencies have a negative impact on performance. In particular, the three second leave latency measured in our IP multicast experiments is far from the ideal range for supporting CSA.

Our results show a steep drop in performance at between two and three seconds of leave latency. The overall trend in performance is important. However, the exact slope of the drop is highly dependent on fraction of time spent on overhead and depends on the specific parameters of the experiment (see Equation 4).

$$Overhead = \frac{SubscriptionOpLatency}{\langle ListenTime \rangle} \quad (4)$$

When the average duration for a single subscription is long, the inefficiency introduced by the leave latency is relatively small and the impact on performance will be lower. Conversely, if the average subscription duration is short, the overhead is large and can dramatically impact performance.

In other research, we have been developing StrandCast [4], a novel ALM algorithm that attempts to minimize leave latency, and therefore overhead, while supporting a high rate of subscription operations. In the future, we plan to evaluate StrandCast as the underlying multicast protocol for CSA and expect that several design properties, including very low leave latencies, will make it ideal for CSA-based applications.

6.4 Granularity of Access

An important parameter in configuring a CSA session is the number of channels in set G. Because G is mapped to the set of clusters C, the number of channels defines the granularity of access to the overall dataset. A small size for G provides relatively few choices for adapting the ACS, reducing the ability of individual clients to customize their incoming data flow. Conversely, a large size of G (noted as $|G|$) provides a great flexibility in ACS management and enables highly customized data flows.

In the extreme, a dataset where $|G| = 1$ corresponds to a single channel and is equivalent to a common monolithic file that all clients must download. When $|G|$ is maximized so that every byte of data is available through a unique channel, clients are given random access to the database.

As a result, additional channels generally result in higher performance. Figure 8 shows a series of experiments performed using four different channel configurations. As expected, the SUM value is highest when the number of channels is greatest.

The benefit of additional channels is greatest when the subscription operation latency is negligible. However, the impact of latency on performance is more pronounced in high-channel configurations. This is evidenced in Figure 8 by the steeper slopes in the high-channel plots.

The steeper decline in performance is a direct result of the increased rate of channel subscription operations for high-channel configurations. The faster pace of subscriptions is exactly what makes large channel sets beneficial: additional channels aid in composing custom data flows. However, the increase in subscription operations reduces the expected listen time for any given channel. As reflected in the results as well as Equation 4, a shorter expected listen time magnifies the impact of changes in the subscription latency.

At first glance, our results seem to hint that it would be desirable to use an enormous number of channels to obtain the best overall performance. In fact, in the absence of any overhead costs, that would be the case. This is why the random-access unicast configuration outperforms both CSA variants for small user groups.

However, in a CSA-based system, which is needed to support large user groups, this would not be practical. First, any multicast infrastructure will introduce some amount of subscription operation latency. Second, the expected listen time in this extreme configuration would be extremely short. Equation 4 shows that these two factors combine to produce extreme amounts of overhead and leave to an inefficient solution.

A practical system design must balance the benefit of a high channel count access with the overhead cost of supporting it. The optimal compromise depends very highly on subscription operation latencies associated with the multicast infrastructure. This conclusion motivates additional work in developing more efficient multicast algorithms with low subscription operation overheads, especially in high-churn environments.

7. CONCLUSIONS AND FUTURE WORK

We have presented Channel Set Adaptation (CSA), a complete solution for scalable and adaptive streaming of non-linear media.

Our work is motivated by a "simple server" design philosophy that achieves scalability by removing all per-client work from the server. In our work, each participating client is responsible for independently performing their own adaptive tasks through management of a set of active communication channels.

We detailed the three primary components of CSA: (1) a data representation abstraction, (2) a channel-based media communication model, and (3) a client-driven adaptation algorithm. The data representation allows us to model a non-linear media database as a set of semantically consistent clusters. Our communication model defines a one-to-one mapping between clusters and channels, creating a semantically meaningful set of distribution channels. The adaptation algorithm, performed independently on each client, performs congestion and content control via channel subscription operations. Taken together, these components provide a framework for scalable and adaptive streaming of non-linear media that allows clients to receive custom data flows in a highly scalable fashion.

We have demonstrated the effectiveness of our approach through experiments using network emulation. We evaluated an experimental prototype that uses CSA for streaming a large Image-Based Rendering dataset to groups of independent clients. Our results show that CSA is an effective technology for supporting large scale non-linear streaming under a wide range of operating conditions.

Despite the promise shown in our initial results, there are several areas that require additional research. For example, our algorithm for congestion control works on the assumption that each client is operating behind its own bottleneck link. This is often the case where last-mile links are responsible for a large fraction of bandwidth bottlenecks. In the future, we would like to move beyond this assumption. We plan to enhance our algorithm to perform well even in the presence of non-shared bottleneck links.

We are also interested in exploring dynamic cluster-to-channel mappings. Our current prototype assumes a static mapping that is predefined in the RG Index. A dynamic mapping could enable more interesting and dynamic datasets, as well as allow servers to reallocate communication resources to better serve high-demand portions of the overall dataset.

Another important area for future work is evaluation with very large user groups. Extrapolation of the results presented in this paper to very large user groups hints at strong performance benefits. However, our experiments were limited to group sizes of 65 because of the required infrastructure. Deployment of CSA in real applications with significantly larger user populations would be an effective way to gather statistics among larger user populations.

Finally, our results highlight the need for an efficient multicast protocol that has very low subscription operation latency and that can support high churn environments. Most of the existing protocols have fairly high subscription latencies and are designed to support long-term sessions. CSA places fundamentally new demands on the multicast infrastructure and new protocols can be designed that would provide significantly improved performance. We have already begun implementation of StrandCast [4], a novel application-layer multicast protocol, designed specifically to support technologies like CSA. In the future, we will integrate this protocol into our testing environment and anticipate improved results.

8. ACKNOWLEDGMENTS

This project is supported in part by NSF CAREER Grant ANI-0238630. We would also like to thank the Flux Research Group and the University of Utah for providing public access to Emulab Testbed.

9. REFERENCES

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: data management for asymmetric communication environments. In *Proc. of ACM SIGMOD*, pages 199–210, 1995.

[2] G. Al-Regib, Y. Altunbasak, J. Rossignac, and R. Mersereau. Protocol for streaming compressed 3-d animations over lossy channels. In *Proceedings of IEEE International Conference on Multimedia and Expo*, pages 353–356, 2002.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM Sigcomm*, 2002.

[4] B. Begnoche, D. Gotz, and K. Mayer-Patel. The design and implementation of strandcast. Technical Report TR05-004, The University of North Carolina at Chapel Hill Department of Computer Science, 2005.

[5] S. Bischoff and L. Kobbelt. Towards robust broadcasting of geometry data. *Computers and Graphics*, October 2002.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A largescale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 2002.

[7] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM SIGMETRICS*, 2000.

[8] Y.-H. Chu, S. G. Rao, S. Seshan, and H.Zhang. Enabling conferencing applications on the internet using on overlay multicast architecture. In *Proc. of ACM SIGCOMM*, 2001.

[9] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proc. of ACM SIGCOMM*, 1990.

[10] S. E. Deering and D. R. Cheriton. Multicast routing in a datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[11] D. Gotz and K. Mayer-Patel. A General Framework for Multidimensional Adaptation. In *Proc. of ACM Multimedia*, New York, NY, USA, 2004. Association for Computing Machinery.

[12] D. Gotz and K. Mayer-Patel. A framework for scalable delivery of digitized spaces. *International Journal on Digital Libraries*, 5(3):205–218, May 2005. Special Issue on Digital Museums.

[13] G. Herman, K. C. Lee, and A. Weinrib. The datacycle architecture for very high throughput database systems. In *Proc. of ACM SIGMOD*, 1987.

[14] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proc. of SIGCOMM*, pages 89–100, 1997.

[15] L.-S. Juhn and L.-M. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, September 1997.

[16] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proc. of ACM SIGCOMM*, 1996.

[17] C. Perkins, O. Hodson, and V. Hardman. A survey of packet-loss recovery techniques for streaming audio. *IEEE Network*, 12:40–48, Sept./Oct. 1998.

[18] L. Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. *ACM Computer Communication Review*, 27(2):24–36, 1997.

[19] S. Rusinkiewicz and M. Levoy. Streaming qsplat: A viewer for networked visualization of large, dense models. In *Proceedings of ACM Interactive 3D Graphics*, 2001.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson.

RTP: A Transport Protocol for Real-Time Applications. *IETF Netowrk Working Group*, 1996. RFC 1889.

[21] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). *IETF Netowrk Working Group*, 1998. RFC 2326.

[22] E. Teler and D. Lischinski. Streaming of complex 3d scenes for remote walkthroughs. In *Proc. of Eurographics*, 2001.

[23] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4:197–208, 1996.

[24] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.

[25] C. Zhang and J. Li. Interactive browsing of 3d environment over the internet. In *Proc. of Visual Communication and Image Processing*, 2001.