Scalable, Adaptive Streaming for Nonlinear Media

Ketan Mayer-Patel
University of North Carolina at Chapel Hill

David Gotz
IBM T.J. Watson Research Center

Streaming linear media objects has become ubiquitous on today's Internet. Interactive, nonlinear media objects, such as large 3D models and visualization databases, have proven difficult to stream. The Channel Set Adaptation (CSA) framework lets clients request custom data flows for interactive applications using standard broadcast or multicast join and leave operations. CSA scales to support large user groups while providing interactive data access to clients.

More than two decades of research effort have focused on the various challenges presented by scalable streaming of continuous media data types (such as audio and video). (By *scalability*, we refer to the problem of supporting a large number of simultaneous clients by broadcasting video and audio to hundreds, thousands, or someday even millions of users. Although other notions of scale and scalability for multimedia systems are possible and important, our use of these terms in this article is limited to this idea.) To date, scalable media streaming has been largely limited to *linear media* (audio and video), which consists of data arranged in a fixed and linear ordering. (See the "Related Work in Linear Streaming" sidebar for more details.) Every user that accesses a linear media stream receives the same flow of information in the same order.

Recent advances in computing and interactive technology, however, have led to the growing importance of *nonlinear media*, such as video games, interactive visualizations, and virtual environments. Such objects provide individual data orderings to each user in response to the user's local requirements and interactions.

Linear and nonlinear media provide consumers with a fundamentally different experience. Nonlinear media experiences require unique presentations to each participating user. For example, every video game player receives a different flow of information in response to the player's interactions and movements within the game. Therefore, nonlinear media poses fundamentally new challenges for scalable media streaming. In particular, we can't deliver a custom data flow to each member of a large group of independent users using traditional media streaming techniques, which typically rely on common interests across a receiver population for an entire session. (See the "Related Work in Nonlinear Streaming" sidebar on p. 70 for a discussion of related literature.)

To address this problem, we propose Channel Set Adaptation (CSA), a novel approach that provides scalable and adaptive streaming for nonlinear media. Our driving design philosophy involves pushing all per-client work away from the server and toward individual clients. The CSA framework consists of three primary components—a data representation abstraction, a channel-based media communication model, and a client-driven adaptation algorithm—that let us distribute nonlinear media data sets to large groups of individual users. Each user can independently set data requirements and preferences. In response, custom data flows designed to match those individual needs are delivered to each client in a way that scales to support large user groups.

Achieving scalability

A scalable solution for nonlinear media streaming requires a carefully balanced design. Such a solution must manage the tradeoff between providing each client with a custom flow of information tailored to the client's specific needs and aligning the interests of clients so that scalable streaming techniques can be brought to bear.

Example nonlinear application

Let's use a digital museum application example to illustrate our discussion. Consider a digital museum that aims to digitize and share a famous space (for example, the Palace of Versailles) with a group of virtual visitors from around the globe. The application could capture a large set of digital pictures from the scene, store them in an *image-based rendering* (IBR) data set, and make them available online.

IBR is a computer graphics technique that uses real-world pictures from a scene as input and renders novel photorealistic views of the scene in response to a user moving a virtual viewpoint.[1] The views are generated by interpolating between the captured samples. Users can navigate through

1070-986X/07/$25.00 © 2007 IEEE

Published by the IEEE Computer Society

Related Work in Linear Streaming

Much of our work explores efficient streaming techniques for large user groups, but a large body of research exists on linear media streaming for audio and video.

Techniques

Streaming technologies for linear media objects have received much 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 audio and video content.

These technologies are based on several fundamental research efforts, including application-level framing,[1] forward-error correction,[2] and early research initiatives in developing successful streaming protocols.[3] This has led to several protocol standards for supporting real-time streaming, including the Real-Time Transfer Protocol (RTP)[4] and Real-Time Streaming Protocol (RTSP).[5]

Large user group distribution

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,[6] where data streams are efficiently distributed to groups of interested users, was developed as an efficient alternative to unicast.

Multicast lets a user join a group of receivers, all of whom receive an identical flow of data. A multicast server can then 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 deploying IP multicast, the standard version of infrastructure multicast, have led to significant effort in developing *application-level multicast* (ALM).[7,8] Rather than relying on core network resources to perform group management and data replication, ALM performs these tasks at the application level, using the hosts 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 linear data often requires more flexibility. Several researchers have explored novel uses of multicast protocols to provide limited flexibility for linear media access. For example, pyramid broadcasting[9] and its many derivatives[10,11] allow scalable video on demand. Similarly, other work, such as *receiver-driven layered multicast*,[12,13] 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 on the predictable access patterns associated with linear media applications. Our work explores techniques that exploit multicast delivery for scalable and adaptive nonlinear media streaming, where data access patterns aren't known a priori.

References

1. D.D. Clark and D.L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *Proc. ACM SIGCOMM*, ACM Press, 1990, pp. 200-208.
2. L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," *ACM Computer Comm. Review*, vol. 27, no. 2, 1997, pp. 24-36.
3. C. Perkins, O. Hodson, and V. Hardman, "A Survey of Packet-Loss Recovery Techniques for Streaming Audio," *IEEE Network*, vol. 12, Sept./Oct. 1998, pp. 40-48.
4. H. Schulzrinne et al., *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889, 1996.
5. H. Schulzrinne, A. Rao, and R. Lanphier, *Real Time Streaming Protocol*, RFC 2326, 1998.
6. S.E. Deering and D.R. Cheriton, "Multicast Routing in a Datagram Internetworks and Extended Lans," *ACM Trans. Computer Systems*, vol. 8, no. 2, 1990, pp. 85-110.
7. S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. ACM SIGCOMM*, ACM Press, 2002, pp. 205-217.
8. Y. Chu, S.G. Rao, and H. Zhang, "A Case for End System Multicast," *Proc. ACM SIGMETRICS*, ACM Press, 2000, pp. 1-12.
9. S. Viswanathan and T. Imielinski, "Metropolitan Area Video-on-Demand Service Using Pyramid Broadcasting," *Multimedia Systems*, vol. 4, no. 4, 1996, pp. 197-208.
10. K.A. Hua and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-On-Demand Systems," *Proc. SIGCOMM*, ACM Press, 1997, pp. 89-100.
11. L.-S. Juhn and L.-M. Tseng, "Harmonic Broadcasting for Video-on-Demand Service," *IEEE Trans. Broadcasting*, vol. 43, no. 3, 1997, pp. 268-271.
12. S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-Driven Layered Multicast," *Proc. ACM SIGCOMM*, ACM Press, 1996, pp. 117-130.
13. L. Vicisano, J. Crowcroft, and L. Rizzo, "TCP-Like Congestion Control for Layered Multicast Data Transfer," *Proc. INFOCOM*, ACM Press, 1997, pp. 996-1003.

the virtual space interactively, exploring the scene with the same freedom that video game players have while exploring a game's virtual setting.

IBR data sets are typically large. A digital museum would therefore want to stream the data set to each user to avoid long download delays. In addition, because users will navigate the scene independently, they will each require a unique

Related Work in Nonlinear Streaming

Several researchers have explored techniques for *single-user streaming* of nonlinear data sets, particularly in computer graphics. For example, it's possible to stream complex 3D geometric models by selectively transmitting multiresolution models of geometric objects based on the user's navigation of the scene.[1] 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 uses a more generic utility metric.[2]

Researchers have used progressive mesh representations, which prioritize geometric information based on their importance to overall shape, to develop geometric data streams that are resilient to lost packets during transmission.[3] 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.[4] Similar work has addressed the streaming techniques for point-based models.[5]

These techniques, while supporting streaming access to nonlinear 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 small user populations.

Recognizing the need for more scalable solutions, some researchers have explored support for broadcasting geometric data[6] for scalable access. However, this work is limited to broadcast environments and doesn't allow any per-client control over the received data flow. All users receive the same flow of information, making it most applicable to small data sets where last-mile bandwidth efficiency isn't a concern. Unfortunately, the Internet isn't a broadcast medium and the last-mile links are often the primary communication bottleneck link for individual clients.

The database community has also explored scalable access frameworks that attempt to scalably support large numbers of simultaneous queries. Two of these efforts, the Datacycle Architecture[7] and Broadcast Disks,[8] use solutions that draw on concepts that are closely related to our work.

The Datacycle Architecture serves 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, researchers developed Broadcast Disks 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.

References

1. E. Teler and D. Lischinski, "Streaming of Complex 3D Scenes for Remote Walkthroughs," *Proc. Eurographics*, European Assoc. for Computer Graphics, 2001, pp. 17-25.
2. D. Gotz and K. Mayer-Patel, "A General Framework for Multidimensional Adaptation," *Proc. ACM Multimedia*, ACM Press, 2004, pp. 612-619.
3. G. Al-Regib et al., "Protocol for Streaming Compressed 3-D Animations over Lossy Channels," *Proc. IEEE Int'l Conf. Multimedia and Expo*, IEEE CS Press, 2002, pp. 353-356.
4. C. Zhang and J. Li, "Interactive Browsing of 3D Environment over the Internet," *Proc. Visual Comm. and Image Processing*, SPIE, vol. 4310, 2001, pp. 509-520.
5. S. Rusinkiewicz and M. Levoy, "Streaming Qsplat: A Viewer for Networked Visualization of Large, Dense Models," *Proc. ACM Interactive 3D Graphics*, ACM Press, 2001, pp. 63-68.
6. S. Bischoff and L. Kobbelt, "Towards Robust Broadcasting of Geometry Data," *Computers and Graphics*, vol. 26, no. 5, 2002, pp. 665-675.
7. G. Herman, K.C. Lee, and A. Weinrib, "The Datacycle Architecture for Very High Throughput Database Systems," *Proc. ACM SIGMOD*, ACM Press, 1987, pp. 97-103.
8. S. Acharya et al., "Broadcast Disks: Data Management for Asymmetric Communication Environments," *Proc. ACM SIGMOD*, ACM Press, 1995, pp. 199-210.

flow of image data. We therefore need to support scalable, nonlinear streaming to support a large group of museum visitors.

Adaptive extreme

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 (position in space) and syntactic (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. Every client can custom compose the incoming stream of images by specifically requesting each photograph.

However, this approach doesn't 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 supports per-client nonlinear access at the expense of scalability.

Scalable extreme

At the other extreme, the most scalable architecture for the digital museum application is to simply order the images in the database and use existing scalable streaming techniques to distribute the database. Individual clients would then receive the streamed data until they have downloaded the entire data set.

Although this architecture is highly scalable, clients have no options for adapting the flow of images and must settle for the predefined linear ordering. Clients that are lucky enough to receive the images they currently need at the beginning of the stream might be able to immediately present the user with the correct virtual view. Clients that aren't so fortunate will either have to delay rendering or make do with suboptimal images until the images they most need finally arrive.

Broadcast and multicast work well for linear media distribution because all users have identical interests in the order of the media units. It fails, however, to allow the nonlinear data access required for nonlinear media access as in our example digital museum application.

Key insights

To negotiate a scalable and adaptive solution between these two extremes, we must recognize two key insights:

- The server must do as little per-client work as possible.
- We can use application-level knowledge to exploit access coherence.

These two insights are at the heart of the CSA design.

Simple-server design philosophy. Our design goal is to push all computation away from the server and toward the participating clients. Two factors motivate this client-driven approach.

Although client access to the media data might be nonlinear, it is also not random.

First, we're striving for a constant-server-load model. If the server itself handles any per-client tasks, it's impossible to reach that goal. Second, each client has independent data access and adaptation requirements that 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. This design leads directly to a bounded server load that is independent of the number of participating clients and thus achieves scalability.

Exploiting access coherence. Although client access to the media data might be nonlinear, it is also not random. As a result, the data needs of subsets of clients might often come into transient alignment. By employing application-level knowledge about the media data elements and how they are related to each other as well as how clients are likely to access them, we can partition the data set into independent units for which we expect a high degree of access coherence. Imposing a linear ordering on these partitioned units lets us utilize traditional scalable streaming techniques (see the "Media Streaming Techniques" sidebar for more details).

For example, in our sample application, we might group all the low-resolution pictures from one corner of a room as one partition. We could then distribute this group of pictures using multicast or broadcast 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 instead subscribe to an alternate multicast stream with an image cluster that more closely matched their requirements.

Media Streaming Techniques

The result of the media streaming research has been an extensive toolkit of techniques and methods that developers can use to engineer scalable streaming solutions for a range of contexts and applications. Layered video coding[1] and multiple description coding[2] techniques let clients adapt media quality to match channel capacity. Multicast protocols provide efficient distribution.[3,4] When infrastructure support for multicast proved undeployable, researchers developed application-level multicast via overlay or peer-to-peer networks.[5,6] Pyramid broadcasting addresses the problem of providing near-zero delay media on demand for clients that access the same stream but at different starting times.[7,8] Patching and batching techniques can further optimize system scalability.[9] The examples here are only a representative sample of a large body of work for each of these techniques.

References

1. R. Mathew and J. Arnold, "Efficient Layered Video Coding using Data Partitioning," *Signal Processing: Image Comm.*, vol. 14, no. 9, 1999, pp. 761-782.
2. V. Goyal, "Multiple Description Coding: Compression Meets the Network," *IEEE Signal Processing Magazine*, vol. 18, no. 5, 2001, pp. 74-94.
3. S. Deering, "Multicast Routing in a Datagram Internetwork," doctoral dissertation, Stanford Univ., Dept. of Computer Science, 1991.
4. H. Holbrook and D. Cheriton, "IP Multicast Channels: EXPRESS Support for Large-Scale Single-Source Applications," *Proc. SIGCOMM*, ACM Press, 1999, pp. 65-78.
5. M. Castro et al., "Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," *IEEE J. Selected Areas in Comm.*, vol. 20, no. 8, 2002, pp. 1489-1499.
6. S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. SIGCOMM*, ACM Press, 2002, pp. 205-217.
7. C. Aggarwal, J. Wolf, and P. Yu, "A Permutation-Based Pyramid Broadcasting Scheme for Video-On-Demand Systems," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems*, IEEE CS Press, 1996, pp. 118-126.
8. K. Hua and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-On-Demand Systems," *Proc. ACM SIGCOMM*, ACM Press, 1997, pp. 89-100.
9. K. Hua, Y. Chai, and S. Sheu, "Patching: A Multicast Technique for True Video-on-Demand Services," *Proc. ACM Multimedia*, ACM Press, 1998, pp. 191-200.

Thus, clients can adaptively access the data in a nonlinear manner by independently choosing which partitions to access. At the same time, the server can scalably distribute data to subsets of clients that choose to access the same partition.

Within a partition, clients won't have adaptive access. Thus, a partition will only be effective if the subset of clients accessing the data are likely to have similar access requirements for the data within that partition. If so, the server can construct a linear ordering within that partition that might not be perfect for anyone, but it's at least reasonable for all.

For example, as users move from one room to another in an IBR data set, they could change subscriptions to receive streams that contain pictures from their changing location. At first, the users could subscribe to low-resolution and sparsely located image channels for regions along a hallway where they are moving quickly. After arriving and pausing to look around a new room, the client could dedicate all of its bandwidth to dense, high-resolution image channels for the users' new location.

The key to this approach is using application-level knowledge to determine a partitioning that provides the appropriate tradeoff between adaptability and scalability. A few relatively large partitions will provide greater scalability for the server, but this forces clients to choose between coarse subsets of the data linearly ordered in a way that might be less likely to match their adaptive requirements. A larger number of smaller partitions will provide greater adaptability for the clients, but this reduces scalability because the expected number of clients accessing the same partition at the same time will decrease and the overall number of channels the server must be prepared to support increases.

Once this tradeoff is determined and the available data is appropriately partitioned and mapped by the application designers into a set of channels, clients must be made aware of the available communication channels and their associated semantic meanings—that is, which images are in which channel—so they can intelligently subscribe to the multicast streams that contain information most relevant to their needs. As those needs change over time, clients can then quickly subscribe to whichever streams become most appropriate.

In this way, clients can compose a custom flow of images based on 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 streams over a broadcast (similar to a cable TV network) or multicast-enabled (such as an IP or application-layer multicast) network. If each channel remains reasonable in terms of bandwidth, the system can support heterogeneous and time-varying receiver bandwidth by varying the number of channels each client subscribes to. This channel-based approach to providing scalable and adaptive access to nonlinear media forms the conceptual foundation for our research.

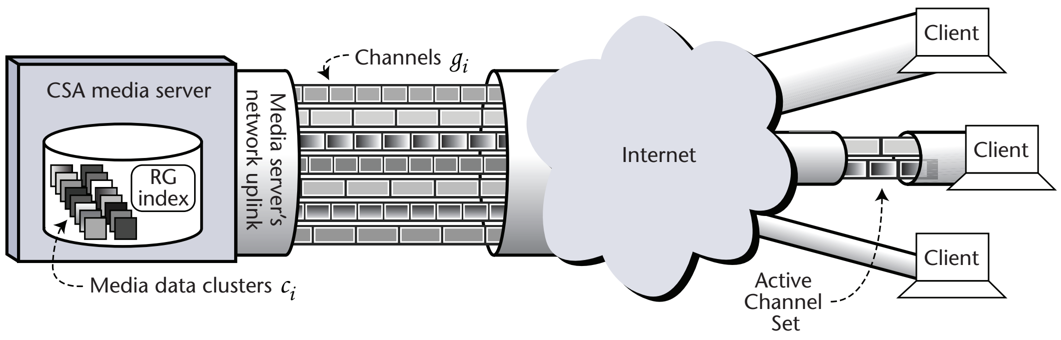


Figure 1. Channel Set Adaptation (CSA) enables efficient streaming of nonlinear media to large groups of independently operating users. We partition the media object into semantically meaningful clusters, labeled c_i***. These clusters are then mapped to a large set of broadcast or multicast channels, 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.***

Channel Set Adaptation

Our CSA framework lets individual clients request custom data flows for interactive applications using standard multicast join and leave operations. The three major portions of the CSA framework include the data representation formalisms we use to express data relationships and dependencies; its communication model, which we designed to provide scalable service to large user groups; and the client-driven adaptation algorithms that perform congestion and content control. Figure 1 gives the overall architecture.

Data representation

The CSA framework requires a data representation that formally expresses syntactic (encoding dependencies) and semantic (similarity in meaning or utility) data relationships. To satisfy this requirement, we use the *representation graph* (RG) abstraction[2] first proposed as a generic representation model for multidimensional adaptation. The RG model is a flexible representation abstraction designed specifically for expressing syntactic and semantic data relationships in multimedia databases. It also provides mechanisms for evaluating the relative utility of individual elements of information in a database based on dynamic system conditions.

An RG consists 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 that are accessed atomically. Each cluster is considered a semantically consistent unit of data. An RG's underlying structure, including the list of nodes and their connectivity, is stored explicitly as an *RG index*. Two parts of the RG abstraction are particularly important within the context of CSA: clusters and the RG index.

In our example digital museum application, the clusters application might represent groups of images captured from nearby locations. When modeled using an RG, a data set is essentially partitioned into a set of clusters, $C = \{c_1, \ldots, c_n\}$. The RG model supports multiple-descriptor encoding, allowing individual elements (such as pictures in the digital museum application) to be encoded in more than one cluster.

The RG index is a specification of the RG's underlying structure. 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 is embedded. The index doesn't include any actual media data and is therefore small compared to the overall data set. For example, in the digital museum application, the RG index could include metadata describing the images that make up the IBR data set (including the image resolution and camera position) and a description of how the images are clustered.

Media communication model

Our media communication model is designed to meet two goals. First, the model must let individual users access the nonlinear 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. Our design consists of three primary components: channel-based transmission, session initiation, and client behavior.

Channel-based transmission. CSA requires the central server to maintain a large set of communication channels, noted as $G = \{g_1, \ldots, g_n\}$. 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 accept the data flow assigned to the active channel or unsubscribe to terminate the flow of information. Both broadcast and multicast networks easily support our channel-based transmission scheme's subscription model.

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 (for example, a set of images captured from the northeast corner of a particular room), the mapping M assigns a semantic meaning to each channel g_i. We append the mapping information in M to the RG index.

At runtime, the server simultaneously transmits all channels in G, with each channel g_i transmitted at a constant bit rate. Although at first this approach might appear to waste significant bandwidth, our experimental results (which we discuss later in this article) show that for large user groups CSA is significantly more efficient in terms of bandwidth than a traditional unicast approach to nonlinear streaming.

Similarly, CSA might seem inefficient with respect to multicast channels, which have traditionally been a scarce resource within a global IP multicast namespace. However, given the lack of deployment for IP multicast, CSA would likely be deployed over application-layer multicast or a dedicated broadcast network where channels are locally defined and plentiful.

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. Each channel is encoded using forward error correction to guard against lost packets. In the worst case, a client can continue receiving a channel for a full carousel cycle to rereceive a critical lost packet.

In the digital museum example, the data assigned to each channel is static: a fixed set of images. In other applications, this data may be dynamic. Although dynamic data is generally compatible with CSA, it introduces additional performance implications, which we don't study in this article. The server isn't responsible for any tasks other than transmitting data over the set of channels, adhering to our simple-server design philosophy.

Session initiation. Clients are responsible for initiating 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 available through a well-known HTTP or FTP host.

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

Client behavior. Following session initiation, a client has all the information it needs to begin receiving the nonlinear data stream. Using the client-driven adaptation algorithm (which we describe later on), 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 dynamically choosing which channels are in the ACS and how many channels are active, clients can compose a unique stream that delivers a custom flow of nonlinear media data that's individually tailored to meet their needs.

At any point, the set of clients subscribed to a particular channel are all individually expressing their interest in a common data stream—the semantic data assigned to the channel. We use this common interest to allow scalable delivery. However, unlike linear media streaming, the overlap in interests between this set of users is transient because individual clients will join and leave channels according to local interests, which evolve independently over time. For example, two users navigating the digital museum application might momentarily explore the same region of virtual space when their paths through the IBR environment cross.

Satisfying design goals. Our media communication model meets our two primary design goals. First, individual users can access the nonlinear media stream interactively and independently by managing the ACS. Second, our model easily supports large groups of independent users because of the channel-based transmission design.

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 a broadcast or multicast network can support. This leads to a highly scalable server-side solution with a performance independent of the number of participating clients.

This independence lets us determine a constant upper bound on computation and bandwidth requirements. As a result, we can properly provision servers with a finite and static level of resources to support, in the ideal case, an infinite number of simultaneous users.

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 it manages its ACS.

ACS management is performed through two fundamental operations. The first operation, Sub$\{g_i, \text{ACS}\}$, is used to subscribe to a new channel. Upon subscription, the new channel is added to the ACS. The second operation, Unsub$\{g_j, \text{ACS}\}$, 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 as

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

$$\text{Unsub}\{g_j, \text{ACS}\} = \text{ACS} \backslash g_j \quad (2)$$

Clients can perform the subscribe and unsubscribe operations 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 and 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 nonlinear media applications. We define both of these adaptive tasks in terms of the subscribe and unsubscribe operations.

Congestion control. A client participating in a nonlinear 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 ACS size.

Under our channel-based transmission scheme, the server offers a large set of constant bit-rate channels, G. Clients subscribe to a subset of this offering, so that $\text{ACS} \subset G$. Because each channel $g_i \in \text{ACS}$ is offered at a constant bit rate, the overall bit rate of the arriving ACS is determined by the size of the set, or $|\text{ACS}|$. The congestion-control problem for CSA is analogous to the problem faced in Receiver-Driven Layered Multicast,[3] and we apply a similar solution.

At runtime, the client adjusts the ACS size through subscribe and unsubscribe operations. At signs of network congestion, such as the detection of lost packets, the client decreases $|\text{ACS}|$ through an unsubscribe operation. To maintain the most useful data flow after the decrease in subscription level, a client unsubscribes from the least useful active channel. We utilize the Utility-Cost Ratio (UCR) metric[2] for this evaluation, which combines application-specific utility and cost functions to determine how best to adapt a multimedia data set.

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

We use a series of timers for each subscription level to improve stability and to allow the system to converge more quickly to an appropriate subscription level.

Content control. In parallel with congestion control, each client must also perform content control. This task is unique to the problem of nonlinear streaming. In traditional linear media applications, data is delivered in a fixed order and there's 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 nonlinear 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 lets us 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

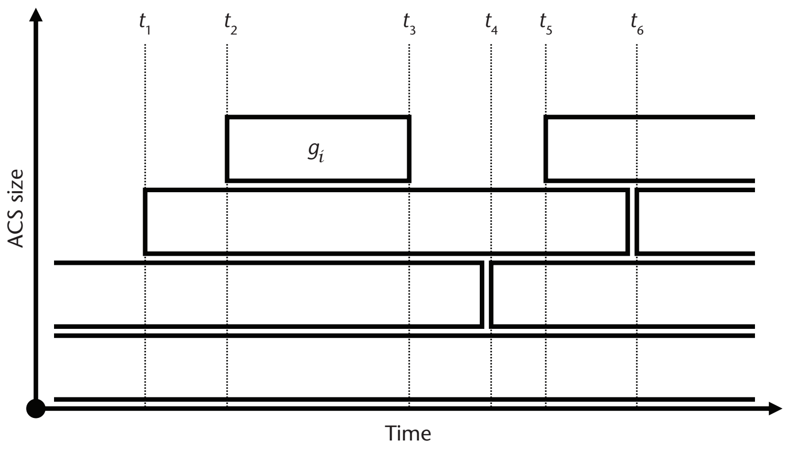


Figure 2. A hypothetical plot that shows the evolution of the Active Channel Set over time.

useful inactive channel, g_{inactive}. Whenever it discovers that the utility of g_{inactive} is greater than that of g_{active}, the two swap positions and g_{inactive} becomes an ACS member.

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

The UCR metric is a spatial measure of utility defined on the RG 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 its own pattern of subscription requests based on its own local needs. For example, Figure 2 illustrates a possible sequence of subscription operations over a small time window. 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 contracts the ACS back down to size three. The content control algorithm initiates a channel swap at time t_4. This 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 data flow delivered to each individual client. In addition, the unique flow is composed without any direct communication between clients and the server.

Experimental prototype

We developed an experimental prototype to evaluate the performance of CSA using an IBR tool for digital museums as the target application.

Prototype application

IBR environments allow free viewpoint exploration of digitized spaces, immersing users in a photorealistic recreation of a virtual place that they can navigate interactively. When combined with digital museums, these environments promise to enable large populations from around the globe to explore remote artifacts and locations.[4]

IBR applications require nonlinear access to possibly massive sets of images as input to their reconstruction algorithms, making them a strong match for our work.

We designed our prototype as a client–server system with a single image server that uses CSA to scalably transmit streams of images to a set of interested clients. The sample data set we used in our experiments consists of approximately 8,000 color images captured from within a library environment. Each image has a 512×512 pixel resolution. The pictures are distributed across a 2D plane at eye level.

We use an RG composed of nearly 16,000 nodes, corresponding to each of the 8,000 images stored at two resolutions. The number of clusters in the RG varied across experiments. The overall RG was embedded within a 5D utility space, defined by three spatial dimensions, image resolution, and spatial density.

Experiment testbed

We performed a series of experiments to evaluate CSA's performance. Rather than rely on network simulation, we executed our experiments on the Emulab network testbed,[5] which uses network emulation to achieve more realistic operating conditions. Using emulation as the basis for our evaluation let us send live data flows over real networks to real machines. At the same time, it provided an experimental platform within which we could control several critical parameters to gain a better understanding of CSA's performance across a range of operating environments.

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

Experiment methodology

Our methodology in evaluating the performance of our experimental prototype involved client emulation and formulating the Summed Utility Metric (SUM).

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 10-minute path through the IBR data set. The path included various movement types including fast and slow movements and direction changes.

During the 10-minute execution time, we computed a performance metric once per second. (We describe the performance metric in more detail later on.) When presenting average performance values, we considered only the second 5 minutes of performance data from each 10-minute session. This let us avoid any transient events that might occur during the early and less stable moments of a session when analyzing average behavior.

SUM performance metric. Throughout our evaluation, we measured performance using the SUM, an application-independent performance metric for evaluating our prototype's behavior based on the abstract adaptation framework.[2] The SUM measures system performance as a function of the RG's current state.

The SUM requires no domain-specific knowledge because it's 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 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 as long as the underlying RG and utility metrics remain the same.

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 lets us mark obtained data elements by placing the elements in the resolved state. We can then apply the utility metric to each resolved node. The SUM is the sum of all the resolved node utility values. (A full discussion on possible node states and state transitions is beyond the scope of this article. See related work[2] for more information.) We formally define the SUM metric as

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

where R is the set of all resolved nodes and UtilMetric is the application-specific utility metric.

The SUM is a measure of performance at a single point in time and is therefore highly dynamic. To capture a reliable measure of system behavior, we must repeatedly evaluate the SUM metric over a period of time. For this reason, the results presented in the next section are evaluated using either a cumulative or average value of the numerous instantaneous SUM values captured during each experiment.

Results

We performed a series of experiments using our prototype and the Emulab network testbed. These experiments highlight CSA's capability. We also explored the impact of certain engineering parameters on overall performance.

Scalable delivery

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 nonlinear media distribution. The remaining two configurations were CSA-based, using multicast and broadcast networks.

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 TV). 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 one to 65. Because we used emulation with actual hardware resources, the upper limit in this range was determined by the testbed's size. We used the same adaptation library and identical utility metrics across all experiments, letting us 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

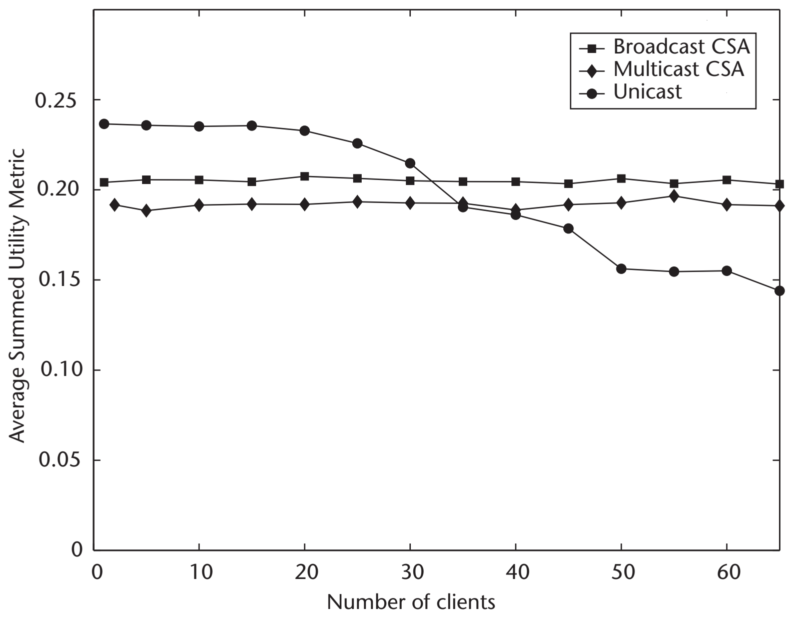


Figure 3. CSA system performance for group sizes up to 65.

RG, but 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.

Figure 3 shows the results of these experiments. For all experiments, we provisioned the server with 100 Mbps of bandwidth and each client with 5 Mbps. As a result, the unicast server was able to fully support nearly 20 clients ($n = 20$) before saturating its network resources. At small group sizes of $n < 20$, the unicast configuration outperforms the broadcast-based CSA variant by roughly 14 percent. 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 let the clients 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 its saturation point. Past this point, the performance drops as the group size increases. Performance will asymptotically approach zero as the server's bandwidth is divided in service 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 nonlinear media streams to large user groups without saturating the central server. This is highlighted by the flat plot for broadcast performance in Figure 3. Although 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 3 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 3 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 principal adaptive mechanism. Slower adaptation leads to a drop in performance. (We further explore the impact of these overheads shortly.)

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.

Adaptation for congestion control

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

We evaluated our congestion-control algorithm's performance by observing its behavior in the face of 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. Figure 4 shows the results.

In the first 30 seconds, the ACS size 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 Figure 4 shows, the client initially hesitates to back down from $|ACS| = 10$. The delayed response is because 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 $t = 330$, when we removed the HTTP traffic from the bottleneck link, the client detected the improved network conditions. Very rapidly, it increased the ACS size to 10, following the same probing pattern as at the start of the session.

Clients can tune the timers used to govern the rate of increase and decrease in ACS size and configure them 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. In the future, we will explore coordinated approaches to achieving more TCP-friendly congestion control[6] by assigning partial priorities to channels, for example, for high- versus low-resolution pictures.

Adaptation for content control

Clients perform content control in CSA by managing the set of active channels over time to ensure that the data flow arriving at a particular client matches that client's application requirements. This task is a requirement specific to nonlinear media that typical media streaming architectures need not address. In contrast, content adaptation, which we described earlier, is essential for supporting access to nonlinear media. Two system parameters directly impact content-control performance.

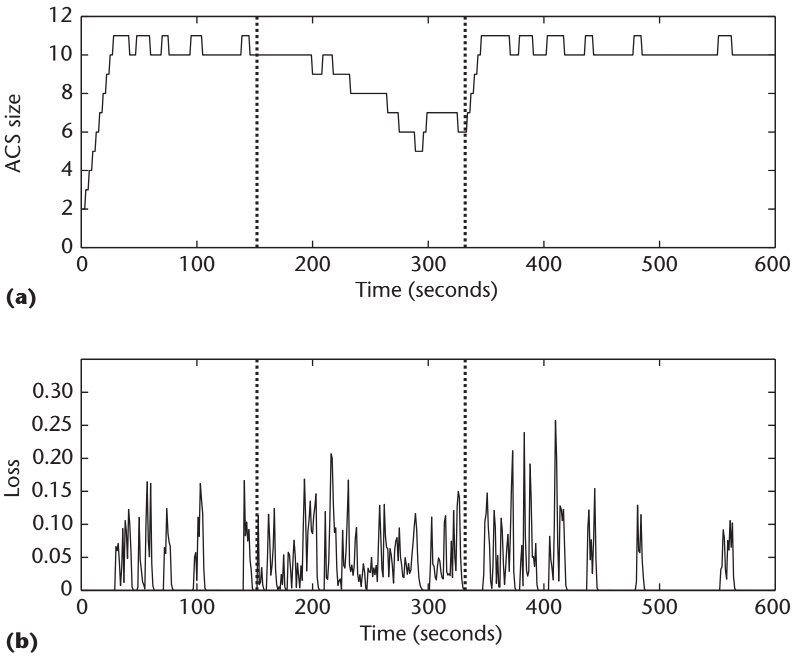


Figure 4. Rate adaptation in response to HTTP cross traffic. The adaptation constants for altering the Active Channel Set (ACS) size can be tuned to provide quicker back-off at the expense of data-rate stability. The plots show (a) ACS size versus time and (b) loss versus time.

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

In particular, various multicast implementations (such as IP multicast and *application-level multicast* protocols) exhibit a range in *leave latency*—the time it takes between an unsubscribe 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 5,000 milliseconds. Figure 5 (next page) gives the results. The experiment shows that longer latencies have a negative impact on performance. In particular, the 3-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 2 and 3 seconds of leave latency. The overall trend in performance is important. However, the drop's exact slope highly depends on the fraction of time spent on overhead and on the experiment's specific parameters:

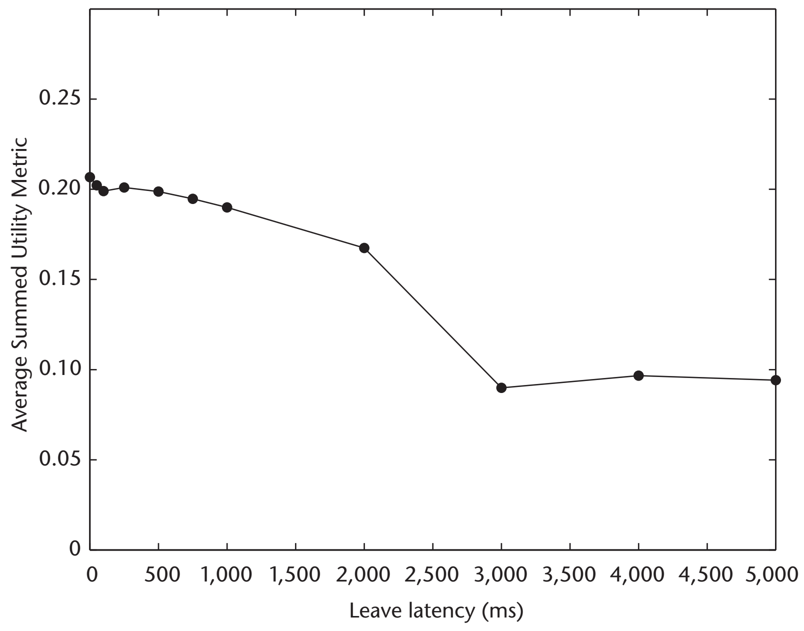


Figure 5. Impact of leave latency on performance.

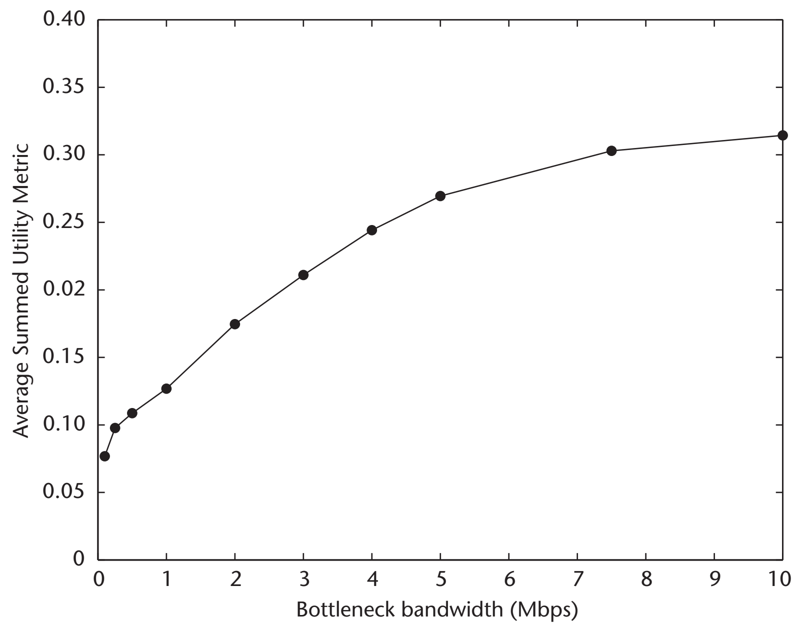


Figure 6. Bandwidth impact on content control.

$$\text{Overhead} = (\text{SubscriptionOpLatency}/\text{ListenTime}) \quad (4)$$

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're developing StrandCast,[7] an 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 low leave latencies, will make it ideal for CSA-based applications.

Bandwidth impact. Bandwidth is a second parameter that has a direct impact on content-control performance. Figure 6 shows the results from a series of experiments in which we measured average performance across a broad spectrum of network bottleneck bandwidths, ranging from 0.1 to 10 Mbps. As expected, content control on clients with a higher bit rate outperformed those provisioned with lower capabilities. The adaptation algorithm uses additional bandwidth to obtain additional channels of data, yielding higher SUM values.

More importantly, the results indicate that marginal utility gained from each unit of additional bandwidth decreases. The shape of the curve in Figure 6 illustrates this behavior. We find the steepest gains in utility when increasing bandwidth from a low value. The gains tend to plateau as the bottleneck link speed increases.

Although not necessarily intuitive, the decrease in marginal utility at higher bit rates is the designed result for effective content control. For example, the ideal content-control algorithm would ensure that for a bottleneck bit rate of x, the system can receive the n most useful units of media data (*nodes* in the RG data model). In this scenario, there wouldn't be a single node of data yet to be received that had a higher utility than any of the n nodes already received by the client.

Meanwhile, assuming all nodes are the same size, a system with a bottleneck bit rate of twice the previous example would be able to resolve those same n nodes plus n additional nodes. However, due to the content-control algorithm, the additional n nodes wouldn't be as useful as the first n. Therefore, although the system with twice the bandwidth was able to resolve twice as much data, the additional data wasn't as useful as the first n nodes.

The results of this experiment show that the CSA content-control algorithm exhibits the desired sublinear growth in utility. Figure 6 shows that the marginal increase in utility drops continuously as bandwidth increases. The shape implies that at each configuration the CSA content-control algorithm can effectively access the most useful data, leaving less desirable data to be received at higher data rates.

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 data set. 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 data set 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 7 shows a series of experiments performed using a range of channel set configurations. We performed all these experiments using an idealized network environment with negligible channel subscription overheads. As expected, we achieved the best performance with the highest number of clusters, each of which maps to an individual channel.

In practice, the benefit of additional channels is greatest when the subscription operation latency is negligible. However, the impact of latency on performance is far more pronounced in high-channel configurations. Figure 8 demonstrates the results of several experiments where we artificially controlled the subscription operation latency within the range of 0 to 1,000 ms for four different channel set size configurations.

All configurations performed more poorly as the subscription operation latencies increase, as the overhead model defined in Equation 4 predicted. However, the rate of decline was significantly steeper 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.

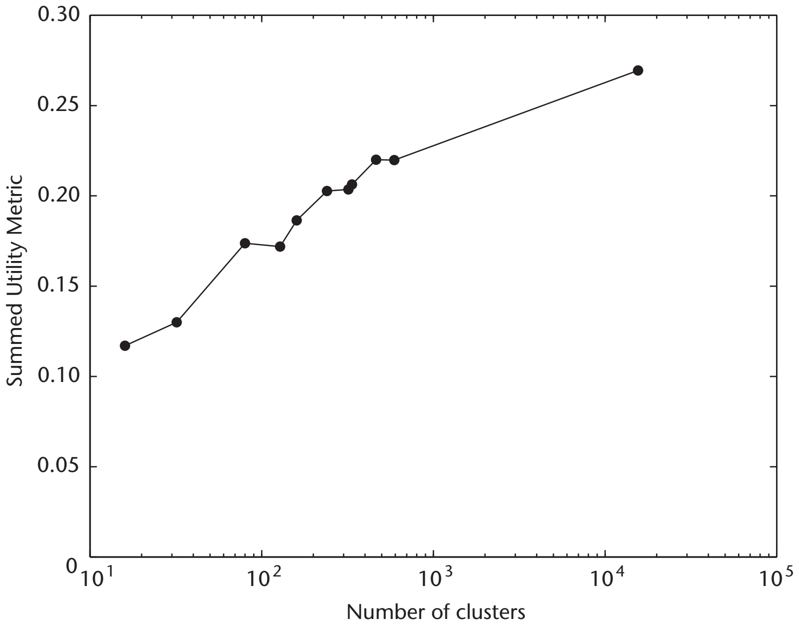


Figure 7. Impact of the cluster count, which determines the size of the channel set G, on CSA performance when subscription operation latency is negligible.

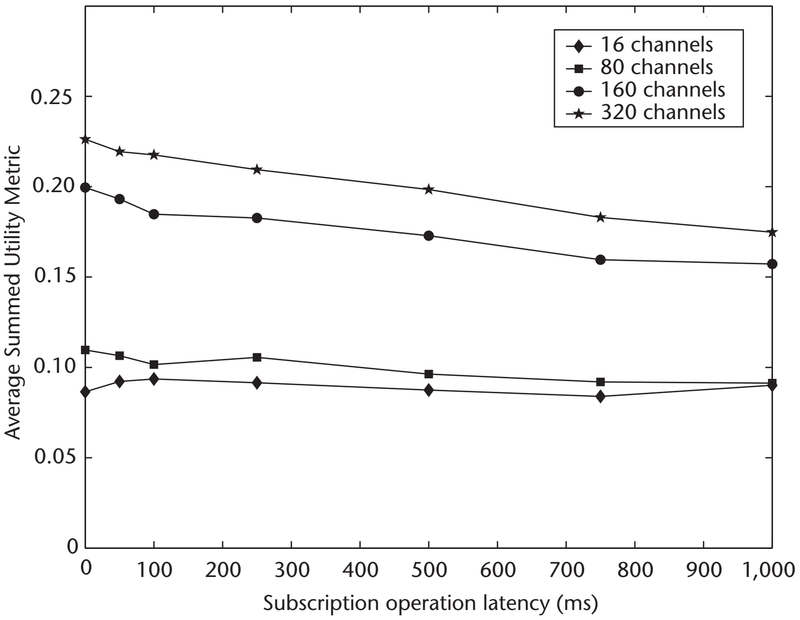


Figure 8. Relationship between performance, latency, and the number of available channels.

As these results and Equation 4 reflect, a shorter expected listen time magnifies the impact of changes in the subscription latency.

At first glance, these 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 outper-

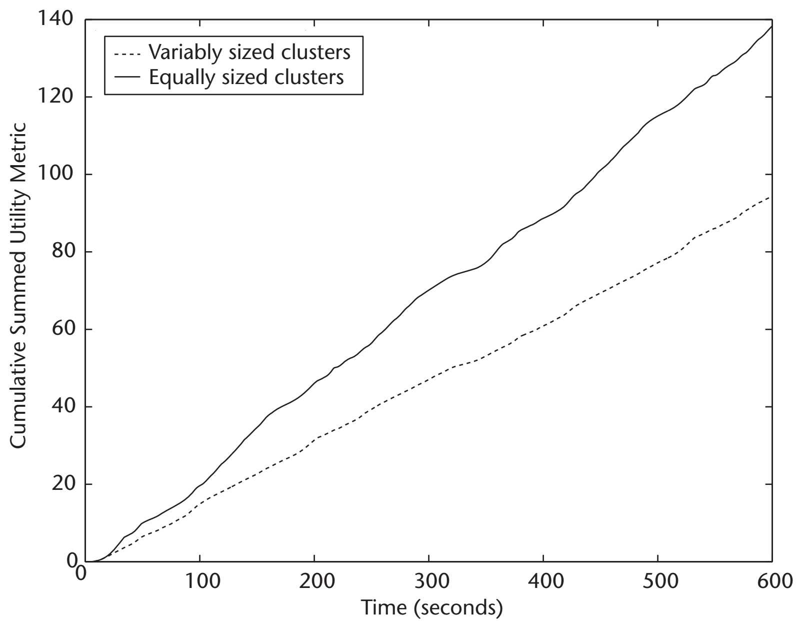


Figure 9. Performance impact of data-to-channel assignment strategies.

forms both CSA variants for small user groups.

However, in a CSA-based system supporting large user groups, this wouldn't 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 lead 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 highly depends 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.

Channel cycle size distribution

The amount of data assigned to each CSA channel is a critical design decision in formulating an effective mapping $M : C \mapsto G$, which binds specific clusters of media data to individual channels, as we described earlier. Just as the number of channels influences overall performance, so too does the relative size of each cluster. For example, one mapping might use clusters of data that are all the same size. Conversely, the clusters could be specified so that their size greatly varies.

The amount of data in a cluster directly determines the cycle time for the associated communication channel. This is because CSA transmits the data for an individual channel using a carousel schedule, where every byte is sent out over the channel before starting once again from the beginning.

The time required to send out all bytes over the channel is the *cycle time*. For clusters with little data assigned to them, the cycle time can be short. For larger clusters, the cycle time can be long. The cycle time is critical because it determines the expected time required to access a specific node of data. Because CSA transmits data cyclically and clients can join a channel at any time, the expected access time for any node is equal to one half the cycle time. Therefore, the expected access time for a node in a large cluster is much longer than the expected access time for data in a small cluster.

In practice, the proper configuration of edges to clusters highly depends on the application. Certain applications might perform best with relatively even access times across clusters. Other applications might require fast access to certain data elements and would therefore perform better using variably sized clusters.

Figure 9 highlights the impact of channel cycle size on our prototype application. We measured the application's performance using two different configurations. In one session, we arranged the system with clusters of roughly equal size. This provided a nearly constant expected access time across the entire data set. In the second session, we variably sized the channels to allow faster access to low-resolution data (at the expense of higher access times for high-resolution data).

The two plots in Figure 9 show the cumulative SUM value over the life of each of the two sessions. The plots indicate that for the prototype application, equally sized channels have superior performance when compared to variably sized channels. The results in this experiment indicate that, for this application, the benefit of faster access to certain data isn't worth the penalty of longer access times to the remaining data.

Generally, the impact of cluster size on performance is highly application dependent, and careful experimentation should be performed to determine the optimal configuration. As these results highlight, the allocation of data across the set of channels can have a significant impact on overall performance.

Future work

Despite the promise of our initial results, several areas still 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. However, in the future, we want to move beyond this assumption and plan to enhance our algorithm to perform well even in the presence of nonshared bottleneck links.

We're 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 distribution of dynamic data sets where content assigned to each channel changes over time. In addition, dynamic mappings could help reallocate communication resources to better serve high-demand portions of the overall data set—that is, the *Mona Lisa* effect, where certain parts of a data set get disproportionate attention from users just as the famous painting gets far more attention than others in the Louvre.

Another important area for future work is evaluating our framework with large user groups. Extrapolating the results we present here hints at strong performance benefits for large group sizes, but our experiments were limited to groups of 65 because of the required infrastructure.

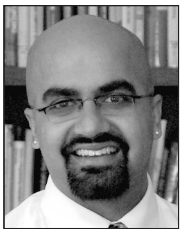
Finally, our results highlight the need for an efficient multicast protocol that has low subscription operation latency and that can support high-churn environments. Most of the existing protocols have fairly high subscription latencies and are designed to support long-term sessions. CSA places fundamentally new demands on the multicast infrastructure, and new protocols can be designed that would provide significantly improved performance. We've already begun implementing StrandCast[7] to support technologies like CSA. **MM**

Acknowledgments

This work was partially supported by the National Science Foundation grant #ANI-0238260. This article is an expanded and revised version of a paper presented at the ACM International Conference on Multimedia 2006.[8] Here we've expanded and reorganized our discussion of the driving design principles and provided additional experimental results.

References

1. D.G. Aliaga and I. Carlbom, "Plenoptic Stitching: A Scalable Method for Reconstructing 3D Interactive Walkthroughs," *Proc. ACM SIGGRAPH*, ACM Press, 2001, pp. 443-450.
2. D. Gotz and K. Mayer-Patel, "A General Framework for Multidimensional Adaptation," *Proc. ACM Multimedia*, ACM Press, 2004, pp. 612-619.
3. S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-Driven Layered Multicast," *Proc. ACM SIGCOMM*, ACM Press, 1996, pp. 117-130.
4. D. Gotz and K. Mayer-Patel, "A Framework for Scalable Delivery of Digitized Spaces," *Int'l J. Digital Libraries*, special Issue on digital museums, vol. 5, no. 3, 2005, pp. 205-218.
5. B. White et al., "An Integrated Experimental Environment for Distributed Systems and Networks," *Proc. 5th Symp. OS Design and Implementation*, Usenix Assoc., 2002, pp. 255-270.
6. L. Vicisano, J. Crowcroft, and L. Rizzo, "TCP-Like Congestion Control for Layered Multicast Data Transfer," *Proc. INFOCOM*, ACM Press, 1998, pp. 996-1002.
7. B. Begnoche et al., *The Design and Implementation of Strandcast*, tech. report TR05-004, Dept. Computer Science, Univ. North Carolina Chapel Hill, 2005.
8. D. Gotz, "Scalable and Adaptive Streaming for Nonlinear Media," *Proc. 14th Ann. ACM Int'l Conf. Multimedia*, ACM Press, 2006, pp. 357-366.



Ketan Mayer-Patel is an associate professor of computer science at the University of North Carolina at Chapel Hill. His research focuses on multimedia systems and networking. He received a PhD in computer science from the University of California, Berkeley.

David Gotz is a research staff member at the IBM T.J. Watson Research Center. His research interests include multimedia systems and networking, visualization, and computer graphics. He received a PhD in computer science from the University of North Carolina at Chapel Hill.

Readers may contact David Gotz at the IBM T.J. Watson Research Center, 19 Skyline Dr., Hawthorne, NY 10532; dgotz@us.ibm.com.