Algorithms for Multiplex Scheduling of Object-Based Audio–Visual Presentations

Hari Kalva, *Member, IEEE,* and Alexandros Eleftheriadis, *Senior Member, IEEE*

***Abstract*—Object-based representation of audio–visual (AV) presentations provides a flexible scheme to create interactive content that lends itself to resource-driven adaptation. The content adaptation needs of mobile devices can be met well with the use of object-based AV presentations. The main distinguishing feature of object-based AV presentations is the scene composition at the user terminal. In this paper, we discuss the problem of scheduling the delivery of object-based AV presentations under resource constraints. We explore the similarities with the problem of job sequencing on a single machine. We present a family of algorithms to determine the schedulability of AV presentations, and for unschedulable presentations, we present algorithms to compute a schedule that minimizes the additionally acquired resources. We present algorithms for computing incremental schedules for applications such as content authoring that require immediate feedback on resource consumption. The algorithms can be used to schedule object-based MPEG-4 presentations. We discuss the algorithms and results by considering a relatively complex MPEG-4 presentation with 16 objects, including audio, video, and images.**

***Index Terms*—Delivery scheduling, MPEG-4, object-based content, scheduling algorithms.**

NOMENCLATURE

N	Set of objects to be scheduled.
N	Number of objects to be scheduled.
n_i	Number of access units (AUs) per object ($1 \leq i \leq N$).
$A_j(k)$	Access unit k of object j.
A	$\equiv \bigcup_{j,k} A_j(k)$. Set of all AUs in the presentation.
$T_j^d(k)$	Decoding time of $A_j(k)$.
$T_j^s(k)$	Send time of $A_j(k)$.
σ	$\equiv \bigcup_{j,k} T_j^s(k)$. Send-time schedule.
C	Transmission channel of capacity C.
$s_j(k)$	Size in bytes of $A_j(k)$.
$d_j(k)$	Duration (channel occupancy) of AU k on the wire; $d_j(k) = C \div s_j(k)$.
T_s	Startup delay.
$T_s^{\max}$	Maximum startup delay.
$T_s^{\min}$	$= \sum_k d_k(0) \ni T_k(0) = 0$. Time to transmit AUs of all objects with DTS/CTS of zero.

Manuscript received April 10, 2000; revised July 27, 2003. This paper was recommended by Associate Editor A. Puri.

H. Kalva is with the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: hari@cse.fau.edu).

A. Eleftheriadis is with the Department of Electrical Engineering, Columbia University, Columbia, NY 10027 USA (e-mail: eleft@ee.columbia.edu).

Digital Object Identifier 10.1109/TCSVT.2004.837020

I. INTRODUCTION

IMAGE and video encoding has been totally transformed with the advent of new coding and representation techniques [28]. These next-generation coding techniques have made possible encoding and representation of audio–visual (AV) scenes with semantically or structurally meaningful objects. Such a new paradigm of object-based representation of AV scenes and presentations will change the way AV applications are created. MPEG-4 is a standardization activity, under the auspices of the International Standards Organization, specifying tools to enable object-based AV presentations [25]. These include tools to encode individual objects, compose presentations with objects, store these object-based presentations, and access these presentations in a distributed manner over networks. The main distinguishing feature of object-based AV presentations is the scene composition at the user terminal. The objects that are part of a scene are composed and displayed at the user end as opposed to encoding the composed scenes as is done in the case of MPEG-2 systems. Such object-based representation and presentation has several benefits including compression efficiency and the capability to interact with individual objects.

The MPEG-4 Systems specification [1], [12], [13], [26] defines an architecture and tools to create AV scenes from individual objects. The scene description and synchronization tools are at the core of the systems specification. The MPEG-4 scene description, also referred to as binary format for scenes (BIFS), is based on the virtual reality modeling language (VRML) and specifies the spatiotemporal composition of objects in a scene [26]. MPEG-4 also specifies the delivery multimedia integration framework (DMIF), a general application and transport delivery framework [14]. In order to keep the user unaware of underlying transport details, MPEG-4 defined an interface between user level applications and the underlying transport protocol called the DMIF application interface (DAI). The DAI provides the required functionality for realizing multimedia applications with quality-of-service (QoS) support. This architecture allows creation of complex presentations with wide-ranging applications. As the complexity of the content increases, so does the complexity of the servers and user-terminals involved. The servers now have to manage multiple streams (objects) to deliver a single presentation.

The flexibility of MPEG-4 enables complex interactive presentations but makes the content creation process nontrivial. Unlike MPEG-2, the content creation process involves much more than multiplexing the media streams. Determining the schedulability, i.e., whether objects in a presentation can be delivered in realtime, of a presentation is also important during the content creation process to determine if the presentation being designed

can be scheduled for specific channel rates and client buffer capacity. It may not be possible to schedule a presentation with a given set of resources. In order to create a schedulable presentation, some constraints may be relaxed. In the case of scheduling objects, relaxing a constraint may involve increasing the buffer capacity, increasing the channel capacity, not scheduling some object instances, or removing some objects from a presentation.

In this paper, we discuss the problem of scheduling AV objects and present algorithms for optimal scheduling of AV objects. We present new algorithms, based on job sequencing on a single machine proposed by Carlier [3], for scheduling objects in a presentation. The original contributions of this paper include: algorithm to determine the schedulability of AV presentations, algorithm to compute startup delay optimal schedules, algorithm to compute schedules that minimize the required channel capacity, and algorithms to compute incremental schedules. A more detailed discussion including the issues in scheduling interactive presentation can be found in [17]. This paper is organized as follows. The general problem of scheduling AV objects and related earlier work is presented in Section II. The characteristics of startup delay and terminal buffer are discussed in Section III. In Section IV, we present several algorithms to schedule AV presentations. Discussions and results are presented in Section V. We conclude the paper in Section VI.

II. Scheduling AV Objects

Scheduling and multiplexing of AV objects in a presentation is a complex problem. Scheduling of AV objects has been the subject of study in [1], [23], and [27]. In [23], Little and Ghafoor present synchronization of multiobject presentations using Petri-net models to describe timing relations in multimedia presentations. They present network-level and application-level synchronization protocols for multiobject presentations. The problem considered is delivering objects from multiple sources to a single destination. The problem we are considering is the network-independent scheduling of interactive AV objects on the server side. We assume the use of underlying network services for establishing connections for data transport. We also show that scheduling objects jointly results in bandwidth savings. In the Firefly system [2], the issue addressed was scheduling a set of local objects to ensure synchronization by adjusting the duration of the media objects involved. The authors address the issue of meeting specified timing constraints on the presentation by adjusting the playrate of objects (speeding up or slowing down playback) but do not consider network delivery issues. We address the problem of synchronization by assuming a constant delay network, a terminal buffer, and time stamps associated with the objects in the presentations. In [27], Song *et al.* describe the JINSEL system that uses bandwidth profiles (BPs) to reserve bandwidth for media objects on a delivery path. The JINSEL system computes the bandwidth required on the network segments on the delivery path using the amount of buffer available on the switch/component. This approach to delivering object-based presentations is not practical because of the processing required at the intermediary nodes. The approach reserves a constant bit rate (CBR) channel assuming only CBR sources while it has been shown that multiple CBR sources, when combined, result in a variable bit rate (VBR) source [9]. We consider a generic MPEG-4 presentation and present algorithms for determining a delivery schedule.

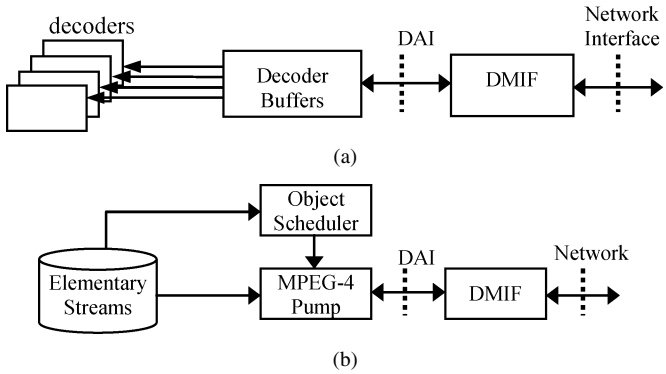

Fig. 1(a). Terminal model. (b) Server model.

In the following, the problem is explained in the context of MPEG-4 Systems. MPEG-4 Systems specifies an architecture to describe scenes and communicate AV data that corresponds to the objects in a scene [1]. A scene consists of one or more AV objects with each of these objects associated with an elementary stream that carries the corresponding data. All the elementary streams are typically multiplexed in a transport multiplex. A server that is transmitting objects (elementary streams) should make sure that an *access unit* (AU) (an AU is the smallest data entity to which timing information can be attributed, e.g., frames in an elementary stream) arrives at the terminal before its decoding time. The constraints on the server transmission are the channel capacity and buffer capacity at the receiving terminal. This problem has similarities with VBR scheduling [24] where the goal is to maximize the number of streams supported by a server. One of the main differences is that in VBR scheduling discussed in [24] and references therein, the assumption is that the video data being handled is periodic (e.g., 30 f/s). In a general architecture such as MPEG-4, such an assumption is not valid as the data may consist of only still images and associated audio. Furthermore, the multiple streams in MPEG-4 presentations are synchronized at the same end-user terminal using a single clock or possibly multiple clocks whereas there are no interdependencies when scheduling multiple VBR video streams. This puts tighter restrictions on the scheduling of an AV presentation. In such cases, the decoding times of individual AUs have to be considered for efficient scheduling. Furthermore, the delay tolerances and relative priorities of objects in an AV presentation can be used to schedule objects for delivery. To make a presentation schedulable, objects of lower priority could be dropped. Even different instances of an object may be assigned different priorities (e.g, higher priority for I and P frames and a lower priority for B frames in an MPEG video stream). These characteristics of the AV services can be used to efficiently schedule a presentation with minimal resource consumption.

A. System Model and Assumptions

We discuss the scheduling of AV objects in the context of a system consisting of client (end-user) server, and network components as shown in Fig. 1(a). Fig. 1(b) shows the server model. The server delivers objects in a presentation as scheduled by

the scheduler. The scheduler uses the decoding timestamps to schedule the delivery of AUs. A decoder is assumed at the far end that decodes the objects for real-time playback. On the client side, data is retrieved from the network and provided to the decoders at decoding time of that AU. Any data that arrives before its decoding time is buffered at the terminal. The terminal buffer model is not considered to keep the schedule independent of terminal designs. However we need the minimum buffer size for a class of terminals to compute object schedules. The data delivered from the server is transported on the channel established between the client and the server. The following assumptions are made about the content, decoders, network, and the server.

Content:

- An AV presentation is composed of one or more objects (AV Objects).
- An AU is the smallest piece of data that can be associated with a decoding time.
- An AV object contains one or more AUs.
- Objects and their AUs may be assigned relative priorities.

Terminal/Decoders:

- The decoders have given, limited memory for receiving and decoder buffers.
- The object data is removed instantaneously from the buffer at the decoding time given by the object's decoding timestamp.
- An object/instance that is received before the decoding time is buffered in the decoder-input buffers until its decoding time.
- More than one object instance may be present in the decoder-input buffers.

Channel/Network:

- End-to-end delays from the server to the player (including the transmission delay) are assumed to be constant.
- The capacity required for the signaling channel is assumed to be negligibly small.
- The transport layer is work conserving, and delivers the packets to the network instantaneously.

Server:

- Audio-visual objects are available at the server in the form of time-stamped AUs.
- All the AUs of an object are delivered in their decoding order.
- A server presents an AU to the transport layer at the send time determined by the scheduler.

B. Problem Formulation

Given a set of N objects that comprise an AV presentation, with each object containing n_i AUs each with a decoding time T_j^d, of kth AU of object j, a transmission channel of capacity C, terminal buffer of size B, allowed startup delay of $T_s^{\max}$, and duration (channel occupancy) of each AU on the channel, $d_j(k)$, is there a schedule σ that satisfies the following constraints:

$$T_j^s(k) \le T_j^d(k) - d_j(k) \tag{1}$$

$$T_j^s(k+1) \ge T_j^s(k) + d_j(k) \tag{2}$$

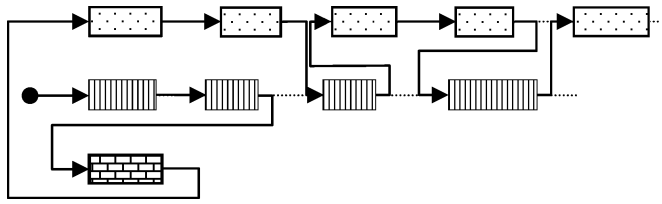

Fig. 2. Sequencing of AUs in a 3-object presentation.

$$\begin{aligned} &\text{if } A_i(k) \in A - A_j(l), \text{ then} \\ &\quad \text{either } T_i^s(k) + d_i(k) \le T_j^s(l) \\ &\quad\quad \text{or } T_i^s(k) \ge T_j^s(l) + d_j(l) \end{aligned} \tag{3}$$

$$B(t) = \sum_{j,k} C * d_j(k), \quad j \ni T_j^s(k) + d_j(k) \le t; T_j^d(k) \ge t \tag{4}$$

$$T_s \le T_s^{\max}? \tag{5}$$

Constraints (1)–(5) represent the conditions for transmission and playback of object based AV presentations. Constraint (1) enforces the on-time delivery of AUs. Ignoring the constant end-to-end delays, (1) gives the latest time an AU can be transmitted. Constraint (2) imposes intraobject synchronization by enforcing precedence constraints among the AUs. Access units are never transmitted out of order; they are transmitted in their decoding order. Since a single channel is used for transmission, channel occupancy of any two AUs cannot overlap. Constraint (3) ensures that data is delivered on a single channel between a server and a client, i.e., if we consider any two AUs in the presentation, only one of them can be transmitted at any time. Constraint (4) gives the buffer occupancy at the end-user terminal at time t. Constraint (5) gives a bound on the startup delay for the given presentation. If the problem cannot be solved, i.e., a schedule that satisfies the given resource constraints cannot be found, some of the constraints could be relaxed in order to find a schedule. The constraints can be relaxed by reducing the number of objects, increasing the startup delay, or increasing the channel capacity.

Example: Consider the scheduling of a presentation with three objects as shown in Fig. 2. Object 1 has five AUs, object 2 has three AUs, and object 3 has one AU. The AUs are shown with increasing decoding time stamps from left to right. We have to find a schedule, if one exists, that sequences the AUs starting with the first AU of one of the three objects and satisfying the constraints. Fig. 2 shows one such sequence. The general problem of determining the existence of such a sequence is NP-complete. We prove that in *Theorem 1*.

Scheduling is a complex problem and has been widely studied [4]–[6], [8]. Many of the scheduling problems are NP-complete and a number of approximation algorithms are developed trading off optimality for tractability [10], [30]. The scheduling problem closest to the AV object scheduling is job-shop scheduling on a single machine. There has been earlier work on scheduling on single machines. Complexity of machine scheduling problems is studied in [21]. Carlier proved the NP-hardness of one-machine sequencing problem in [3] and some approximation algorithms are discussed in [10]. Another problem with similarities to AV scheduling is job

scheduling with temporal distant constraints. NP-completeness results and polynomial time algorithms for a restricted instance of the problem are given in [11]. In spite of the similarities to the current problem, the approximation results for single machine scheduling problems cannot be applied to AV object scheduling because of an entirely different problem domain and additional constraints on AV object scheduling. Approximation algorithms are based on heuristics and domain knowledge is essential to develop good designs. Even though the results of single-machine scheduling are not directly applicable to AV presentations, some of the results can be used in scheduling individual objects on a channel. The results of single-machine scheduling problems as formulated by Lawler in [19], [20] may be used to determine the schedulablity of individual objects.

C. Complexity of Audio Visual Object Scheduling

Theorem 1: Scheduling of an AU in AV presentations (SAV) is NP-complete in the strong sense.

Proof: We prove this by transforming the problem of sequencing within intervals (SWI), proven to be NP-complete in the strong sense [8].

We restate SWI below:

Instance: A finite set T of tasks and, for each $t \in T$, an integer release time $r(t) \geq 0$, a deadline $d(t) \in Z^+$, and a length $l(t) \in Z^+$.

Question: Does there exist a feasible schedule for T, i.e., a function $\sigma : T \rightarrow Z^+$, such that for each $t \in T$, $\sigma(t) \geq r(t)$, $\sigma(t) + l(t) \leq d(t)$, and if $t' \in T - \{t\}$, then either $\sigma(t') + l(t') \leq \sigma(t)$ or $\sigma(t') \geq \sigma(t) + l(t)$?

The basic units of the SWI problem are the tasks $t \in T$. The local replacement for each $t \in T$ is a single AU $A_j(k)$ with $r(A_j(k)) \geq T_j^s(k-1)$, $d(t) = T_j^d(k)$, $l(t) = d_j(k)$. We disregard the buffer and startup delay constraints. It is easy to see that this instance can be created from SWI in polynomial time. Since SWI can be transformed to SAV, SAV is at least as hard as SWI.

Since SAV is NP-complete in the strong sense, it cannot be solved by a pseudopolynomial-time algorithm. We present several polynomial-time algorithms based on heuristics and constraint relaxation and evaluate their performance with respect to speed and efficiency.

III. Bounds on Startup Delay and Terminal Buffer

An MPEG-4 terminal has a finite buffer to store the received data until they are decoded. The amount of buffer capacity required depends on the type and number of elementary streams being buffered. Since there are usually no limits on the number of objects in AV presentations, it is not practical to have sufficient buffer for *all* presentations. A terminal should be designed to support a *class* of presentations. The amount of buffer available also determines the upper bound on the startup delay for a session. The higher the startup delay, the higher the buffer capacity required (with channel capacity remaining the same). When scheduling presentations, a scheduler should assume the minimum allowable buffer for terminals in order to support all terminal types. Even though the knowledge of the buffer occupancy at a terminal may help improve the schedule, it makes the schedule dependent on the buffer model used by the terminals. Since the buffer model and management in a terminal depends on terminal design, we designed the scheduler to be buffer model independent.

Startup delay can be defined as the time a user has to wait from the time a request is made until the time the presentation starts. A startup delay of T_s is *not* equal to buffering T_s seconds of the presentation. The amount of startup delay varies from presentation to presentation and even for the same presentation, it may vary with varying resources (e.g, bandwidth and buffer). Startup delay can be viewed as preloading the beginning of a presentation so that the presentation is played back continuously once the playback starts. The amount of startup delay required for the smooth playback of a presentation depends on the channel capacity. For any channel, the minimum startup delay is the time needed to transmit (buffer) AUs that are presented at time 0 (AUs with timestamp 0).

Consider a presentation composed of several images displayed on the first screen, followed by an audio track. The images to be displayed on the first screen should reach the terminal before the presentation starts, resulting in a startup delay. If the channel bandwidth reserved for the presentation is allocated based on the low bitrate audio stream that follows the images, the startup delay will be higher. On the other hand, if the higher bandwidth is reserved to minimize the startup delay, the capacity may be wasted during the remainder of the presentation when low bitrate audio is delivered. The tradeoff depends on resource availability and startup-delay tolerance of the application.

Given a startup delay T_s, the buffer required is equal to the size of the objects that can be loaded (transmitted to the client) in time T_s. The minimum buffer required for this delay is $T_s^* C$. The minimum startup delay for any presentation is equal to the time required to transmit (load) the objects/instances to be displayed at time 0. We refer to this time as T_s^0. T_s^0 is the optimal startup delay for startup delay-optimal schedules and is the lower bound on startup delay for bandwidth-optimal schedules.

A. Residual Data Volume

We introduce the notion of *data volume* to quickly compute the minimum startup delays needed for a presentation and determine the nonschedulability. Data volume (V_d) is the amount of data (in bits) transferred during a session. The amount of data that can be carried by a channel during a session is the data pipe volume ($V_p = C * D_p$). The amount of data volume exceeding the data pipe volume is the residual data volume ($V_{\text{res}} = V_d - V_p$). A positive V_{res} gives the lower bound on the amount of data to be loaded during startup and hence determines the lower bound on the startup delay for a presentation. A negative value of V_{res} indicates unused channel capacity during the session. We prove the lower bound on channel capacity required in Theorem 2.

Theorem 2: For a presentation of duration D_p, the lower bound on channel capacity required for a startup delay-optimal schedule is

$$C \geq C_{\min}, \quad \text{where } C_{\min} = \frac{V_d}{D_p}$$

and the bound is tight.

Proof:

$$V_d = \sum_{j,k} s_j(k)$$
$$D_p = \max_j \left\{T_j^d(n_j)\right\} - \min_j \left\{T_j^d(n_j)\right\}.$$

For a presentation of length D_p, the data pipe volume at the given pipe capacity is

$$V_p = C * D_p.$$

Assuming that the buffers are filled up at a rate C, the startup delay due to V_{res} is

$$T_s^{\text{res}} = \frac{(V_d - V_p)}{C}.$$

To minimize the startup delay

$$T_s^{\text{res}} = \frac{(V_d - V_p)}{C} = 0 \Rightarrow V_p = V_d.$$

Since $V_p = C * D_p$, substituting V_p we get the lower bound on the channel capacity

$$C_{\min} = \frac{V_d}{D_p}.$$

From constraint (1)

$$T_j^s(1) \le T_j^d(1) - d_j(1) \Rightarrow T_j^s(1) \le T_j^d(1) - \frac{s_j(1)}{C}.$$

From constraint (2)

$$T_j^s(1) \ge T_j^s(0) + \frac{s_j(0)}{C}.$$

Assuming that the presentation starts at time 0

$$T_j^s(0) = 0, \quad \Rightarrow T_j^s(1) \ge \frac{s_j(0)}{C}.$$

From constraints (1) and (2)

$$T_j^s(2) \ge T_j^s(1) + \frac{s_j(1)}{C} \Rightarrow T_j^s(2) \ge \frac{s_j(0)}{C} + \frac{s_j(1)}{C}$$
$$T_j^s(2) \le T_j^d(2) - \frac{s_j(2)}{C} \Rightarrow T_j^d(2) \ge T_j^s(2) + \frac{s_j(2)}{C}$$
$$\Rightarrow T_j^d(2) \ge \frac{s_j(0)}{C} + \frac{s_j(1)}{C} + \frac{s_j(2)}{C}.$$

Similarly

$$T_j^d(n_j) \ge \frac{s_j(0)}{C} + \frac{s_j(1)}{C} + \ldots + \frac{s_j(n_j)}{C}.$$

Since the AUs are transmitted on a single channel

$$D_p = \max\left\{T_j^d(n_j)\right\} \ge \sum_j \frac{s_j(0)}{C} + \frac{s_j(1)}{C} + \ldots + \frac{s_j(n_j)}{C}$$
$$D_p \ge \frac{1}{C}\sum_{j,k} s_j(k) \Rightarrow D_p \ge \frac{1}{C} * C_{\min} * D_p \Rightarrow C \ge C_{\min}.$$

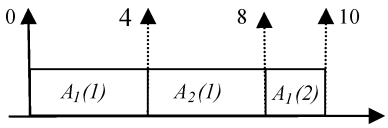

Fig. 3. Example to prove the tightness of the bound.

We can show that the bound is tight by considering the example as shown in Fig. 3.

Object 1 has two AUs and object 2 has one AU. The decoding times and sizes of AUs in bytes are

$$T_1^d(1) = 4, \quad T_1^d(2) = 10, \quad T_2^d(1) = 8,$$
$$s_1(1) = 10, \quad s_1(2) = 5, \quad s_2(1) = 10.$$

With these values, the send times and channel capacity are $T_1^s(1) = 0$, $T_1^s(2) = 8$, $T_2^s(1) = 4$ and $C = C_{\min} = 2.5$ bytes/s.

The actual channel capacity required to minimize startup delay may be higher depending on the timing constraints of AUs. Note that irrespective of the channel capacity, the minimum startup delay remains nonzero and is equal to T_s^0.

Thus, for any given presentation with resource constraints, the minimum startup delay is $T_s^{\min} = \max\{T_s^{\text{res}}, T_s^0\}$, the minimum buffer capacity required is $B_{\min} = T_s^{\min} * C$, and the presentation is schedulable only if the available buffer is at least equal to $B_{\min}$.

IV. Scheduling Algorithms

In this section we describe a family of scheduling algorithms for AV object scheduling. Given an AV presentation, the scheduling algorithms compute a delivery schedule according to the selected criteria. We assume that the terminal buffer is fixed and compute startup delay-optimal or bandwidth-minimizing schedules. The algorithms can also be repurposed to compute the minimum terminal buffer required for a given channel capacity.

A. FullSched Algorithm

This algorithm is based on the last-to-first idea mentioned in [20] for scheduling jobs on a single machine. The main principle behind this algorithm is scheduling an AU with latest deadline first and scheduling it as close to the deadline as possible. The algorithm computes the schedule starting with an AU with the latest decoding time in the presentation. This algorithm computes the schedule, the required startup delay, and any channel idle times. The channel idle times computed are used in the gap-scheduling algorithm described in Section IV-B.

Let S be the set of current AUs to be scheduled. Initialize S to contain the last AU of each of the objects to be scheduled. Let x_j be the index of the next AU of object j to be scheduled

$$x_j = n_j, \quad 1 \le j \le N.$$

Initialize $S = \{A_j(x_j)\}$, $1 \le j \le N$. $S(j)$ is the AU of object j to be scheduled next. S contains at most one AU for every

object j. G is the set of channel idle times. Idle time is given by a tuple $\langle t, d\rangle$, i.e, the channel is idle for duration d starting at time t. Initialize $G = \{\phi\}$. Set current time $i = \infty$. Sort AU of objects in the decreasing order of their decoding times.

```
BEGIN
  while (S ≠ φ) {
    i = min{i, max{T_j^d(k)}}, T_j^d(k) ∋ A_j(k) ∈ S
    T_j^s(x_j) = i − d_j(x_j); //send time for
A_j(x_j)
    i− = d_j(x_j); // Update i
    x_j--;
    // Update S by removing S(j) from S
    S− = S(j);
    // add A_j(x_j) to S
    if (x_j ≠ 0)
      S+ = AU(j, x_j)
    if (i > max{T_j^d(k)}), T_j^d(k) ∋ A_j(k) ∈ S
      // there is a gap on the channel
      G+ = ({max{T_j^d(k)}, i − {max{T_j^d(k)})
  }
  if T_first^s < 0
  then T_s = |T_first^s|
  T_j^d(k)+ = T_s,        ∀j, k
END
```

The process begins with S initialized with the last AU of each of the objects in the presentation and G initially empty. In each of the iterations, the AU with the latest decoding time is scheduled as close to the decoding time as possible. Ties are broken arbitrarily. Once the AU of an object is scheduled, the next AU of that object is added to S as long as there are AUs to be scheduled. The current time is given by i. A value of i greater than the largest decoding time of AUs in $S(\max\{T_j^d(k)\})$ indicates idle time on the channel (gaps or slots). The channel is idle because nothing can be scheduled between $\max\{T_j^d(k)\}$ and i. This is illustrated in the example below.

Example: Consider two objects: object O_1 with two AUs and object O_2 with one AU with duration on channel, d, and decoding time stamp, T given as a set of tuples $\langle d, T\rangle$. $O_1 = \{\langle 7,7\rangle, \langle 10,21\rangle\}$ and $O_2 = \{\langle 5,6\rangle\}$. After $A_1(2)$ is scheduled, at time $T_1^s(2) = 11$, nothing can be scheduled between $T_1^d(1) = 7$ and current time $i = 11$ resulting in a gap on the channel. A negative value of the send time indicates that the AU has to be transmitted before the presentation starts giving rise to a startup delay. The FullSched example is depicted in Fig. 4.

When S becomes empty, i.e., all AUs are scheduled, a negative value of i indicates the required startup delay for the presentation and G gives the set of gaps on the channel. Since the decoding times $T_j(k)$ are all nonnegative, once i becomes negative, there are no gaps on the channel indicating that the AUs are tightly packed. A gap is not an indication of the suboptimality of the schedule. However, it may indicate the suboptimality of the bandwidth-optimized schedule, i.e, it may be possible to schedule the presentation at a lower bandwidth. When $N = 1$, this algorithm can be used to determine the schedulability of individual objects and determine the unschedulability

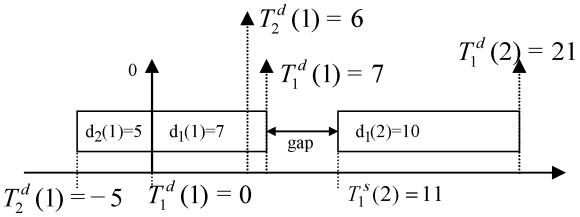

Fig. 4. FullSched example.

of a presentation. This is especially useful during the content creation process where objects are added to create presentations. When an object is added during an editing operation, it is faster to determine the unschedulability of a presentation by computing the independent schedules of objects and adding the startup delays of the independent schedules. However, a full schedule should still be computed after the editing operations to determine the schedulability of the presentation under given resource constraints. This algorithm is not efficient in computing the schedulability during the content creation process, as the full schedule needs to be recomputed every time an object is added. We next present a gap-scheduling algorithm that computes incremental schedules to determine the schedulability of a presentation and is well suited for the content creation process. We also prove that FullSched and gap-scheduling algorithm compute startup delay optimal schedules.

Theorem 3: Algorithm FullSched produces a startup delay-optimal schedule.

Proof: The algorithm selects an AU with the latest decoding time and schedules it as close to the deadline as possible, i.e., the algorithm schedules the AUs in nonincreasing order of their decoding times. On a conceptual timeline, with time increasing from left to right, we are stacking the AUs as much to the right as possible. Gaps occur only when there is nothing to be scheduled in that gap. Any (or part of) AUs that appear to the left of the origin (time = 0) give the startup delay. Since the algorithm always moves the AUs to the right whenever possible, the startup delay is minimized. A smaller startup delay is not possible because, it would mean moving the AUs to the right implying that there is a usable gap on the channel. This cannot be the case because the algorithm would have scheduled an AU in that gap!

B. GapSched Algorithm

The gap-scheduling (GapSched) algorithm schedules AUs in the available gaps on a channel. It starts with available gaps on a channel and tries to fit an AU or a partial AU using the SplitAndSchedule procedure. The initial set of gaps may be obtained by using FullSched to schedule a single object. The algorithm looks for the first available gap starting at a time less than the decoding time of the AU to be scheduled. Since G is already sorted in the decreasing order of gap times, the look up can be done very efficiently. If the gap duration is not long enough to fit an AU, the AU is split, with one part scheduled in the current gap and the other added to S to be scheduled next. The AUs in the presentation are iteratively scheduled until S becomes empty.

S contains all the AU of the object j $S = \{A_k(k)\}$, $1 \le k \le n_j$, and $j \in \{N\}$. Sort AUs in S in the decreasing order

of their decoding times. G = set of available slots $\neq \{\phi\}$. $G(l)$ is the lth tuple in G with start time $G(l).t$ and duration $G(l).d$. $k = n_j$.

```
BEGIN
  while (S ≠ φ) {
    find a slot l, G(l), such that
T_j^d(k) > G(l).t
    if (G(l).d ≥ d_j(k)) {
          T_j^s(k) = G(l).t − d_j(k)  //send
        time for A_j(k)
          k--;
          // update the gap
          if (G(l).d − d_j(k) > 0)
      G(l).d = G(l).d − d_j(k)
     else
      G− = {G(l)};
          // remove AU from the set
        S− = A_j(k);
        }
        else{
          PROCEDURE   SplitAndSchedule
      (A_j(k), G(l));
        }
  }
END
```

Split the AU into two parts, one part that is scheduled in $G(l)$ and the other that is placed back in S.

```
PROCEDURE SplitAndSchedule (A_j(k), G(l))
{
  Create a subAU of length G(l).d with
the last G(l).d * C bytes of the AU.
  t'_j(k) = G(l).t;
  d'_j(k) = d_j(k) − G(l).d;
  G− = {G(l)};
}
```

C. IncSched Algorithm

The incremental scheduling (IncSched) algorithm computes the schedule for a presentation by considering one object at a time. This is a typical content creation scenario where objects are composed to create a presentation. Instead of recomputing the full schedule with FullSched algorithm each time an object is added, this algorithm computes the schedules incrementally by scheduling the AU in the available gaps. Note that not all the gaps are schedulable. A gap is unschedulable if there are no AUs with decoding times greater than the gap time. An unschedulable gap indicates unused bandwidth, which is either due to the structure of the presentation or due to a suboptimal schedule. The IncSched algorithm uses FullSched and GapSched algorithms to schedule a presentation. This algorithm appears to be more efficient than FullSched as it schedules parts of AUs and fills all the gaps. However, this is only as efficient as FullSched as far as startup delay is concerned. Splitting the AUs in order to pack the gaps is not going to decrease the startup delay, as the available channel capacity is the same. The startup delay, like in other cases, is given by the send-time of the first AU transmitted.

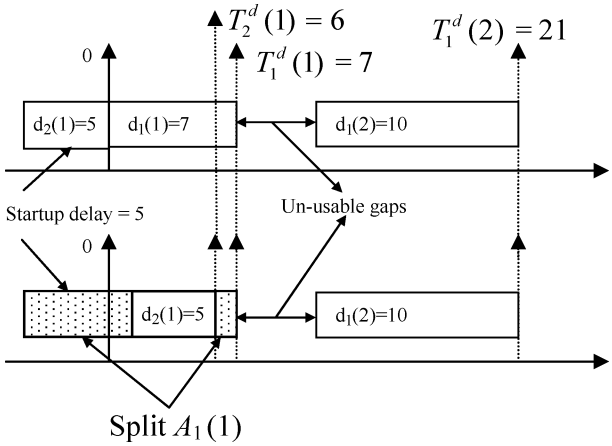

Fig. 5. Schedules computed using FullSched and IncSched.

OBJ is the set of objects in the presentation.

```
BEGIN
   Apply FullSched and compute
schedule for object 1.
   //LRG is a sufficiently large
number to accommodate startup delay.
  G+ = {⟨−LRG, i − LRG⟩}
  for j ∈ OBJ − {1}, apply gap scheduling
GS to j.
  for j   ∈   OBJ, find t_first, the send
time of the first AU to be transmitted
(smallest t_j(k))
  if T^s_first < 0
  then T_s = |T^s_first|
  T_j^d(k)+ = T_s,      ∀j, k
END
```

Theorem 4: The IncSched algorithm is startup delay-optimal.

Proof: The first object is scheduled using FullSched producing a startup delay-optimal schedule for that object. GapSched, when applied iteratively to the remaining objects, packs the AUs tightly, i.e., an AU is scheduled if the gap time is less than the decoding time for that AU. The resulting startup delay is optimal because the algorithm would reduce the startup delay by moving the AU to the right on the timeline if any schedulable gap is available.

Example: Consider two objects: O_1 with two AUs, and O_2 with one AU with duration on channel d and decoding time stamp T given as a set of tuples $\langle d, T\rangle$. $O_1 = \{\langle 7, 7\rangle, \langle 10, 21\rangle\}$ and $O_2 = \{\langle 5, 6\rangle\}$. The top part of Fig. 5 shows the schedule computed using FullSched, and the bottom half shows the schedule computed with IncSched, with object 2 scheduled first using FullSched. The figure also shows unschedulable gaps in both the schedules.

There may be cases where splitting the AUs is necessary, for example, the underlying transport layer may not be able to

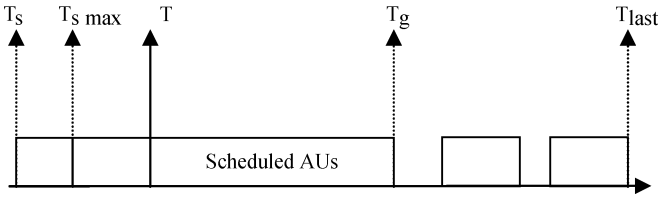

Fig. 6. First gap-time and startup delay of a presentation.

handle large AUs. This result shows that AUs can be split while maintaining the optimality of the schedule. Although the IncSched algorithm produces an optimal schedule and is useful in determining the schedulability of a presentation in applications such as content creation, the schedule generated by FullSched may be more efficient when the overhead due to splitting and packetizing is significant.

D. MinC Algorithm

In scheduling AV objects, we have so far answered two questions: 1) Is the given presentation schedulable under the given resource constraints, and 2) what is the minimum startup delay required for this presentation? If the answer to question 1 is negative (or if bandwidth consumption needs to be minimized), the question we need to address is what are the minimum amounts of resources required to schedule the presentation? Since we cannot make assumptions about decoder buffers in order to keep the schedules player-independent, the only resource that can be acquired is the bandwidth (C). We next present the MinC algorithm that computes the minimum bandwidth (CBR) required to schedule a presentation.

This algorithm is based on the premise that there is a gap on the channel only when everything else *after* the gap-time has been scheduled. Otherwise, an unscheduled AU would have taken up the gap. The presentation is not schedulable because there is not enough channel capacity until the first gap time, T_g (smallest gap time). Consider the case in Fig. 6. T_g is the first gap-time $T_s^{\max}$ is the maximum allowable startup delay with the current channel capacity, and T_s is the current startup delay. The channel capacity should be increased to accommodate $T_s - T_s^{\max}$ in the duration $T_g - T_s^{\max}$. The new value of C then is $C_{\text{new}} = C * ((T_g - T_s)/(T_g - T_s^{\max}))$. The additional bandwidth necessary is therefore equal to $C * ((T_s^{\max} - T_s)/(T_g - T_s^{\max}))$. The algorithm also outputs the BP for the presentation in the form of three tuples ⟨capacity, start, end⟩. Note that the increased channel capacity is not going to affect the schedule from T_g to T_{last}. A finer BP can be obtained by initializing C with $C_{\min}$ and increasing C by a small value in each iteration.

```
BEGIN
  G_c = |G|=gap count gap count, number
of gaps on the channel.
  BP = {φ}
  T_last is the decoding time of the
first AU scheduled (=duration of the
presentation)
  C = C_min, computed using the results
of theorem 2.
    SCHEDULE: compute schedule using
FullSched
    If schedulable goto END
    if (G_c == 0)
      T_g = T_last
    else {
      T_g−old = T_g
      Find the smallest gap time, T_g
      BP+ = {⟨C, T_g, T_g−old⟩}
    }
    C = C + C * ((T_s^max − T_s)/(T_g − T_s^max))
    goto: SCHEDULE
END
```

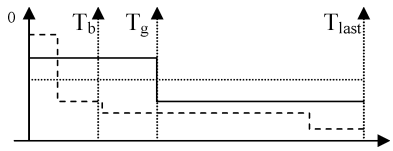

Fig. 7. Typical BP generated by MinC.

The channel capacity output by the algorithm is in the form of a set of three tuples forming a BP. The minimum CBR channel required is given by the maximum value of C in the BP. This profile may also be used to reserve session bandwidth efficiently. Since the schedule is computed from last to first (right-to-left on the timeline), the BP will always be a step function with possible steps (decreasing) from left to right. Fig. 7 shows some sample profiles. This algorithm does not give the best profile to reserve variable session bandwidth since the algorithm does not reduce the bandwidth when it is unused. Consider the example shown in the Fig. 7. At T_g, a capacity increase is necessary. Suppose the increase in C at T_g is sufficient to schedule the presentation. It is possible that the presentation from 0 to T_b could have been scheduled with a much smaller capacity.

E. BestSched Algorithm

When a presentation cannot be scheduled with the given resources, and additional resources cannot be acquired, the only way to schedule the presentation is to drop some AUs. AUs cannot be dropped arbitrarily as they have different effects on the presentation. Content creators should assign priorities to objects and possibly AUs of objects to help a scheduler in determining the AUs to be dropped. The following algorithm schedules a presentation by dropping lower priority objects.

```
BEGIN
SCHEDULE: Compute schedule using
FullSched.
  if (B <= T_s * C) {
    Remove A_j(k) of lower priority ob-
jects such that,
    R = {A_j(k)} ∋ C * ∑ d_j(k) ≥ T_s * C − B
    A− = {R}
    goto: SCHEDULE
  }
END
```

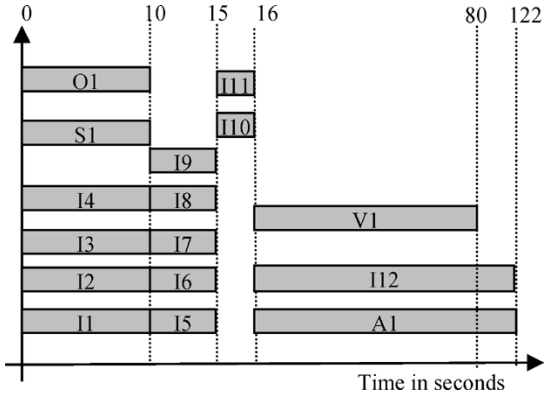

Fig. 8. Structure of the presentation in the example.

V. Results and Discussion

Determining the schedulability of a presentation is of $O(n)$ complexity, where n is the number of AUs in the presentation. Both FullSched and GapSched fall under this category. These algorithms are used to determine the schedulability, compute an optimal startup delay for the given channel capacity, and for computing incremental schedules. The MinC algorithm, used to compute the minimum channel capacity required to schedule the presentation, calls FullSched iteratively with channel capacity incremented in each iteration. The number of iterations depends on the structure of the presentation and the initial value of C. The complexity of this algorithm is $O(Kn) = O(n)$, where K is a constant determined by the structure of the presentation and the initial channel capacity. The proposed algorithms are fast enough to determine the schedulability of the presentations in real-time.

The structure of the presentation has significant impact on the performance of MinC algorithm. To aid the discussion, we consider a relatively complex MPEG-4 presentation with structural overview as shown in Fig. 8. The properties of the objects in the presentation are tabulated in Table I.

In the following discussion we refer to objects by the codes shown in the first column of the table. The presentation is made up of 16 objects including scene description, object description, images, audio, and video. A screenshot of the presentation as seen in an MPEG-4 player is shown in Fig. 9. The presentation is composed of three scenes. Before the scenes are loaded, the scene description and object description streams are received and decoded by the terminal. The first scene consists of four jpeg images (I1–I4) animated to give a breakout effect. The scene is encoded to animate the images for 10 s and then load the second scene. The second scene consists of a background image (I5), four logos with animation effects (I6–I9), and two images (I10 and I11) with descriptive text of the following AV scene. The last scene consists of a background image, an audio stream, and a video stream. The temporal layout of the presentation is shown in Fig. 8. The times indicated are the decoding times of the first AUs of the objects starting at that time. Thus, the first four images (I1–I4), the scene description (S1) and the object descriptor stream (O1) should reach the decoder before anything is displayed on the screen. This amounts to the minimum startup delay for the presentation. The objects I5–I9 should reach the decoder by the time $t = 10$, I10 and I11 by 15, and the first AU of V1 and A1, and the object I12 should reach the terminal by the time $t = 16$. The video ends at $t = 80$ while the audio stream continues until the end of the presentation. The total length of the presentation is 122. This temporal ordering of objects in the presentation results in higher data rates toward the beginning of the presentation (object data to be delivered in the first 16 s: 261 Kb$\sim$ $j = 130$ Kb/s).

TABLE I
Properties of Objects in the Example

Object FileName (ID)	Size (KB)	Start Time	AU Count
Scene.od (O1)	0.5	0	1
Scene.bif (S1)	1 (1025 Bytes)	0	4
Main1.jpg (I1)	25	0	1
Main2.jpg (I2)	22	0	1
Main3.jpg (I3)	19	0	1
Main4.jpg (I4)	20	0	1
main_ui.jpg (I5)	39	10	1
Advent_logo.jpg (I6)	7	10	1
CU_logo.jpg (I7)	9	10	1
Lm_logo.jpg (I8)	6	10	1
Xbind_logo.jpg (I9)	8	10	1
geo_pict.jpg (I10)	23	15	11
dance_pict.jpg (I11)	29	15	1
next_page.jpg (I12)	51	16	1
clip01.h263 (V1)	552	16	974
clip01.g723 (A1)	64	16	3233

Fig. 9. Snapshot of the player with the scheduled presentation.

A. Startup Delay and Capacity Computation

Fig. 10 shows the plot of the minimum channel capacity required for a given startup delay. This is a scenario with variable buffer at the terminal. We assume a work-conserving transport layer that delivers the objects at the minimum required capacity. The amount of buffer available at the terminal should be at least sufficient to store the data during the startup. For a startup delay of T_s, if $C_{\min}$ is the min capacity required, then the buffer at

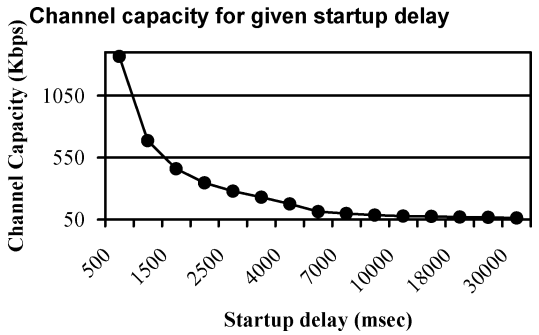

Fig. 10. Computing min capacity using MinC.

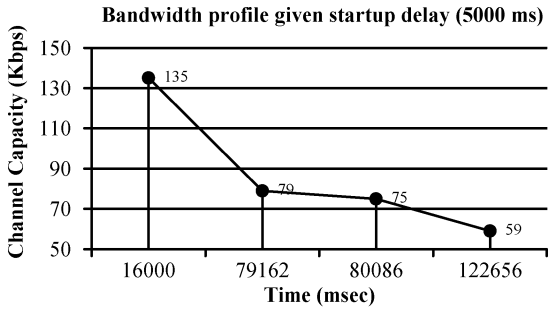

Fig. 11. Computing BP using MinC.

the terminal $B_{\min} > T_s * C_{\min}$. This curve is useful to determine the amount of buffer (delay) required based on the available network capacity, especially with terminals such as PC's with sufficient memory. As mentioned earlier in the discussion of the MinC algorithm, the MinC algorithm also computes the BP for presentations.

Fig. 11 shows the BP computed for a startup delay of 5 s. The minimum capacity in the profile is 59 Kb/s even for the segment (80–120 s) that only has low bit rate audio (6 Kb/s). This is because MinC starts with an initial value of C computed using the residual data volume as described in Section III. This starting point is acceptable for computing a CBR channel required; for a profile to be used in reserving variable network resources, a lower initial value of C should be selected. The final bandwidth jump in the profile gives the minimum channel capacity required for the given delay or buffer.

B. Buffer and Capacity Computation

The available buffer at the terminal determines the amount of startup delay a terminal can support. The available channel capacity imposes a lower limit on the buffer required. Lower channel capacity implies higher startup delays, and hence, larger required buffer. Fig. 12 gives the required buffer at various channel capacities. This can be directly converted to the startup delay at that capacity. Computing the capacity for a given buffer is bit more computationally intensive. Unlike the previous case where we assumed enough capacity to support the required startup delay, the buffer capacity is fixed in this case. This typically the scenario when using dedicated devices such as set-top-boxed with fixed receiving buffers. Since the buffer is fixed, the supported startup delay decreases as channel capacity increases. For MinC algorithm to complete, the reduction in startup delay due to increased channel capacity should

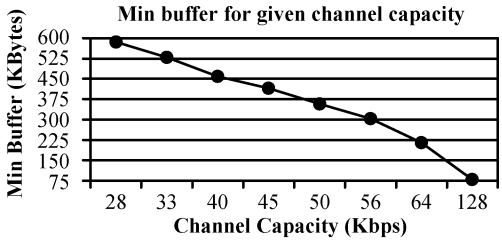

Fig. 12. Minimum required buffer.

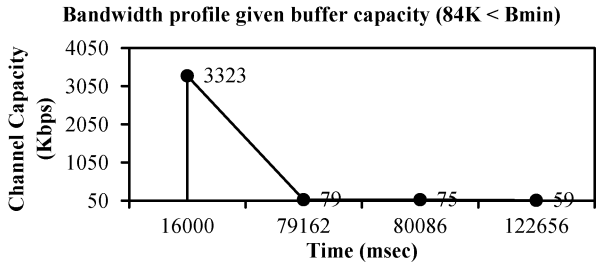

Fig. 13. Partial profile for low terminal buffer.

be greater than the reduction in startup delay supported by the buffer.

Fig. 13 shows a case with terminal buffer $< B_{\min}$. For the given presentation $B_{\min} = 85$ Kb, the size of objects to be decoded at time 0. As discussed in Section III, for a presentation to be schedulable, the available buffer should be greater than $B_{\min}$, the lower bound on the required buffer capacity for the presentation. Since the terminal buffer is less than the required buffer, at any given capacity C, the supported startup delay $(B_{\text{term}}/C) < (B_{\min}/C)$, the required startup-delay. The presentation is hence unschedulable with terminal buffer capacity of 84 Kb. This is depicted in Fig. 13, which shows the presentation is unschedulable even at 3323 Kb/s. The plot shows that no matter how much the channel capacity is increased, the presentation cannot be scheduled because of limited terminal buffer. To avoid infinite loops in MinC, the scheduler should first examine the available and required buffer capacities.

VI. Conclusion

We presented the problem of scheduling object-based AV presentations under resource constraints. The problem is NP-complete in the strong sense. We explored similarities with the problem of sequencing jobs on a single machine and used the idea of last-to-first scheduling to develop heuristic algorithms to determine schedulability and compute startup delay-optimal schedules. The proposed algorithms are applicable in optimizing the delivery of any generic object-based AV presentations and also mixed media presentations described with formats such as SMIL [30]. The algorithms can be applied in scheduling object-based MPEG-4 presentations.

We introduced the notion of residual data volume to compute lower bounds on buffer, channel capacity, and startup delay. Determining the schedulability of presentations online is important for applications like content creation where an additional object may make the presentation unschedulable. We presented an algorithm that computes incremental schedules and produces a startup delay optimal schedule. The incremental scheduling is

very useful in application such as content authoring where authors add and remove objects to presentations during creation. The IncSched algorithm computes the incremental resources required to schedule the additional object. Starting with a lower bound on the channel capacity for scheduling a presentation, the MinC algorithms minimizes the CBR channel capacity required to schedule the presentation. The proposed algorithms are of low complexity and can be implemented efficiently.

REFERENCES

[1] O. Avaro, A. Eleftheriadis, C. Herpel, G. Rajan, and L. Ward, "MPEG-4 systems: Overview," *Signal Process. Image Commun.*, vol. 15, no. 4–5, pp. 281–298, Jan. 2000.

[2] M. C. Buchanan and P. T. Zellweger, "Scheduling multimedia documents using temporal constraints," in *Proc. NOSDAV*, 1992, pp. 223–235.

[3] J. Carlier, "The one machine sequencing problem," *Eur. J. Oper. Res.*, vol. 11, pp. 42–47, 1982.

[4] T. L. Casvant and J. G. Kuhl, "A taxonomy of scheduling in general purpose distributed computing systems," *IEEE Trans. Software Eng.*, vol. 14, no. 2, pp. 141–154, Feb. 1988.

[5] P. Chretienne, E. G. Coffman, Jr., J. K. Lenstra, and Z. Liu, Eds., *Scheduling Theory and Its Applications*. New York: Wiley, 1995.

[6] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*. Reading, MA: Addison-Wesley, 1967.

[7] M. L. Escobar-Molano, "Management of resources to support coordinated display of structured presentations," Ph.D. dissertation, Univ. Southern California, Los Angeles, CA, 1996.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. San Francisco, CA: Freeman, 1979.

[9] L. Grossglauser and S. Keshav, "On CBR service," in *Proc. INFOCOM*, Mar. 1996, pp. 129–137.

[10] L. A. Hall, "Approximation algorithms for scheduling," in *Approximation Algorithms for NP-Hard Problems*, D. S. Hochbaum, Ed. Boston, MA: PWS-Kent, 1997, pp. 1–45.

[11] C.-C. Han, K.-J. Lin, and J. W.-S. Liu, "Scheduling jobs with temporal distance constraints," *SIAM J. Comput.*, vol. 24, no. 5, pp. 1104–1121, Oct. 1995.

[12] C. Herpel and A. Eleftheriadis, "MPEG-4 systems: elementary stream management," *Signal Process. Image Commun.*, vol. 15, no. 4–5, pp. 299–320, Jan. 2000.

[13] *Generic coding of moving pictures and associated audio (MPEG-4 Systems)—ISO/IEC 14386-1*, ISO/IEC/SC29/WG11, Apr. 1999.

[14] *Delivery multimedia integration framework (DMIF)—ISO/IEC 14 496-6*, ISO/IEC/SC29/WG11, Feb. 1999.

[15] N. S. Jayant, "Signal compression: technology targets and research directions," *IEEE J. Select. Areas Commun.*, vol. 10, pp. 796–818, June 1992.

[16] H. Kalva, L.-T. Cheok, and A. Eleftheriadis, "MPEG-4 systems and applications," in *Demonstration, ACM Multimedia*, Orlando, FL, 1999.

[17] H. Kalva, *Delivering MPEG-4 Based Audio Visual Services*. Norwell, MA: Kluwer, 2000.

[18] L. Kleinrock and A. Nilsson, "On optimal scheduling algorithms for time-shared systems," *J. ACM*, vol. 28, no. 3, pp. 477–486, July 1981.

[19] E. L. Lawler, "A functional equation and its application to resource allocation and sequencing problems," *Manage. Sci.*, vol. 16, no. 1, pp. 77–84, Sept. 1969.

[20] ——, "Optimal sequencing of a single machine subject to precedence constraints," *Manage. Sci.*, vol. 19, no. 5, pp. 544–546, Jan. 1973.

[21] J. K. Lenstra, A. H. G. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annal. Discrete Mathematics 1*, pp. 343–362, 1977.

[22] T. D. C. Little and A. Ghafoor, "Synchronization and storage models for multimedia objects," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 413–427, Dec. 1990.

[23] ——, "Multimedia synchronization protocols for broadband integrated services," *IEEE J. Select. Areas Commun.*, vol. 9, pp. 1368–1382, Dec. 1991.

[24] S. Paek and S. F. Chang, "Video server retrieval scheduling and resource reservation for variable bit rate scalable video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, pp. 460–474, Apr. 2000, to be published.

[25] A. Puri and A. Eleftheriadis, "MPEG-4: A multimedia coding standard supporting mobile applications," *ACM Mobile Networks Applicat. J.*, vol. 3, no. 1, pp. 5–32, June 1998.

[26] J. Signes, Y. Fisher, and A. Eleftheriadis, "MPEG-4's binary format for scene description," *Signal Process. Image Commun.*, vol. 15, no. 4–5, pp. 321–345, Jan. 2000.

[27] J. Song, A. Dan, and D. Sitaram, "Efficient retrieval of composite multimedia objects in JINSIL distributed system," in *Proc. ACM SIGMETRICS*, June 1997, pp. 260–271.

[28] L. Torres and M. Kunt, Eds., *Video Coding: The Second Generation Approach*. Norwell, MA: Kluwer, 1996.

[29] D. Trietsch, "Scheduling flights at hub airports," *Transpor. Res.*, vol. 27, no. 2, pp. 133–150, 1993.

[30] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, no. 10, pp. 384–393, 1975.

[31] "Synchronized multimedia integration language (SMIL 1.0)," W3C Recommendation, June 1998.

Hari Kalva (M'00) received the B.Tech. degree in electronics and communications engineering from N.B.K.R. Institute of Science and Technology, S.V. University, Tirupati, India, in 1991, the M.S. degree in computer engineering from Florida Atlantic University, Tallahassee, in 1994, and the M.Phil. and Ph.D. degrees in electrical engineering from Columbia University, New York, in 1999 and 2000, respectively.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, Florida Atlantic University (FAU), Boca Raton, FL. Prior to his appointment at FAU, he was a consultant with Mitsubishi Electric Research Labs, Cambridge, MA, where he worked different projects including MPEG-2 to MPEG-4 real-time video transcoding. He was a Cofounder and the Vice President of Engineering of Flavor Software, a New York company founded in 1999, that developed MPEG-4 based solutions for the media and entertainment industry. He has over 24 published works and three patents (eight pending) to his credit. He is the author of one book and coauthor of five book chapters. His research interests include video compression and communications, content representation, content adaptation, and mobile multimedia.

Dr. Kalva is a member of the IEEE Communications Society, the IEEE Signal Processing Society, and the ACM.

Alexandros Eleftheriadis (M'95–SM'02) was born in Athens, Greece, in 1967. He received the Diploma in electrical engineering and computer science from the National Technical University of Athens, Athens, Greece, in 1990, and the M.S., M.Phil., and Ph.D. degrees in electrical engineering from Columbia University, New York, in 1992, 1994, and 1995, respectively.

Since 1995, he has been with the faculty of the Department of Electrical Engineering, Columbia University (currently as an Associate Professor), where he is leading a research team working on multimedia signal processing, with emphasis on multimedia software, video signal processing and compression, and video communication systems, and most recently, music signal processing. In 1999, he cofounded Flavor Software Inc., New York, a company that builds software for creating and distributing rich media content using MPEG-4. In the summers of 1993 and 1994, he was with the Signal Processing Research Department, AT&T Bell Labs, Murray Hill, NJ, where he performed research on very low bit rate coding systems. He has published more than 100 papers in international journals and conferences, and has been awarded 11 patents.

Dr. Eleftheriadis is a recipient of an National Science Foundation CAREER Award and has been elected member of the Multimedia Signal Processing Technical Committee of the IEEE Signal Processing Society. He also served as the Editor of the MPEG-4 Systems (ISO/IEC 14 496-1) specification. He serves in the editorial board of the *Multimedia Tools and Applications Journal*, and has served as a Guest Editor, Committee Member, and Organizer for several international journals and conferences. He is a member of the AES, the ACM, and the Technical Chamber of Greece.