

# **Algorithms for Multiplex Scheduling of Object-Based Audio-Visual Presentations**

**Hari Kalva and Alexandros Eleftheriadis**

*Dept. of Electrical Engineering, Columbia University, New York*

## **Abstract**

**With the emergence of standards such as MPEG-4, systems and applications using object-based audio-visual presentations will become more and more common. The main distinguishing feature of object-based audio-visual presentations is the scene composition at the user terminal. In this paper we discuss the problem of scheduling object-based audio-visual presentations with 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 audio-visual presentations, and for un-schedulable 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.**

## **1. 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 scenes with semantically or structurally meaningful objects. Such a new paradigm of object-based representation of audio-visual scenes and presentations will change the way audio-visual applications are created. MPEG-4 is a standardization activity, under the auspices of the ISO, specifying tools to enable object-based audio-visual 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 audio-visual 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 audio-visual 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 spatio-temporal composition of objects in a scene [26]. MPEG-4 also specifies DMIF or the Delivery Multimedia Integration Framework, 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 non-trivial. Unlike MPEG-2, the content creation process involves much more than multiplexing the media streams. Determining the schedulability 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.

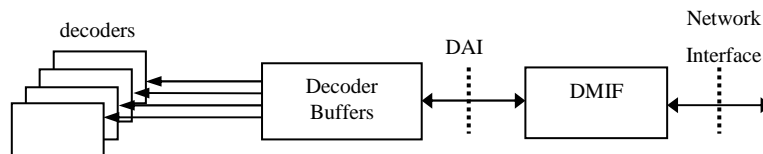
In this paper we discuss the problem of scheduling audio-visual objects and present algorithms for optimal scheduling of audio-visual objects. We present new algorithms, based on job sequencing on a single machine proposed by Carlier [3], for scheduling objects in a presentation. The original contributions of this paper include: algorithm to determine the schedulability of

audio-visual 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]. The paper is organized as follows: the general problem of scheduling audio-visual objects and related earlier work is presented in Section 2. The characteristics of startup delay and terminal buffer are discussed in Section 3. In Section 4 we present several algorithms to schedule audio-visual presentations. Discussion and results are presented in Section 5. We conclude the paper in Section 6.

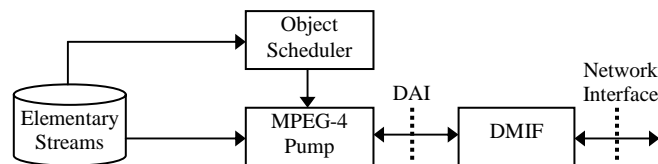
## 2. Scheduling Audio-Visual Objects

Scheduling and multiplexing of audio-visual (AV) objects in a presentation is a complex problem. Scheduling of audio-visual objects has been the subject of study in [1][23][27]. In [23] Little and Ghafoor present synchronization of multi-object presentations using Petri-net models to describe timing relations in multimedia presentations. They present network-level and application-level synchronization protocols for multi-object 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 audio-visual 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 play-rate 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 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 practicable because of the processing required at the intermediary nodes. The approach reserves a CBR channel assuming only CBR sources while it has been shown that multiple CBR sources, when combined, result in a VBR source [9]. We consider a generic MPEG-4 presentation and present algorithms for determining a delivery schedule.

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 audio-visual data that corresponds to the objects in a scene [1]. A scene consists of one or more audio-visual 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* (access unit 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. The difference 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 fps). In a general architecture such as MPEG-4, such assumption is not valid as the data can 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 inter-dependencies 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 access units have to be considered for efficient scheduling. Furthermore, the delay tolerances and relative priorities of objects in an audio-visual 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 audio-visual services can be used to efficiently schedule a presentation with minimal resource consumption.



**Figure 1.a. Terminal Model**



**Figure 1.b. Server Model**

## 2.1. System Model and Assumptions

We discuss the scheduling of audio-visual objects in the context of a system consisting of client (end-user), server, and network components as shown in Figure 1.a. Figure 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 access units. 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 access unit. 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 audio-visual presentation is composed of one or more objects (AV Objects).
- An access unit (AU) is the smallest piece of data that can be associated with a decoding time.
- An audio-visual object contains one or more access units.
- Objects and their access units 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 access units.
- All the access units of an object are delivered in their decoding order.
- A server presents an access unit to the transport layer at the send time determined by the scheduler.

## 2.2. Notation

The following notation is used in this paper.

$\mathbb{N}$  – set of objects to be scheduled

$N$  – number of objects to be scheduled

$n_i$  – number of access units 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 access units in the presentation

$T_j^d(k)$  – decoding time of  $A_j(k)$ .

$T_j^s(k)$  – send time of  $A_j(k)$ .

$\sigma \equiv \bigcup_{j,k} T_j^s(k)$  – the send-time schedule

$C$  – a transmission channel of capacity  $C$ .

$s_j(k)$  – size in bytes of  $A_j(k)$

$d_j(k)$  – duration (channel occupancy) of access unit  $k$  on the wire;  $d_j(k) = C \div s_j(k)$ ;

$T_s$  – startup delay.

$T_s^{\max}$  – max startup delay.

$T_s^{\min} = \sum_k d_k(0) \quad \ni T_k(0) = 0$  – time to transmit AUs of all objects with DTS/CTS of

zero.

### 2.3. Problem Formulation

Given a set of  $N$  objects that comprise an audio-visual presentation, with each object containing  $n_i$  access units each with a decoding time  $T_j^d(k)$ , of  $k^{\text{th}}$  access unit of object  $j$ , a transmission channel of capacity  $C$ , terminal buffer of size  $B$ , allowed startup delay of  $T_s^{\text{max}}$ , and duration (channel occupancy) of each access unit on the channel,  $d_j(k)$ : is there a schedule  $\sigma$  that satisfies the following constraints?

$$T_j^s(k) \leq T_j^d(k) - d_j(k) \quad (1)$$

$$T_j^s(k+1) \geq T_j^s(k) + d_j(k) \quad (2)$$

if  $A_i(k) \in A - A_j(l)$ , then, (3)

$$\text{either } T_i^s(k) + d_i(k) \leq T_j^s(l) \text{ or } T_i^s(k) \geq T_j^s(l) + d_j(l)$$

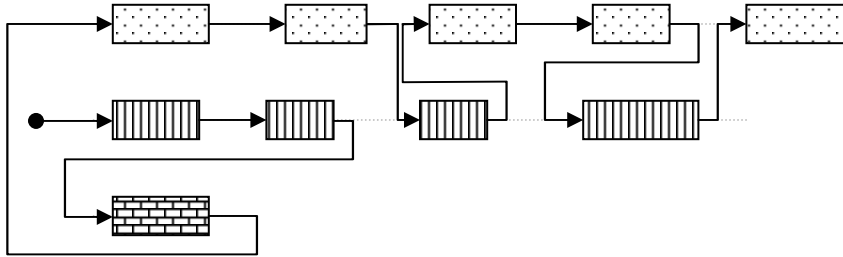
$$B(t) = \sum_{j,k} C * d_j(k) \quad j \ni T_j^s(k) + d_j(k) \leq t, T_j^d(k) \geq t \quad (4)$$

$$T_s \leq T_s^{\text{max}} \quad (5)$$

Constraints (1)-(5) represent the conditions for transmission and playback of object based audio-visual presentations. Constraint (1) enforces the on-time delivery of access units. Ignoring the constant end-to-end delays, (1) gives the latest time an access unit can be transmitted. Constraint (2) imposes intra-object synchronization by enforcing precedence constraints among the access units. 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 access units 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 access units in the presentation, only one of them can be transmitted at any time. Equation (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 Figure 2. Object 1 has five access units, object 2 has three, and object 3 has one access unit. 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. Figure 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* below.



**Figure 2. Sequencing of access units in a 3-object Presentation**

Scheduling is a complex problem and has been widely studied [4][5][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 audio-visual 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 audio-visual 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 audio-visual 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 audio-visual presentations, some of the results can be used in scheduling individual objects on a channel. The results of single machine scheduling problem as formulated by Lawler in [19][20] may be used to determine the schedulability of individual objects.



## 2.4. Complexity of audio-visual object scheduling

*Theorem 1:* Scheduling of access units in Audio-Visual 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  $d(t) \in \mathbb{Z}^+$ , and a length  $l(t) \in \mathbb{Z}^+$ .

*QUESTION:* Does there exist a feasible schedule for  $T$ , i.e., a function  $\sigma: T \rightarrow \mathbb{Z}^+$ , 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 access unit  $A_j(k)$  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 pseudo-polynomial time algorithm. We present several polynomial-time algorithms based on heuristics and constraint relaxation and evaluate their performance with respect to speed and efficiency.

## 3. 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 audio-visual 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) access units 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.

### 3.1. Residual Data Volume

We introduce the notion of *data volume* to quickly compute the minimum startup delays needed for a presentation and determine the non-schedulability. 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 ( $V_{res} = V_d - V_p$ ). 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 below.

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

$C \geq C_{\min}$  where  $C_{\min} = \frac{V_d}{D_p}$ , and the bound is tight.

*Proof:*

$$V_d = \sum_{j,k} s_j(k)$$

$$D_p = \max_j \{T_j^d(n_j)\} - \min_j \{T_j^d(n_j)\}$$

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

$$T_s^{res} = (V_d - V_p) / C$$

To minimize the startup delay,

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

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)

$$T_j^s(1) \leq T_j^d(1) - d_j(1) \Rightarrow T_j^s(1) \leq T_j^d(1) - s_j(1) / C$$

From constraint (2)

$$T_j^s(1) \geq T_j^s(0) + s_j(0) / C$$

Assuming that the presentation starts at time 0

$$T_j^s(0) = 0, \Rightarrow T_j^s(1) \geq s_j(0) / C$$

From constraints (1) and (2),

$$T_j^s(2) \geq T_j^s(1) + s_j(1) / C \Rightarrow T_j^s(2) \geq s_j(0) / C + s_j(1) / C$$

$$\begin{aligned} T_j^s(2) \leq T_j^d(2) - s_j(2) / C &\Rightarrow T_j^d(2) \geq T_j^s(2) + s_j(2) / C \\ &\Rightarrow T_j^d(2) \geq s_j(0) / C + s_j(1) / C + s_j(2) / C \end{aligned}$$

Similarly,

$$T_j^d(n_j) \geq s_j(0) / C + s_j(1) / C + \dots + s_j(n_j) / C$$

Since the AUs are transmitted on a single channel,

$$D_p = \max\{T_j^d(n_j)\} \geq \sum_j s_j(0) / C + s_j(1) / C + \dots + s_j(n_j) / C$$

$$D_p \geq \frac{1}{C} \sum_{j,k} s_j(k)$$

$$\Rightarrow D_p \geq \frac{1}{C} * C_{\min} * D_p$$

$$\Rightarrow C \geq C_{\min}$$

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

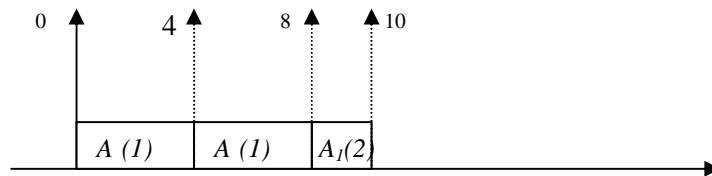
Object 1 has 2 AUs and object 2 has 1 AU. The decoding times and sizes of AUs in bytes

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

With these values the send times and channel capacity are:

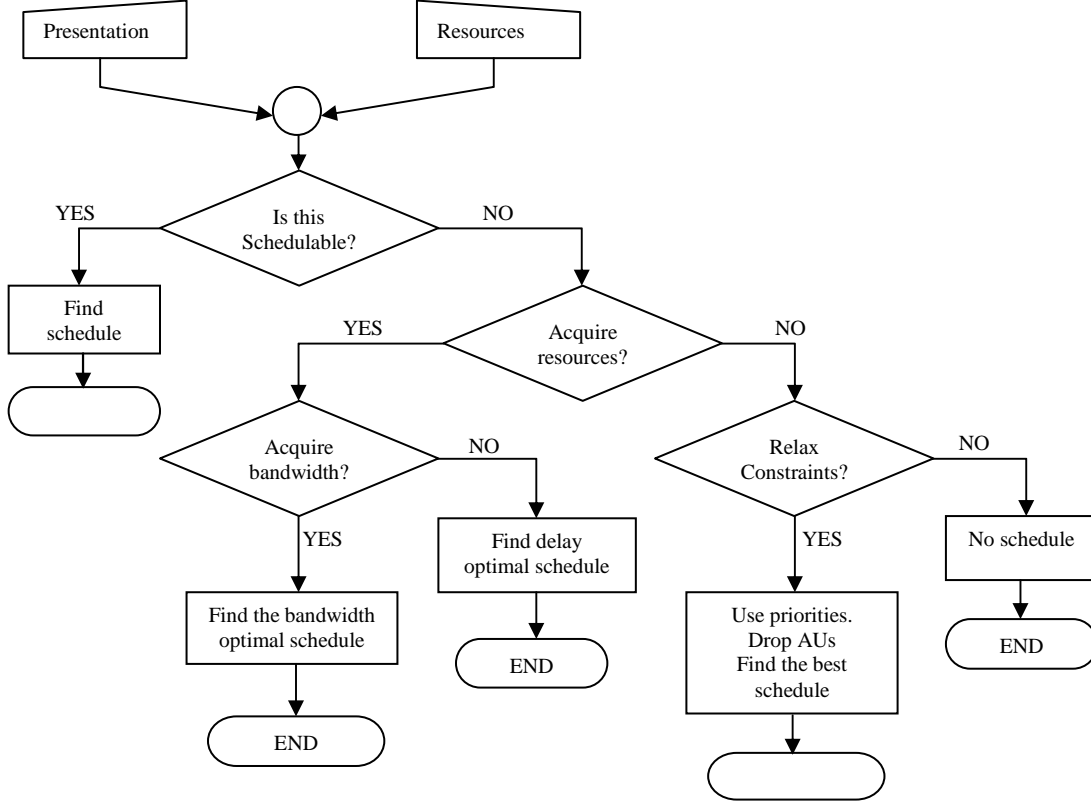
$T_1^s(1) = 0$ ,  $T_1^s(2) = 8$ ,  $T_2^s(1) = 4$ , and

$C = C_{\min} = 2.5 \text{ bytes / sec}$ .



**Figure 3. Example to prove the tightness of the bound**

The actual channel capacity required to minimize startup delay may be higher depending on the timing constraints of access units. Note that irrespective of the channel capacity, the minimum startup delay remains non-zero and is equal to  $T_s^0$ .



**Figure 4. Determining the Schedulability of a Presentation**

Thus for any given presentation with resource constraints:

$$\text{the minimum startup delay is, } T_s^{\min} = \max\{T_s^{\text{res}}, T_s^0\}$$

$$\text{the minimum buffer capacity required is, } B_{\min} = T_s^{\min} * C$$

the presentation is schedulable only if the available buffer is at least equal to  $B_{\min}$ .

#### 4. 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 re-purposed to compute the

minimum terminal buffer required for a given channel capacity. Figure 4 shows the flowchart for selecting an appropriate scheduling algorithm. The choice of the algorithm depends on the applications, resource availability, and constraints.

#### 4.1. Algorithm FullSched

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 4.2.

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.

$$x_j = n_j, \quad 1 \leq j \leq N$$

$$\text{Initialize } S = \{A_j(x_j)\}, \quad 1 \leq j \leq N$$

$S(j)$  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  $G = \{\emptyset\}$

Set current time  $i = \infty$

Sort AU of objects in the decreasing order of their decoding times.

BEGIN

```

while ( $S \neq \emptyset$ ) {
     $i = \min\{i, \max\{T_j^d(k)\}\}, \quad T_j^d(k) \ni A_j(k) \in S$ 
     $T_j^s(x_j) = i - d_j(x_j)$ ; // send time for  $A_j(x_j)$ 
    // Update  $i$ 
     $i -= d_j(x_j)$ ;
     $x_j--$ ;
    // Update  $S$  by removing  $S(j)$  from  $S$ 
     $S -= S(j)$ ;
    // add  $A_j(x_j)$  to  $S$ 
    if ( $x_j \neq 0$ )
         $S += AU(j, x_j)$ 
}

```

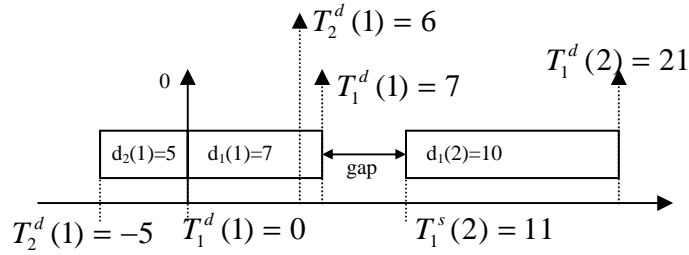
```

if ( $i > \max\{T_j^d(k)\}$ ),       $T_j^d(k) \ni A_j(k) \in S$ 
    // 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  $T_s = |T_{first}^s|$ 
 $T_j^d(k) += T_s, \quad \forall j, k$ 
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  $O_1$  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.



**Figure 5. Applying FullSched to a two-object Presentation**

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,  $T_j^d(k)$ , are all non-negative, 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 sub-optimality of the

schedule. However, it may indicate the sub-optimality 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 un-schedulability 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 un-schedulability 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 re-computed 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 non-increasing 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!*

## **4.2. The 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 access unit 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  $j$

$$S = \{A_j(k)\}, \quad 1 \leq k \leq n_j, j \in \{N\}$$

Sort AUs in  $S$  in the decreasing order of their decoding times

$G$  = set of available slots  $\neq \{\emptyset\}$ .

$G(l)$ , is the  $l^{\text{th}}$  tuple in  $G$  with start time  $G(l).t$  and duration  $G(l).d$ .

$$k = n_j$$

BEGIN

```

    while ( $S \neq \emptyset$ ) {
        find a slot  $l$ ,  $G(l)$ , such that  $T_j^d(k) > G(l).t$ 
        if ( $G(l).d \geq d_j(k)$ ) {
             $T_j^s(k) = G(l).t - d_j(k)$  //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 sub - AU of length  $G(l).d$  containing the last  $G(l).d * C$  bytes of the AU
     $t_j^s(k) = G(l).t$ ;
     $d_j(k) = d_j(k) - G(l).d$ ;
     $G - = \{G(l)\}$ ;
}
```

### 4.3. The IncSched Algorithm IS

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 re-computing 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 un-schedulable if there are no AUs with decoding times greater than the gap time. An un-schedulable gap indicates unused bandwidth, which is either due to the structure of the presentation or due to a sub-optimal 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.

$OBJ$  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  $j \in OBJ - \{1\}$ , apply gap scheduling GS to  $j$ .

    for  $j \in OBJ$ , find  $t_{first}$ , the send time of the first AU to be transmitted (smallest  $t_j(k)$ )

    if  $T_{first}^s < 0$

    then  $T_s = |T_{first}^s|$

$T_j^d(k) += T_s, \quad \forall j, k$

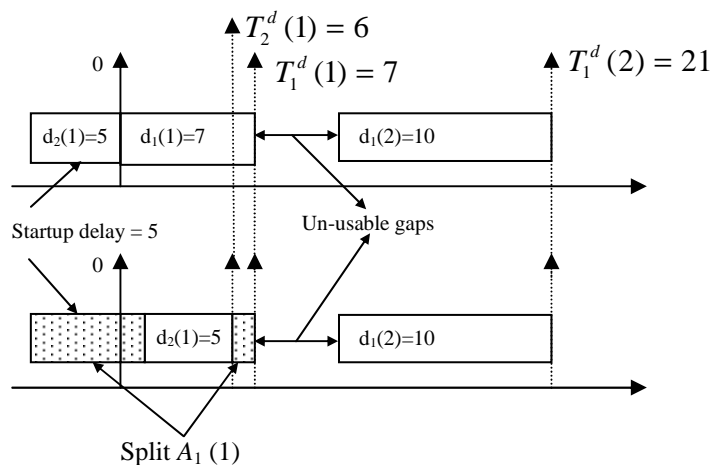
END

*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 access unit 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 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  $O_2 = \{ \langle 5, 6 \rangle \}$ . The top part of the figure shows the schedule computed using FullSched and the bottom half shows the schedule computed with IS, with object 2 scheduled first using FullSched. The figure also shows un-schedulable 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 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

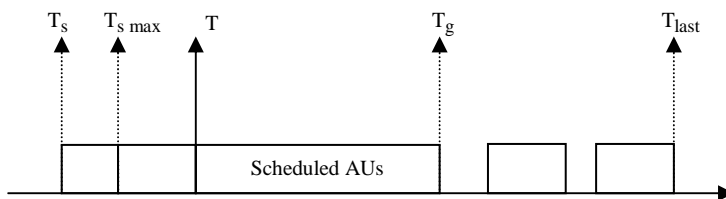


**Figure 6. Schedules Computed Using FullSched and IS**

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.

#### 4.4. Algorithm MinC

In scheduling audio-visual 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.



**Figure 7. First gap-time and startup delay of 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 un-scheduled 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 Figure 7.  $T_g$  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 start-up 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 bandwidth profile ( $BP$ ) for the presentation in the form of 3-tuples  $\langle \text{capacity, start, end} \rangle$ . Note that the increased channel capacity is not going to affect the schedule from  $T_g$  to  $T_{\text{last}}$ . A finer bandwidth profile can be obtained by initializing  $C$  with  $C_{\min}$  and increasing  $C$  by a small value in each iteration.

BEGIN

$G_c = |G| = \text{gap count, number of gaps on the channel.}$

$BP = \{\phi\}$

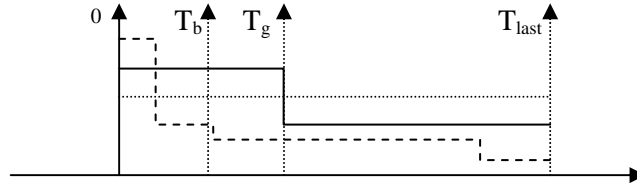
$T_{\text{last}}$  is the decoding time of the first AU scheduled (= duration of the presentation)

```

C = Cmin, computed using the results of theorem 2.
SCHEDULE: compute schedule using FullSched
If schedulable goto END
if (Gc == 0)
    Tg = Tlast
else {
    Tg-old = Tg
    Find the smallest gap time, Tg
    BP += {<C, Tg, Tg-old>}
}

C = C + C * ((Tsmax - Ts) / (Tg - Tsmax))
goto: SCHEDULE
END

```



**Figure 8. Typical shapes of bandwidth usage generated by MinC**

The channel capacity output by the algorithm is in the form of a set of 3-tuples forming a bandwidth profile. The minimum CBR channel required is given by the maximum value of  $C$  in the bandwidth profile. 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 bandwidth profile will always be a step function with possible steps (decreasing) from left to right. Figure 8 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 Figure 8. 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.

#### 4.5. Algorithm BestSched

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 access units. 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 ( $B \leq T_s * C$ ) {
    Remove  $A_j(k)$  of lower priority objects such that,
     $R = \{A_j(k)\} \ni C * \sum d_j(k) \geq T_s * C - B$ 
     $A = A - \{R\}$ 
    goto: SCHEDULE
  }
END

```

## 5. Discussion and Results

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.

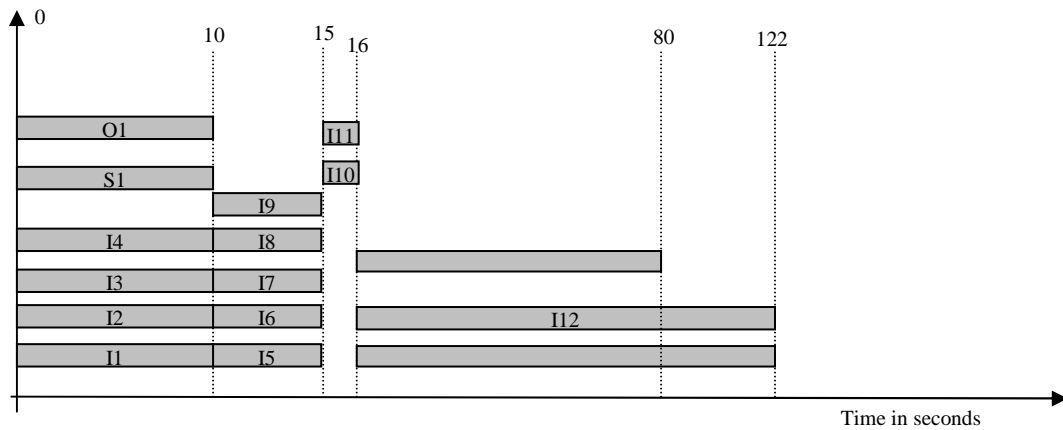


Figure 9. Structure of the presentation in the example

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 Figure 9. The properties of the objects in the presentation are tabulated in Table 1.

**Table 1: 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

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 Figure 10. 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 seconds 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 audio-visual 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 Figure 9. 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 seconds. 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 seconds: 261 KB  $\approx$  130 Kbps).



**Figure 10. Snapshot of the Player with the Scheduled Presentation**



### 5.1. Startup Delay and Capacity Computation

Figure 11 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 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 PCs with sufficient memory. As mentioned earlier in the discussion of the MinC algorithm, the MinC algorithm also computes the bandwidth profile for presentations. Figure 12 shows the bandwidth profile computed for a startup delay of 5 seconds. The minimum capacity in the profile is 59 Kbps even for the segment (80 - 120 secs) that only has low bit rate audio (6 Kbps). This is because MinC starts with an initial value of C computed using the residual data volume as described in Section 3. 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.

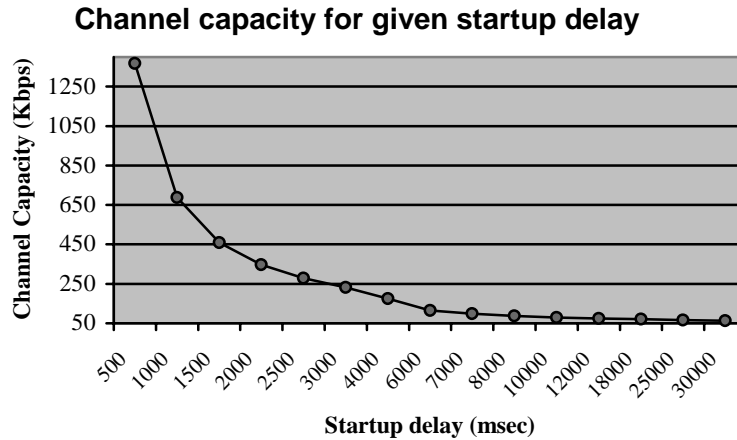
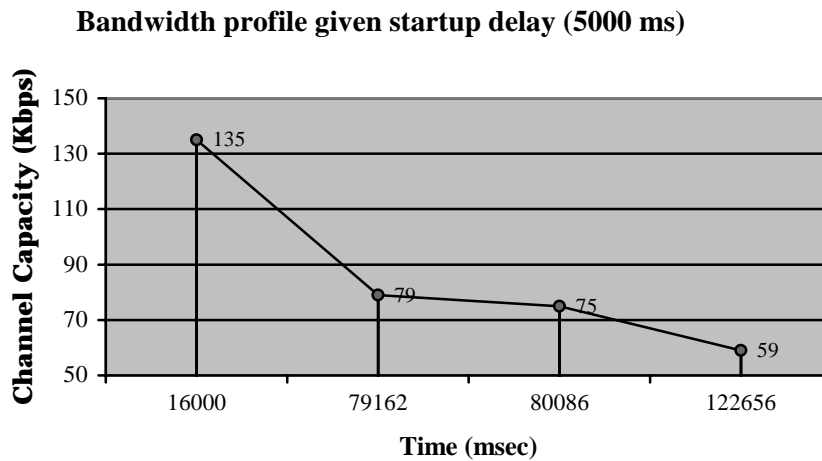


Figure 11. Computing min capacity using MinC

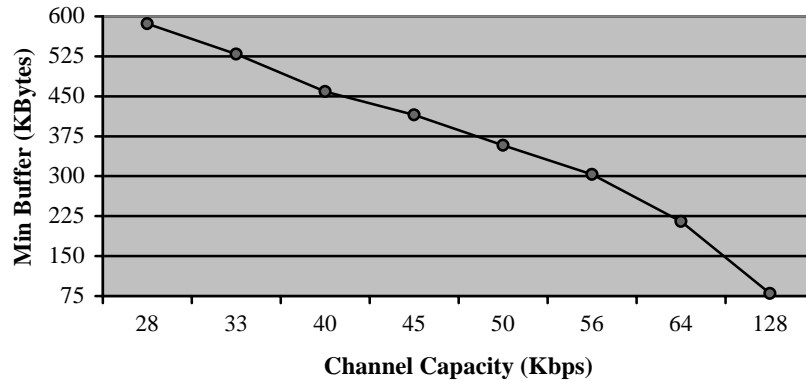


**Figure 12. Computing the bandwidth profile using MinC**

## 5.2. 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. Figure 13 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-boxes 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 be greater than the reduction in startup delay supported by the buffer.

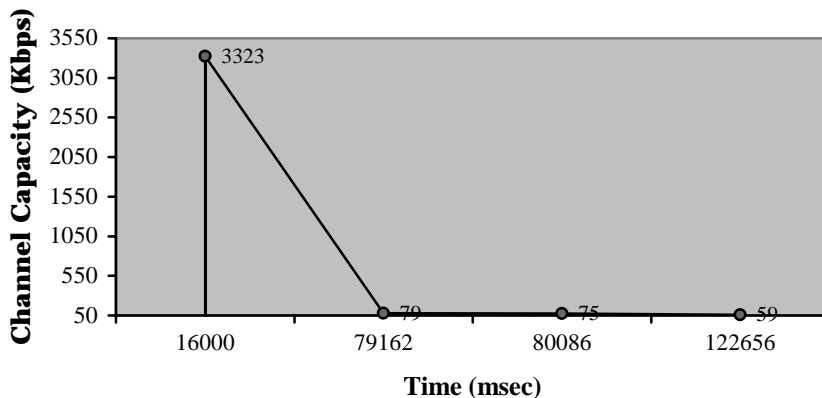
**Min buffer for given channel capacity**



**Figure 13. Minimum required buffer**

Figure 14 shows a case with terminal buffer  $< B_{\min}$ . For the given presentation,  $B_{\min} = 85KB$ , the size of objects to be decoded at time 0. As discussed in Section 3, 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  $\frac{B_{term}}{C} < \frac{B_{min}}{C}$ , the required startup-delay. The presentation is hence un-schedulable with terminal buffer capacity of 84 KB. This is depicted in Figure 14, which shows the presentation is un-schedulable even at 3323 Kbps. 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.

**Bandwidth profile given buffer capacity (84K < Bmin)**



**Figure 14. Partial profile for low terminal buffer**

## 6. Conclusion

We presented the problem of scheduling object-based audio-visual 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 audio-visual presentations and also mixed media presentations described with formats such as SMIL[31]. 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.

## 7. References

- [1] O. Avaro, A. Eleftheriadis, C. Herpel, G. Rajan, and L. Ward, "MPEG-4 Systems: Overview", *Signal Processing: Image Communication*, Tutorial Issue on the MPEG-4 Standard, Vol. 15, Nos. 4-5, January 2000, pp.281-298.
- [2] M. C. Buchanan and P. T. Zellweger, "Scheduling Multimedia Documents Using Temporal Constraints," NOSDAV 92, pp. 223-235.
- [3] J. Carlier, "The One Machine Sequencing Problem," *European Journal of Operational Research*, No. 11, 1982, pp. 42-47.
- [4] T.L. Casvant and J.G. Kuhl, "A Taxonomy of Scheduling in General Purpose Distributed Computing Systems," *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, Feb 1988, pp. 141-154.

- [5] P. Chretienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, *editors*, "Scheduling Theory and its Applications," John Wiley & Sons, 1995.
- [6] R.W. Conway, W.L. Maxwell, and L.W. Miller, "Theory of Scheduling," Addison-Wesley Publishing Company, 1967.
- [7] M.L. Escobar-Molano, "Management of Resources to Support Coordinated Display of Structured Presentations," Ph.D. Dissertation, Graduate School, University of Southern California, 1996.
- [8] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness," W. H. Freeman, San Francisco, 1979.
- [9] L. Grossglauser and S. Keshav, "On CBR Service," Proceedings of the INFOCOM, March 1996.
- [10] L.A. Hall, "Approximation Algorithms for Scheduling," Approximation algorithms for NP-hard problems," D. S. Hochbaum edts., PWS Pub. Co., c1997, pp. 1-45.
- [11] C.-C.Han, K.-J.Lin, J.W.-S. Liu, "Scheduling Jobs with Temporal Distance Constraints," SIAM Journal on Computing, Vol. 24, NO. 5, October 1995, pp. 1104-1121.
- [12] C. Herpel and A. Eleftheriadis, "MPEG-4 Systems: Elementary Stream Management", Signal Processing: Image Communication, Tutorial Issue on the MPEG-4 Standard, Vol. 15, Nos. 4-5, January 2000, pp.299-320.
- [13] ISO/IEC/SC29/WG11, "Generic Coding of Moving Pictures and Associated Audio (MPEG-4 Systems) - ISO/IEC 14386-1," International Standards Organization, April 1999.
- [14] ISO/IEC/SC29/WG11, "Delivery Multimedia Integration Framework (DMIF) - ISO/IEC 14496-6," International Standards Organization, February 1999.
- [15] N. S. Jayant, "Signal compression: Technology targets and research directions," IEEE Journal on Selected Areas in Communications, Special issue on speech and image coding, June 1992.
- [16] H. Kalva, L-T. Cheok, A. Eleftheriadis, "MPEG-4 Systems and Applications," Demonstration, ACM Multimedia '99, Orlando, FL.
- [17] H. Kalva, "Delivering MPEG-4 Based Audio Visual Services," ISBN 0-7923-7255-7, Kluwer Academic Publishers, Nov. 2000.

- [18] L. Kleinrock and A. Nilsson, "On Optimal Scheduling Algorithms for Time-Shared Systems," *Journal of the ACM*, Vol. 28, No. 3, July 1981, pp. 477-486.
- [19] E.L. Lawler, "A Functional Equation and its Application to Resource Allocation and Sequencing Problems," *Management Science*, Vol. 16, No. 1, September 1969, pp. 77-84.
- [20] E.L. Lawler, "Optimal Sequencing of a Single Machine Subject to Precedence Constraints," *Management Science*, Vol. 19, No. 5, January 1973, pp. 544-546.
- [21] J.K. Lenstra, A.H.G.R. Kan, and P. Brucker, "Complexity of Machine Scheduling Problems," *Annals of Discrete Mathematics* 1, 1977, pp. 343-362.
- [22] T.D.C. Little and A. Ghafoor, "Synchronization and Storage Models for Multimedia Objects," *IEEE Journal on Selected Areas in Communications*, Vol 8, No. 3, Dec 1990, pp. 413-427.
- [23] T.D.C. Little and A. Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services," *IEEE Journal on Selected Areas in Communications*, Vol 9, No. 9, Dec 1991, pp. 1368-1382.
- [24] S. Paek and S.F. Chang, "Video Server Retrieval Scheduling and Resource Reservation for Variable bit Rate Scalable Video," To be published in *IEEE Transactions on Circuits and Systems for Video Technology*.
- [25] A. Puri and A. Eleftheriadis, "MPEG-4: A Multimedia Coding Standard Supporting Mobile Applications" *ACM Mobile Networks and Applications Journal*, Special Issue on Mobile Multimedia Communications, Vol. 3, No. 1, June 1998, pp. 5-32 (invited paper).
- [26] J. Signes, Y. Fisher, and A. Eleftheriadis, "MPEG-4's Binary Format for Scene Description," *Signal Processing: Image Communications*, Tutorial Issue on the MPEG-4 Standard, Vol. 15, Nos. 4-5, January 2000, pp.321-345.
- [27] J. Song, A. Dan, and D. Sitaram, "Efficient Retrieval of Composite Multimedia Objects in JINSIL Distributed System," *ACM SIGMETRICS*, June 1997.
- [28] L. Torres, M. Kunt, eds., "Video Coding : The Second Generation Approach," Kluwer Academic Publishers, 1996.
- [29] D. Trietsch, "Scheduling Flights at Hub Airports," *Transportation Research*, Vol 27, No. 2, 1993, pp 133-150.

- [30] J.D. Ullman, "NP-Complete Scheduling Problems," Journal of Computer and System Sciences, No. 10, 1975, pp. 384-393.
- [31] W3C, "Synchronized Multimedia Integration Language (SMIL 1.0)," W3C Recommendation, June 1998.