# Remote Access to Interactive Media[1]

**Roger B. Dannenberg**

Carnegie Mellon University, School of Computer Science
Pittsburgh, PA 15213 USA
Email: dannenberg@cs.cmu.edu

**ABSTRACT**

Digital interactive media augments interactive computing with video, audio, computer graphics and text, allowing multimedia presentations to be individually and dynamically tailored to the user. Multimedia, and particularly continuous media pose interesting problems for system designers, including those of latency and synchronization. These problems are especially evident when multimedia data is remote and must be accessed via networks. Latency and synchronization issues are discussed, and an integrated system, Tactus, is described. Tactus facilitates the implementation of interactive multimedia computer programs by managing latency and synchronization in the framework of an object-oriented graphical user interface toolkit.

## 1. Introduction

High-bandwidth digital communication is creating many new possibilities for interpersonal and human-computer interaction. An important dimension of the new technology is the ability to transmit multimedia information, information that engages the senses with moving color images and high quality sound. *Digital* multimedia is interesting not so much because of images and sound but because this content can be manipulated and stored directly on computer systems. This brings tremendous new flexibility and integration.

In particular, digital media can be interactive. Unlike traditional analog storage and playback formats, digital media are amenable to non-sequential access, synthesis, indexing, cross-linking, annotation, excerpting, and transmission. Interactive systems can tailor multimedia presentations individually, rather than relying on the ‘‘common denominator’’ approach seen in mass media. Individual tailoring can be based in part on models of the user; that is, the computer retains a model of a user preferences, knowledge, and abilities in order to deliver information efficiently. Interactive multimedia is important for education, entertainment, and communication, to name just some of the large application areas.[1]

What are the technical requirements of interactive multimedia? Media such as audio and video are characterized by a relatively constant flow of data over time, e.g. 30 video frames per second or 44100 audio samples per second. These are often referred to as ‘‘continuous media’’ and require a sustained and guaranteed throughput. Continuous media usually have either a constant or a *maximum* data rate[2]. System performance with continuous media is measured by the reliability of transmission. Other media, such as text, numerical data, and still images, present transient throughput demands. In these media, there is no upper bound on data rate, and system performance is measured in terms of throughput or response time.

In interactive media, a timely startup of media from the moment the user makes a request is important. Long delays limit the granularity of interaction and the degree with which the user can interact with the system. For example, a delay of several seconds for a database search may be be tolerable, but delays of even tenths of seconds are annoying in direct manipulation editing systems. The reaction time of the system must be suited to the style of interaction the system supports.

---

[1]Published as: Dannenberg, ‘‘Remote Access to Interactive Media,’’ in ***Enabling Technologies for High-Bandwidth Applications*** (September 1992), Proceedings of the SPIE 1785, pp. 230-237 (1993).

[2]In particular, compression schemes often generate variable data rates that depend on the complexity of the data being compressed. A maximum rate is often dictated by hardware limitation.

### 1.1. Organization

This paper has two main sections. Section 2 gives an overview and tutorial on the issues of latency and synchronization in computer systems. Section 3 describes how these issues are treated in the Tactus system, which supports the implementation of interactive media. Readers with real-time software backgrounds should feel free to skip ahead to read about Tactus, while others may choose to read the overview (Section 2) and skip the more detailed description of Tactus (Section 3). The last section presents a summary and conclusions.

## 2. Issues

What are the problems that arise in interactive multimedia? In the following sections, we will visit some of the problems and the relevant techniques and solutions that are known. We will pay particular attention to the cases where media sources are not physically near the presentation site(s). This implies a need for communication and distributed processing, a source of many interesting problems.

### 2.1. Latency

Latency is the delay from the time an event or stream of data is requested to when it is actually delivered. There are many sources of latency, including operating system task scheduling, computation, disk rotational latency, data buffers, and signal propagation. Many techniques are used to overcome or minimize the effect of latency. Among these are local computation, local buffering, precomputation, scheduling, and faster systems. These are often used in combination.

One solution is to perform computation locally. Often, inexpensive local processors can be dedicated to a time-critical task much more cheaply and effectively than providing a shared central processor. With many interactive media systems, all processing is performed locally by personal computers and special hardware. As applications and processing become distributed, local processors can still manage the user-interface and perform time-critical low-latency feedback. For example, local computation can render a 3D view of a graphical database as the user changes orientation. Only changes to the database need to be transmitted to or from a remote central site.

Local buffering is another important technique for managing latency. Whenever multimedia information access times are high, it may be helpful to prefetch the information, buffer it locally, and then present it. Local buffering has two benefits: First, it can eliminate some of variance in data rates that might otherwise be present. Often, high-speed networks transmit data in large bursts rather than in more frequent smaller packets. Second, local buffering may provide data at a higher rate than the direct source. For example, a movie could be transmitted over phone lines not in real time and stored locally for real-time playback. This same principle is at work on a different time scale in the memory cache of a high-speed processor: The cache has a faster access time than primary memory.

Another important technique is precomputation to minimize the amount of time-critical computation.[2, 3] For example, quality video and animation requires that a new frame be made available during the vertical retrace interval; otherwise, transitions between frames might become visible. However, it may not be possible to decompress or render an entire frame during this relatively short interval. Instead, frames are rendered one frame (or more) ahead and then copied to the display buffer during the vertical retrace. Thus precomputation removes all computation from the time-critical vertical retrace interval except for that of a final copy operation. Note that precomputation is always associated with buffering (to hold the precomputed results), but buffering may sometimes be useful without precomputation.

In a typical digital communication and computing system, many tasks are performed at once, requiring that hardware be multiplexed to service each task in turn. When computations and data become ‘‘bursty,’’ momentary or even extended system overloads occur. Scheduling mechanisms and policy then determine which, if any, tasks will receive low-latency service. In computer-based systems, large amounts of communication and computation are ‘‘non-real-time’’ tasks that can be delayed with little or no cost. Other tasks, such as the delivery of continuous multimedia information are time-critical. Appropriate prioritizing and scheduling is necessary to avoid too much latency.

Finally, faster systems can reduce latency. Especially in the areas of digital computation and communication, the technology has shown a steady trend toward faster systems, leading to higher data rates and lower computation times, with orders of magnitude improvements still to come.

## 2.2. Synchronization

Synchronization between various media is critical for effective communication. The degree of synchronization required varies with the medium. Still graphics may only need to be synchronized within 1 second of an audio narrative, while lip-synchronization in video requires on the order of 0.1 second synchronization. Multiple music streams such as audio and MIDI[3] require synchronization on the order of 1ms to 10ms, and stereo channels should be synchronized at the sample level. In this section, we examine some of the techniques used to achieve synchronization. In all cases we assume a specification for the desired synchronization, e.g. ‘‘play video from frame 10 at 29.97 frames/s and audio from sample 1 at 22050 samples/s.’’

In a low-latency system, explicit synchronization is not always necessary. For example, the MIDI protocol[4] used by music synthesizers has no timing information. When a performer pushes down a key, a MIDI note-on message is sent immediately from the keyboard to a synthesizer, which immediately plays the note. Similarly, ‘‘slide show’’ graphics can be synchronized with audio by simply issuing commands to display the graphics at the appropriate time since the latency of putting up an image is low compared to the relatively lax synchronization requirement.

On the other hand, synchronizing audio with video requires more careful management to achieve tight synchronization. A popular technique for synchronization is the ‘‘master/slave’’ approach. One continuous medium serves as the ‘‘master timekeeper’’ while other media are synchronized to it. For example, digital audio playback hardware often serves as a master while video is synchronized to it. Relative time is computed by counting audio samples and dividing by the sample rate. The corresponding video frame number is then computed by multiplying time by the frame rate. If there is a discrepancy between the desired and actual frame numbers, a frame is skipped or doubled to reestablish synchronization. This technique assumes that audio rates are not adjustable and that video hardware allows frames to be skipped or doubled.

Another possibility is synchronization to a real-time clock. In this approach, all media are synchronized to a system-wide clock. This is really a special case of the ‘‘master/slave’’ approach, where the ‘‘master’’ is time itself. The advantage of this approach is that there is no need to derive time from a master i/o device or to communicate this time to other devices. This is particularly attractive in distributed systems where it is cheaper to maintain a local clock than to broadcast time information from the master. The disadvantage is that every device must be able to synchronize to real time, whereas the master/slave approach allows one device, the master, not to synchronize to anything.

In a distributed system, synchronization requires a global time reference. This is less difficult to achieve than one might think: computers with clocks can exchange messages with readings from their local clocks. Timings of round-trip message transmissions are used to correct for network delay, leading to higher accuracy.

To summarize, synchronization of media requires a *specification* of the desired synchronization consisting of sample rates, frame rates, and explicit timestamps for discrete event data. Synchronization is then achieved by dynamically adjusting the presentation rates and/or times of separate media streams during playback to maintain the specification. Adjustment is relative to a *master*, which may be a specific device or time itself. All of the latency reduction techniques, including local computation, local buffering, precomputation, and scheduling, can be used to improve synchronization by allowing more timely corrections.

## 2.3. Other Issues

Moving multimedia data across digital networks poses many problems and is the focus of a vast amount of research. Here, we will mention some major issues, but we will not attempt to cover in detail the many technologies intended to solve these problems.

[3]Musical Instrument Digital Interface

#### 2.3.1. Network Congestion

From the multimedia application point of view, reliable communication is a network problem, to be solved (as suggested earlier) by appropriate scheduling mechanisms and policies. On the other hand, the network designer's point of view may be that reliable communication is at least partially an application problem.

There are several ways that applications can enhance performance and reliability. One technique is to adapt to the available resources. Many compression schemes allow a tradeoff between bandwidth and quality, so it is possible to design system with ''graceful degradation'' when resources are limited. Another technique is to reserve resources, allowing networks to anticipate loads and make early scheduling decisions to avoid subsequent failure. Finally, when resources are not available, the level of interaction must be sacrificed. For example, there is a tradeoff between a carefully specified database search (limiting communication costs) and casual database browsing (enhancing interactivity).

#### 2.3.2. Resource Reservation

Resource reservation is a general technique that can be applied to network transmission media, routers, file servers, and others resources. Obtaining an end-to-end reservation is a complicated process that may involve negotiating and bidding schemes to establish priority and billing. Alternatively, some network technologies, particularly some implementations of ATM, are not very suitable for reservation schemes and rely instead on probabilistic arguments that data will be delivered in a timely manner.

#### 2.3.3. Scheduling

Regardless of whether or not resources are reserved, contention for resources must be resolved by some priority system as implemented by a scheduler. Schedulers use a variety of schemes to optimize the aggregate benefit of a system to its many clients, and scheduling in a distributed communication system is a relatively unexplored but active area of research.

## 3. The Tactus System

We now turn to a case study of Tactus, a system designed to facilitate interactive multimedia interfaces. We assume that data may be distributed, leading to relatively long latencies. Tactus consists of two parts, the application toolkit and the presentation server. The application toolkit helps software developers combine multiple media into synchronized presentations. The presentation server uses local computation and buffering to solve the latency problems associated with remote access to interactive media (Anderson[5] has done related work on servers). Apple's Quicktime[6], Microsoft's MPC[7], and Dec's XMedia are examples of commercial systems that support interactive media, although these systems are less integrated with application toolkits.

From a system perspective, Tactus works as follows (see Figure 1): The application program generates media, adding timestamps and sending the data to a presentation server at the site of the multimedia display[4]. The application computes media slightly ahead of realtime so that data arrives at the server before the display time provided in the timestamps. At the presentation server, data is buffered until the the display time and then forwarded to presentation devices. If one or more media streams fall out of real time, Tactus stops the presentation until the late streams ''catch up,'' at which time the presentation can be restarted. Other recovery policies, such as allowing one stream to fall behind, are also possible.

### 3.1. The Application Toolkit

Object-oriented toolkits are a standard technique for developing interactive graphical programs.[8] Graphical entities are represented by software objects, and there is typically a ''view tree'' of these objects to manage the hierarchy of windows, subwindows, and interactors like buttons, sliders, and text boxes. These systems are event driven: The application is idle until input arrives. When the user clicks a mouse or types a character, the ''target'' or ''handler'' for that event is located by searching the view tree. An event method associated with the target is run, graphics are updated, and the application waits for the next event.

---

[4]We use the term ''display'' to refer to presentation devices for any medium, including video, graphics, audio, music synthesis, etc.

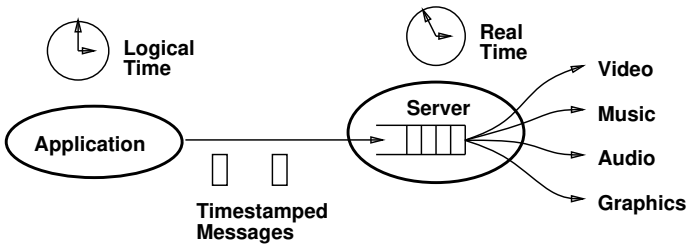


**Figure 1:** Tactus applications are scheduled according to logical clocks running in advance of real time. Data is sent with timestamps to a server, where it is dispatched to multimedia devices at the indicated time.

For continuous media, we want applications to be *time driven* as well as event driven. Tactus extends a toolkit with *time events* to supplement input events, and a *clock tree* to supplement the view tree (see Figure 2). The leaves of the clock tree are subclasses of the *active objects* class[9, 10]. These active objects can request *wakeup* events at a specific time. Active objects handle these events by performing some action, such as generating the next frame of an animation. The parents of active objects in the clock tree are clock objects, which translate real time into the logical time observed by their children in the tree. Clocks are often used to shift the time origin so that, say, an animation starts at the current time instead of at absolute time zero. Clocks can also control the ‘‘rate of passage of time.’’ If a clock is set to run at half speed, the presentation generated by its children will run in slow motion. At the root of the clock tree is a special clock representing real time.

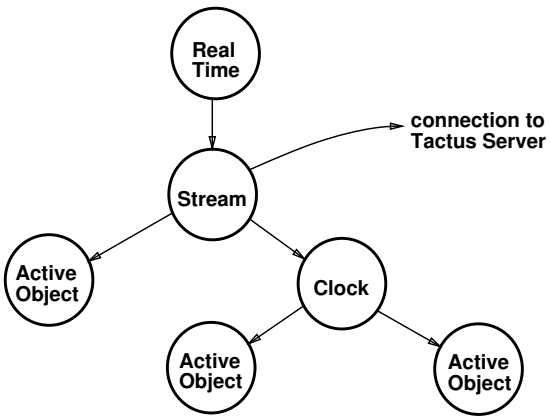


**Figure 2:** A clock tree: active objects compute timed data by requesting ‘‘wakeup’’ messages from their parents. Clock objects apply transformations (shifting and scaling) to time, Stream objects are clocks that also maintain a connection to the Tactus Server, and Real Time is a special clock that interfaces to the system real time clock.

A special clock subclass is the *stream*, which manages connections with the Tactus server. Because a stream is a subclass of clock, it can take the place of a clock in the clock tree. A stream schedules its children to run earlier than real time so that results can be delivered to the Tactus server and buffered before their display times. The stream also provides timestamps for outgoing messages. These timestamps are automatically generated based on the requested wakeup times of the children in the clock tree. The children (active objects) are programmed as if running in real-time on an infinitely fast computer; the actual non-zero computation times are masked by the precomputation, timestamping and buffering provided by Tactus.

### 3.2. The Tactus Server

The Tactus server runs locally at the site where multimedia information is displayed. The server receives messages with timestamps from multiple streams and buffers the messages until the indicated time. In practice, devices often need data ahead of the presentation time. For example, video decompression hardware may need to look several frames ahead, and our MIDI driver accepts timestamped data ahead of real-time. Tactus is able to dispatch messages to device drivers in advance of timestamps.

The server uses an interesting protocol to detect when a stream falls behind real time. On the application side, an active object implicitly indicates the time of the next message when it makes a wakeup request. (The active object will be passive until the wakeup time, so we know there will be no more output until then.) When all of its children are waiting, the stream object sends the the earliest child wakeup time on an empty message to the Tactus server. Assuming messages arrive in order, the server then knows that it has all data until the timestamp of the empty message. If new data does not arrive by that time, the buffer has fallen behind real time.

### 3.3. Enhancing Interactivity

So far, Tactus has been described strictly as a media output system, but interactivity requires input handling and rapid system response. A number of extensions to the Tactus concept are under development at this time.

One interesting problem that arises with Tactus is stream startup. Since an application normally computes data ahead of time by an interval greater than the worst-case latency of the system, the response time from pushing a ‘‘start’’ button to receiving media could be quite long. Tactus solves this problem by acting as if the application has fallen out of real-time upon stream startup. That means that the application computes and sends data as fast as possible to Tactus, and as soon as a ‘‘low water mark’’ is reached in the server buffers, output can commence. Thus, startup is limited only by the time it takes to prime the buffers in the server.

Another issue is that when the application receives input, it will correspond to displays that were computed at some time in the past. Therefore, input events are all timestamped locally (for accuracy) by the server and sent to the potentially remote application. It is up to the application to keep track of enough history to use the input information. For example, if the user clicks on a video frame, the application must map the input click timestamp to a video frame number (which will have been sent earlier) and perhaps even ‘‘rewind’’ the video back to the selected frame.

A common interactive technique is to give users choice points in video to allow ‘‘surrogate travel.’’ For example, a video of an approaching intersection is shown, and the user is allowed to select ‘‘left,’’ ‘‘right,’’ or ‘‘straight.’’ If a Tactus application waits too long for the choice, there might not be enough time to start up a new video stream before the intersection is reached. This would result in either a momentary halt in the presentation or the user input would have to be ignored. Our solution is to allow multiple streams to be active but not actually enabled for presentation. In the intersection example, three independent streams are initiated as the intersection is approached. When the user makes a choice, a ‘‘cut’’ command is issued from the application to Tactus, enabling one of the pre-buffered streams to be started with very low latency. The other streams are then deleted.

### 3.4. Discussion

Tactus embodies many of the techniques described in Sections 2.1 and 2.2. Let us review the techniques and how they are used in Tactus. For latency control:

- *Local computation* is supported by the Tactus server. The timing of presentations is handled locally by the server, and the server coordinates local computations including graphics rendering, video decompression, and audio synthesis.
- *Local buffering* is a key element in Tactus: Data is precomputed and sent to local buffers in the Tactus Server to reduce the timing variance due to network transmission.
- *Precomputation* is another key element in Tactus: In order to support interactivity, Tactus assumes that multimedia data may be computed ‘‘on the fly.’’ This data is precomputed slightly ahead of real time for synchronized presentation by the server.

- *Scheduling* is not emphasized by Tactus, but is necessary for proper operation. The clock tree dispatches wakeup messages in time-order, essentially implementing an earliest-deadline-first scheduling policy. The Tactus server assumes that it receives operating system support to run at high priority, thus low latency.
- *Faster Systems* will enable Tactus to support finer grained interaction, and systems that can perform real-time image and sound synthesis will lead to greater interaction than those that can only play back pre-authored video. A premise of Tactus is that the interactive computation and manipulation of media is a very important feature. Otherwise, Tactus could be greatly simplified to obtain a ''video player'' system.

For synchronization, we discussed the ideas of timing specification and the need for a master time reference. In Tactus, the timing specification is explicitly attached to data as timestamps. The ''master'' is time itself, promoting distributed processing and presentation, but requiring globally synchronized clocks.

Tactus is based on the assumption that network latency is an important consideration. As new technology pushes network data rates into the multigigabit range, network latencies will be dramatically reduced because network software, routers, and protocols can no longer afford to buffer large packets as is common now in local area networks. Will this make Tactus obsolete? We believe that low-latency networks may change the nature of Tactus, but latency has many sources, including application computation, disk rotation latency, frame lookahead in video decompression, and operating system overhead. Latency management and synchronization services will be important as long as latency is present in any form.

## 4. Summary and Conclusions

Interactive multimedia systems represent a convergence of traditional media with digital computing. The digital format makes it much easier for computers to manipulate the information, and this facilitates interactivity in which multimedia presentations can be adapted to individual needs.

Continuous media such as audio and video pose new challenges for computer and communication system designers because timing is critical to performance. When data is stored remotely, there is inherent latency associated with data access. This can lead to perceived delays and loss of synchronization; however, a variety of techniques exist to minimize these effects. For latency management, the techniques include local computation, local buffering, precomputation, scheduling, and faster systems. Synchronization is achieved by applying display rate corrections to maintain a correspondence between a specified frame or sample and the actual frame or sample, and this specification can be based upon either a ''master'' device or upon a real time clock. Latency management techniques help with synchronization problems by reducing the timing error (latency) of rate corrections.

Tactus integrates these techniques into a system that helps programmers create interactive multimedia. Tactus includes an extended graphical interface toolkit that helps programmers by providing support for time-driven as well as event-driven software. Tactus also includes a server that helps to overcome network latency and to provide accurate timing and synchronization in multimedia presentations.

A prototype Tactus server based on the Andrew Toolkit[8] and unix-like operating systems is running. Tactus is being ported to the Mach microkernel operating system[11], where we hope to achieve better real-time performance and continue development.

## 5. Acknowledgments

The work was sponsored by the IBM Corporation and was performed at the Information Technology Center at Carnegie Mellon University. I would also like to acknowledge Dean Rubine, Joe Newcomer, Tom Neuendorffer for their design work and contributions to the Tactus project. In addition, David Anderson, Jim Zelenka, and Kevin Goldsmith have worked on various aspects of the Tactus implementation. Finally, Ion Ratiu provided some helpful editing suggestions.

## References


1. Blattner, M. M. and R. B. Dannenberg, editors, *Multimedia Interface Design*, ACM Press, 1992.

2. Anderson, D. P. and R. Kuivila, ‘‘Accurately Timed Generation of Discrete Musical Events’’, *Computer Music Journal*, Vol. 10, No. 3, Fall 1986, pp. 48-56.

3. Anderson, D. P. and R. Kuivila, ‘‘A System for Computer Music Performance’’, *ACM Transactions on Computer Systems*, Vol. 8, No. 1, February 1990, pp. 56-82.

4. Rothstein, J., *MIDI: A Comprehensive Introduction*, A-R Editions, Madison, WI, 1992.

5. Anderson, D. P., R. Govindan, G. Homsy, ‘‘Abstractions for Continuous Media in a Network Window System’’, Tech. report UCB/CSD 90/596, Computer Science Division (EECS), U.C. at Berkeley, 1990.

6. Wayner, P., ‘‘Inside QUICKTIME’’, *Byte*, Vol. 16, December 1991, pp. 189.

7. Yager, T., ‘‘The Multimedia PC: High-powered Sight and Sound on Your Desk’’, *Byte*, Vol. 17, February 1992, pp. 217.

8. Palay, A. J., F. Hansen, M. Kazar, M. Sherman, M. Wadlow, T. Neuendorffer, Z. Stern, M. Bader, T. Peters, ‘‘The Andrew Toolkit - An Overview’’, *Proceedings USENIX Technical Conference*, USENIX, Winter 1988, pp. 9-21.

9. Kahn, K., ‘‘DIRECTOR Guide’’, Tech. report MIT AI Laboratory Memo 482B, MIT, December 1979.

10. Gibbs, S., ‘‘Composite Multimedia and Active Objects’’, *OOPSLA ’91 Conference Proceedings*, A. Paepcke,ed., ACM Press, ACM/SIGPLAN, New York, 1991, pp. 97-112.

11. Accetta, M., R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, M. Young, ‘‘Mach: A New Kernel Foundation for UNIX Development’’, *Proc. of Summer Usenix*, Usenix, July 1986.