Multimedia document retrieval system using partially ordered/ partially reliable transport service

Phillip T. Conrad, Edward Golden, Paul D. Amer, Rahmi Marasli

Phillip T. Conrad, Edward Golden, Paul D. Amer, Rahmi Marasli, "Multimedia document retrieval system using partially ordered/partially reliable transport service," Proc. SPIE 2667, Multimedia Computing and Networking 1996, (25 March 1996); doi: 10.1117/12.235867

SPIE.

Event: Electronic Imaging: Science and Technology, 1996, San Jose, CA, United States

A multimedia document retrieval system using partially-ordered/partially-reliable transport service

Phillip T. Conrad
Edward Golden
Paul D. Amer
Rahmi Marasli

Computer and Information Science Department
University of Delaware, Newark, DE 19716 USA
Email: {pconrad,golden,amer,marasli}@cis.udel.edu

ABSTRACT

We investigate the benefits of using a partially-ordered/partially-reliable (PO/PR) transport service for multimedia document retrieval over the Internet by implementing a prototype system. We introduce PMSL, a document specification language combining aspects of Little and Ghafoor's OCPN with features for graceful degradation. We then show that for retrieval of PMSL documents, our PO/PR transport protocol (POCv2) provides several benefits over traditional protocols such as TCP or UDP. First, POCv2 provides mechanisms to achieve the reliability and order requirements of individual multimedia objects as specified by a document author. Second, when network conditions are poor (e.g., high loss rates), POCv2 provides for graceful degradation. Finally, POCv2 simplifies application development by providing appropriate mechanisms for synchronization.

Keywords: transport protocols, quality of service, multimedia synchronization, multimedia networking, retransmission, partial order, partial reliability, multimedia authoring.

1 INTRODUCTION

Many systems now exist that allow authors to construct pre-orchestrated multimedia documents. These documents consist of objects such as still images, text, audio clips and video clips, which are pre-arranged according to some temporal scenario. Various schemes exist for expressing these temporal scenarios.[21] During playback of such a document, object presentation proceeds according to this temporal scenario until some event occurs which stops or resets it–for example, a user interaction point is reached, or a user presses a "pause" button. Typically, a multimedia workstation with sufficient processing capacity (e.g., CPU, memory) can present a document in compliance with its temporal scenario, provided that the channel delivering the information is ordered and error-free. However, suppose the document is stored on a remote file server, and the channel delivering this information is a network connection. In this case, network errors may wreak havoc with attempts to present the document correctly. We propose that in such situations, it is appropriate to provide for "graceful degradation" of the multimedia document presentation. This recognizes that in many multimedia documents, not all objects have equal importance, or the same quality-of-service requirements; some objects are essential to document content, while others are "nice to have," but completely optional.

Given that we want to provide for graceful degradation, what transport protocol should be used for multimedia objects? In this paper, we argue that classic transport services such as TCP and UDP are suboptimal for this application. We propose an alternative: partially-ordered/partially-reliable (PO/PR) service. We are developing a prototype system for authoring multimedia documents, and retrieving and displaying them across the Internet. Our goals in building this system are: (1) to show how a PO/PR transport service facilitates coarse-grained synchronization of multimedia objects and graceful degradation during times of network stress, (2) to demonstrate the mechanisms needed to implement a PO/PR transport protocol in practice, and (3) to demonstrate and quantify performance improvements when a PO/PR transport service is used instead of an ordered/reliable service (e.g., TCP) or an unordered/unreliable service (e.g., UDP). For this paper, we focus on (1) and (2).

The paper is structured as follows: Section 2 discusses PO/PR service in some detail, including motivation for PO/PR service, and applications to synchronization and graceful degradation of multimedia documents. Section 3

0-8194-2041-7/96/$6.00

describes a multimedia document retrieval system that will be used as a "proof-of-concept" vehicle for PO/PR service. This section includes an overview of Prototype Multimedia Specification Language (PMSL), comparing and contrasting its synchronization model with that of Little and Ghafoors's OCPN. Also included is a description of the design of our prototype client and server, highlighting the ways in which the features of a PO/PR service are applied. We follow this with a summary and discussion of future work on POCv2 in Section 4, and an overview of related work in Section 5.

2 PARTIALLY-ORDERED/PARTIALLY-RELIABLE TRANSPORT SERVICE

2.1 Motivation

The transport layer is the lowest layer responsible for end-to-end quality-of-service (QoS). Typical functions of the transport layer include (1) recovery from data loss, (2) detecting and removing duplicates, (3) resequencing out-of-order data, and (4) flow control/congestion avoidance. The level of service provided by each of these functions is a QoS parameter of the transport layer.

Some authors lump three QoS parameters (loss, order, and duplication) together under the term "reliability", using "reliable" to refer to a transport service where no messages are lost, duplicated, or delivered in a different order from that in which they were transmitted. To classify transport QoS with greater precision, we restrict "reliability" to refer only to data loss; defining a *reliable* service as one that allows no loss whatsoever. In this way, reliability becomes orthogonal to the service features of order and duplication. "Order" refers to whether or not the sequence in which objects are sent is preserved in delivery. Thus, an *ordered* service delivers objects in the same order as presented to that service, while an *unordered* service makes no guarantees about order. "Duplication" refers to whether or not multiple copies of the same object may be delivered. A *no-duplicates* service detects and discards any duplicates, while a *duplicates* service does not detect duplicates.

One fundamental design problem at the transport layer is making appropriate tradeoffs between the four *qualitative* QoS parameters mentioned above, and other *quantitative* QoS parameters, such as delay and throughput. This is because transport services are often called upon to provide a QoS that is an enhancement of the underlying network; however, enhancing one QoS parameter usually involves a tradeoff with another. For example, TCP provides a reliable service on top of an unreliable network protocol[a] by means of retransmissions, but does so at the expense of introducing additional end-to-end delay. The selection of which transport mechanisms are appropriate for a given application is often a matter of considerable debate. For example, while the prevailing view is that retransmission is inappropriate for multimedia data because of its real-time nature,[14] some authors[9] describe circumstances in which it is appropriate.

Commonly, transport protocol selection is a choice between extremes. For example, in the Internet protocol suite we note that the service provided by *Transmission Control Protocol* (TCP)[24] is ordered, reliable, no-duplicates and flow-controlled, while the service provided by *Unit Datagram Protocol* (UDP)[25] is unordered, unreliable, duplicates, not-flow-controlled. The fact that these protocols provide service at the extremes of each of these four QoS axes creates a dilemma for the designer of an application whose needs reside in the middle. When designing an application that will run over Internet protocols, today's implementer typically has three choices: (1) use TCP, (2) use UDP, or (3) build the needed transport functionality as part of the application development effort. As explained below, choosing either TCP or UDP when neither is appropriate has negative consequences. However, correctly designing and implementing an application specific transport protocol that correctly handles retransmission and flow-control may be a larger effort than designing and implementing the application that sits on top of it!

What is desirable is a standardized transport service (e.g. a library of routines) that applications could utilize to gain flexible control over the ordering and reliability of individual objects. This would allow such applications to achieve an appropriate balance among various QoS parameters without having to "reinvent the transport-layer wheel" with every application. Such an approach is consistent with Application Layer Framing as proposed by Clark and Tennenhouse.[6] In this paper we describe work underway to implement such a service. Our protocol, *Partial Order Connection, version 2* (POCv2) provides a *partially-ordered/partially-reliable* (PO/PR) transport service. In addition to providing the precise level of reliability and ordering required, POCv2 provides an additional benefit that neither TCP nor UDP provides: a mechanism that facilitates coarse-grained synchronization of multimedia objects.

[a] namely, Internet Protocol, or IP

The remainder of this section expands on the themes described above. Section 2.2 describes some of the disadvantages of the currently available options at the transport layer (i.e., TCP, UDP, implementing your own). Section 2.3 describes how a PO/PR service can overcome these disadvantages. Sections 2.4 through 2.6 describe additional benefits that accrue from using a PO/PR service; namely synchronization support and graceful degradation. Finally, Section 2.7 provides some service and implementation details concerning POCv2.

2.2 Problems with current protocol options

In a protocol environment where only the extremes of transport service are provided some applications cannot find a perfect home. Consider the retrieval of objects that are part of a multimedia presentation. For some objects, no loss is permissible (e.g., text, some still images, control information) while for others, some loss is permissible (e.g., audio and video streams, images that are merely decorative). Also, the order of presentation of objects may be defined by a partial order rather than a total order, as is the case for document synchronization requirements described by an Object Composition Petri Net (OCPN).[15] We describe the service required by such an application as *partially-ordered/partially-reliable*. Partially-reliable refers to the notion that some objects must be delivered reliably, while for others loss may be permitted. Partially-ordered refers to the fact data sequencing requirements are expressed as a partial order rather than as a linear order.

For such an application, TCP provides more reliability and resequencing than is necessary at the expense of extra delay and reduced throughput. Extra delay may result in annoying discontinuities in the playback of continuous media such as audio or video data. However, TCP has the advantage of providing flow control and congestion control algorithms that have been tested for over a decade, and scale well to a global internet.

UDP, on the other hand, provides a "best effort" service with no guarantees whatsoever. On a lightly-loaded LAN where the underlying network is inherently fairly reliable, this may be perfectly acceptable, but over longer Internet distances, loss and disorder are fairly common. Our anecdotal experience with the MBONE tools for internet video-conferencing (i.e., nv, vat) suggests that sustained packet drop rates between 7-15% are common, and that drop rates as high as 50% occasionally occur. Some argue that since such loss rates are commonly due to buffer overflows, bandwidth reservation and improved congestion control methods will eliminate this problem.[29] Our sense is that in spite of excellent research efforts in this area, there will always be cases where reservation mechanisms are not feasible to implement, or fail to provide the necessary QoS (e.g., a disaster situation or battlefield scenario involving intermittent jamming). In these cases the loss rate of the underlying network may be higher than the application's tolerance for loss. Furthermore, because UDP is not flow-controlled, unless the application implements its own flow-control mechanism, an application using UDP may flood the network and/or the receiver with packets at a rate faster than either can handle; this creates another source of packet loss.

A third choice for the application designer is to build the needed transport functionalities from scratch, perhaps as an application specific transport protocol layered on top of UDP. This way, the application designer can choose exactly what features to implement based on the requirements of the application. However, implementing transport protocol features at user layer is non-trivial. Two issues are particularly difficult: (1) managing the context switching between asynchronous protocol events such as timer expiration and packet arrival and the rest of the application code, and (2) getting flow-control/congestion-avoidance right. In the latter case, to truly test the correctness of the design and implementation requires simulation and/or implementation on a wide scale. The complexity of designing and testing the operation of retransmission timers, resequencing of data, buffers, round-trip-delay estimation, and flow-control/congestion avoidance may exceed the complexity the application itself!

We argue that given a reasonable alternative, application designers would prefer to avoid programming transport layer functionality. Therefore we present an alternative: a standardized transport service providing the flexibility to specify reliability, ordering, flow-control, and duplication at a finer granularity than either TCP or UDP. This would allow application designers to focus their efforts on their application.

2.3 An alternative: partially-ordered/partially-reliable service

Partially-reliable services are those which provide something between reliable service (no-loss) and unreliable service (no guarantees). Dempsey[9] and Gong et al.[11] describe partially-reliable services based a controlled number of retransmissions. Our approach to partially-reliable service combines this idea with the notion of *reliability classes*: the idea that an application designates individual objects (messages) as needing different levels of reliability. Reliability classes are assigned at the object level. By specifying the reliability class "RELIABLE" for a

particular object, the application indicates to the transport layer that that object should be retransmitted an unlimited number of times until it gets to the receiver successfully (or the connection is dropped). By contrast, the application can indicate via the reliability class "UNRELIABLE" that an object should be transmitted only once and never retransmitted, or by the reliability class "PARTIALLY_RELIABLE" that an object should be retransmitted, but only up to a point. (The class "PARTIALLY_RELIABLE" is described in more detail in Section 2.5.) In addition to partial reliability, we propose *partially-ordered* service. An application using partially-ordered service defines a partial order (PO) over the objects to be communicated and provides a representation of this PO to the transport layer. Objects submitted by the sending application may then be delivered to the receiving application in any delivery order that is a *linear extension* of the PO.

Partially-ordered/partially-reliable service was first proposed in [Amer et. al 1994].[2] In that paper, the authors incorporate both partial order and partial reliability into a single transport protocol called POC (Partial Order Connection). Simulation and analysis results based on models of POC have shown that applications using POC can make QoS tradeoffs between {order, reliability} and {throughput, delay, buffer utilization}.[16–18] This first version of POC was conceived as an abstract protocol to illustrate the concept of PO/PR service. However, it lacked certain features needed for practical implementation.

RFC1693 describes an attempt to incorporate partial order and partial reliability into TCP.[7] Modifying TCP to provide PO/PR service presents several difficulties. First, partial order and partial reliability are defined in terms of objects (messages), while TCP is fundamentally a byte-stream protocol. Imposing a message framework on TCP alters one of its foundation principles and proves to be cumbersome. Second, the flow control algorithms in TCP are designed based on the assumption of a linear order. Our work with models of POC leads us to believe that using these algorithms with a partially ordered service severely restricts the potential performance that partial order and partial reliability may offer. Thirdly the need for backward compatibility with TCP makes the design unnecessarily complicated.

Therefore, the authors are instead pursuing the development of a new standalone version of POC (POC version 2, or POCv2) designed to make the benefits of partial order and partial reliability available to application developers. Our overall goal in developing POCv2 is to evaluate whether the theoretical advantages of PO/PR services claimed in previous work can be achieved in practice.

2.4 Coarse-grained synchronization via "explicit release"

Another benefit of a partially-ordered/partially-reliable service is support for coarse-grained synchronization of objects.[b] This is provided in POCv2 via a feature called *explicit release.*

POCv2 uses a transitively reduced precedence graph to represent the partial order; therefore, throughout this paper the terms *predecessor and successor* should be understood to mean *immediate* predecessors and successors. Predecessors of object k are all objects $i : i \prec k, \neg\exists j, i \prec j \prec k$. The definition of "successors" is similar.

The transport receiver normally considers an object *deliverable* when each of its predecessors has been delivered. It keeps track of this by keeping a count of each object's undelivered predecessors. Normally, when an object is delivered, this count is immediately decremented at each of the object's successors; this operation is called "releasing successors". However, when explicit release is used, the transport receiver does not immediately release the successors of delivered objects; instead it requests that the application signal the transport layer when the successors of that object should be released.

By way of motivation, consider a document in which an image is to appear immediately following the completion of an audio clip, and therefore follows it immediately in the partial order. Suppose that as soon as the audio clip is delivered, the application submits it to the audio device for playback. Without an explicit release mechanism, the image becomes deliverable immediately upon delivery of the audio clip. The receiving application may then read the image and present it before the playback of the audio is complete, in spite of the fact that the image was intended to *follow* the audio clip in the presentation order. Note that the application cannot simply wait until the audio clip is complete to read the next object from the transport layer, since the next object *might* be something that is to be presented in parallel with the audio clip. Thus, unless the client application is aware of the partial order and implements its own functionality to preserve synchronization relationships, such

[b] Coarse grained synchronization (also called *temporal alignment*) refers to synchronizing the start and end of objects with respect to one another. This can be distinguished from fine-grained synchronization (also called *stream synchronization*) which refers to keeping parallel streams synchronized with one other (as in lip-sync, for example).[19,27]

relationships may be violated; the client cannot simply read objects from the transport layer and display them as they become available.

By contrast, when explicit release is used, the application explicitly signals the transport layer when the audio playout is complete, thereby releasing the successors of the audio clip for delivery. By waiting until the audio clip is finished to explicitly signal the release of the audio clip's successors in the partial order, it is guaranteed that they will not be released prematurely.

Being able to rely on the explicit release feature for coarse-grained object synchronization removes yet another burden from the application programmer. The code for a multimedia client can be made much simpler if the client can rely on the transport layer to provide basic synchronization support. Because the exchange of the partial order between sender and receiver happens entirely within the transport layer, a multimedia document retrieval *client* using POCv2 can present a document in compliance with a partial order synchronization specification *without having to know what the partial order is.* Instead, the server simply reads the partial order as part of the document specification, and requests that the transport layer use it for object delivery. The client's only responsibility is to signal explicit release when the playback of objects is complete. In Section 3.2 we describe how this mechanism is used in our prototype document retrieval system.

2.5 Graceful Degradation

Integrating coarse-grained synchronization with partial reliability has an additional benefit in terms of graceful degradation. In particular, we can define a "PARTIALLY_RELIABLE" reliability class that has the following semantics:

- The sender will retransmit the object up until the time that it is "declared lost"
- The receiver will declare it lost only at the last possible moment, i.e.:
  - when there is no deliverable data, and
  - when declaring this item lost (and possible other partially-reliable or unreliable successors) will make some data that has arrived deliverable.

By integrating the synchronization mechanism with the decision whether and when to declare an object lost, we can be more conservative about throwing objects away. This is particularly true when a presentation contains objects which have a playout duration that is considerably longer than their transmission time; e.g., pause objects which cause a presentations flow to suspend for some period of time, or user interactions that suspend a presentation until the user presses a key or clicks a mouse.

Consider, for example, a sequence of audio following an object that requires the user to click a button in order to continue. Depending on how long the user takes to click the button, there may be no time to retransmit a lost audio object, or there may be time for many retransmissions. By integrating the processing that synchronizes and schedules object presentation with the decision as to when to declare an object lost, we get the best of both worlds: we can take advantage of slack time to do retransmission of lost objects, and improve presentation quality. On the other hand, when a user demands fast response, we can provide it by dropping objects rather than incurring any delay introduced by doing retransmissions.

2.6 Graceful Degradation: Example[c]

Consider a system where multimedia documents are being retrieved from a remote server and displayed in real-time as the contents arrive at a user's workstation. Suppose we desire to retrieve and display a document which describes three routine maintenance procedures that should be performed on a car: checking the oil, adding oil, and changing the air filter. Figure 1 illustrates the objects that might be present in such a document, as well as the sequence in which they might be transmitted over the network for display.

First, four graphic objects are placed on the screen: a picture of the engine (object A, which is critical to the presentation content), along with three objects (B,C,D) that are merely decorative. Next, an audio sequence plays out (E,G,I,K) that describes the three maintenance procedures. Synchronized with this audio, arrows pointing to the parts of the engine affected (F1,H1,J1) appear at the appropriate times, along with text labels that describe the engine parts (F2,H2,J2). Finally, a "Press to Continue" appears (L).

[c]This section appeared previously in a position paper by three of the authors.[8]

A plan for graceful degradation of this document is represented by Figure 2. This diagram illustrates a partial order for this document, represented as a precedence graph. The reliability class of each object is represented by the different colors of circles. **Black** circles represent `RELIABLE` objects; these objects may not be lost and will be retransmitted until they arrive. For each of these objects, if necessary, the presentation will be stopped until the critical object arrives and all predecessors have been delivered or declared lost. In the given example, the "critical" diagram of the engine is such an object, as are the arrows pointing to the various engine parts. It is assumed that the points in the audio track that are separated by these arrow objects represent places where a brief pause for retransmission of an object would be acceptable (e.g., a pause between complete sentences.) **White** circles represent `UNRELIABLE` objects. These objects will be transmitted on a best effort basis, and the presentation will not stop if they do not arrive. If they have not arrived by the time they are scheduled for presentation, the receiver simply declares them lost. This class includes the individual 1/50 second audio objects and the decorative graphics. **Gray** circles represent `PARTIALLY_RELIABLE` objects. These objects are valuable for some period of time, but subsequently are of no value; furthermore, they are not important enough to hold up the presentation if they do not arrive on time. The text labels at the ends of the arrows fall into this class, presumably because they are redundant—the audio track will also identify the engine parts as they are indicated by the synchronized arrows. We would like to retransmit them any time up until the user presses the button (L); after that, they are useless and should be dropped.

Consider what may happen to this presentation using three transport layer alternatives: UDP, TCP, and a PO/PR transport service. We Assume that a throughput of 128kbps is available, so there is clearly sufficient bandwidth to send the document and display it in real time. Suppose that the underlying network has a non-negligible loss rate such that it is expected that at least one network packet will be lost during the presentation.

Under UDP, we run the risk that certain critical objects (e.g., the picture of the engine, or the button that allows the user to continue) may be lost. In this case, the user of the program is likely to experience severe frustration. Unless the application provides partial reliability itself, graceful degradation is not possible: UDP has no features to support it.

Under TCP, we will retransmit any missing objects, even if they are objects for which loss may be tolerated. Furthermore, because the service is an ordered stream, any objects following the lost object will be delayed until the retransmission is successful. Suppose that one of the objects in the middle of one of the audio sequences (e.g., E250) is lost. In this case, there will be a gap in the playback of this audio—in the middle of a sentence, or possibly even in the middle of a word—while packet E250 is retransmitted. Our measurements of TCP retransmissions on wide-area Internet routes indicate that this gap could be as long as half a second.

With a PO/PR service, however, we have the flexibility to meet our goals. If any black object is lost, the presentation schedule will stop until the object can be retransmitted. Since this occurs only at logical stopping points in the audio track, and only for essential objects, the effect of this on the user is minimal. If audio objects (white) are lost, each one is sufficiently small that the user is unlikely to mind the loss in audio quality; furthermore, since they are loseable, the presentation will not grind to a halt while a retransmission takes place. Loss of a "loseable" object will not delay delivery of other objects. Finally, the loss of a gray object will not delay subsequent audio objects; in fact, assuming there is adequate idle bandwidth, a lost gray object can likely be retransmitted in time to make an appearance while it is still useful.

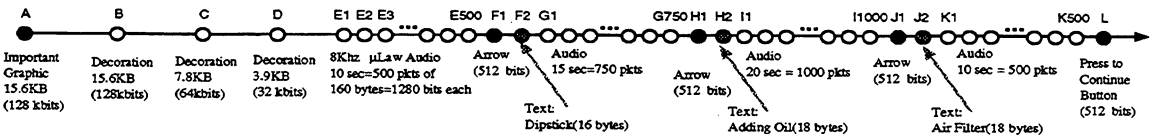


Figure 1: Example document as a stream

2.7 An overview of Partial Order Connection, version 2 (POCv2)

POCv2 is a connection-oriented, full-duplex, message-based transport service, which can be used by a pair of applications to exchange data objects according to a partial order and the reliability requirements of the individual objects.

For each direction of the full-duplex connection, the application defines a partial order over some set of objects

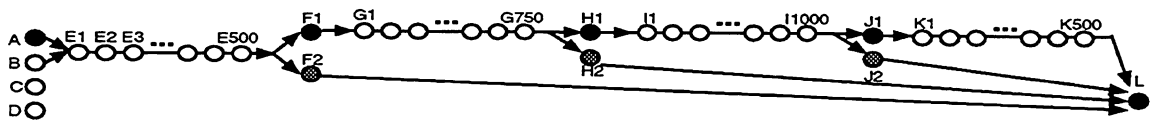


Figure 2: Example document as a partial order

to be communicated. This set is referred to as a *period.* In addition, the application defines the reliability class of each object in the period. The application communicates the partial order and the reliability class of each object through a structure called a *service profile.* At the end of each period, the current service profile can be reused, or a new one can be defined by the application. A set of sequential periods sharing a single service profile is called an *epoch.*

Associated with each object is an ordered pair $\langle period, objectNum \rangle$ called an *object identifier*, or *oid.* The $objectNum$ is a value between $0..n-1$ that uniquely identifies the object within the period numbered by *period.* Periods are strictly ordered; that is, at the sender, all objects for period i must be submitted before any objects from period $i+1$, and likewise at the receiver all objects for period i must be delivered before any objects from period $i+1$.

There are two types of objects in POCv2: block and stream. Block objects are treated as atomic; they cross the boundary between the application and transport layers as a single unit and are either presented in their entirety or not at all. Stream objects provide a convenient means of representing long chains of sequential data: for example, streams of audio or video packets. Rather than having to represent each audio packet or video frame as an individual object, a stream object can be used. A stream object consists of a sequence of totally ordered *cells.* All cells which make up a stream object have the same *objectNum*; however they are assigned sequential *cellNum*'s. At the receiver, the delivery of the individual cells is strictly ordered, however their delivery may be interleaved with the delivery of other objects (or cells of other stream objects) that are in parallel with the stream object in the PO.

The POCv2 architecture divides the transport layer into three sub-layers, two of which are proposed to be implemented at user layer as an application library, and one of which is proposed to be implemented in the Unix Kernel (see Figure 3.B.) The top layer provides the partial ordering mechanism. The middle layer provides for object segmentation and reassembly; this layer divides objects into segments based on the Path Maximum Transfer Unit (MTU) of the connection. The lowest layer is an unordered/partially-reliable segment delivery service; this layer is responsible for retransmission and flow control. We are currently developing this system for Solaris 2.4 and SunOS 4.1.3. For the time being, we are implementing the lower layer at user level; with the intention of moving this layer into the kernel at a later stage if this would significantly improve performance.

3 PROTOTYPE MULTIMEDIA DOCUMENT AUTHORING AND RETRIEVAL SYSTEM

To test the idea of applying partial order and partial reliability to multimedia synchronization and graceful degradation we have designed a test system. Figure 3.A. shows the architecture of this system, which consists of:

- a Prototype Multimedia Specification Language (PMSL) that allows authors to express document synchronization and reliability requirements in a way that allows for graceful degradation,
- a client/server system for retrieval of PMSL documents.

The interaction between client and server is similar to that between Web browsers and HTTP servers. When the user specifies the URL of a document; the client makes a connection to the server, which in turn transmits the elements of the document. Our prototype client and server are designed with POCv2 features in mind and written to the POCv2 application programming interface. We have also written an adaptation layer that allows us to run these applications directly over TCP or UDP. This allows us to make comparisons among the three transport protocols.

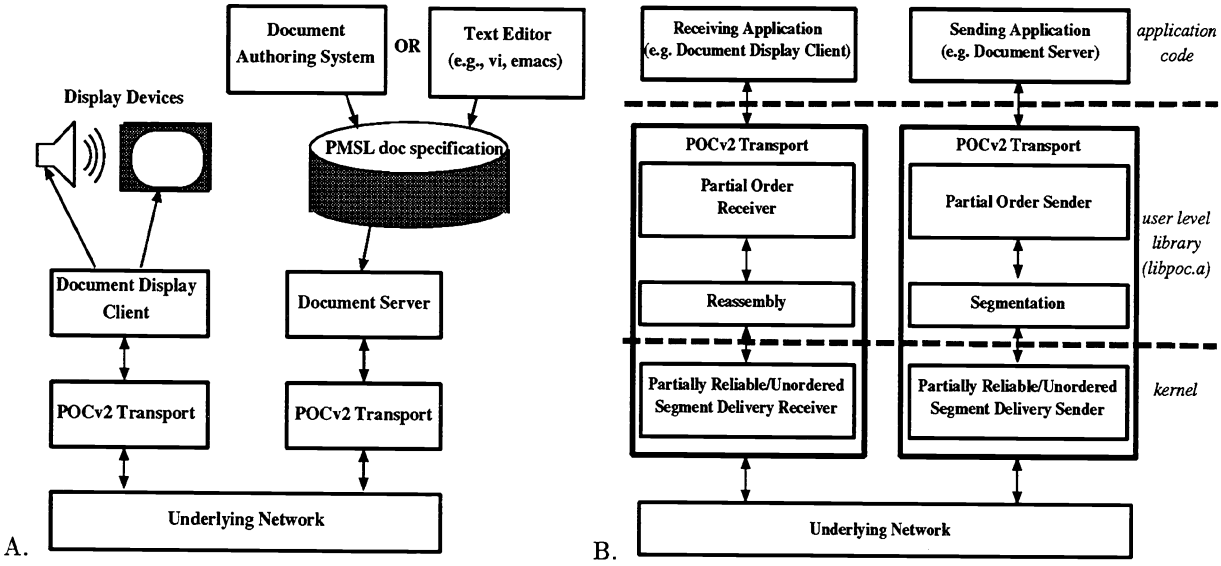


Figure 3: (A) Doc Retrieval System Architecture (B) POCv2 Architecture

3.1 Prototype Multimedia Specification Language (PMSL)

The purpose of PMSL is only to serve as a vehicle for experimenting with applying PO/PR protocols to multimedia document retrieval. Choosing to call it a "prototype" is quite deliberate, for we have consciously chosen to keep it very simple. We hope to demonstrate through simple experimentation with PMSL that the concepts of PO/PR are worth incorporating into more elaborate and powerful multimedia authoring systems. Our future directions for PMSL are to investigate ways to express the concepts of the PMSL model within the framework of existing and emerging standards such as HTML and HyTime,[3,10] and to consider ways of incorporating the PO/PR concepts in PMSL into graphical authoring tools.

For reasons of space we do not present a complete syntax and semantics for PMSL, but rather sketch the main ideas. A PMSL document specification consists is an ASCII description of a multimedia document. It consists of a list of COLORDEF's, FONTDEF's, and ELEMENTS. Each COLORDEF or FONTDEF defines a color or font that can be used later in the document to specify the attributes of graphic elements; these are transmitted to the client at the beginning of document transmission. ELEMENTS are objects which comprise the presentation itself–for example: audio clips, video clips, fixed objects (images, text, geometric shapes), pauses, user interactions (e.g., buttons), and erase events (which remove fixed objects from the screen). ELEMENTS are organized into a partial order which defines the presentation synchronization and ordering requirements. In addition, PMSL allows the author to define a reliability class for each object.

The syntax of the ELEMENT definition is shown in Figure 4. An element definition starts with the keyword ELEMENT followed by its unique elementID. The successor list, which may be empty, consists of the elementID's of elements which should follow that element in the partial order (PO).[d] The successor list and reliability class are followed by the element type. HOTSPOTS are screen regions where a user can click to cause some interaction to occur; for example, they can be used to define a "click here to continue" button that pauses the presentation until the users clicks, or to provide links to other documents. PAUSE elements indicate a pause of some fixed duration in the presentation (along that "branch" of the partial order). The other element types are self-explanatory.

The semantics of the PMSL model are similar to those of Object Composition Petri Net (OCPN).[13] In both OCPN and PMSL, objects can be composed in sequence and parallel to express synchronization relationships. A basic idea in both models is that no object should be presented until all of its predecessors have been presented. In PMSL, a partial order is used to represent this fact, while OCPN uses a petri-net representation. Given an

[d]Note that it is only necessary to specify elements which follow immediately in the PO. It is not necessary to specify elements which follow in the PO via transitive relationships; if these are specified, they are essentially ignored, since the parser computes the transitive reduction of the PO before using it to set the POCv2 service profile.

```
<Element> ::= "ELEMENT" <ElementID> {<Successors>} <ReliabilityClass> <ElementData>
<Successors> ::= ":" <SuccessorList>
<SuccessorList> ::=   <ElementID> {","<SuccessorList>}
<ElementData> ::=  AUDIO ... | GRAPHIC ... | ERASE ... | PAUSE ... | HOTSPOT ... | NULL | END
```

Figure 4: Syntax of Element Definition. Ellipsis ("...") indicates elements in the syntax not fully specified in this example. Braces ("{}") indicate optional syntax elements.

OCPN, we can construct an equivalent PMSL representation by constructing the partial order that corresponds to the petri-net, and setting the reliability class to RELIABLE; an algorithm to do this appears in Figure 5. Therefore, by the results in [Little and Ghafoor 1990],[13] PMSL can represent all thirteen temporal intervals between any two elements.

PMSL differs from OCPN in two significant ways. First, while both OCPN and PMSL objects have a duration, in PMSL, this duration may be indeterminate (for example, an audio clip whose length is unknown). This feature allows PMSL to express interactivity; for example, the duration of an object may be the length of time between initial presentation of a button, and the time the user clicks on the button. Secondly, PMSL allows the document author to specify the transport reliability for each object using POCv2 reliability classes. In this way, PMSL allows for the possibility that some objects may be lost, whereas OCPN assumes that all objects will be presented.

- **for** each place t_i
  create a PMSL element with elementID i, reliability class RELIABLE.
- to construct the partial order P:
  for each transition t_i
    for each arc (p_j, t_i)
      for each arc (t_i, p_k)
        add a relation $j \prec k$ to P.

Figure 5: Algorithm to convert an OCPN into a PMSL document specification

3.2 Prototype PMSL server and client

The server accepts connection requests from document display clients. When the server accepts a connection, it looks for a request for a specific PMSL document. It reads the requested document from disk and transmits it in encoded form across the transport connection.[e] Audio objects are sent using POCv2 stream objects consisting of audio cells of 1/50th of a second each; other objects are sent as block objects. The server uses two POCv2 epochs; one for the fontdefs/colordefs (sent reliably, but unordered), and another for the document itself (sent with a service profile that matches the PMSL spec.) To order the objects and cells within each period for transmission, the server constructs a graph representation of the partial order and uses a breadth first traversal.

The fact that fontdefs and colordefs form a separate period ensure that all of these resources arrive at the beginning of the transmission; this avoids delays during the presentation caused by loading fonts or colors. However the fact that they are unordered within the period provides an advantage; if one is lost on its original transmission, the client can proceed with processing the others while the lost one is retransmitted. This is an application of a fundamental principle of Clark and Tennenhouse's Application Layer Framing concept[6]: processing application layer data units out-of-order whenever possible.

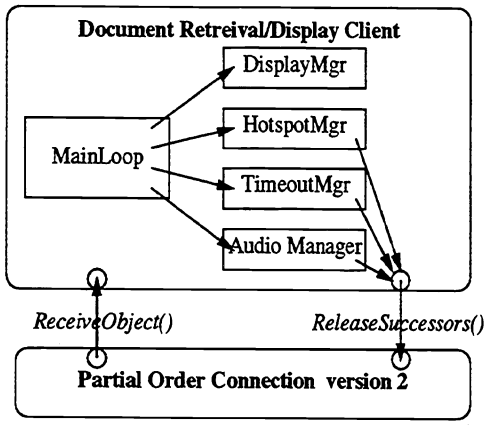
The client allows a user to connect to the server and request a document. The client then presents the document elements as they are delivered by the transport connection. One of our design goals was to take advantage of the synchronization features of POCv2 in order to keep the code in the client as simple as possible. The client consists of user interface code (which we do not consider further in this paper) and presentation display code. The structure of the presentation display code is illustrated in Figure 6. It consists of a main "read and display" loop, and four submodules. The pseudocode of this main loop is illustrated in Figure 6.

The main loop code submits the oid of the object along with each call to one of the four manager routines. The audio manager, timeout manager and hotspot manager all provide a callback service when respectively, an audio

[e] The encoding is done using a Prototype Multimedia Transfer Protocol (PMTP) designed for encoding PMSL documents and managing PMSL document requests and responses. We omit a discussion of PMTP for reasons of space.

element reaches the end of its playback, a timeout for a pause event expires, or a user interaction is complete (e.g., a user clicking on a "continue" button.) The purpose of this callback service is to make the *ReleaseSuccessors(oid)* call; this ensures that object synchronization is maintained. For our prototype system GRAPHIC and ERASE elements do not need a callback; since they are assumed to be presented instantaneously, there is no need to use explicit release. This may be changed in future versions to provide support for presentation of a graphic element that requires a significant duration of time to display (e.g., a large interleaved or compressed graphic that might take several seconds to present in its final form.)

Note that the client has no need to have any knowledge of the order and reliability QoS expressed by the author, nor to manage retransmission or flow control; this is maintained by the POCv2 service. In this way, POCv2 simplifies the task of coding a multimedia document retrieval application.



A.

```
while not end of file
    ReceiveObject
    switch (object.type)
        case audio:
            submit to the audio manager
        case graphic:
        case erase of a graphic:
            submit to the display manager
        case hotspot:
        case erase of a hotspot:
            submit to the hotspot manager
        case pause:
            schedule a timeout event
        case null:
        case end:
            do nothing
```

B.

Figure 6: The Document Retrieval Client: (A) Diagram (B) Main Loop Pseudocode

4 SUMMARY AND FUTURE WORK

We have shown how a partially-ordered/partially-reliable service provides a better match with the QoS requirements of multimedia document retrieval than either ordered/reliable service (e.g., TCP) or unordered/unreliable service (e.g., UDP). We have shown how the PO/PR service provided by our protocol, POCv2, provides support for coarse-grained synchronization through the explicit release feature, and explained how integrating this support with the PO/PR transport can provide support for graceful degradation. Finally, we have presented the architecture of a multimedia document retrieval system where the client need not include any transport functionality or complex synchronization code; the underlying POCv2 service provides enough functionality that a simple "read and display" loop suffices.

Our study of POCv2 is ongoing; our next focus is a comparison of performance metrics such as delay and throughput for POCv2 vs. TCP and UDP at various levels of loss. As part of this study, we also plan to further investigate various flow control/congestion avoidance algorithms for POCv2, since this will make performance comparisons with TCP more meaningful.

We are also considering the usefulness of additional reliability classes for POCv2. For example, while POCv2 currently has a `UNRELIABLE` class which indicates that a single transmission of a packet should be made but no retransmissions, it may be desirable to add another class which has these semantics: never retransmit this object, and do not even transmit it *once* if the sender determines the bandwidth is severely restricted. The sender could make this determination based on feedback from the receiver. Such a reliability class would be particularly useful during times of severe network service degradation. Another feature that may be desirable for supporting MPEG video streams is the ability to specify different reliability classes for different cells in a stream; this way, I, P and B frames could potentially be assigned to different reliability classes as appropriate.

A third area for further study is the authoring of documents for graceful degradation. We have argued that graceful degradation is feasible provided that the author allows for this possibility. However, we acknowledge that document authors are likely to demand more convenient and intuitive means of specifying synchronization and reliability requirements than a text-based language such as PMSL. Furthermore, many document authors may be

accustomed to thinking in terms of a system that can deliver documents flawlessly; thinking in terms of graceful degradation may be a new and unfamilar concept. This creates the need for an authoring tool with an interface that facilitates and promotes the specification of documents in a way that allows for graceful degradation.

5 RELATED WORK

Other authors have considered theoretical consequences of channel ordering, or lack thereof, in the context of designing and verifying distributed algorithms.[12,20] Ahuja et. al. showed that some conclusions derived on the design of distributed algorithms need not have required FIFO ordering as a base assumption.[1] Ahuja et. al. propose a mechanism called *flush channels*, which provides a restricted form of partially ordered service. However Ahuja's flush channels assume a reliable service, and unlike POC, restrict the partial orders that can be specified to those that can be represented by the four message types provided in Flush Channels; this differs from POC service, which allows any partial order to be used, and allows for partial reliability. Peterson et. al.[22] define a partial order on the messages communicated by a set of distributed processes and implements a protocol *Psync* that encodes the partial ordering within each message. The partial order is defined by the interleaved times that messages are sent and received in the shared message space of the multiple communicating processes and dynamically changes with each newly sent message. The work of Birman et. al. on ISIS proceeds along a similar line.[4] Our work differs in its assumption of a *point-to-point* connection, rather than multipoint-to-multipoint communication. In general, our assessment of the previous work on partial order communication that we are aware of is that the focus is on the correctness and coherence of distributed systems. This is fundamentally different from our emphasis on tradeoffs among QoS parameters for a single point-to-point transport connection.

Other research has considered adding more flexibility for point-to-point transport layer connections. The Versatile Message Transaction Protocol (VMTP, defined in RFC1045) is a transaction-oriented transport protocol which provides a rich selection of security, real-time, asynchronous message exchange, streaming, multicast and idempotency features.[5] The MultiStream Protocol (MSP) provides seven kinds of service (STREAM1 thru STREAM7) with varying characteristics regarding reliability and ordering.[23] However, neither of these protocols provides a partial order service. Also, both protocols continue to view reliability as an "either/or" situation; that is, either "reliable" or "unreliable" service is provided.

The notion of adding transport QoS to document specification schemes has been addressed by Woo, Qazi and Ghafoor,[28] and by Rody and Karmouch in the MEDIADOC/MEDIABASE project.[26] Woo, et al. propose an Extended OCPN (XOCPN) which "takes into account the demands of isochronous data". Our work differs from that of Woo et al. in at least two respects. First, their work addresses situations where network delay characteristics can be known *a priori*, and this information can be used to pre-schedule transmission of multimedia objects at the source. However, we consider the case of an unpredictable network, and provide mechanisms that allow the transport layer and application to respond to changing conditions. Second, because the underlying network in their model is more predictable, they are able to address both coarse-grained and fine-grained synchronization, while at the present our focus is only on coarse-grained synchronization.

The MEDIABASE/MEDIADOC project has several features in common with our work. In both projects (1) a client/server system is used to serve multimedia documents over a network, (2) QoS (e.g., reliability) can be defined on a per object level, and (3) the transport layer assists with synchronization. Our work differs in both focus and architecture. The MEDIABASE/MEDIADOC project focuses on the design of "an advanced high-performance distributed multimedia information and communications system [DMIS] with a particular focus on document architectures, database models, communication and synchronization, and ... storage." As such their architecture is quite sophisticated. By contrast, our emphasis is on exploring the usefulness of partial order and partial reliability mechanisms in the transport layer; we use multimedia document retrieval as an example application to explore this concept. Hence we limit the scope of our document model and application architecture to the minimal framework necessary to test certain ideas. Our hope is that by developing PO/PR mechanisms and demonstrating their utility for multimedia document retrieval, we can provide useful tools to the developers of advanced DMIS's such as MEDIABASE/MEDIADOC.

6 ACKNOWLEDGMENTS

This work supported, in part, by the National Science Foundation (NCR-9314056) and by the US Army Research Office (ARO) (DAAL03-92-G-0070, DAAH04-94-G-0093).

7 REFERENCES

[1] M. Ahuja. Flush primitives for asynchronous distributed systems. *Info Processing Letters*, 34(1):5–12, Feb 1990.

[2] P.D. Amer, C. Chassot, T. Connolly, M. Diaz, and P. Conrad. Partial order transport service for multimedia and other applications. *IEEE/ACM Trans on Networking*, 2(5):440–456, Oct 1994.

[3] T. Berners-Lee and D. Connolly. Hypertext markup language specification – 2.0. (Internet) Network Working Group, Request for Comments RFC1866, November 1995.

[4] Kenneth P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, December 1993.

[5] D. Cheriton. Vmtp: Versatile message transaction protocol specification. (Internet) Network Working Group, Request for Comments RFC1045, April 1993.

[6] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM 90 Proceedings*. ACM, 1990.

[7] T. Connolly, P.D. Amer, and P. Conrad. Internet RFC1693: An Extension to TCP: Partial Order Service. Internet Request for Comments RFC1693, Nov 1994.

[8] Phillip T. Conrad, Paul. D. Amer, and Rahmi Marasli. Graceful degradation of multimedia documents via partial order and partial reliability transport protocols. In *IEEE Workshop on Multimedia Synchronization*, Tysons Corner, VA, May 1995. URL:<http://spiderman.bu.edu/sync95/papers/conrad.ps>.

[9] Bert J. Dempsey. *Retransmission-Based Error Control For Continuous Media Traffic In Packet-Swithced Networks*. PhD thesis, University of Virginia, 1994.

[10] Steven J. DeRose and David G. Durand. *Making Hypermedia Work: A User's Guide to HyTime*. Kluwer Academic Publishers, Boston, 1994.

[11] F. Gong and G. Parulkar. An application-oriented error control scheme for high-speed networks. Technical Report WUCS-92-37, Department of Computer Science, Washington University in St. Louis, November 1992.

[12] L. Lamport. Time, clocks and the ordering of events in a distributed system. *CACM*, 21(7):558–565, Jul 1978.

[13] T. Little and A. Ghafoor. Synchronization and storage models for multimedia objects. *IEEE J on Selected Areas in Comm*, 8(3):413–427, Apr 1990.

[14] T. Little and A. Ghafoor. Multimedia synchronization protocols for broadband integrated services. *IEEE Journal on Selected Areas in Communication*, 9(9):1368–1382, December 1991.

[15] T. Little and A. Ghafoor. Spatio-temporal composition of distributed multimedia objects for value-added networks. *IEEE Computer*, 24(10):42–50, October 1991.

[16] R. Marasli, P. D. Amer, P. T. Conrad, and G. Burch. Partial order transport service: An analytic model. In *Ninth Annual IEEE Workshop on Computer Communications*, Marathon, Florida, October 1994.

[17] Rahmi Marasli. *Partially Ordered and Partially Reliable Transport Protocols: Performance Analysis*. PhD thesis, University of Delaware, (In progress).

[18] Rahmi Marasli, Paul D. Amer, and Phillip T. Conrad. An analytic study of partially reliable transport services. (Submitted for publication).

[19] Michael Wynblatt. Position statement on multimedia synchronization. In *IEEE Workshop on Multimedia Synchronization*, Tysons Corner, VA, May 1995. URL:<http://spiderman.bu.edu/sync95/papers/wynblatt.ps>.

[20] G. Neiger and S. Toueg. Substituting for real time and common knowledge in asynchronous distributed systems. In *Proc 4th Symp on Principles of Distributed Computing*, pages 281–293, 1987.

[21] M. Pérez-Luque and T. Little. A temporal reference framework for multimedia syncrhonization. In *IEEE Workshop on Multimedia Synchronization*, Tysons Corner, VA, May 1995. URL:<http://spiderman.bu.edu/sync95/papers/mjpl_posit.ps>.

[22] L. Peterson, N. Buchholz, and R. Schlighting. Preserving and using context information in interprocess communication. *ACM Trans on Computer Systems*, 7(3):217–246, Aug 1989.

[23] Thomas F. La Porta and Mischa Schwartz. The multistream protocol: A highly flexible high-speed transport protocol. *IEEE Journal on Selected Areas in Communications*, 11(4):519–530, May 1993.

[24] J.B. Postel. Transmission Control Protocol. Internet Request for Comments RFC793, Sept 1981.

[25] J.B. Postel. User Datagram Protocol. Internet Request for Comments RFC768, Aug 1981.

[26] J.A. Rody and A. Karmouch. A remote presentation agent for multimedia databases. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 223–230, Washington, DC, May 1995. IEEE.

[27] James Schnepf, Joseph A Konstan, and David Du. Doing FLIPS: Flexible interactive presentation synchronization. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 213–222, Washington, DC, May 1995. IEEE.

[28] Miae Woo, Naveed U. Qazi, and Arif Ghafoor. A synchronization framework for communication of pre-orchestrated multimedia information. *IEEE Network*, pages 52–61, January/February 1994.

[29] L. Zhang, S. E. Deering, D. Estrin, S. Shenker, and D. Zappala. Rsvp: A resource reservation protocol. *IEEE Network*, September 1993.