RTP/I - Towards a Common Application Level Protocol for Distributed Interactive Media

Martin Mauve, Volker Hilt, Christoph Kuhmünch, Wolfgang Effelsberg

Abstract—**Distributed interactive media are media that involve communication over a computer network as well as user interactions with the medium itself. Examples of this kind of media are shared whiteboard presentations and networked computer games. One key problem of this media class is that a large amount of common functionality is currently redesigned and redeveloped for each single medium. In order to solve this problem we present a media model and an application level protocol called RTP/I. Derived from the experience gained with audio and video transmission using RTP, RTP/I is defined as a new protocol framework which reuses many aspects of RTP while it is thoroughly adapted to meet the demands of distributed interactive media. By identifying and supporting the common aspects of distributed interactive media RTP/I allows the reuse of key functionality in form of generic services. Furthermore RTP/I makes it possible for applications of different vendors to interact with each other in a standardized way.**

Index Terms—**Distributed Interactive Media, Multimedia Communication and Networking, Transport Protocols. Shared Whiteboard Systems, Networked Computer Games, Distributed Virtual Environments**

I. Introduction

THE class of *distributed interactive media*, i.e., networked media involving user interaction, has received increasing interest over the past decade. Important representatives of this media class are shared workspaces [1], [2], distributed virtual environments [3] and networked computer games [4]. These media often require common functionality. A prominent example is late join support, which allows late coming participants to synchronize with the current state of the session. Support for late comers is needed by almost all distributed interactive media, ranging from shared whiteboards to networked computer games. Other functionality frequently needed is support for consistency, timing, and session recording. Currently this common functionality has to be designed and implemented individually for each application. This leads to a tremendous overhead in the development of applications for this media class and, at the same time, to many incompatible proprietary implementations. Thus even applications for the same media type but developed by different vendors cannot interact with each other.

The main reason for this problem is the lack of a standardized protocol framework upon which applications and reusable functionality could be based. To solve this problem we developed (1) a media model, so that it is possible to discuss problems and solutions independent of an individual medium and (2) a standardized application level protocol framework to be able to make applications interoperable and to allow the direct reuse of common functionality as so called *generic services.* The protocol framework is called *"Real-Time Application Level Protocol for Distributed Interactive Media" (RTP/I).*

In Section Two we discuss existing approaches that have been proposed for communication in distributed interactive media. In the third section we investigate important aspects of the distributed interactive media class and describe how it differs from other media classes. Section Four covers important design considerations that should be taken into account when developing a protocol for this media class. This includes issues like identification of data units, consistency, fragmentation and meta information. With these requirements explained, we introduce the "Real-Time Application Level Protocol for Distributed Interactive Media" (RTP/I) in Section Five. The sixth section describes how we have used RTP/I for the development of applications and generic services. Section Seven concludes this paper with a summary and an outlook.

II. Existing Approaches

A large number of proprietary application level protocols for specific distributed interactive media have been developed, including protocols for shared whiteboard applications, networked computer games or distributed virtual environments. These protocols are application or media dependent and they do not capture the common aspects of the distributed interactive media class. Therefore it is not possible to develop generic and reusable functionality based on these protocols.

For distributed non-interactive media, such as audio and video, the Real-Time Transport Protocol (RTP) [5] has been developed. RTP is a flexible protocol framework that can be tailored to the specific needs of different non-interactive media. RTP has been very successful, it is currently used by the majority of tools for streaming audio and video over the Internet. RTP has also been used for some distributed interactive media (e.g., the digital lecture board (dlb) [2] and DIVE [3]).

The main reason why it is not a good solution to use RTP for distributed interactive media is that audio and video follow a fundamentally different media model than distributed interactive media. Audio and video transmissions have an ephemeral state which is frequently and redundantly transmitted. In addition the state of the medium is influenced only by a single entity - the sender of the stream. In contrast distributed interactive media are about managing the shared state of a medium. All participants are potentially able to change that state. The key problem of this media class is to keep the shared state reasonably similar for all participants of a session, i.e., to maintain consistency. These fundamental differences in the media models prevent RTP from being an optimal fit for distributed interactive media. For an in-depth discussion on why not to use RTP for distributed interactive media the reader is referred to [6].

Despite of the fact that RTP has been developed with the audio/video media model in mind, it includes many aspects which are also useful for distributed interactive media. Examples are participant identification, time-stamps, and the general concept of a flexible protocol framework. However, because of RTP's audio/video based media model it is not appropriate to use this protocol directly for distributed interactive media. For these reasons the Real-Time Application Level Protocol for Distributed Interactive Media (RTP/I) presented here is a separate protocol which reuses many aspects of RTP while it is thoroughly adapted to meet the demands of distributed interactive media.

It has also been proposed to use a reliable multicast framework like the Reliable Multicast Framing Protocol (RMFP) [7] as a foundation for the communication in distributed interactive media. This approach is mainly based on the observation that many elements of this media class require some degree of reliability. Furthermore, reliable multicast protocols typically provide a number of mechanisms that are directly applicable to distributed interactive media, such as application-level naming, message ordering, and participant identification.

There are two main reasons why a reliable multicast framework is not an appropriate base for a common application level protocol for distributed interactive media: first, there exist many distributed interactive media which either do not need reliability or which do not use multicast. Second, a reliable multicast framework is generally applicable to several media classes, ranging from bulk-data transfer to distributed interactive media. Therefore, it should not capture aspects which are only relevant to a single media class. However, this is exactly the information required to develop reusable functionality.

III. Distributed Interactive Media

A. Classification of Distributed Media

In order to define the scope of our work, we distinguish between distributed media types by means of two criteria. The first criterion ascertains whether the medium is discrete or continuous. The characteristic of a *discrete medium* is that its state is independent of the passage of time. Examples of discrete media are the shared viewing of still images or shared whiteboard presentations. While discrete media may change their state, they do so only in response to external events, such as a user drawing on a digital whiteboard. The state of a *continuous medium*, however, depends on the passage of time and can change without the occurrence of external events. Video and animations belong to the class of continuous media.

The second criterion establishes whether media are interactive or non-interactive. *Non-interactive* media may change their state only in response to the passage of time and they do not accept external events. Typical representations of non-interactive media are video, audio and images. *Interactive media* are characterized by the fact that their state can be changed by external events such as user interactions. Shared whiteboard presentations and distributed virtual environments represent interactive media. Figure 1 depicts how the criteria characterize different media types.

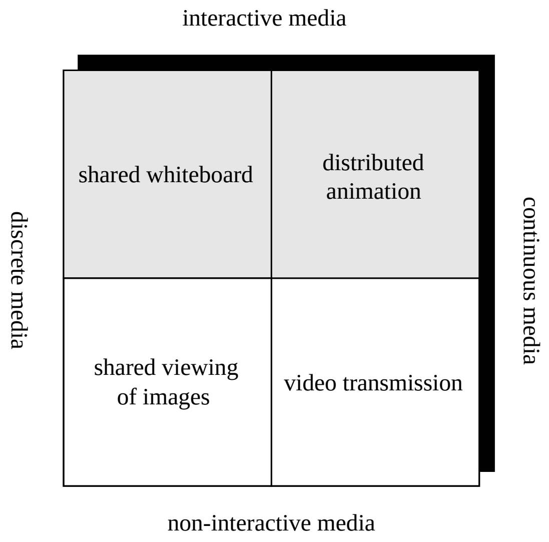

Fig. 1. Examples of Distributed Media

Distributed media that are neither interactive nor continuous are well understood and therefore discussed no further here. Media types that are non-interactive and continuous have already been investigated to a large extent in the context of audio and video transmission. Especially RTP [5] provides a solid base for the development of applications and services for this media class.

In contrast, proprietary protocols are used by almost all applications for distributed interactive (continuous, as well as discrete) media. The usage of proprietary protocols prohibits the development of generic services for important aspects such as recording or support for late comers. However, generic services like these would be within reach if distributed interactive media would share a common application level protocol for the data they transmit. RTP/I is such a protocol.

B. Model for Distributed Interactive Media

A *distributed interactive medium* has a *state*. For example, the state of a shared whiteboard presentation is defined by the content of all pages present in the shared whiteboard. In order to perceive the state of a distributed interactive medium a user needs an *application*, e.g., a shared whiteboard application is required to see the pages of a shared whiteboard presentation. This application generally maintains a local copy of (parts of) the medium's state. The architecture of applications for distributed interactive media is therefore said to be *replicated*. For all applications participating in a session the state of the medium should be at least reasonably similar. It is therefore necessary to synchronize the local state of the distributed interactive medium among all participants, so that the state of the medium is *consistent*.

The state of a distributed interactive medium can change for two reasons, either by the passage of time or by events. The state of the medium between two successive events is fully deterministic and depends only on the *passage of time*. Generally, a state change caused by the passage of time does not require the exchange of information between applications, since each user's application can independently calculate the required state changes. An example of a state change caused by the passage of time is the animation of an object moving across the screen.

Any state change that is not a fully deterministic function of time is caused by an *event*. Examples for events are user interactions with the medium, e.g. an annotation on a shared whiteboard page or the generation of a random number. Whenever events occur, the state of the medium is in danger of becoming inconsistent because the local copies of the state might cease to be synchronized. Therefore, an event usually requires that the applications exchange information - either about the event itself or about the updated state once the event has taken place.

In order to provide for a flexible and scalable handling of state information, it is often desirable to partition an interactive medium into several *sub-components*. In addition to breaking down the complete state of an interactive medium into more manageable parts, such partitioning allows the participants of a session to track only the states of those sub-components in which they are actually interested. Examples of sub-components are 3D objects (an avatar or a car) in a distributed virtual environment, or the pages of a shared whiteboard.

While it would be conceivable to declare all the information that is required by the application to display the distributed interactive medium, to be part of the medium's state, this is generally not desirable. Often a substantial part of the information needed by a participant in order to render a medium's state remains constant over the course of a session. We call this constant information the *environment* of a distributed interactive medium. Examples of environments are the base world descriptions of distributed virtual environments, or the postscript slides of shared whiteboard presentations. Since the environment stays constant, there are no mechanisms required to synchronize it among the participants of a session - the environment information just needs to be received once by each participant of a session. Since the environment does not need a real-time protocol, we do not further investigate how the environment is shared between participants.

It can be derived from our model that four main elements are involved in a distributed interactive media session: the application, the environment, states, and events. As described above, only the transmission of states and/or events needs the support of a common application level protocol. Event and state transmissions form the basis of a *distributed interactive media stream*. Note, that the term stream is used in a somewhat different fashion than for audio and video: a distributed interactive medium's stream can have multiple senders - all participants of a session could potentially contribute to the stream.

IV. Design Considerations

In this section we examine important considerations that should influence the design of an application level protocol for distributed (continuous as well as discrete) interactive media. These considerations can be grouped into the following categories:
- core functionality for distributing event and state information,
- support for maintaining the consistency of the shared state,
- support for fragmentation of oversized states and events,
- getting the current state of a sub-component in a standardized way,
- conveying meta information, and
- flexibility to customize the protocol to the specific needs of individual media.

In the following sub-sections we will examine each of these categories in detail.

A. *Core Functionality*

The core functionality of an application level protocol for distributed interactive media is to enable the dissemination of event and state information that are framed by a common header. The common header needs to exhibit enough information for generic services to take appropriate actions without knowing anything about the medium-specific encoding of states and events. Two obvious pieces of information that should be visible to a generic service are the type of the data (event vs. state) and an identification of the sub-component to which they refer. With these two pieces of information a generic service can interpret the semantics of a distributed interactive media stream to a high degree.

B. *Consistency*

Besides the identification of events and states, the chief aim of an application level protocol for distributed interactive media is to make it possible that the applications of all participants can maintain a consistent, synchronized state of the medium. Depending on the medium and the application the consistency criterion can vary. The strongest level of consistency would require that at all sites all events and/or state updates be applied in the correct order at the correct point in time. This level could be relaxed by allowing events and state updates to get lost, or be processed out of order, or at a "wrong" point in time. Obviously this would require either some sort of repair mechanism or the willingness of the user to accept a certain amount of inconsistency. Generally it can be said that diverse *consistency policies* exist, that realize different levels of consistency. An application level protocol for distributed interactive media should provide the information required to realize these policies without imposing a certain policy on the medium or the application.

Up to three types of mechanisms can be involved in achieving consistency for a distributed interactive medium:
• reliability may be required so that each participant learns about all state changes caused by remote actions,
• ordering may be required if events are not commutative (i.e., if the order in which events are applied to a state matters),
• timing may be required since events and states might be valid only at a single point in time.
In the following we will consider each of these mechanisms and discuss whether and how they should be supported by a general application level protocol for distributed interactive media.

B.1 Reliability

There exist two main approaches to establish reliability in distributed interactive media. The first is to rely on reliable transport protocols. Typical examples are TCP or generic reliable multicast libraries on top of (multicast) IP. The benefit of this approach is a simple TCP-like interface to the library providing reliability as well as a very clean (layered) software architecture. Using transport level reliability does not place specific requirements on an application level protocol for distributed interactive media.

The drawback of a generic reliable transport protocol is that it usually enforces ordered packet delivery and might therefore be inefficient. For example, consider that a single participant transmits an event for sub-component 1 and then an event for sub-component 2. Imagine that the first event gets lost for any one receiver. Now the second event will be buffered by the reliable transport service until the packet loss has been repaired. This is inefficient since the application most likely would have been able to process the event for sub-component 2 without waiting for the lost event to be retransmitted. If the medium is continuous the situation might be even worse. The time at which the event for sub-component 2 should have been executed could have been missed just because it was buffered by the reliable transport service. This would most likely result in the triggering of a state repair mechanism that could have been avoided had the transport service not buffered the event.

The second approach to establish reliability tries to avoid this problem by requiring that the application is "network aware" and helps with repairing packet loss. This approach is commonly known as application level reliability. Application level reliability is enabled by an architectural principle called Application Level Framing (ALF) [8]. With ALF the application transmits data in form of application data units (ADUs). An application data unit is a piece of information which is framed in a way that allows the application to interpret and possibly use the information independent of other ADUs. ADUs can therefore be passed to the application without enforcing a particular ordering. Examples of ADUs for distributed interactive media are events and states. The drawback of application level reliability is the increased complexity of the application. This drawback can be somewhat alleviated by using appropriate libraries such as libsrm [9].

An application level protocol for distributed interactive media can be regarded as a specialization of the ADU framing that is used for application level reliability mechanisms. It extends the framing of these mechanisms to the specific needs of distributed interactive media. The application level protocol should not try to specify reliability mechanisms. This would be inappropriate since the reliability requirements of distributed interactive media vary widely. Instead there should be a well defined way that allows arbitrary application level reliability mechanisms to use and extend the information provided by an application level protocol for distributed interactive media. This leads to the typical benefits of Integrated Layer Processing [8], namely the avoidance of redundant header information, as well as the reduction of copy operations.

In summarizing, it can be noted that both approaches to achieve reliability should be usable with an application level protocol for distributed interactive media. The protocol should not try to realize reliability, this is done either by a transport level protocol or by the application, possibly with the support of appropriate libraries. However, it is the task of an application level protocol to capture the common aspects of a media class. This information is frequently required by applications and generic services, even when a reliable transport protocol such as TCP is used. An application level protocol therefore provides some of the information required to deploy application level reliability, independent of how reliability is actually realized.

B.2 Ordering

Sequence numbers are an important tool for the ordering of transmitted events and states. One might argue that sequence numbers are not required if a reliable transport protocol is used, since such a protocol will ensure the reliable and ordered delivery of data. However, sequence numbers are not only needed for the detection of lost and misordered packets. But they are also useful to address and identify individual ADUs. This is required by a number of applications and services (e.g., consistency mechanisms, recording, floor control), independent of the transport protocol. Therefore sequence numbers should be part of the framing specified by an application level protocol for distributed interactive media. Generally it is a good idea to have distinct sequence numbers for each sub-component and ADU type (e.g., events and states). This allows application level reliability to reuse the sequence numbers for fine-grained packet loss detection and recovery.

While sequence numbers can solve the ordering problem for events and states from a single source, many distributed interactive media additionally require that the messages sent by all participants of a session are ordered. Such a total ordering of all events and state transmissions can be established by including a timestamp into each ADU. In general it is possible to use either the timestamp values of a physical clock or of a logical clock [10] for this purpose. If the ADUs of two participants carry the same timestamp a contention criterion such as a unique participant identifier can be used to break the tie.

B.3 Timing

Only timestamps that refer to a physical clock are usable for the timing in continuous distributed interactive media, since an event or state for this class of media is only valid at a single point in (real) time. This requires the existence of a common physical time base, established using protocols like NTP [11] (typical deviation from the real time: less than 50ms) and/or GPS receivers (deviation of less then 1ms). If a message arrives late in a continuous medium, it cannot be directly applied to the local copy of the medium's state. Instead there need to be taken actions to repair the problem, before the local copy of the medium's state is consistent again.

A physical timestamp is also required if the distributed interactive medium is to be synchronized with other media such as audio and video streams. Logical time, as described in Lamport's work on virtual clocks [10], could be used in the absence of a common physical time. However, they would only be appropriate for discrete interactive media. Generally it is preferable to use a physical clock, since distributed interactive media are very likely to be used in combination with continuous non-interactive media such as audio and video.

C. Fragmentation

In the case that large ADUs are transmitted it is necessary to fragment the information into smaller chunks that fit into a network packet. While it is generally possible to let the network layer handle this fragmentation, this is often considered inefficient. The main reason for this is as follows: if IP fragments transport layer (TCP/UDP) packets and a single IP packet gets lost, then the complete TCP/UDP packet is discarded, even though other parts of the transport packet might have arrived intact [12].

If distributed interactive media could guarantee that each ADU fits into a network layer packet, then there would be no need to support fragmentation in a application level protocol. Unfortunately, it is very likely that many distributed interactive media will transmit states (and maybe even events) that do not fit into a single network packet. An application level protocol for this media class should therefore support fragmentation of ADUs.

D. Getting the Current State of a Sub-Component

In a number of situations an application or a generic service might need to get the current state of a certain sub-component. Examples of those situations are: as means of resynchronization if an event has been lost or received late, as access point for random access in a distributed interactive media recorder, or as an initial state for a latecomer joining an ongoing session. Since this functionality is universally needed by many distributed interactive media, it should be supported directly by a standardized state query packet type.

E. Meta Information

Applications and generic services frequently need additional information about the medium and the participants of a session. For example, an application that joins an ongoing session typically requires an overview of the sub-components present in that session. It should be able to decide for each sub-component whether the sub-component is relevant for the local presentation of the medium. Equipped with this information, the application can then take the required actions to get the state of those sub-components that are of interest to the local participant. Since the meta information about sub-components is generally required for distributed interactive media, it should be transported in a standardized way by an application level protocol for this media class.

Information about the participants is desirable since many applications that use distributed interactive media require only a very simple session control functionality. An application level protocol should support this by providing a standardized means to communicate information about participants. Typical participant information includes the name, e-mail, and the phone number of session participants. This concept is very successfully used with RTP, where many audio and video tools rely on the participant description that is transported by RTP means.

Another interesting type of meta information that is conveyed in RTP are reception quality reports. Examples of this kind of information are loss rates, latency and jitter. These values are periodically transmitted by each participant. However, such a quality feedback is generally inappropriate as a standard component in an application level protocol for distributed interactive media. The reason for this is that ADUs might be retransmitted either by a reliable transport protocol or by a library for application level reliability. This would render reports on loss rates, latency, and jitter invalid. Instead such feedback should be part of whatever mechanism is used to achieve reliability. Alternatively it could also be an optional protocol element used only by those distributed interactive media that realize reliability without the help of retransmissions.

F. Flexibility

A "good" application level protocol for distributed interactive media will provide a large amount of flexibility, especially in consistency related issues and for the actual encoding of states and events. It is therefore necessary to specify an application level protocol for distributed interactive media as a protocol framework rather than as a monolithic protocol. One kind of flexibility has already been discussed: the ability to use different reliability mechanisms.

Additional flexibility should be provided so that the protocol can be tailored to the specific needs of diverse distributed interactive media. Ideally this should be a two-step process, as it is also used for RTP. The first step should be a specification document that defines the commonalities of a sub-class of distributed interactive media. A possible sub-class may exhibit the following characteristics: continuous medium, frequent transmission of sub-component state information to achieve consistency, no reliability at the transport level, no event transmission. This sub-class would contain the majority of battle field simulations and some networked action games.

In analogy to RTP we call such a specification document a profile. A profile may specify which reliability mechanism(s) are to be used and how consistency is realized. A profile may also define additional information that is included in the framing of the transmitted information. It can define additional meta information that is important for this sub-class of media, e.g., quality feedback for media sub-classes that do not use packet retransmission. However, a profile may not change the meaning of the information that is specified for the core protocol, since this would prevent generic services from working properly for the profile.

The second step in customizing an application level protocol for distributed interactive media should be the specification of how a single medium is carried using the framework set up by the application level protocol and possibly by a profile. This definition contains the encoding of event and state information, as well as the specification of reliability, timing and consistency constraints that are not already covered by a profile. Such a definition is called a payload type specification. Theoretically a payload type specification can be used without a profile or with multiple profiles. However, the usual case is a hierarchical relationship of a payload type definition that operates under exactly one profile specification. This allows services to be specified on three levels:

- they can be fully generic services and, as such, be usable by any medium transported over the given application level protocol for distributed interactive media,
- they can specify certain profiles for which they are usable,
- or they can be limited to a number of specific payload types.

V. Real-Time Application Level Protocol for Distributed Interactive Media - RTP/I

RTP/I has been specifically designed to meet the demands discussed in the previous section: framing of event and state data, support for consistency and fragmentation, a standardized way to query the state of a sub-component, the ability to convey meta information, and a flexible protocol design. RTP/I reuses many aspects of RTP, including the concept of two distinct protocols for the transportation of the data and meta information. The protocols used for the transmission of the data is called the RTP/I data protocol while the protocol used for meta information is called RTP/I control protocol (RTCP/I). These two protocols are carried over distinct transport addresses in order to use the demultiplexing performed by the transport service for the distinction between RTP/I and RTCP/I packets.

We start the presentation of RTP/I by discussing the data part of RTP/I. In a second step we move on to explain how meta information is transported. In both parts we reuse aspects of RTP when it is appropriate. A more formal definition of RTP/I can be found in the RTP/I Internet Draft [13]. An Open Source implementation of the RTP/I protocol can be downloaded from our webpages [14].

A. RTP/I Data Transfer Protocol

The bulk of data for a distributed interactive medium - states, events and requests for state information - are carried in RTP/I data packets. Essentially RTP/I data packets contain medium-specific information that is framed by common header fields. In RTP/I there exist four distinct data packet types: event, state, delta state, and state query.

A.1 Event Packet Type

An event packet carries an event or a fraction of an event. It is structured as depicted in Figure 2. The first two bits of an RTP/I event packet contain the version number (V) of the protocol. The E (end) and the fragment count fields are used for fragmentation and reassembly of ADUs that do not fit into a single network packet. The fragment count is set to 0 for the first packet of an ADU and it is increased for each packet transmitted for that ADU. The end bit is set to one if this packet is the last packet of an ADU. Recipients of a fragmented ADU know that they have received all parts of an ADU when they have received a packet with the fragment count 0, a packet with the end bit set, and all packets in-between as identified by the fragment count.

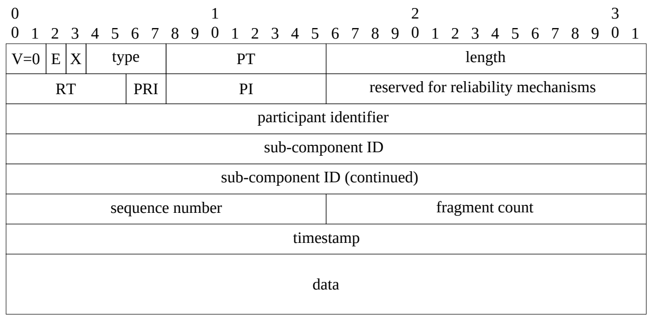

Fig. 2. RTP/I Data Transfer Protocol Packet

The type field identifies the content of the packet. There are four values used by RTP/I: EVENT, STATE, DELTA_STATE, and STATE_QUERY. For event packets the type field contains the value EVENT. In the type field the values of 8 and above are reserved for use with application level reliability mechanisms, e.g., this can be used to identify an additional packet type for the detection of tail loss. Furthermore, application level reliability mechanisms may blend into the RTP/I header. The RT field identifies the reliability mechanism that is used to transmit the packet (e.g., libsrm [9]). The 16 bits that are reserved for reliability can be used by reliability mechanisms to store additional information. Examples are flags to mark packets for forward error correction or retransmitted packets. If a given reliability mechanism needs more than 16 additional bits, it may append an extension header to the regular RTP/I header. This is signaled through the reliability header eXtension (X) bit.

An application level reliability mechanism that is used together with RTP/I is specified in an RTP/I reliability specification document. The reliability specification document completely defines the employed mechanism and its cooperation with RTP/I. It may do so by referencing an existing specification of the reliability mechanism and by providing information on how it cooperates with RTP/I. This includes the mapping of the information provided by RTP/I (framing and meta information) to the information required by the reliability mechanism, the specification of any additional information that is to be stored in the RTP/I framing, and the definition of any additional packet types that may be required. Each RTP/I reliability specification is assigned a unique number in the range of 2-63. There exist two predefined reliability mechanisms that can be used with RTP/I. The first one is no reliability (e.g., raw UDP). The second one is the usage of a reliable transport protocol that is transparent to the application (e.g., TCP). These mechanisms are assigned the numbers 0 and 1, respectively.

The PT (payload type), timestamp and participant identifier fields are similar to the corresponding fields in the RTP framework. The payload type field identifies the payload type transported in this packet (e.g., a specific shared whiteboard encoding). The timestamp indicates the point in time when the event must be applied to the medium. Generally this value should be expressed in milliseconds of a physical clock synchronized by NTP or a GPS receiver.

The 32-bit participant identifier field makes it possible to identify participants independent of their network/transport layer addresses. In contrast to RTP the participant identifier needs to be unique and stays constant for the whole lifetime of a session. How to choose a unique identifier is outside of the scope of RTP/I. The sub-component ID is a 64-bit value that uniquely identifies the sub-component the ADU refers to. As for the participant identifier, the mechanism used to choose unique sub-component identifiers is outside of the scope of RTP/I.

Additional information that is defined by a profile may be stored in a field which is reserved for profile information (PI). This allows profiles to include important information in the framing without having to add another 32-bit word to the header. The last field relevant to the event packet is the sequence number. It is increased by one for each event ADU. The medium-dependent event data follows the RTP/I header, its encoding is specified in the payload type definition. The PRI (priority) field is not used by event packets.

A.2 State Packet Type

The RTP/I state packet type is used to transmit a sub-component's complete state (or a fraction thereof if the state is to large to fit into a single network layer packet). A state packet has the same structure as an event packet with the exception that an additional priority (PRI) field is present. This field is necessary because setting the state of a sub-component can be costly and might not always be reasonable for all participants. A packet with high priority should be examined and applied by all communication peers who are interested in the specific sub-component. Situations where a high priority is recommended are re-synchronization after errors, or packet loss. A state transmitted with low priority can be ignored at will by any participant. This is useful if only a subset of communication partners is interested in the state. An example of this case are late joins where only applications joining the session might be interested in certain state transmissions.

The timestamp value of state packets has a different meaning than that for event packets. It denotes the point in time when the state contained in the packet was extracted from the medium. When such a state is applied to a sub-component of a continuous distributed interactive medium at a later time, the time difference needs to be accounted for.

A.3 Delta State Packet Type

In cases where a complex sub-component state of an interactive medium is transmitted frequently by an application, it may be desirable to be able to send only those parts of a state that have changed since the last state transmission. This is similar to the concept of P frames in an MPEG encoded video stream. RTP/I supports this by providing a delta state packet type. A delta state ADU (possibly consisting of more than one packet) can only be interpreted if the preceding full state ADU is also available (see Figure 3). The main advantages of delta state ADUs are their smaller size and that they can be calculated faster than full state ADUs. The delta state packet has the same format as the state packet, too.

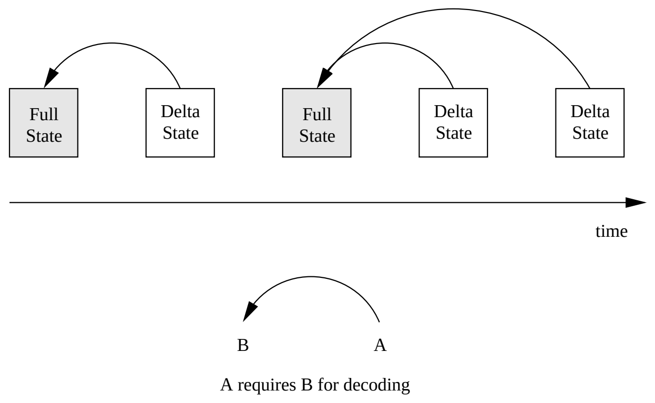

Fig. 3. Decoding of Delta States

A.4 State Query Packet Type

The state query packet type is used by a participant to indicate that the transmission of a certain sub-component's state is required. The state query is part of the data protocol (and not of the control protocol) for two reasons. First, it requires the same header information as the other RTP/I data packets. Second, it may make use of a reliability service that could exist for the data part of RTP/I. The structure of this packet type is the same as for all RTP/I data packets (see Figure 2). However, it does not contain any medium specific data.

Instead of being a regular request - as found in other protocols - a state query packet is only an indication to the receivers that a participant would like to get the state of the concerned sub-component. A regular request/reply mechanism would be inappropriate for a protocol that should work between more than two participants. The decision whether any given receiver of a state query packet will reply is made locally by the receiver. This decision is influenced by the priority and should made such that it will prevent a reply implosion if multicast is used. In detail the meaning of the priority contained in a state query packet is as follows:

- Priority 3 is the highest priority and is used when the request needs to be satisfied immediately, e.g., for resynchronization after errors.
- Priority 2 is used when a response is required, but a short delay is acceptable, e.g., for a late-join service.
- Priority 1 is used when a response is desirable but not required, e.g., pre-fetching of sub-component state, which might be needed later.
- Priority 0 is used when the state request is issued periodically, e.g., for a recording service.

B. RTP/I Control Protocol - RTCP/I

In order to convey meta information RTP/I uses the RTP/I Control Protocol (RTCP/I). RTCP/I compound packets are transmitted by each participant in regular intervals. The interval between the transmission of two compound packets is called report interval. In order to keep the overall data rate consumed by RTCP/I constant, the duration of the report interval is inverse proportional to both, the average size of the compound packets and the number of users in the session. An RTCP/I compound packet typically consists of two different packet types: source description packets and sub-component report packets. The source description packets convey information about the identity of the participant, such as a unique name (e.g. mauve@pilatus.uni-mannheim.de) or an e-mail address or a telephone number. Sub-component report packets contain meta information about the sub-components present in the session. In the following we will only describe the sub-component report packet, the source description packet is used as defined in the RTP Internet Draft [5].

B.1 Sub-Component Report Packet

The sub-component report packet is used to announce the sub-components present in a session, to indicate which sub-components are actively used to display the medium to the users, and to provide information for the mapping from sub-component IDs to application level names. For each report interval every participant checks whether any sub-components that it tracks the state of have not been reported by other participants in the last two report intervals. All unreported sub-components are then reported using a sub-component report packet. This algorithm is simple, robust, and scalable - in most cases each sub-component is reported exactly once per report interval, independent of the number of participants and packet loss.

Applications and services can use the received sub-component report packets to learn which sub-components are actively used to display the medium to at least one user. This information is very important since these sub-components are likely to be of immediate relevance for the session. Sub-components which are not currently used to display the medium might be handled with a lower priority, e.g., for applications that join an ongoing session, or for a recording service. In order to make this distinction possible the report for a sub-component includes its current status: active or passive. The active sub-components of an application at any point in time are those sub-components that are required by the application to present the interactive medium at that specific time.

The third information carried in sub-component report packets are application level names for sub-components. This functionality is required since partitioning the state of a distributed interactive medium into several sub-components leads to the question how an application can decide which sub-component it is interested in. Typically this problem can be solved by providing each sub-component with an application-level name. This name should carry enough information about the sub-component so that the applications or the users can decide whether they are interested in the sub-component or not. Examples for application level names are the titles of shared whiteboard pages, or the textual description of 3D objects in a distributed virtual environment.

The mechanism to convey application level names is optional, a payload type may specify that it does not require application level names for sub-components. One possible reason for not using application level names could be that the sub-component identifier holds enough information for the application to decide whether it is interested in the sub-component or not.

As depicted in Figure 4, the sub-component report packet starts with the protocol version number (V). The name bit (N) signals whether the reported sub-components are accompanied by an application level name. The sub-component count (SC) field holds the number of sub-components reported in this packet; note that one sub-component report packet can carry information about several sub-components. The type field is used to distinguish different RTCP/I packet types (sub-component report packets, source description packets, and bye packets). In order to enable the combination of several RTCP/I packets into one compound packet, a length field is included in each RTCP/I packet. The length field is followed by the participant identifier of the participant sending the packet. The main part of the packet starts with a list of active (A) flags. An active flag is set if the corresponding sub-component is active for the sender. Finally the packet contains a list of sub-component IDs that may be followed by their application level names. The sub-component IDs appear in the same order as the active flags, so that a receiver of the packet can associate the appropriate active flags with a sub-component ID.

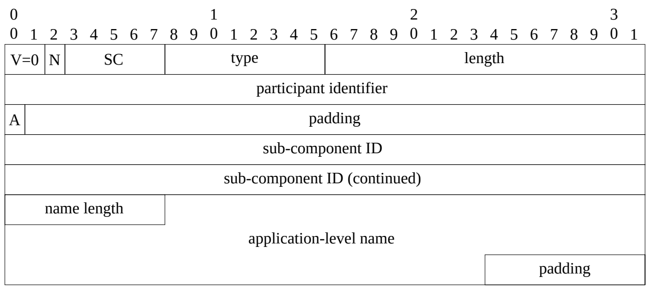

Fig. 4. RTCP/I Sub-Component Report Packet

VI. Applications and Services

RTP/I has not been created in a "clean-room" environment - it rather is the solution to a "real-world" problem we encountered when developing applications for distributed interactive media. Examples for these applications are a shared whiteboard [2], remote controlled Java animations for teleteaching [15] and a 3D telecooperation application [16]. For each of these applications we had to re-implement a substantial part of similar functionality (e.g., session recording, floor control, support for late comers). The prime reason for this problem was that diverse application level protocols were used to realize the communication between session participants.

RTP/I is designed to solve this problem. Currently we are in the process of replacing the proprietary application level protocols of our applications with RTP/I. At the same time we extract the common functionality from existing application specific services and expand them to generic, RTP/I-based, services.

The first application for which the integration of RTP/I has been completed is a 3D telecooperation application called TeCo3D. The first generic service which has been defined is an RTP/I recording service. Other services like a simple floor control and a late-join service have also been developed. In order to demonstrate how RTP/I can be used for the development of applications and services, we will briefly outline TeCo3D and a generic recording service in the following two sub-sections.

A. Using RTP/I for a 3D Telecooperation Application

TeCo3D is a shared workspace for dynamic and interactive 3D models [16]. Its aim is to allow users to share collaboration-unaware VRML (Virtual Reality Modeling Language) models, i.e., models which have not been specifically developed to be used by more than one user at a time. With this functionality it is possible to include arbitrary VRML content, as generated by standard CAD or animation software, into teleconferencing sessions.

TeCo3D uses the Java3D VRML loader [17] as 3D presentation and execution engine. It employs a completely replicated distribution architecture with reliable multicast at the transport layer. When a user imports a local VRML object into the shared workspace, the VRML code is parsed and the parts which are responsible for user interactions are replaced with custom components. This turns the collaboration-unaware 3D objects into a collaboration-aware model. User initiated operations are captured by the custom components and are transmitted to all participating applications in the session, where they are injected into the local model. In order to provide remote users with the initial state of the imported objects and to recover from inconsistencies we have enhanced the VRML loader by a method to get and set the state of arbitrary VRML objects [18].

The media model for distributed interactive media can be applied to TeCo3D as follows: the sub-components are the (independent) interactive 3D objects loaded into the shared workspace. The state of a sub-component (3D object) can be accessed through our enhancement of the VRML loader. Events are the user interactions which are captured by the custom components inserted into the 3D objects. Since the state of dynamic and interactive 3D objects can change because of the passage of time as well as because of user actions, TeCo3D can be regarded as a *continuous distributed interactive medium*.

We have defined the encoding of events and states within a TeCo3D payload type definition. This definition also specifies how consistency is realized. TeCo3D uses a reliable multicast protocol at the transport layer and requires that events are applied to the model of each participant in the same order at the same time. This is a very strong consistency criterion, which can be realized by voluntarily delaying local actions until it is likely that they have been received by all participants. This concept, called local-lag, is explained in detail in [19]. If resynchronization is required because a participant has received an event late, then the state request mechanism of RTP/I is used to request the state of a specific 3D object. First experiments have shown that TeCo3D provides a highly accurate synchronization between participants. Currently there are efforts under way to use the TeCo3D prototype (including the mechanisms for consistency and the usage of RTP/I) as a foundation for a product of the SIEMENS AG.

B. Generic Recording Service for RTP/I

The ability to record a session is one of the most important and universally needed aspects of video-conferencing. So far the generic recording of audio and video streams has been investigated and understood to a large extent [20]. There also exist first approaches to record the data transmitted by a specific shared whiteboard [21]. Due to the lack of a common application level protocol, a *generic* recording service for distributed interactive media was long thought to be infeasible. However, since RTP/I provides the required information, we have been able to develop a generic recording service for this media class. The service is based exclusively on the common aspects of distributed interactive media as they are exposed by RTP/I.

The main idea of the generic recording service is to record the ADUs (states and events) as they are transmitted during a live session. At a later time, the stream of ADUs can be replayed with the proper order and timing as provided by the RTP/I header information. The timestamps of the replayed ADUs need to be re-calculated so that they match the replay time. The stream produced in this way can be interpreted and presented to a user by an unmodified application, just like the stream of a regular session. It is therefore not necessary to develop special viewers for the recorded sessions.

One challenging problem with the development of a generic recording service for distributed interactive media is the realization of random access to the recorded stream. The main problem here is that the listening applications need to get the current state of the medium before they are able to follow the recorded stream. Since the recorder is generic, it will not be able to calculate this required state information. Instead it needs to transmit a combination of recorded ADUs which puts the receivers into the desired state at the access position. If the medium has just one sub-component and if the state of that sub-component is inserted frequently into the live stream, a very simple mechanism for random access would be: start the reply at a time close to the desired access point, where a state transmission was recorded. In the case that the live stream of a medium does not normally contain frequent state transmissions, the recording service is able to request this information by issuing state query packets. A more sophisticated random access methods which supports realistic applications with more than one sub-component or discrete media types with heavy weight states can be found in [22].

The development of generic services such as the recording service and the integration of RTP/I into real applications has provided us with the required knowledge to complete the first cycle of protocol design, implementation, testing and re-design. Therefore we are confident that the main functionality of RTP/I, as described in this work, is both appropriate and useful for the class of distributed interactive media. At the same time we are very much aware of the fact that the adoption of RTP/I by other applications will likely result in new challenges and possibly enhancements and additions to RTP/I.

VII. Conclusion

In this paper we discussed the development of a real-time application level protocol for distributed interactive media. Typical examples for this media class are shared whiteboards, distributed virtual environments and networked computer games. The key advantage of a common application level protocol is the ability to develop generalized and reusable functionality, such as session recording, support for late comers or consistency mechanisms.

As a starting point we defined a general media model. This model identified the common aspects which are shared by all distributed interactive media. Based on the model as well as on the needs of applications and generic services we investigated the requirements that an application level protocol for distributed interactive media has to satisfy. Most importantly these requirements included framing the core data, support for consistency, conveying meta information and the ability to tailor the protocol to the specific needs of different distributed interactive media.

In order to match the media model and the requirements we proposed the development of a new application level protocol inspired by the Real Time Transport protocol (RTP). The new protocol is called "Real-Time Application Level Protocol for Distributed Interactive Media" (RTP/I). RTP/I reuses many aspects of RTP while it is thoroughly adapted to meet the specific demands of distributed interactive media. We have proven that RTP/I is viable by using it for a 3D telecooperation application as well as for the development of a number of generic services. Other applications are currently in the process of adopting RTP/I as their application level protocol.

Acknowledgments

We would like to thank Colin Perkins for many inspiring discussions about RTP and RTP/I. These discussions have been very valuable for the design of RTP/I. Furthermore the comments of the anonymous reviewers were a great help for improving the paper.

This work was supported by the European MECCANO Telematics for Research Project 4007 and by the German BMBF (Bundesministerium für Forschung und Technologie) with the "V3D2 Digital Library Initiative".

References

[1] M. Handley and J. Crowcroft, "Network Text Editor (NTE): A scalable shared text editor for the MBone," in *Proc. of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM) '97*, 1997, pp. 197–208.

[2] W. Geyer and W. Effelsberg, "The Digital Lecture Board - A Teaching and Learning Tool for Remote Instruction in Higher Education," in *Proc. of the 10th World Conference on Educational Multimedia (ED-MEDIA) '98*, 1998.

[3] E. Frécon and M. Stenius, "DIVE: A Scalable network architecture for distributed virtual environments," *Distributed Systems Engineering Journal*, vol. 5, no. 3, pp. 91–100, 1998.

[4] L. Gautier and C. Diot, "Design and Evaluation of MiMaze, a Multi-player Game on the Internet," in *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, 1998, pp. 233–236.

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," Internet Draft, Audio/Video Transport Working Group, IETF, draft-ietf-avt-rtp-new-08.txt. Work in prgress, 2000.

[6] C. Perkins and J. Crowcroft, "Notes on the use of RTP for shared workspace applications," *ACM Computer Communication Review*, vol. 30, no. 2, 2000.

[7] J. Crowcroft, L. Vicisano, Z. Wang, A. Ghosh, M. Fuchs, C. Diot, and T. Turletti, "RMFP: A Reliable Multicast Framing Protocol," Internet Draft, IETF, draft-crowcroft-rmfp-02.txt, work in progress, 1998.

[8] D. Clark and D. Tennenhouse, “Architectural Considerations for a New Generation of Protocols,” in *Proc. of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM) '90*, 1990, pp. 201–208.

[9] S. Raman, “Design and analysis of a framework for reliable multicast,” Master's Thesis, University of California Berkeley, Berkeley, 1998.

[10] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[11] D. L. Mills, “Network Time Protocol (Version 3) specification, implementation and analysis,” DARPA Network Working Group Report RFC-1305, University of Delaware, 1992.

[12] C. A. Kent and J. C. Mogul, “Fragmentation considered harmful,” in *Proc. of the ACM workshop on Frontiers in computer communications technology (ACM SIGCOMM) ‘87*, 1987, pp. 390–401.

[13] M. Mauve, V. Hilt, C. Kuhmünch, J. Vogel, W. Geyer, and W. Effelsberg, “RTP/I: An Application Level Real-Time Protocol for Distributed Interactive Media,” Internet Draft, IETF, draft-mauve-rtpi-00.txt. Work in progress, 2000.

[14] RTP/I, “The RTP/I homepage,” http://www.informatik.uni-mannheim.de/informatik/pi4/projects/RTPI/index.html, 2000.

[15] C. Kuhmünch, T. Fuhrmann, and G. Schöppe, “Java teachware - the java remote control tool and its applications,” in *Proc. of the 10th World Conference on Educational Multimedia (ED-MEDIA) ‘98*, 1998.

[16] M. Mauve, “TeCo3D - A 3D Telecollaboration Application Based on VRML and Java,” in *Proc. of Multimedia Computing and Networking (MMCN'99 at SPIE'99)*, 1999, pp. 240–251.

[17] VRML Consortium, “The java3d and vrml working group,” http://www.vrml.org/WorkingGroups/vrml-java3d.

[18] M. Mauve, “Transparent Access to and Encoding of VRML State Information,” in *Proc. of the Fourth Symposium on the Virtual Reality Modeling Language (VRML'99)*, 1999, pp. 29–38.

[19] M. Mauve, “Consistency in Replicated Continuous Interactive Media,” To appear in: Proc. of the ACM Conference on Computer Supported Cooperative Work (CSCW'2000), 2000.

[20] W. Holfelder, “Interactive Remote Recording and Playback of Multicast Videoconferences,” in *Proc. of the International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services 1997 (IDMS'97)*, 1997, pp. 450–463.

[21] MASH, “The Mash Archive System Documentation,” http://mash.cs.berkeley.edu/mash/software/archive-usage.html.

[22] V. Hilt, M. Mauve, C. Kuhmünch, and W. Effelsberg, “A Generic Scheme for the Recording of Interactive Media Streams,” in *Proc. of the International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services 1999 (IDMS'99)*, 1999, pp. 291–304.