HIGH PERFORMANCE RECORDING AND MANIPULATION OF DISTRIBUTED STREAMS

Hasan Bulut

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of Computer Science
Indiana University
May 2007

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Geoffrey C. Fox, Ph.D. (Principal Advisor)

Randall Bramley, Ph.D.

Dennis Gannon, Ph.D.

Donald McMullen, Ph.D.

Beth Plale, Ph.D.

April 24, 2007

I would like to dedicate this dissertation to

my father, Ibrahim

my mother, Sariye

my brothers, Naif and Omer

my son, Omer Naif

my daughter, Hatice Zulal

and

my wife, Vahide

Acknowledgements

Through out this dissertation, I have worked on interesting topics in various projects at Indiana University Community Grids Lab. I have received experiences not just while working in my profession but also while collaborating with people. Also without the support of my family, I could never have overcome the challenges of Ph.D. life. It is a pleasure to acknowledge the debts that I have accumulated over these years.

I would like to express my appreciation to my advisor, Professor Geoffrey C. Fox for supporting and steering this research in the right way. Community Grids Lab has been an excellent research environment.

I would like to thank my research committee members, Prof. Geoffrey C. Fox, Prof. Beth Plale, Prof. Dennis Gannon, Prof. Randall Bramley and Prof. Donald McMullen for their suggestions and feedback on this dissertation.

I want to express my thanks to research associates of Community Grids Lab, Dr. Shideep Pallickara and Dr. Wenjun Wu for their advising and help throughout my research. I would also like to thank Dr. Marlon Pierce for helping me with the issues I encountered during tests.

I would like to thank Choonhan Yoon from San Diego Super Computer Center, technical staff of Welsh e-Science Centre at Cardiff University UK and Computer Science Department at Florida State University for providing their machines to conduct my tests.

I would like to thank my friends in Community Grids Lab. I owe a special thanks to my friends Galip Aydin, Mehmet Aktas and Beytullah Yildiz for their insightful discussions in the final stages of my research. I am also grateful to Mehmet Akif Nacar and other members of the lab for their company and for providing a friendly atmosphere.

I would like to express my profound gratitude to my father Ibrahim, who recently passed away, and my mother Sariye and my brothers Naif and Omer for all the sacrifices they have made and for their unconditional support. I also give thanks to my mother-in-law Hatice and father-in-law Huseyin for taking care of my family during the last year of my Ph.D. My friends Ali Mermer, H. Ali Mantar and Yusuf Erdemli also deserve my special thanks.

Finally, and most importantly I want to thank my wife Vahide for her love. She has been very patient and supportive through these stressful years. Her love and support made it possible for me to complete my Ph.D. We had to be apart from each other in the last 18 months of my research and she took care of my son Omer Naif and daughter Hatice Zulal during that time. She has been an excellent mother and a most loving friend. I cherish my son Omer Naif and daughter Hatice Zulal, with whom I have lived the most joyful years of my life.

Abstract

Collaboration and videoconferencing systems have become important applications over the Internet. They have been used in many areas such as distance education, telemedicine and business sectors. For videoconferencing systems, highly sophisticated audio and video codec have been designed to transfer audio and video content efficiently, standards have also been developed to initiate and manage real-time videoconferencing sessions. Another emerging application area for video streams is annotation systems, where video streams are annotated.

In this dissertation we present a scalable, fault tolerant and services-based streaming architecture based on messaging systems that integrates videoconferencing, streaming and annotation systems into an interoperable collaboration environment. Our architecture is a generic streaming framework which is independent of media type and format. So it provides a generic delivery mechanism which can be used not just for multimedia content but also for any type of streaming content. One of the strengths of this architecture is to allow instant replay of real-time, live streams. We have developed services within a publish/subscribe messaging scheme to support synchronization of streams, reduction of jitter caused by the underlying network and to increase fault tolerance and scalability of streams.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Collaboration and videoconferencing systems have become important applications over the Internet. They have been used in many areas such as distance education, telemedicine and also in various business sectors [1]. For videoconferencing systems, highly sophisticated audio and video codec have been designed to transfer audio and video content efficiently and standards have been developed to initiate and manage real-time videoconferencing sessions.

The well known standards are H.323 [2] and IP-Multicast [3]. However, these standards emerged from different multimedia communities and serve different purposes.

Videoconferencing systems provide a framework to send/receive audio and video streams. These systems consist of three parts: clients, session servers and communication channels. For example, in an H.323 based system, a client refers to the H.323 endpoint that is capable of sending/receiving audio and video. H.323 client can only send/receive one audio and video stream at a time. A session server refers to the Multipoint Controller that can generate multipoint sessions. A multipoint communication channel provider is the Multipoint Processor that can mix audio and video streams from different clients. H.323 based systems such as Polycom [4] and Radvision [5] are used in business sectors to enable companies to hold online meetings with other companies and with their own clients to reach business agreements and/or discuss business projects. These H.323 systems can also be used in distance education. Session Initiation Protocol (SIP) [6] is a text based protocol and is usually used in IP Telephony [7]. Another example is the Access Grid [8] system where a client is based on the Multicast Bone (MBONE) [9] audio/video tools such as Robust Audio Tool (RAT) [10] for audio and Videoconferencing Tool (VIC) [11] for video. Furthermore there is a venues server in the Access Grid, which is responsible for scheduling meetings. Multicast Real-time Transport Protocol (RTP) channels are the communication infrastructure for the Access Grid. The Access Grid clients VIC and RAT can receive and play multiple audio/video (A/V) streams in the session so that they can interact with each other. The Access Grid makes long distance collaboration between researchers possible only if their clients are MBONE clients.

Systems based on the H.323, SIP, IP-Multicast and other such standards cannot communicate with each other since the collaboration environment in these systems is homogenous where all clients would have to use the same protocol to communicate with each other. Among videoconferencing systems the Virtual Rooms Videoconferencing System (VRVS) [12, 13] is the only one that integrates different collaboration environments with each other.

There are also some client types which do not need to send streams to interact with the others but only need to receive streams. Such clients can receive streams with some delay. These clients do not want flickering in the video or noise in the audio if their bandwidth is fluctuating often. Streaming provides a framework to deliver media stream across large distances over the Internet while trying to minimize aforementioned problems that clients might have.

A streaming client connects to a streaming server, primarily using Real Time Streaming Protocol (RTSP) [14], to establish a session and receive the stream. Streaming is primarily used for receiving media that resides on a streaming server (Media-On-Demand) whenever a client wants to play a stream. It is also possible to make live streaming broadcasts. RTSP is a client-server multimedia presentation control protocol. It provides Video Cassette Recording (VCR)-like control, so that clients can pause, fast forward, reverse, absolute position and can perform other such functions. RTSP also maintains the state between client and server. Streaming media aims to improve the quality-of-service of the stream, by using various streaming codec to improve stream quality and a buffering mechanism to reduce the jitter [15]. The buffering mechanism causes the stream to be delayed for some time. Hence, streaming media does not provide a good interactive environment compared to the interactivity in a videoconferencing system, despite the fact that the stream quality is improved.

Developments in cellular networks and cell phone technology provide new application areas for streaming on cellular phones. Currently cellular phones are capable of playing video streams. While some of them are capable of playing only local video files others are also capable of receiving streams from streaming servers.

1.1 Motivation

The collaboration systems discussed above do not provide an architecture or framework against network failures. For instance, in these kinds of systems, clients who join late miss some of the session, and broken clients suffer from poor and broken connections.

In order to support late joining and broken clients, the system should have an archive and replay service so that missed parts of the stream can be played by the late joining or broken clients. *IG Recorder* [16], developed by *inSORS Integrated Communications* [17] or its open source equivalent *Voyager* [18], developed by *Argonne National Laboratory* [19] is used in the Access Grid sessions and records streams as they are. Those recorded streams are available only after the session is over. Late or broken clients cannot receive the past streams while the session is on. Multiple instances of *Voyager/IG Recorder* can run to record the same session independently. But there is no mechanism for recovery from failures while a session is recorded. There is a need for a distributed and replicated archiving system to improve the fault tolerance of the system. H.323 based systems such as ref. [20] and ref. [21] do not specify an archiving and replay capability within the standard and VRVS does not provide such service either.

In a collaboration system, support for different types of clients is an issue. The Access Grid needs MBONE tools and does not provide streams in other streaming formats. Although H.323 terminal (client) is capable of playing the stream format in the Access Grid sessions, since the Access Grid uses multicast network, H.323 terminal cannot receive the Access Grid streams. VRVS provides streams for QuickTime [22] and PocketPC clients. But there are no stream format conversions for QuickTime. It only receives the original stream available. Clients with low bandwidth will not be able to receive this stream. PocketPC also uses wireless networks. Cellular phones cannot receive those streams since they do not integrate web casting systems and videoconferencing systems to enable streaming clients such as RealMedia [23] clients to receive streams.

In a large collaboration environment stream quality is also another major issue. A high value of jitter may be introduced in packets. Packets lose their order en route to the client and this may result in ignoring those packets if they arrive too late [24]. Hence, there is a need for improving stream quality by reducing jitter and providing services and mechanisms to improve packet ordering. None of the above collaboration systems provide such mechanisms.

Another emerging application area for video streams is annotation systems. Examples for these systems are IBM's VideoAnnEx Annotation Tool [25], Microsoft Research Annotation System [26], Classroom 2000 [27], Intelligent Video Annotation System [28] and Synchronous Multimedia and Annotation Tool [29]. Annotation systems may annotate video streams. However current annotation systems use previously archived video files and they cannot get streams from videoconferencing sessions such as the Access Grid sessions or H.323 terminals.

Each of these systems target a specific community providing a capability that may not exist in the other system. These systems also do not benefit from the developments in publish/subscribe systems.

Publish/subscribe systems provide loosely coupled messaging middleware to facilitate group communications. Today there are many publish/subscribe systems from research to the industrial community. Some of the well known systems are NaradaBrokering, Gryphon [30], SIENA [31], SonicMQ [32], Java Message Service (JMS) [33], Sun Java System Application Server [34], IBM WebSphere Application Server [35], and BEA WebLogic Server [36].

Communication in publish/subscribe systems is asynchronous. Event producers which publish messages on topics on the broker network and event consumers which receive those messages by subscribing to those topics through the broker network are completely decoupled. There may be many subscribers subscribed to the same topic so when the publisher publishes a message, the broker network delivers it to the subscriber. The benefits of publish/subscribe systems are as follows:

- They can transport any type of message, including messages from collaborative applications. This reduces overall system complexity.
- They provide software multicast. So when a publisher publishes a message that message is received by many subscribers without affecting the publisher.

1.2 Research Issues

In this dissertation we investigate the issue of developing a fault tolerant and service oriented streaming architecture based on messaging systems that integrates videoconferencing, streaming and annotation systems into an interoperable collaboration environment. The research issues in this thesis can be divided into four categories: session management, metadata management, multiple stream support and investigation of application scenarios devised to demonstrate the framework explained in this dissertation. Session and metadata level issues are related to signaling among components, management of sessions and streams. Multiple stream level issues are related to data level support for fault tolerant streaming with quality-of-service.

1.2.1 Session Management

Videoconferencing systems have their own conference control framework which makes them accept only clients that use their protocols. In such a system the environment is homogenous, that is only one type of client exists in the videoconferencing session. Even when the systems define some gateways for other client types, such as SIP-H.323 gateway [37, 38] to enable SIP clients talk to H.323 terminals, the architecture makes it difficult to extend it to other types of clients and also these gateways are limited to only a handful of client types. This is because the protocol that they define either does not allow them to extend the architecture to cover heterogeneous client types or makes it too difficult to integrate with other systems. So we propose XML Based General Session Protocol (XGSP) for audio and video sessions to overcome these difficulties in achieving an integrated videoconferencing system. XGSP allows the system to be extended easily to different client types. We investigated how this is achieved to handle H.323 terminals, SIP clients and as well as MBONE clients.

We then propose a streaming gateway to investigate how flexible this protocol is in integrating videoconferencing systems and streaming systems which are two different technologies in multimedia streaming and collaboration. We demonstrate this for RealMedia [39-41] streams and also investigate other client types such as cellular phone clients that can be included in the system.

We also extend the XGSP framework to include RTSP functionality to support archiving and replay of streams. We propose Archived Streaming Service which provides metadata support and messaging middleware topic management for archiving streams and replaying archived streams.

1.2.2 Metadata Description and Management

We divide metadata into two categories; static metadata and dynamic metadata. Static metadata is the metadata that does not change during the lifetime of the session while dynamic metadata changes. We investigate session and stream metadata from these two categories.

We describe all metadata in XML format and store them in WS-Context Service [42] which provides a distributed fault tolerant metadata repository that can be shared among services and clients in the system. We investigate the impact of this on the design and architecture of the streaming service.

1.2.3 Multiple Stream Support

Since we will use messaging middleware, particularly NaradaBrokering publish/subscribe system, we propose developing new services and extending some existing ones in order to provide archive and replay streaming functionality within NaradaBrokering system.

We transport data whether it is audio/video or text or any binary data (i.e. image data) with the NaradaBrokering native event called “NB Event”. The data is encapsulated inside the event and published to a topic. At this stage, the only parameter needed to provide streaming functionality for the events published to a topic is the timestamp information so that we can preserve the ordering and timespacing between the events. This lets us provide functionalities such as seeking within a stream, playing a stream at its original rate etc. The services that we propose are as follows:

- *Time Service*: Unsynchronized clocks in a distributed environment will result in inconsistency in timestamping events generated from different sources. In order to avoid this inconsistency in archiving and replay services, we propose using the time service within messaging environment which is provided to entities in the messaging system so that entities can timestamp the events. We will investigate the behavior of local clocks on Linux and Solaris machines.
- *Jitter Reduction Services:* Jitter is considered to be one of the most important quality of service measurements within audio/video collaborative systems. As a part of these services we propose a Buffering Service and Time Differential Service. These two services require the use of the Time Service mentioned above. We will demonstrate how Jitter Reduction Service using the following services is effective in reducing jitter in an intercontinental environment which we set up between Cardiff, UK and Bloomington, IN, USA. We investigate the delay introduced by this service to the stream.
    1. *Buffering Service*: The goal of this service is to order events that clients receive. Unordered events mean that the original order has changed. Especially if the publisher and the subscriber are located at a large distance from each other it is possible that some of the later events reach the subscriber earlier than the former.

2. *Time Differential Service*: The ordering of events is not sufficient to reduce jitter. If the timespacing of events is not preserved this will also increase jitter. We propose Time Differential Service that timespaces events before releasing them to the subscriber. This service is another means to preserve the rate at which the stream is published.

We also propose Generic Streaming Framework which can archive and replay any event type within the messaging environment. This framework also provides guidelines on how to archive and replay raw RTP packets as well as major RTSP functionalities (play, pause, rewind and forward).

We also investigate how we can synchronize streams using the services proposed in this research.

We also propose a replication service by introducing a replication scheme for messaging systems that can store events in redundant repositories to improve the fault tolerance of the system in the presence of repository failures.

1.2.4 Application Scenarios

We investigate application scenarios from different collaboration systems. From videoconferencing systems, we will archive and replay GlobalMMCS [43-49] sessions. GlobalMMCS sessions contain audio and video streams. We will also archive and replay sessions from Anabas eLearning and Collaboration [50] system. Anabas is a web based collaboration system which provides many different tools such of shared display, text messaging and audio conferencing. These two collaboration systems demonstrate different characteristics, since in GlobalMMCS sessions the number of streams (hence topics) changes as clients join or leave sessions, in Anabas there is a fixed number of topics.

Another important application area is annotation systems. We initiate an annotation system named “eSports Collaborative Multimedia Streaming and Annotation System” based on the streaming architecture proposed in this dissertation. In eSports, we will replay real-time live A/V streams and annotate them. We shall also replay annotated streams with the annotations.

1.3 Organization of the Thesis and Summary of the Chapters

In the next chapter we provide background information on topics such as videoconferencing standards and systems, streaming technologies and protocols used. We will also give the overview of NaradaBrokering messaging middleware which we use to implement our streaming architecture.

In chapter three, we will give the overview of GlobalMMCS collaboration system and explain XGSP Session Server. GlobalMMCS is the prototype system that is built around XML Based General Session Protocol (XGSP) framework. XGSP Session Server is the session management unit within GlobalMMCS that implements XGSP management and conference control framework. XGSP Sessions Server provides an administrative interface for session activation and deactivation, controls XGSP sessions over media servers and helps application endpoints to join and leave sessions. We will explain the control framework, messages exchanged with clients, gateways and administrator components and how XGSP Session Server manages media servers.

Chapter four presents the XGSP Streaming Gateway. XGSP Streaming Gateway extends XGSP framework by defining and designing a novel Streaming Gateway in order to deliver audiovisual streams in a XGSP session to streaming media clients. Streaming format chosen to implement the idea presented in this chapter is RealMedia format which is a proprietary format of RealNetworks [23]. But the idea can be implemented for other streaming formats as well.

In chapter five, we will explain the Time Service which we incorporated within NaradaBrokering messaging middleware to achieve synchronization of streams from different sources. This Time Service utilizes Network Time Protocol (NTP). Entities running on different machines in NaradaBrokering environment can timestamp events with NTP timestamps which provides more consistent timestamps than local clocks. We also provide test results that we run on different Linux and Solaris machines to show different clock behaviors.

In chapter six, we will present the Jitter Reduction Service we developed for NaradaBrokering messaging system. Jitter is usually introduced to events due to the underlying network delay or processing node latencies when the events are traversing to the client. Jitter is a quality-of-service criterion in streams. Jitter Reduction Service reduces jitter to a minimum. Jitter Reduction Service is composed of Buffering Service which time orders events and Time Differential Service which preserves the timespacing between events while releasing them to client. The timestamps used in events are NTP timestamps that are provided by Time Service explained in chapter five. We provide test results in Local Area Network (LAN) environment and in Wide Area Network (WAN) environment.

NaradaBrokering provides reliable delivery of messages for entities within the system. However, it does not ensure reliable delivery if there is repository failure. In chapter seven, we present our strategy to make this scheme even more failure resilient, by incorporating support for repository redundancy. Each repository functions autonomously and makes decisions independently. If there are N available repositories, reliable delivery guarantees will be met even if N-1 repositories fail. We present test results for different topologies explained in the chapter.

In chapter eight, we present one of the components of the archive and replay streaming architecture, Archive and Replay Service which manages archive and replay streaming sessions by providing session metadata and topic information during session initiation. An important aspect of this architecture is that it utilizes a fault tolerant distributed metadata repository, WS-Context Service, to maintain session and stream metadata. In this chapter we also explain how we obtain GlobalMMCS real-time videoconferencing session metadata and GlobalMMCS archive manager which initiates recording of GlobalMMCS sessions.

Another component of the streaming architecture is explained in chapter nine, Generic Streaming Framework. In chapter nine, we explain initiating media delivery for streaming sessions; delivery and control of the archive and replay stream. This Generic Streaming Framework is independent of media type and format, so it provides a generic delivery mechanism which can be used not just for multimedia content but also for any type of streaming content. It utilizes messaging middleware to achieve this generic framework. It supports both archiving and replay of streams with replay of real-time live streams.

Chapter ten provides an application scenario where we use all of the services and components described in this thesis. We present “eSports Collaborative Multimedia Streaming and Annotation System” which is built on top of the archiving and replay streaming architecture we explained in this thesis. eSports system is an annotation system that can be used in distance education especially for distance sports training or education. eSports system can provide new features, which are not found in other annotation systems, with this streaming architecture. In eSports system clients can collaboratively and synchronously play and archive real-time live video, take snapshots, and annotate those snapshots using whiteboard application. Annotated stream can be replayed synchronously with the original video stream.

Finally in chapter eleven, we provide concluding remarks, the contribution of this thesis and future directions.

Chapter 2

Background

2.1 Videoconferencing

The well-known solutions for multimedia collaboration framework are H.323, SIP and the Internet Engineering Task Force (IETF) Multi-Party Multimedia (MMUSIC) [51]. IETF MMUSIC's proposed solution, Simple Conference Control Protocol (SCCP) [52], targeted lightweight conference management for multicast. In the year 2000, Multiparty Multimedia Session Control (MMUSIC) Working Group (WG) gave up and removed conference control from WG charter, since multicast was not deployed widely in the Internet. The Access Grid project started for videoconferencing over multicast internet using MBONE tools, VIC and RAT, but they try to define and use their own conference control framework rather than using SCCP.

These standards, hence collaboration and conferencing systems based on them, cannot communicate with each other. So the collaboration environment is a homogenous environment where all clients use and understand one common protocol to communicate with each other. It is difficult to extend the system to other types of client from other systems. Moreover it is difficult to extend the videoconferencing session to include sessions from other videoconferencing systems. Among collaboration systems VRVS provides an environment that different clients can join in a videoconferencing session. In this section we will examine videoconferencing standards and some major systems used in research and group collaboration.

2.1.1 H.323

H.323 is an International Telecommunications Union (ITU) standard aiming multimedia communication over Local Area Networks (LAN). H.323 [2] is defined as an umbrella standard which specifies the components to be used within an H.323-based environment. It provides conference management functionality for audio/video conferences using the call signaling functionality of H.225 [53], H.245 [54]. H.225 and H.245 provide call set-up and call transfer of real-time connections to support small-scale multipoint conferences. The H.243 [55] protocol defines some commands between the H.323 Multipoint Control Unit (MCU) and H.323 terminals to implement audio mixing, video switch and cascading MCU. Codecs used within H.323 system are G.711 [56] for audio, H.261 [57] and H.263 [58] for video and T.120 [59] for data. T.120 recommendation contains a series of communication and application protocols. It also includes services to support real-time, multi-point data applications in collaborations sessions.

H.323 system is designed for packet switched networks. The components of H.323 are:

- Terminals are endpoints that provide audio/video/data to another endpoint.
- Gatekeepers (GK) provide admission and call control services to endpoint. They also provide services such as address translation, RAS control, call redirection and zone management to H.323 participants.
- Multipoint controller (MC) establishes a H.245 control channel with H.323 participant to negotiate media capabilities.
- Multipoint processor (MP) provides media switching and mixing functionalities.
- Multipoint control unit (MCU) is an endpoint that enables three or more endpoints to participate in a conference.
- Gateway (GW) provides real-time, two-way communication between H.323 endpoints and non-H.323 endpoints.

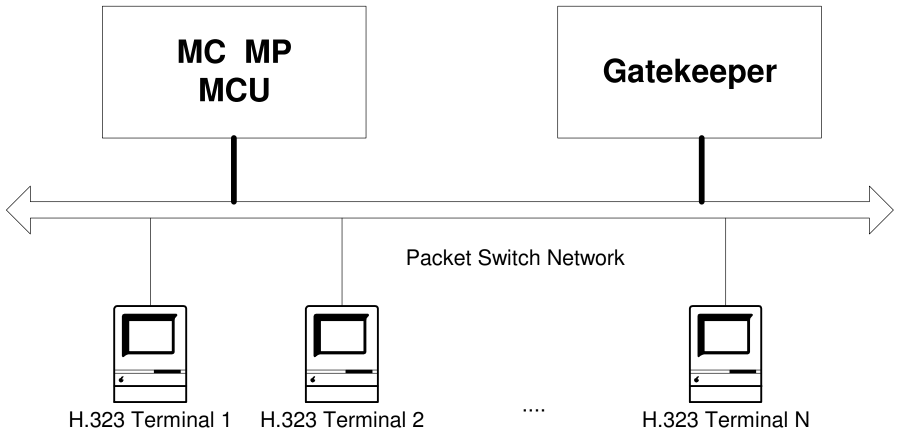


Figure 2-1: Typical H.323 architecture

H.323 protocol defines how these components communicate with each other. H.323 control communication is in binary format. Figure 2-1 shows the architecture of a typical H.323 system.

2.1.2 SIP

The Session Initiation Protocol (SIP) is a text based and HTTP-like style (request-response) application layer protocol for establishing, modifying and terminating sessions. SIP was designed to solve problems for IP telephony. It provides functions such as user location resolution, capability negotiation, and call management. SIP capabilities are basically equivalent to the services H.225 and H.245 in H.323 protocol. Although SIP does not define the conference control procedure like H.243, there is some research for SIP based conference control protocol [60, 61].

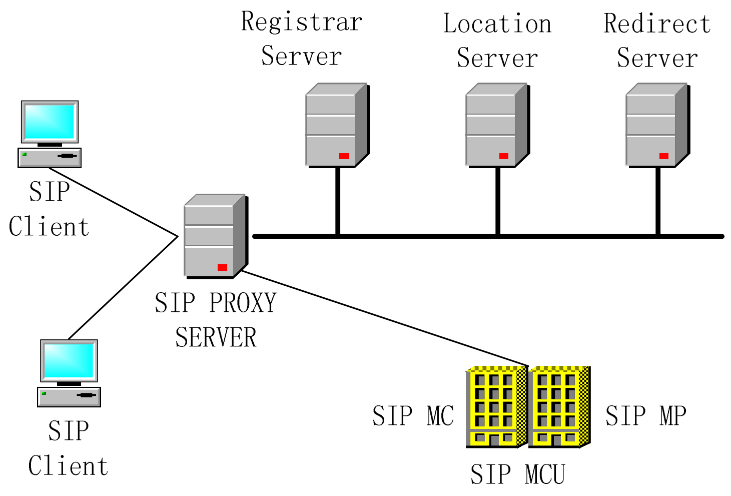


Figure 2-2: Typical SIP based system architecture

A typical SIP system is shown in Figure 2-2. Main components of a SIP system are: SIP clients, a SIP Proxy Server, a Registrar Server, a Location Server, a Redirect Server and a SIP MCU. The SIP Proxy Server primarily plays the role of routing and enforcing policy of call admission. It provides an instant messaging service, forwarding SIP Presence Event messages and SIP text messages to SIP clients. The SIP registrar accepts REGISTER requests and saves the received information in location server.

2.1.3 The Access Grid

Multicast systems are suitable for high bandwidth users. So they can concurrently receive many media streams. Such systems do not need a media distribution server. Participants send their audio and video streams to a multicast address and every client receives those streams through multicast enabled routers. However this does not work for low bandwidth clients, because they cannot receive many multicast streams at once. As a consequence there needs to be a unicast bridge to stream those multicast streams to the client.

The Access Grid uses MBONE tools and can support a large scale audio/videoconference based on a multicast network. The Access Grid provides group-to-group collaborations among more than 100 nodes connected to world wide Internet. Most of these nodes are educational and research institutions.

The Access Grid uses separate multicast IP address and port pair for video and audio streams. Access Grid uses MBONE Videoconferencing Tool (VIC) for video and Robust Audio Tool (RAT) for audio. These tools can connect to these multicast IP address/port number pair and receive streams. A permanent virtual meeting room in Access Grid is called a “virtual venue”. Virtual venues are used for collaboration services management. Users are allowed to establish their own venue server which hosts the information about the user registration and venue addressing and offers rendezvous service to all the users. Users have to log onto the venues server and start the multimedia clients in their nodes for communication through multicast and unicast bridges.

The Access Grid 2.0 provides a security mechanism through Globus Toolkit's [62] authentication and user identification mechanisms [63]. Although they can encrypt RTP streams with a shared key and authorize users, one can send malicious traffic to the multicast session and overwhelm the bandwidth of participants.

The Access Grid provides a chat application for node operators, a distributed power point and whiteboard application.

2.1.4 The Virtual Rooms Videoconferencing System

The Virtual Rooms Videoconferencing System (VRVS) is a web-oriented system for videoconferencing and collaborative work over IP networks. Using VRVS, users from disparate geographic locations can meet and participate in MBONE, H.323, SIP or MPEG2 multipoint videoconferences. It also supports QuickTime player. Participants can make use of different collaborative tools (sharing their desktops, broadcast any local application, participate in a chat, etc.). VRVS also integrates MBONE and H.323 tools with the Access Grid Virtual Venues. H.323 Clients are connected to the Virtual Rooms by the H.323–VRVS Gateway.

VRVS builds its collaboration service on top of pure software reflector infrastructure which is a kind of software multicast. Reflectors play an important role in VRVS design. A reflector is a host that connects each client to a Virtual Room by a permanent IP tunnel. The reflectors for a virtual room have many IP tunnels among themselves and clients multiplex these tunnels. The reflectors and their links form an overlay network through which audio, video or data flows. The use of the reflector technology assures the quality needed for videoconference transmissions. Reflectors are responsible for transmitting the streams. VRVS is not an open project and has few documents detailing its architecture and conference control framework.

2.2 Streaming

Videoconferencing systems provide a framework to send/receive audio and video streams. On the other hand there are also some clients who do not need to send streams to interact with another client but only receive a stream. These clients can receive streams with some delay since there is no need to interact with other clients. Also these clients do not want flickering in the video or noise in the audio if the bandwidth to the client is variable. Streaming provides a framework to deliver media across large distances over the Internet while minimizing the problems that clients may have.

Streaming media aims to improve the quality of service of the stream, by using various streaming codecs and a buffering mechanism to reduce the jitter. The buffering mechanism causes the stream to be delayed for some time. Hence, streaming media does not provide a good interactive environment compared to the interactivity in a videoconferencing system, but the stream quality is improved.

Another feature of streaming that distinguishes it from conferencing is that streaming uses compression algorithms that cause streams to be compressed more so that less bandwidth is required in order to transmit a stream across the network. The side effect of this is that compression/decompression requires high CPU utilization because it involves much computation. Also the buffer delay in streaming is much longer than the buffer delay in conferencing. While in conferencing it is recommended that mouth-to-ear delay of audio be less than 300 msec [64], the delay in streaming buffer in today's streaming players is around 2 – 15 sec.

2.2.1 Real Time Streaming Protocol

A streaming client connects to a streaming server primarily using Real Time Streaming Protocol (RTSP) [14], to establish a session and receive the stream. Streaming is primarily used for Media-On-Demand that is receiving media that resides on a streaming server whenever a client wants to play. RTSP is a client-server multimedia presentation control protocol and is stateful. It provides VCR-like control, so that clients can pause, fast forward, reverse, and absolute position etc.

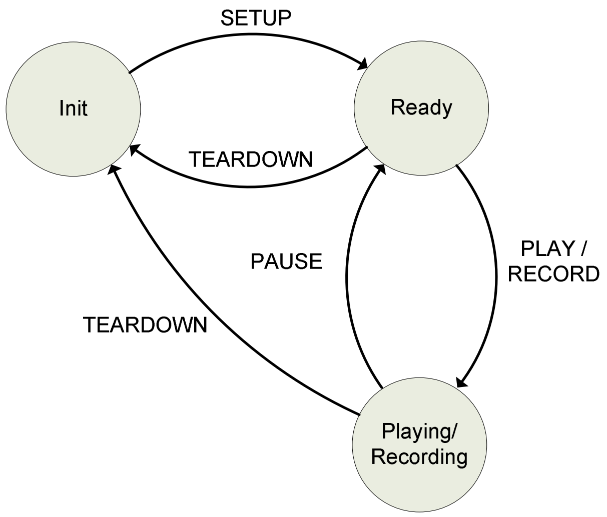


Figure 2-3: RTSP states

RTSP is a protocol specified in the Internet Engineering Task Force's RFC 2326 for control over the delivery of live or stored data with real-time properties. RTSP is similar in syntax and operation to HTTP/1.1 [65], which allows RTSP to be extended by HTTP, but as opposed to HTTP, RTSP maintains state by default. States involved in

RTSP are init, ready, playing/recording. Basic RTSP control functionalities are play, pause, seeking (absolute positioning to a specific point in the stream) etc. While some methods cause state change, some do not. RTSP states are shown in Figure 2-3.

RTSP uses Session Description Protocol (SDP) [66] to describe the multimedia session. It is also used for session announcements and session invitation. SDP is a text based protocol. Some of the metadata included in the SDP description are multimedia session owner information, stream information such as media type and media format, human readable session description, session begin time and end time, URLs for session resources.

RTSP does not mandate any data packet format for sending media to client but usually standard Real-Time Transport Protocol (RTP) [15] is used. Some vendors like RealNetworks may have their own data packet formats, RealNetworks' Real Data Transport (RDT) [67]. This information is exchanged during RTSP session setup.

There are other streaming protocols such as Windows Media Services HTTP 1.1 (WMS HTTP 1.1) [68] which is also an extension of HTTP 1.1 with similar functionality to RTSP.

Figure 2-4 displays RTSP signaling between client and server. State of the connection describes which methods the client can request. Depending on the state, client may be limited to some of the methods provided to it.

Any type of media format can be used for data. RTSP Server is responsible for sending data to client. In this case it has to be capable of understanding the media format of the stream, that is, it has to support the format otherwise it cannot process the request.

In case of a heavy load on the server, the server can redirect the client to another RTSP server. The client needs to be provided with the address of the new RTSP server in order to make requests to the new server. RTSP does not describe any load balancing mechanism among servers; the client needs to find one that can accept its request through trial and error.

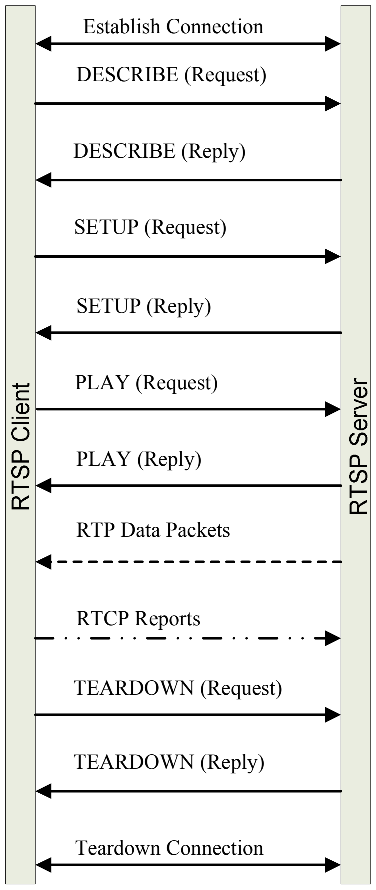


Figure 2-4: A typical RTSP session

2.2.2 Handling Streaming Formats

Streaming servers may support different media formats and encodings such as RealNetworks' RealMedia [39-41], Apple's QuickTime [22], Microsoft's WindowsMedia [69] or Moving Picture Experts Group (MPEG) audio and video formats [70] and AVI format [71]. Vendor specific encodings are usually supported by a vendor's streaming servers while other encodings may be supported by any streaming server. But streaming servers are not required to support all of the formats and encodings. For this reason, vendors have their own specific clients to play their specific formats and encodings.

A streaming server may use files or multimedia storages to archive and replay streams. Multimedia storages are special type of storage designed specifically to achieve high performance during retrieval of multimedia files. Audio and video data consists of frames that must be retrieved in sequence in order to play continuously. So media frames need to be grouped into storage blocks in a continuous manner to improve storage performance. There are two approaches to store multimedia data described below.

In the first approach the media server needs to know the details of the encoding and the format in order to map media frames to storage blocks. For example, an MPEG-encoded video contains I, B, and P frames. While I frames can be decoded independently, P frames depend on the previous I frame and B frames depend on the preceding and the following I or P frame. Consequently the multimedia server needs to know the details of the standard.

In the second approach a multimedia file is considered as a stream of bytes and is partitioned into fixed size storage blocks. In this case during replay the average compression ratio of the multimedia file, which also depends on the media format and encoding, is taken into account while reading the storage blocks during the delivery of the media to the client.

2.3 Annotation Systems

There is several annotation systems developed for digital video. Some of them are used in a stand-alone environment in which the annotations can be saved and shared asynchronously and some others allow collaborative annotation.

IBM's VideoAnnEx Annotation Tool [25] is based on MPEG video and MPEG-7 [72-75] metadata framework. It takes an MPEG video file and segments it into smaller units called shots. It allows each shot to be annotated with static scene descriptions, key object descriptions and event descriptions. It stores those annotations as MPEG-7 descriptions in an XML file and associates them with the original MPEG video. MPEG-7 file is replayed along with the corresponding MPEG file to show the annotations on the original MPEG video.

Microsoft Research Annotation System (MRAS) [26] is an asynchronous annotation system which lets its users download a lecture along with the comments added by other users such as lecturers and students. After adding the annotation to the lecture it is saved onto the annotation server.

Classroom 2000 [27] captures interactions with cameras, microphones and pen-based computers to annotate slides during a typical lecture. All activities are captured and recorded with timestamps. After recording is done, students can replay the recorded lecture.

Intelligent Video Annotation System (iVas) [28] system is a stand alone system that can associate archived digital video clips with various text annotations using the client server architecture, The system analyzes the video content to acquire cut/shot information and color histograms. Then it automatically generates a Web document that allows the users to edit annotations.

Synchronous Multimedia and Annotation Tool (SMAT) [29] is a collaborative annotation system which allows users to collaboratively add annotations to multimedia contents such as archived video clips using text and whiteboard.

2.4 Time Services and Event Ordering

Time ordering of events generated by entities existing within a distributed infrastructure is far more difficult than time ordering of events generated by a group of entities having access to the same underlying clock. Because of the unsynchronized clocks, the messages (events) generated at different computers cannot be time-ordered at a given destination if local time is used for timestamping.

2.4.1 Computer Clocks

On a computer, there are two types of clocks, hardware clock and software clock. Computer timers keep time by using a quartz crystal and a counter. Each time the counter is down to zero, it generates an interrupt, which is also called one clock tick. A software clock updates its timer each time this interrupt is generated. Computer clocks may run at different rates. Since crystals usually do not run at exactly the same frequency, two software clocks gradually get out of sync and give different values. The accuracy of computer clocks can vary due to manufacturing defects, changes in temperature, electric and magnetic interference, the age of the oscillator, or computer load [76]. Another issue is that an ill-behaved software program can use the timer's counter and change the interrupt rate. This could cause the clock to rapidly gain or lose time. A software clock loses state when the machine is turned off and it synchronizes itself with the hardware clock at reboot. Furthermore, hardware clock itself may not be synchronized with the real time and may be seconds, minutes or even days off. Finally, a software clock's accuracy would be bounded by the accuracy of the hardware clock at the reboot.

Because of hardware or software reasons, computer clocks may run slower or faster than they should. An ideal clock is a clock whose derivative with respect to real time is equal to 1. If this derivative is smaller than 1 it is considered a slow clock, and if this derivative is greater than 1 it is considered a fast clock.

Let the clock value on a machine *m* at real time *t* be $C_m(t)$. If the clock is a slow clock then $dC_m(t)/dt < 1$. If the clock is fast clock then $dC_m(t)/dt > 1$. If the clock is ideal clock then $dC_m(t)/dt = 1$. This relationship is depicted at Figure 2-5.

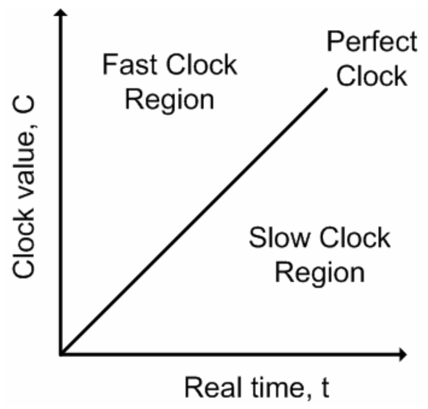


Figure 2-5: Relation between local computer clock time and real time. Adapted from ref. [77].

It is thus necessary to synchronize clocks in a distributed system, if the time-based order of events matter. One cannot rely on the underlying hardware or software clocks to provide synchronization. Hence, time ordering of events generated by entities existing within a distributed infrastructure is far more difficult than time ordering of events generated by a group of entities having access to the same underlying clock.

2.4.2 Event Synchronization in Distributed System

Different approaches exist to synchronize events in a distributed system. One of them is to use logical clocks, which was first presented by Lamport [78], and the other one is to synchronize system clocks so that clocks running on different machines are synchronized with each other.

Using logical clocks guarantee the order of events among themselves. They do not need to run at a constant rate, but they must increase monotonically. Using Lamport timestamps, Lamport synchronizes logical clocks by defining a relation called "happens-before". Vector clocks [79, 80] have also been introduced because Lamport timestamps cannot capture causality. But a major drawback is that vector clocks add a vector timestamp, whose size is linear with the number of processes, onto each message in order to capture causality. Vector clocks thus do not scale well in large settings.

Various algorithms have been devised to synchronize physical clocks in a distributed environment. In Cristian's algorithm [81], all of the machines in the system synchronize their clocks with a time server. Each machine asks the current time from the time server by sending a message to it. The time server responds to that message including its current time as fast as it can. Time server in Cristian's algorithm is passive. In Berkeley algorithm [82], time server is active. It polls every machine periodically, and then computes the average time based on the times received from them and tells them to adjust their times to the recent computed time. Another type of synchronization algorithm is the averaging algorithm, which are also known as decentralized algorithms. An example to a decentralized algorithm might be to average the time received from other machines. Each machine broadcasts its time and when the resynchronization interval is up for a machine it computes the average time from the samples received in that time interval. In averaging those times one might just take the average of them or can discard the *m* highest and *m* lowest samples and then average the rest.

Hardware approaches [83-86], are also available. But hardware approaches might require some custom made hardware components such as Network Time Interface (NTI) M-Module. NTI M-Module [86] is a custom very-large-scale integration (VLSI) chip that has interfaces to Global Positioning System (GPS) receivers. It uses the time received by GPS receivers to achieve synchronization. Obviously hardware solutions are expensive and might require changes to the underlying platform.

Network Time Protocol (NTP) [87, 88] can also be used to synchronize clocks in a distributed system.

2.4.3 Network Time Protocol (NTP)

There are also other solutions available, but one that we are most interested in is the Network Time Protocol (NTP). NTP is one of the most widely used algorithms in the Internet. NTP uses filtering, selection and clustering, and combining algorithms to adjust the local time. NTP receives time from several time servers. A filtering algorithm selects the best from a window of samples obtained from a time server. Selection and clustering algorithms pick best truechimers and discard the falsetickers. Combining algorithm computes a weighted average of the time offset of the best truechimers. An adaptation of NTP, Simple Network Time Protocol (SNTP) [89] can also be used to synchronize computer clocks in the Internet. The major difference between SNTP and NTP is that

SNTP does not implement the algorithms mentioned above. It just uses the time obtained from a time server.

NTP daemons are implemented for Linux, Solaris and Windows machines and are also available online [90]. These NTP daemons sometimes adjust the system clock every 1 sec. This synchronization interval might be too frequent, and can place strains on bandwidth and CPU utilization. The decision of the synchronization interval is also another issue. Setting this to a high value might cause the clocks to get out of sync too much while setting this to a low value might cause performance degradation. If two clocks need to be synchronized with δt time apart, then the synchronization interval, Δt, should be chosen as $\delta t/(\sigma_1+\sigma_2)$ where σ_1 is the drift rate[1] of $clock_1$ and σ_2 is the drift rate of $clock_2$ [77].

2.5 Replica Services

The virtual synchrony model, adopted in Isis [91], works well for problems such as propagating updates to replicated sites. This approach does not work well in situations where the client's connectivity is intermittent, and where the clients can roam around the network. Systems such as Horus [92] and Transis [93] manage minority partitions and can handle concurrent views in different partitions. The overheads to guarantee consistency are however too strong for our case. Spinglass [94] employs gossip-style algorithms, where recipients periodically compare the message digest of the received message with one of the group members. Deviations in the digest result in solicitation

[1] The maximum drift rate of a hardware clock is provided by the manufacturer and indicates how many microseconds a hardware clock drifts apart from real-time per second.

requests (or unsolicited responses) for missing messages between these recipients. This approach is however unsuitable when memberships are very fluid and hence a recipient is unaware of other recipients that should have received the same message sequences.

Distributed Asynchronous Computing Environment (DACE) [95] introduces a failure model that tolerates crash failures and partitioning, while not relying on consistent views being shared by the members through a self-stabilizing exchange of views. DACE achieves its goal through a self-stabilizing exchange of views through the Topic Membership protocol. This however may prove to be very expensive if the number and rate at which the members change their membership is high. The Gryphon [30] system uses knowledge and curiosity streams to determine gaps in intended delivery sequences. This scheme requires persistent storage at every publishing site and meets the delivery guarantees as long as the intended recipient stays connected in the presence of failures. It is not clear how this scheme will perform when most entities within the system are both publisher and subscribers, thus entailing stable storage at every node in the broker network. Furthermore it is conceivable that the entity itself may fail and the approach does not clearly outline how it handles these cases.

Since message queuing products (MQSeries) [35] are statically pre-configured to forward messages from one queue to another they generally do not handle network changes (node/link failures) very well. Furthermore, these systems incur high latency since they use the store-and-forward approach, where a message is stored at every stage before being propagated to the next one. Queues need to also recover within a finite amount of time to resume operations. The WS-ReliableMessaging [96] specification provides a scheme to ensure reliable delivery of messages between the source and the sink for a given message. The specification provides an acknowledgement based scheme to ensure that data is transferred reliably between the communicating entities. The specification, though it is for point-to-point communications, supports composition and interoperates with specifications pertaining to policies, transactions, coordination and metadata exchanges. Also of interest is WS-TransmissionControl, which provides a set of constructs controlling message exchanges between services to improve reliability.

The Data Replication Service (DRS) [97], within the Globus Toolkit, leverages the Replica Location Service which is a distributed registry that keeps track of replicas on storage systems, and facilitates queries to locate replicated files. The Storage Resource Broker (SRB) [98] is a middleware that provides applications a uniform interface to access heterogeneous distributed storage systems. It utilizes a metadata catalog called MCAT which manages descriptive and system metadata associated with data collections and system resources. Both DRS and SRB transfer and replicate files and rely on a separate service to locate replicated files. In our system, we replicate messages. Messages are stored at repositories where the underlying stable storage could be based on databases or flat-files with no need to maintain a separate registry or metadata service to manage the replications.

2.6 NaradaBrokering

NaradaBrokering [99-105] is a distributed messaging infrastructure based on the publish/subscribe paradigm. It provides two capabilities. First, it provides a message oriented middleware (MoM). Second, it provides a notification framework by efficiently routing messages from the originators to only the registered consumers of the message.

Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized messages (*events*). These events can encapsulate information pertaining to transactions, data interchange and system conditions.

The NaradaBrokering substrate provides several advantages over traditional multicast. NaradaBrokering relies on a software multicast for communications. This prevents the need for MBONE which is required for multicast communications. The NaradaBrokering substrate provides support for transport protocols such as the Transmission Control Protocol (TCP), Parallel TCP, the User Datagram Protocol (UDP), Multicast, HTTP and the Secure Sockets Layer (SSL). It also supports communications across network address translation (NAT) and firewall/proxy boundaries.

A drawback of multicast is the need to negotiate a unique multicast group for every collaborative session. For example for N entities within a system there could conceivably be 2N multicast groups. In publish/subscribe systems multiple collaborative groups are typically managed through the use of different topics for publishing and subscribing to messages.

NaradaBrokering allows clients to register their subscriptions using a variety of formats. The subscriptions can be in the form of String, Integer, Long and <tag, value> based topics, or in the form of XPath, SQL and Regular expression queries. Support for this variety of subscription formats also implies richer collaborative interactions since actions may be triggered only under very precise conditions. The complexity of managing these subscriptions and routing relevant messages is delegated to the middleware substrate. Since the individual entities do not need to cope with the complexity of constraints, this in turn facilitates easier development of collaborative applications which enables these complex interactions.

Chapter 3

GlobalMMCS Overview and XGSP Session Server

XML Based General Session Protocol (XGSP) conference control framework is a generic and easy to extend framework that defines signaling protocol for H.225, H.245 (H.323 signaling protocols) and SIP as well as the Access Grid. It also provides services to A/V and data application endpoints and communities, controlling multipoint A/V RTP and data channels.

Global Multimedia Collaboration System (GlobalMMCS) is the prototype system that is built around XGSP framework. XGSP Session Server is the session management unit within GlobalMMCS that implements XGSP management and conference control framework. It also provides a development environment to extend XGSP framework for videoconferencing systems.

XGSP Session Server provides an administrative interface for session activation and deactivation, controls XGSP sessions over media servers and helps application endpoints to join and leave sessions. Also it supports gateways to let H.323 and SIP clients join and leave sessions. XGSP Session Server also manages media server elements.

In this chapter we will present an overview of GlobalMMCS and will explain XGSP Session Server. XGSP Streaming Gateway support which provides RealMedia streams to RealPlayer clients will be explained in chapter 4.

3.1 GlobalMMCS Design

In GlobalMMCS, session management unit and media processing units are separated from each other. In H.323, these two functionalities are handled by H.323 MCU which includes both multipoint controller and multipoint processor.

GlobalMMCS uses NaradaBrokering messaging middleware to transport data between components in the system including clients in the sessions. As we will explain managing media processing unit in section 3.2.4 NaradaBrokering provides GlobalMMCS components to transport their audio/video data with special events called RTPEvents.

Figure 3-1 shows the components of GlobalMMCS. The GlobalMMCS media processing unit shown in the figure is explained in ref. thesis [106]. The XGSP Web Server provides an easy-to-use web interface for users to join multimedia sessions and for administrators to perform administrative tasks. In addition, users can start some audio and video clients through these web pages such as VIC, RAT and Real Player. XGSP Session

Server which is the session management unit of GlobalMMCS and implements XGSP control framework for videoconferencing sessions is explained throughout this chapter. Gateways which bridges GlobalMMCS sessions to legacy RTP clients such as H.323, SIP or Access Grid clients are explained in section 3.3.

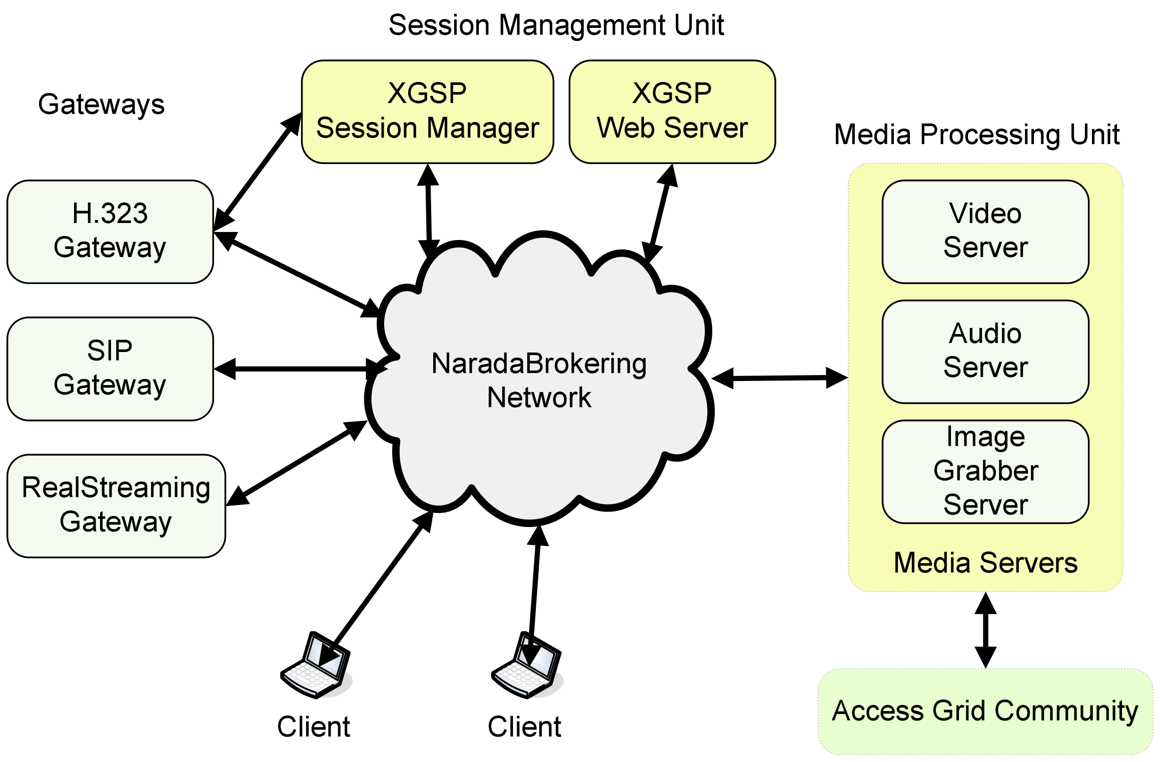


Figure 3-1: GlobalMMCS overall architecture

XGSP defines XML messages for control signaling among components in GlobalMMCS videoconferencing sessions. A request to XGSP Session Server requires an action in the session by the XGSP Session Server. When we explain XGSP Session Server, we will also explain the messages and the actions taken by the server. Figure 3-2 below shows XGSP Session Server components. XGSP Session server has a control unit for each XGSP session. These session control units also use media server management instances such as VideoSession and AudioSession instances to manage audio and video servers.

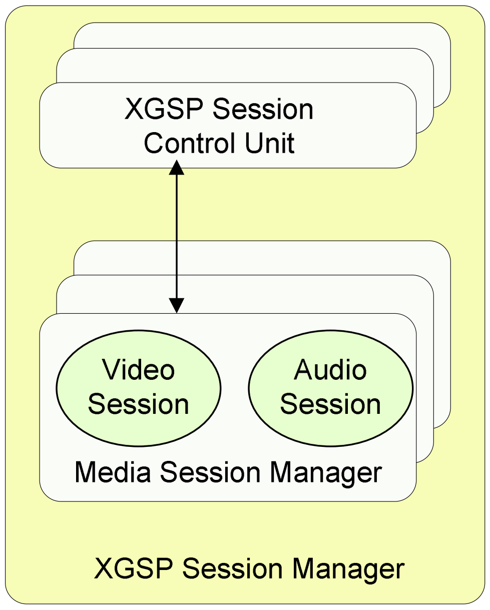


Figure 3-2: XGSP Session Server components.

3.2 XGSP Session Management

XGSP sessions are managed by XGSP Session Server together with the XGSP Conference Manager, which is also referred as XGSP Web Server. XGSP Conference Manager maintains a calendar system to schedule and advertise meetings and also provides administration web pages.

3.2.1 NaradaBrokering Communication Topics

XGSP Session Server uses NaradaBrokering topics for message exchange with other components in the GlobalMMCS. The topics used are listed in Table 3-1. Throughout this chapter we will use topic name column to point the topics used.

Topic Name	Topic	Components Using
TOPIC_SS_WS	"/xgsp/av/ss-ws"	XGSP Session Server and Web Server
TOPIC_SS_CL	"/xgsp/av/"+sessionID+"/ss-client"	XGSP Session Server and clients in the session with sessionID
TOPIC_CL_SS	"/xgsp/av/client-ss"	Client and XGSP Session Server
TOPIC_SS_RG	"/xgsp/av/streaming/ss-rg"	XGSP Session Server and Streaming Gateway
TOPIC_RG_SS	"/xgsp/av/streaming/rg-ss"	Streaming Gateway and XGSP Session Server
TOPIC_SS_RC	"/xgsp/av/streaming/ss-rc"	XGSP Session Server and Streaming clients
TOPIC_SS_HG	"/xgsp/av/streaming/ss-helix"	XGSP Session Server and Helix Gateway

Table 3-1: NaradaBrokering communications topics used by XGSP Session Server

3.2.2 XGSP Sessions

XGSP Session Server manages audiovisual sessions in GlobalMMCS. XGSP Session Server maintains session descriptions defined by the administrator. A session is described as an XML description as shown in Figure 3-3. Session media description includes audio and video session media descriptions together. Media is described by the media type, media format and transport information as shown in Figure 3-4. Each session has audio and video streams. XGSP Session Server handles these streams separately but a client may have both audio and video streams and can join audio and video sessions of a XGSP session at the same time. How XGSP Session Server handles these clients will be explained while we explain join and leave operations in section 3.2.5.

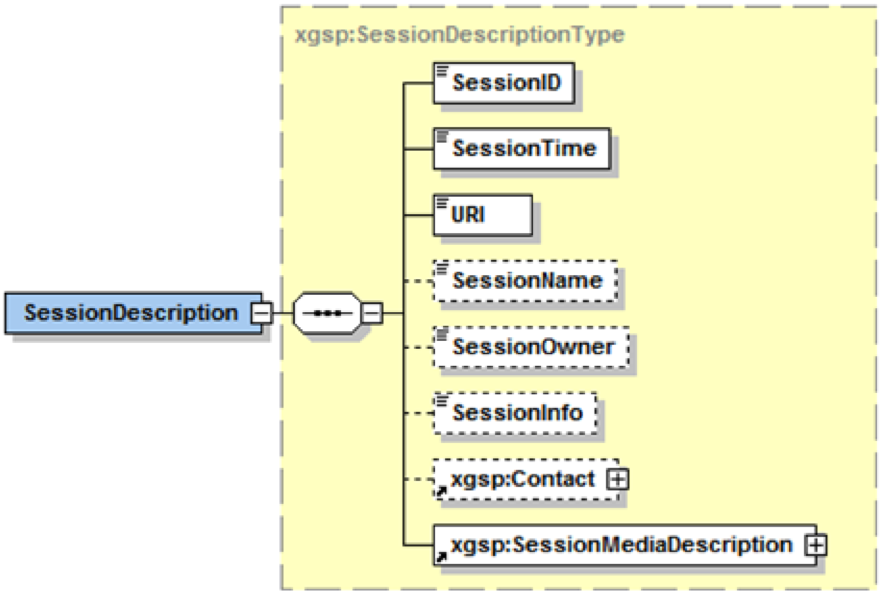


Figure 3-3: Representation of XGSP Session XML description

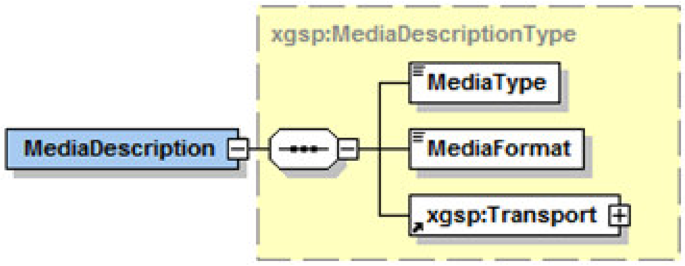


Figure 3-4: Representation of MediaDescription XML message used for both audio and video sessions media descriptions and client media descriptions

Media description provides address of a multicast session. If this field is not provided, the XGSP session is considered a unicast session. Usually the multicast session is an Access Grid room.

The Access Grid rooms have separate multicast addresses for audio and video streams. For this reason a session description has fields for both types of streams. XGSP Session Server passes these multicast addresses to media servers so they can receive audio and video streams in that Access Grid room. Section 3.2.4 will explain this further.

3.2.3 Activate and Deactivate XGSP A/V Sessions

Based on the session schedules specified by the administrator, the XGSP Conference Manager will ask XGSP Session Server to activate/deactivate XGSP sessions. The topic TOPIC_SS_WS shown in Table 3-1 is used for message exchange between XGSP Session Server and Conference Manager. Once the session is activated, XGSP Session Server reserves necessary resources such as AudioSession and VideoSession instances to process requests issued. XGSP Session Server only processes further administrator and user requests for activated sessions.

XGSP Session Server expects an ActiveSession message with fields shown in Figure 3-5. The Conference Manager sends an activation command to the XGSP Session Server to activate an audiovisual session. The XGSP Session Server generates the associated AudioSession and VideoSession instances to manage audio and video servers. We will explain managing media servers in section 3.2.4.

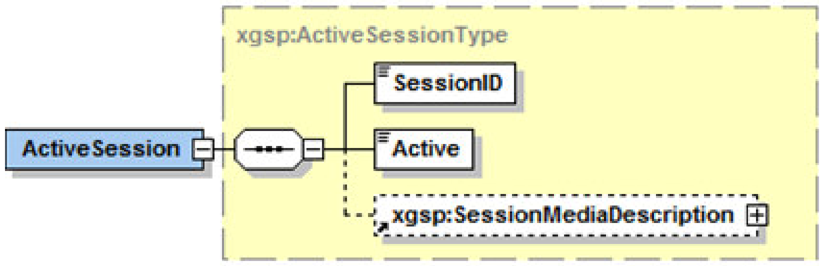


Figure 3-5: Representation ActiveSession XML message.

Once a session is activated by XGSP Session Server, Conference Manager needs to send a deactivate-session message to XGSP Session Server to close the session and release resources reserved by XGSP Session Server. Deactivate-session message is usually sent by Conference Manager when the time for the session is over. Administrator can also send it before the session time is over. Deactivate-session message is the same message shown in Figure 3-5 with “Active” field set to false value.

3.2.4 Managing Media Processing Units

XGSP audiovisual session supports multiple kinds of clients which use RTP packets for data transportation. Among these clients are H.323, SIP and Access Grid clients. Since we use NaradaBrokering messaging middleware to transport these RTP packets to legacy RTP clients, a special NaradaBrokering event named RTPEvent is used to transport within NaradaBrokering messaging middleware. RTP packets are encapsulated into RTPEvents. RTPEvents can be routed within NaradaBrokering. When RTPEvents leave NaradaBrokering for their subscriber, they are transformed back into RTP packets. In order to achieve that NaradaBrokering sets up RTPLinks for every legacy A/V endpoint.

NaradaBrokering provides long type topics for RTPEvents to achieve a better data transport performance instead of using other topic types such as String or XML type topics. Since a legacy RTP client has both RTP and RTCP data, there are two topics associated to one stream.

Media servers are audio servers, video servers and snapshot (image grabber) servers. Details of the media servers, RTPEvents, RTPLinks and how media servers manage those links can be found in ref. thesis [106]. In this section after the overview above, we explain how XGSP Session Server interacts with media servers to manage audio and video processing units.

3.2.4.1 Managing Audio Processing Units

XGSP Session Server maintains an AudioSession instance for each XGSP session. Through AudioSession, it accesses audio servers and manages audio clients. An audio session can be either a multicast audio or a unicast audio session. If it is a multicast audio session, XGSP Session Server provides the multicast address to the AudioSession instance which requests initializations in audio server for that multicast session. Multicast audio clients are treated differently from other unicast audio clients. Audio server receives audio streams in the multicast audio session and adds them as a participant to the XGSP audio session.

Audio servers are audio mixers that mix all of the audio streams in that session into one audio stream, which is called a speaker. So clients in the audio session are added to the speaker to receive that mixed audio stream. New joining clients are added to the audio session. When a client leaves the XGSP session, the client is removed from the audio session.

Participant information is returned to AudioSession instance by audio servers. During initialization, AudioSession instance registers itself to audio server in order to receive the participant information. XGSP Session Server updates the inner metadata tables with these streams information.

When the XGSP session is deactivated, corresponding audio session is also ceased through AudioSession interface and resources on the audio server are terminated.

3.2.4.2 Managing Video Processing Units

Managing video servers is similar to managing audio servers in some way. XGSP Session Server maintains a VideoSession instance for each XGSP session and through this, it accesses video servers and manages video clients. A video session can also be a multicast session. As in a multicast audio session, multicast video session also contains multicast video clients. If XGSP session contains a multicast session address, VideoSession instance is initialized with that multicast address and video server receives those multicast streams and adds them as a participant to the video session.

A video client can usually send and receive one video stream. So the client needs to choose the video stream it wants to display. XGSP Session Server assigns a topic number (or IP/port number pair, if it is a legacy RTP client) for the stream and video server publishes the video stream selected by the client for the topic (or IP/port number pair) provided by XGSP Session Server.

XGSP Session Server can also add video mixers to video servers. Up to four streams can be added to a video mixer. Because video mixing is a CPU intensive process, the number of video mixers is limited on a video server. XGSP Session Server manages those video mixers as well. It can add any stream to and remove any stream from the video mixer. Video mixer output is a single video stream which is considered as a video publishing client by XGSP Session Server.

Video stream information is returned to VideoSession instance by video servers. During initialization, VideoSession instance registers itself to video server topics and receives participant information. XGSP Session Server updates its metadata with these streams information.

When the XGSP session is deactivated, the corresponding video sessions are terminated on the video server using VideoSession interface.

3.2.4.3 Snapshot Servers

Snapshot servers take snapshots from video streams. When XGSP Session Server generates a video session as described in section 3.2.4.2, a session is also generated on snapshot server for the activated XGSP session. When video stream information is received through VideoSession instance, URLs that point to those snapshots (JPEG images) are also received as part of the stream information. These URLs are sent to clients by XGSP Session Server to help them choose which streams they want to receive.

3.2.5 Join and Leave XGSP A/V Sessions

After the session is activated, users can join the session by sending JoinSession messages as shown in Figure 3-6 to the XGSP Session Server. The XGSP Session Server processes JoinSession request for both audio and video sessions. In JoinSession message, the MediaDescription field contains media information of the requesting client.

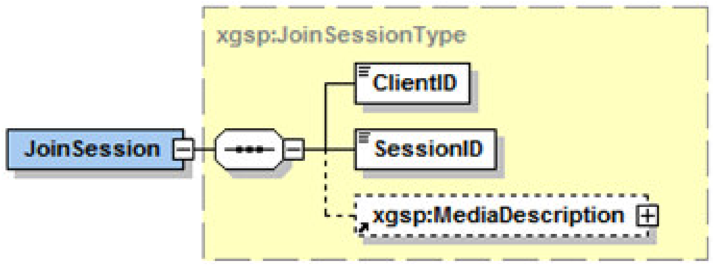


Figure 3-6: Representation of JoinSession XML message

When a speaker joins an audio session, a topic number (or IP address/port number pair for legacy RTP clients) is assigned for this user to publish its audio stream. For legacy RTP clients, XGSP Session Server passes the IP address/port number pair to audio server through AudioSession instance to receive the audio stream from the client. If the client is capable of receiving RTPEvents, it only subscribes to the mixed audio topic since there is only one mixed audio topic for each session. The interaction between the AudioSession and AudioMixerSession components is transparent to the user. If the joining user is a listener then it is only given the mixed stream topic number to receive the audio of all speakers in the session, for RTPEvent capable clients. Legacy RTP clients receive audio stream from the IP address/port number pair they provided to XGSP Session Server. Since they will not publish any audio, they are neither assigned a topic number, nor added to the mixer.

When a client joins a video session, it is assigned a topic number to publish its video stream. A legacy RTP client publishes its video stream to IP address/port number which is the IP address of the video server and available port number on that server. At the same time, an image grabber is started to construct the snapshots of the video stream. This user is given a list of available video streams in the session. The client can subscribe to these streams by sending VideoSourceSelection message to the XGSP Session Server. Video selection is explained in section 3.2.6.

When client wants to leave the session, it simply sends a LeaveSession message as shown in Figure 3-7 to the XGSP Session Server and the server removes it from the XGSP session and hence, from the corresponding media session by calling interfaces on AudioSession and/or VideoSession instances.

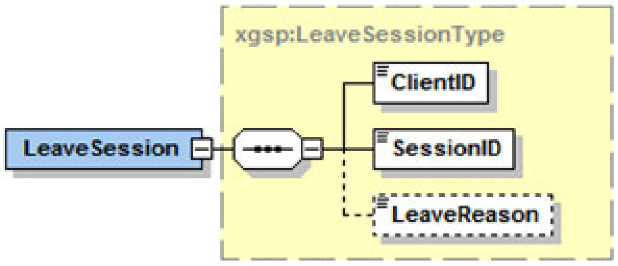


Figure 3-7: Representation of LeaveSession XML message

Both JoinSession and LeaveSession messages are sent to the XGSP Session Server from topic TOPIC_CL_SS by the client. Response message for these messages are SessionSignalResponse shown in Figure 3-8. The response is sent to the client from topic TOPIC_SS_CL. When it is a response to a JoinSession message, MediaDescription field in the response provides the session media description for the client so that it can publish the stream to the given topic or IP address / port number in the specified format.

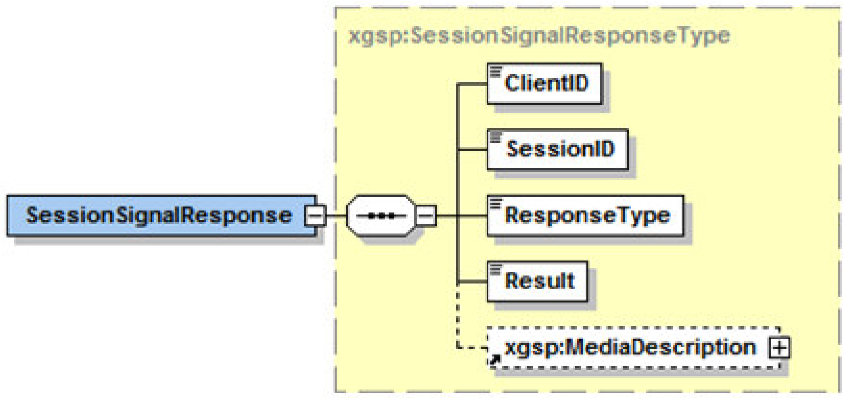


Figure 3-8: Representation of SessionSignalResponse XML message

3.2.6 Video Selection

In audio streaming, clients receive a mixed audio stream of all audio streams in the XGSP session. This cannot be expected for video streams. Usually the number of video streams that can be displayed by clients are limited to a few. Because of this XGSP sessions provide video selection functionality for clients in the XGSP session. When a client wants to select or deselect a video stream, it needs to send a VideoSourceSelection message (Figure 3-9) to XGSP Session Server. If the Active field is set to true, it means that the client wants to subscribe to the video stream. If it is set to false, it means that the client wants to unsubscribe from the video stream. If it is a subscription request, using VideoSession instance, XGSP Session Server tells the video server to subscribe the client to that video stream. If it is an unsubscription request, XGSP Session Server similarly tells the video servers to unsubscribe the client from the video stream. Client sends its requests from TOPIC_CL_SS topic to XGSP Session Server and receives the reply from TOPIC_SS_CL topic.

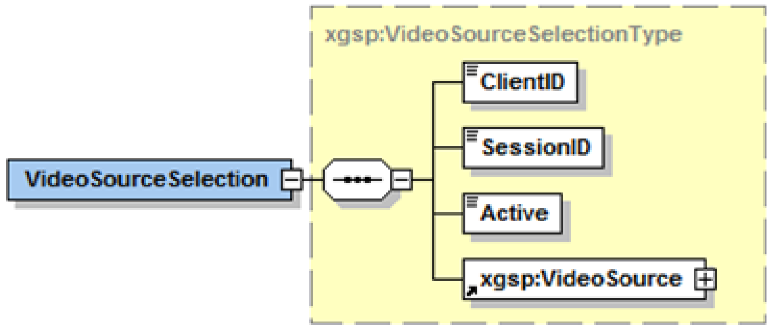


Figure 3-9: Representation of VideoSourceSelection XML message

3.2.7 Managing Video Mixers

In GlobalMMCS, there are VideoMixer servers which are part of the video server group. The video mixers combine up to four video streams into one video stream. The result is a new video stream. These video mixers are managed through XGSP Session Server by the administrator.

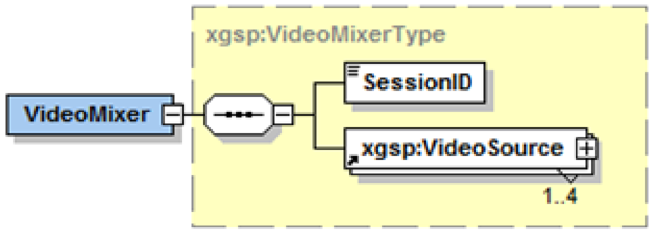


Figure 3-10: Representation of VideoMixer XML message

When the administrator wants to generate a mixed video, he first sends a VideoMixer message which is shown in Figure 3-10. He sets the Active field to true to indicate that the VideoMixer is either an existing mixer or a new mixer. XGSP Session

Server maintains video mixers metadata for each XGSP session. XGSP Session Server sends a VideoMixerReply message as shown in Figure 3-11 to the administrator upon receipt of the request.

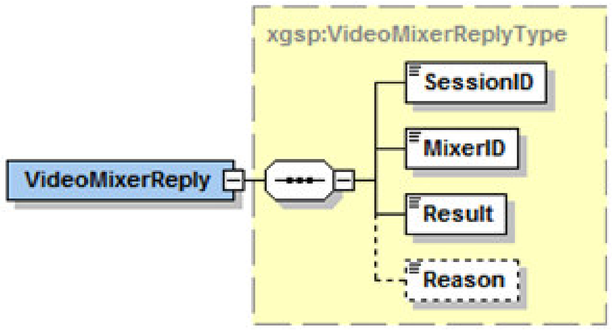


Figure 3-11: Representation of VideoMixerReply XML message

Each video mixer has a unique mixerID which works like clientID. XGSP Session Server accesses video mixers using mixerIDs. XGSP Session Server uses VideoSession instance to manage mixers on video servers. When a VideoMixer is generated, XGSP Session Server keeps the metadata of the mixed video, for example streams included in that video mixer. So when the administrator wants to modify video mixer streams, he just sends another VideoMixer message with the updated stream list. XGSP Session Server checks the list with the current video mixer stream list. If there are streams that no longer exist in the new list, it removes them from video mixer. Then it adds the new streams to the video mixer.

The message exchange between the XGSP Session Server and the administrator is done through TOPIC_SS_WS topic. Since the sender does not receive the message it publishes to the topic, the same topic can be used between two components.

3.2.8 Stream List

Administrator and clients need to know the streams in the session so that functionalities within the session can be provided to them. For example, clients need to know the metadata of the stream in order make a request. When a client gets this information it can see other clients in the session and can make successful video selections as explained in section 3.2.6. The administrator also needs to generate and modify video mixers in the session as explained in section 3.2.7. In order to do that, he also needs to know stream metadata.

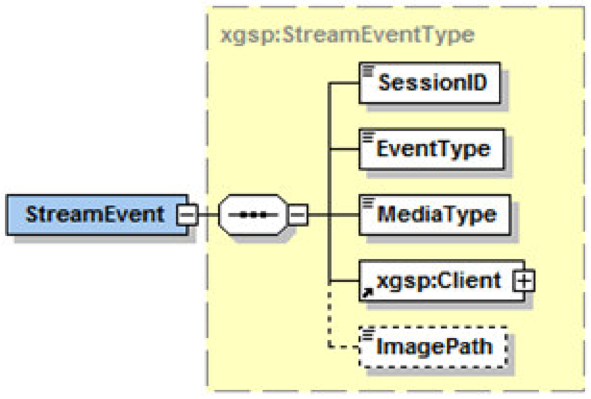


Figure 3-12: Representation of StreamEvent XML message

StreamEvent provides metadata of a stream is shown in Figure 3-12. In order to receive the stream list from XGSP Session Server, the administrator or the client sends a RequestAllStreams message as shown in Figure 3-13 to XGSP Session Server. XGSP Session Server waits for a fixed amount of time before it replies to the client or to the administrator. Since, there may be other requests from other clients who are just joining the session. When the time is up, XGSP Session Server sends RequestAllStreamsReply message shown in Figure 3-14 to a common topic in the session including audio and video stream list of the session. So clients making the stream list request receive the same message that contains the list of all streams in the session.

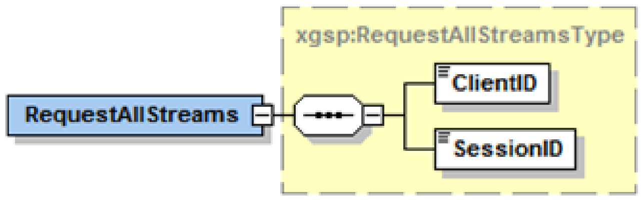


Figure 3-13: Representation of RequestAllStreams XML message

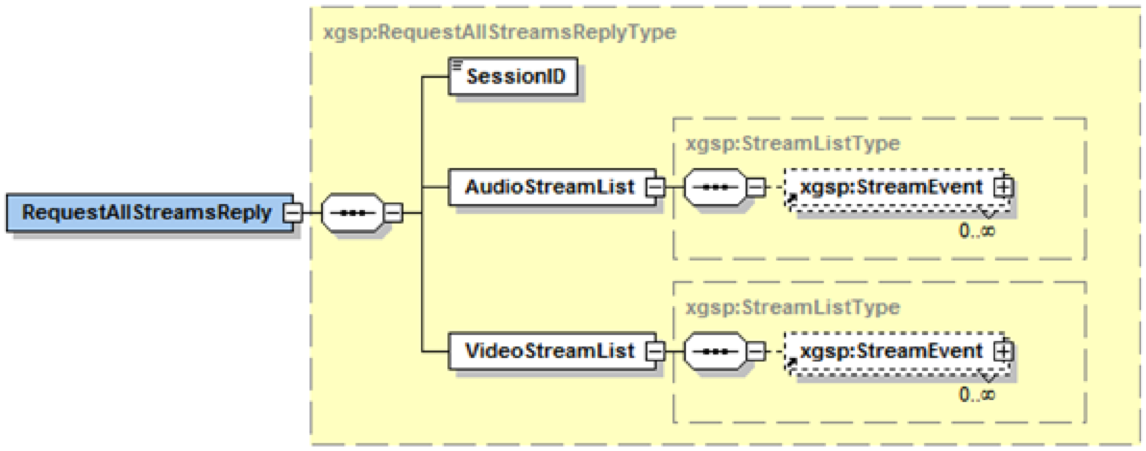


Figure 3-14: Representation of RequestAllStreamsReply XML message

3.3 Support for Gateways

GlobalMMCS has H.323, SIP and RealStreaming Gateways for adapting H.323 and SIP terminals and RealPlayer clients. These gateways play an important role for legacy H.323, SIP and RealPlayer clients. The XGSP Session Server needs to collaborate with these gateways to manage and control a session in this heterogeneous collaboration system.

To support H.323 and SIP audiovisual endpoints in XGSP A/V sessions, some transformations are necessary for both control and data transport. As we explained in section 3.2.4 there are RTPLinks to transport RTP packets within NaradaBrokering.

H.323 and SIP gateways enable H.323 and SIP terminals in order to let H.323 and SIP clients interact with other clients in the GlobalMMCS. They also provide H.323 / SIP clients H.323 / SIP conference control services within GlobalMMCS. The H.323 and SIP gateways transform H323 and SIP messages into XGSP signaling messages.

In this section we will explain the XGSP Session Server communication with H.323 and SIP gateways. Please refer to ref. [46] and ref. [47] for details of SIP Gateway and H.323 Gateway. XGSP Streaming Gateway is explained in chapter 4.

3.3.1 Support for H.323 Gateway

H.323 Gateway is used to translate H.323 messages into XGSP XML messages and it also hides the complexities of H.323 protocol from XGSP Session Server. XGSP Session Server does not need to understand the H.323 protocol. H.323 terminal also does not need to understand XGSP messages, since H.323 Gateway translates XGSP XML messages into H.323 binary messages and sends H.323 terminals necessary information based on the received responses from XGSP Session Server.

When a session is activated H.323 Gatekeeper keeps the alias name of the session, which is the sessionID of the active session. In addition to that, H.323 Gatekeeper also keeps H.323 terminal registrations. So when a H.323 terminal wants to join a XGSP session it needs to use the sessionID of the active session when calling H.323 MCU. H.323 Gatekeeper translates this sessionID into H.323 MCU address and routes the call to the registered H.323 MCU.

3.3.1.1 H.323 Call Setup and Termination

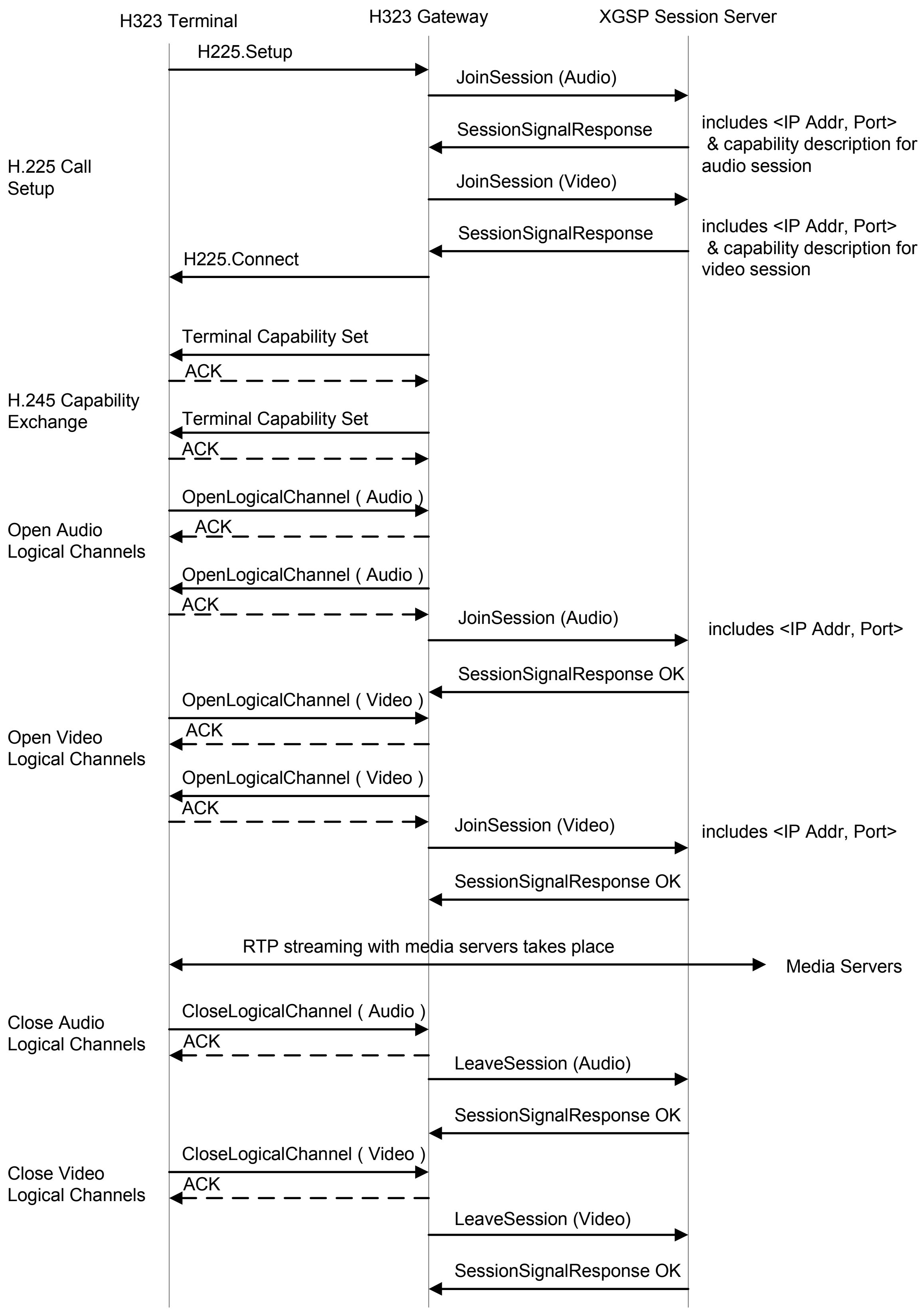
In order to establish a call between two endpoints, information needed is signaling destination address, media capabilities and media transport addresses at which both endpoints can receive RTP packets. When H.323 call is received by H.323 MCU, it parses the information received from client and exchanges XGSP XML messages with XGSP Session Server. Figure 3-15 illustrates a H.323 call setup with XGSP Session Server. This procedure has three important steps: H.225 call setup, H.245 capability exchange and audiovisual logical channel creation.

The first step in H.323 call is the call setup. At this step when H.323 Gateway receives a call setup from a H.323 terminal, it sends a JoinSession (Figure 3-6) message with the sessionID of the active session to tell XGSP Session Server that a H.323 terminal wants to join the XGSP session. If XGSP Session Server permits the client, it sends a SessionSignalResponse (Figure 3-8) as a reply while setting the Result field to OK and setting the media description of the session to let the client know what media formats are supported by the XGSP session. This step is repeated for each media type, audio and video as seen in Figure 3-15. At the end of this step H.323, Gateway maintains session media description for audio and video and H.323 terminal is considered as made a successful connection.

In the second step, H.323 Gateway and H.323 terminal exchanges the media capability. Client chooses one of the media formats supported by XGSP session. Usually XGSP Session Server provides only one media format which is supported by all clients in videoconferencing sessions, for instance for audio ULAW and for video H.261 formats.

In the next step, H.323 terminal makes a request to establish channel for one of the media types, i.e., audio. H.323 Gateway also sends H.323 terminal an open channel request for incoming stream. H.323 Gateway sends another JoinSession message to XGSP Session Server with media type and format provided by H.323 terminal. The same procedure is repeated for other media types, i.e. video. During the JoinSession procedure, XGSP Session Server maintains the state of the call. So it expects that the second JoinSession request will include the media capability of the H.323 terminal. It also waits for a fixed amount of time between audio and video channel establishments. If within that amount of time, H.323 Gateway does not send JoinSession message for the other media type, it concludes that the H.323 terminal will only receive and/or send one type of media.

When H.323 terminal wants to leave the session, the signaling between H.323 terminal and H.323 Gateway takes place according to H.323 protocol. H.323 Gateway sends LeaveSession (Figure 3-7) message to XGSP Session Server for the channels opened and XGSP Session Server unsubscribes H.323 terminal from media servers.



H323 Terminal
H323 Gateway
XGSP Session Server
H225.Setup
JoinSession (Audio)
SessionSignalResponse
includes <IP Addr, Port> & capability description for audio session
H.225 Call Setup
JoinSession (Video)
SessionSignalResponse
includes <IP Addr, Port> & capability description for video session
H225.Connect
Terminal Capability Set
ACK
H.245 Capability Exchange
Terminal Capability Set
ACK
OpenLogicalChannel (Audio)
ACK
Open Audio Logical Channels
OpenLogicalChannel (Audio)
ACK
JoinSession (Audio)
includes <IP Addr, Port>
SessionSignalResponse OK
OpenLogicalChannel (Video)
ACK
Open Video Logical Channels
OpenLogicalChannel (Video)
ACK
JoinSession (Video)
includes <IP Addr, Port>
SessionSignalResponse OK
RTP streaming with media servers takes place
Media Servers
CloseLogicalChannel (Audio)
Close Audio Logical Channels
ACK
LeaveSession (Audio)
SessionSignalResponse OK
CloseLogicalChannel (Video)
Close Video Logical Channels
ACK
LeaveSession (Video)
SessionSignalResponse OK

Figure 3-15: H.323 Call setup and termination with XGSP Session Server

3.3.1.2 H.323 Terminal Video Selection

XGSP Session Server provides audio and video stream lists in the session. Stream lists can be obtained as explained in section 3.2.8. Also a H.323 console can be launched from GlobalMMCS web interface. This console is a basic audio/video control client for H.323 terminal. Using this console, video stream displayed on H.323 terminal can be switched.

The video selection for XGSP clients is explained in section 3.2.6. For H.323 terminals, H.323 console can make the video selection requests. When a video stream is selected, H.323 console sends a VideoSourceSelection (Figure 3-9) message to XGSP Session Server. When a new video stream is selected and the request is processed by XGSP Session Server, H.323 terminal needs to be notified for this selection so that it can refresh its incoming video stream. For this reason, XGSP Session Server sends a VideoSwitch message as shown in Figure 3-16 to H.323 Gateway to notify H.323 terminal. Then H.323 Gateway notifies H.323 terminal to start video switching.

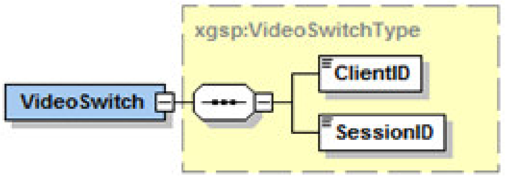


Figure 3-16: Representation of VideoSwitch XML message

3.3.2 Support for SIP Gateway

SIP Gateway is similar to H.323 Gateway. SIP does not support video streams, so SIP clients are only capable of receiving and sending audio streams. Because of this reason, SIP clients can only join XGSP audio sessions.

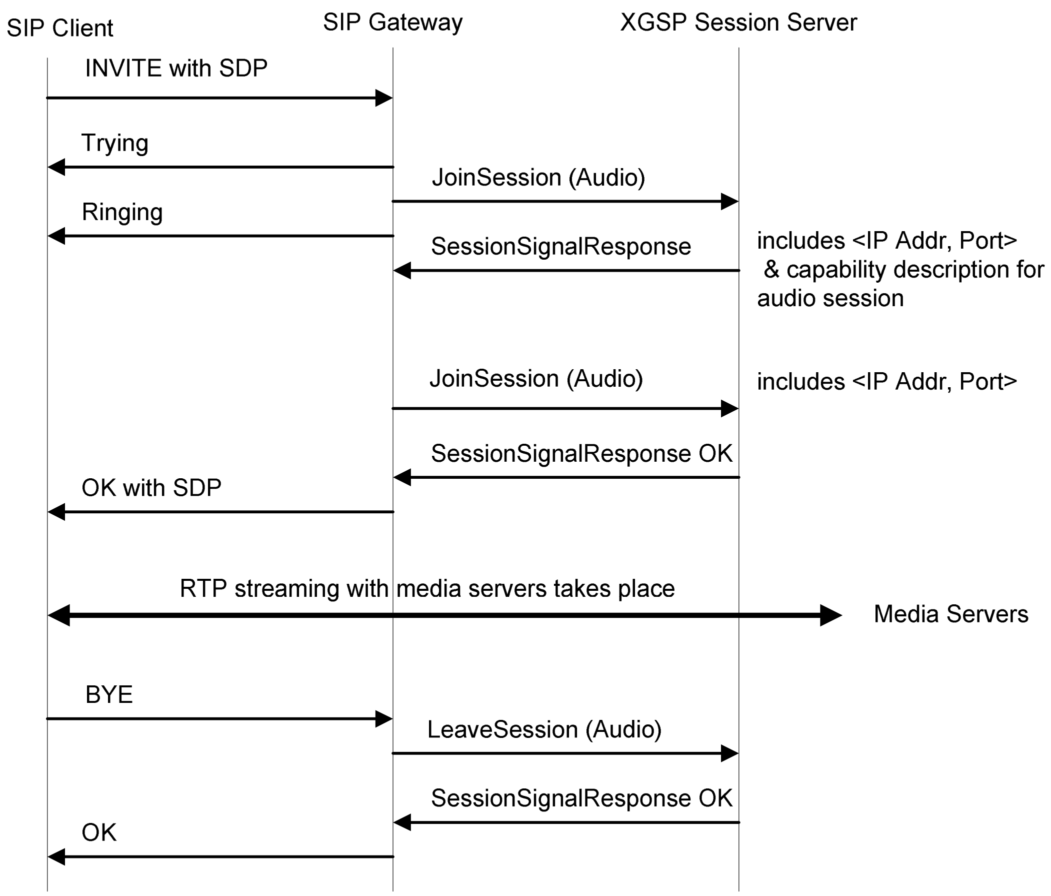


Figure 3-17: SIP client joining and leaving XGSP session

Since SIP is a simple text based protocol SIP Gateway can easily translate SIP messages into XGSP messages in a straightforward way. When SIP Gateway receives an INVITE request from SIP client, it sends a JoinSession message to XGSP Session Server as in the case when H.323 Gateway sends the first JoinSession message to XGSP Session Server, XGSP Session Server sends SessionSignalResponse to SIP Gateway specifying session media description. SIP Gateway receives the media description of the SIP client by parsing the SDP body in the INVITE message. If XGSP Session Server's response is OK, SIP Gateway sends another JoinSession message including an audio description of the client. When an SIP client leaves the session by sending a BYE message to SIP Gateway, SIP Gateway sends a LeaveSession message for the SIP client to the XGSP

Session Server. Then the XGSP Session Server unsubscribes the client from the audio server's AudioSession instance. This is illustrated in Figure 3-17.

3.4 Summary

In this chapter, we gave an overview of GlobalMMCS and explained its session management unit, XGSP Session Server. XGSP Session Server implements XGSP session management and conference control framework. It also manages media processing units for joining and leaving clients. This chapter also explained the interactions of XGSP Session Server with H.323 Gateway and SIP Gateway to describe how H.323 terminals and SIP clients join and leave XGSP sessions.

Chapter 4

XGSP Streaming Gateway

In the previous chapter, we explained XGSP framework for videoconferencing sessions. Videoconferencing systems provide a framework for send/receive audio and video streams. Clients can receive and play audio and video streams in the session with very little latency so that they can interact with each other. These clients usually require high bandwidth so that they can send and receive streams at the same time.

In this chapter, we explain XGSP Streaming Gateway which extends the XGSP framework with a novel XGSP Streaming Gateway to deliver A/V streams in a session to streaming media clients. We have developed Helix Streaming Engines to convert raw data into RealNetworks Streaming formats. Helix Streaming Server [107] is used by RealPlayers to receive those streams.

4.1 Streaming Gateway within XGSP and GlobalMMCS

Within the XML Based General Session Protocol (XGSP) conference control framework, XGSP Streaming Gateway has been introduced and XGSP has been extended with additional signaling protocols for Media-On-Demands systems to deliver the media in real-time videoconferencing systems.

Streaming Gateway is implemented as part of GlobalMMCS to verify and refine the extension we made to XGSP framework. The GlobalMMCS prototype system is shown in Figure 4-1 with XGSP Streaming Gateway added to the system. Other gateways and A/V processing components, such as the video mixer, audio mixer and image grabber servers are also shown in Figure 4-1.

XGSP Streaming Gateway displays characteristics similar to H.323 / SIP Gateways and A/V endpoints due to the streaming requirements. On one side, it extends the control framework similar to H.323 / SIP Gateways because input signal of XGSP Streaming Gateway is in XGSP format while output signals are in RTSP to negotiate with Helix Server, but on the other side it is like A/V processing endpoint because it needs to convert the received streams into RealStream format. Section 4.2 describes XGSP Streaming Gateway more in detail.

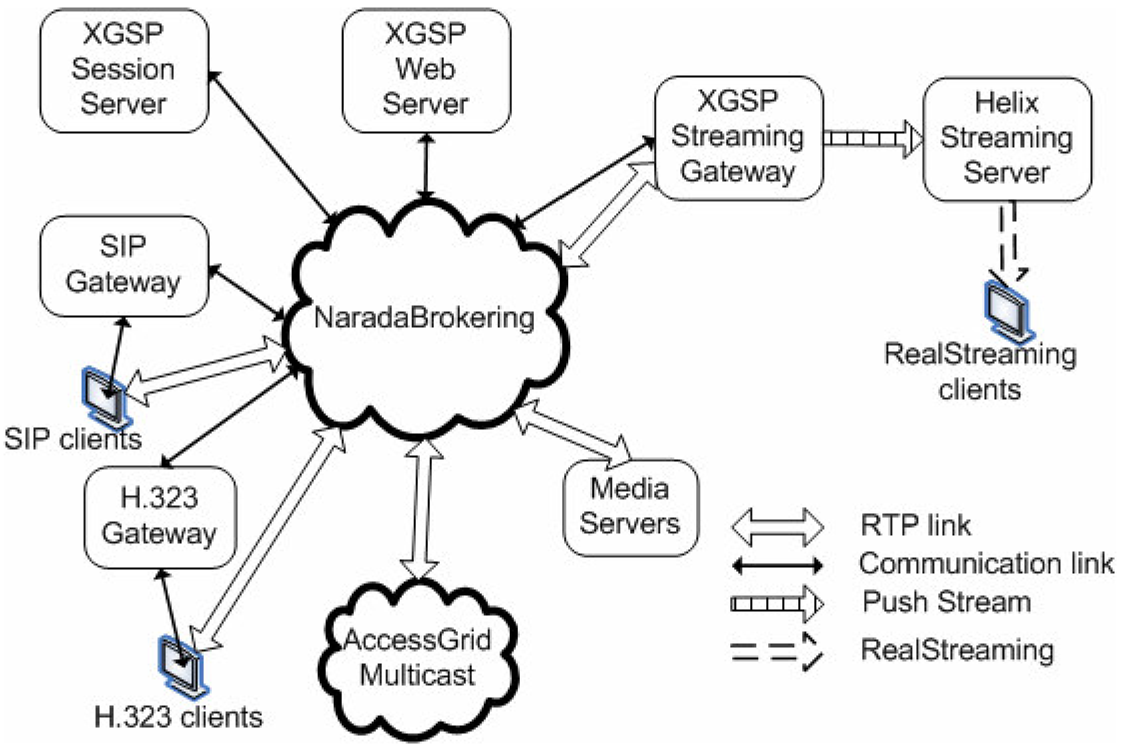


Figure 4-1: XGSP prototype systems

4.2 XGSP Streaming Gateway

The implementation of XGSP Streaming Gateway is different from other gateways, i.e. H.323 and SIP, because the requirements of RealStreaming clients are different than the requirements of other clients in the system. H.323 and SIP gateways transform communication signals from one format to XGSP messages or from XGSP messages to another format. Also they hide some communication details from XGSP Session Server. For example, H.323 is a complicated binary protocol because of the way it's implemented. The H.323 gateway transforms some H.323 binary messages into XGSP messages and from XGSP messages to H.323 binary messages and also hides the complexities from XGSP Session Server. XGSP Session Server only receives information that it requires from an H.323 client and sends information that will be needed by the client to join the session. Other signaling details are hidden from Session Server. SIP uses text messages but it requires different format than XML. Both protocols are signaling protocols and enable their clients to establish session and exchange streams. The Access Grid uses multicast, so AG clients just send/receive streams to/from a multicast address.

In RealStreaming case, clients require different stream format and use RTSP to receive streams. So, one of the jobs of XGSP Streaming Gateway is to convert the stream from formats like H.261 to RealStream format. This conversion process requires two transcoding stages: first step is to decode the received stream and the second stage is to transcode the output of the first stage into RealStreaming format. This conversion process causes delays in the generated stream. Other delays may be introduced from the system that XGSP Streaming Gateway is running and from the underlying network to transmit those streams to Helix Streaming Server. Due to these delays in the system and the network the generated A/V streams and other collaboration events should be synchronized. Streaming Gateway is also responsible for this synchronization.

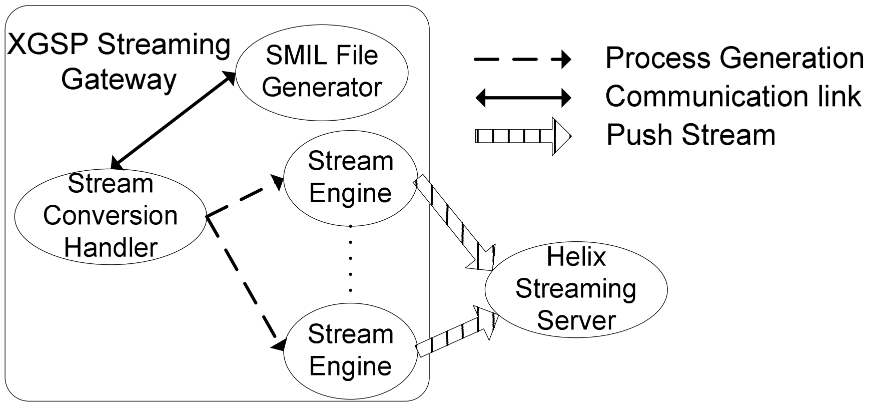


Figure 4-2: XGSP Streaming Gateway components

RealStreaming clients use RTSP to receive streams from RealStreaming Servers. We used Helix Streaming Server, which is available under RealNetworks public and community source licenses. XGSP Streaming Gateway does not only provide a signaling mechanism but also provides conversion mechanisms. This aspect of XGSP Streaming Gateway makes it different from other gateways. A RealStreaming client communicates with the server in order to establish a channel or channels to receive stream and to send/receive control information. Components of XGSP Streaming Gateway are shown in Figure 4-2.

4.2.1 XGSP Streaming Gateway Components and Design

Streaming Gateway is composed of several logical components: stream conversion handler, stream engine and Synchronized Multimedia Integration Language (SMIL) [108] file generator.

4.2.1.1 Stream Conversion Handler

Stream Conversion Handler handles the communication between XGSP Session Server and XGSP Streaming Gateway. It keeps an internal database for the streams being converted. This database is updated when the streaming jobs are started or deleted. In order to start a streaming job, it initiates a Stream Engine for the requested stream and passes required parameters such as conversion format, helix server address, stream name to the Stream Engine.

4.2.1.2 Stream Engine

The stream engine can be considered the most fundamental component of the XGSP Streaming Gateway because it is responsible for converting the received audio or video streams into a specified RealStreaming format and pushes the converted stream to Helix Streaming Server. Streaming Engine is composed of two parts, Java Media Framework (JMF) [109] RTP Handler and HXTA Wrapper. HXTA is a conversion engine provided by Helix Community [110] and converts raw audio and video data into RealStreaming formats.

JMF RTP Handler receives audio and video packets from a local port provided by Stream Conversion Handler. The purpose of this unit is to transform the received packets into a format that HXTA can understand and be able to make the conversion to RealStreaming format. Raw audio and video data can be passed to HXTA Wrapper. There are several color spaces for video representation. But two of the most common are RGB (red/green/blue) and YCrCb (luminance/red chrominance/blue chrominance). HXTA accepts different formats of RGB and YCrCb. As the first conversion step, JMF RTP Handler decodes the received video frames into YCrCb format. YCrCb format has been chosen, since RGB requires more memory to represent video images compared to YCrCb. JMF RTP Handler passes these decoded frames to HXTA Wrapper over a buffer. Audio packets are decoded into raw audio format, for our case WAV format, before passing it to HXTA Wrapper. Another responsibility of JMF RTP Handler is to make sure that packets are processed in an order. If a packet arrives later than a packet that carries bigger timestamp, it is dropped.

JMF RTP Handler gets the media type of the stream from the input provided by Stream Conversion Handler. Based on that information it decodes received packets into raw video or audio format. Each stream, whether it is audio or video, is transcoded into streaming format independently from each other.

4.2.1.3 SMIL File Generator

Streaming Engine receives only one stream, whether it is audio or video, and produces only one stream. In order to enable streaming clients to receive audio and video together, there is a need for SMIL file, which resides on the Helix Streaming Server. Because of this, Stream Conversion Handler provides RTSP links of audio and video streams to SMIL file generator and SMIL file generator produces a SMIL file that includes those RTSP links. SMIL File Generator and Helix Streaming Server share a common directory for generated SMIL files. In our current implementation we have only one audio stream per session, which is a mixture of all available audio streams in that session. So when a video stream is converted into streaming format, the RTSP link for the mixed audio stream is included to the generated SMIL file.

4.2.2 Helix Streaming Server

In our architecture, we have chosen to use Helix Streaming Server as a RTSP server. The main reason behind that is that the Stream Engine which converts streams into RealMedia format which is a proprietary format of RealNetworks and Helix Streaming Server is the one available that supports that format. Also in order to play RealMedia we need RealPlayer. RealPlayer can receive RealMedia stream from Helix Streaming Server and replay that stream in RealMedia format.

4.3 Streaming Gateway Message Exchange Types

This section explains XML message formats added to XGSP in order to enable the communication between XGSP Streaming Gateway and GlobalMMCS. In order to define the required parameters for a streaming job, an XML structure named *StreamingInput* is defined. StreamingInput fields are depicted in Figure 4-3. It provides the necessary information, such as broadcast address, stream name, stream server IP address and port number, media type, required to start a stream conversion job.

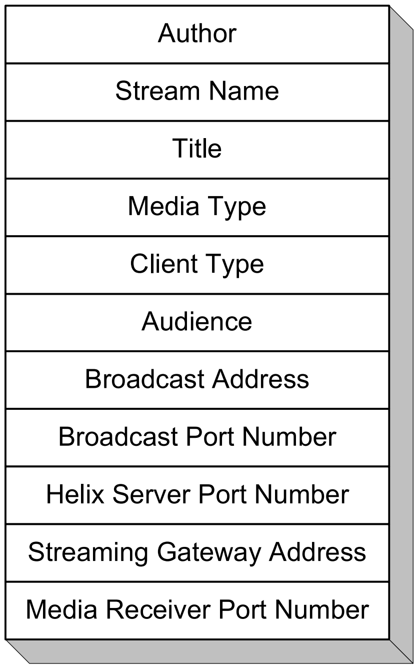


Figure 4-3: StreamingInput message fields

XML message formats added to XGSP are as follows:

InitializeRealGateway: This message causes Streaming Gateway to delete all current jobs in the session specified.

JoinStream: When XGSP Streaming Gateway receives this message, it starts a job for the stream specified. Besides *StreamingInput* information, the message also includes session ID and other stream specific information, such as username, SSRC. JoinStream message is depicted in Figure 4-4.

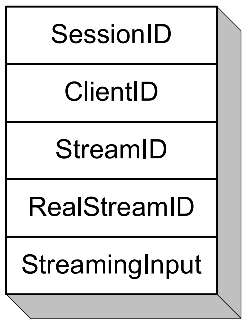


Figure 4-4: JoinStream message fields

JoinStreamReply: This is the reply message for JoinStream. If somehow the job cannot be started, it returns FAIL, otherwise it returns OK to indicate the job is successfully started.

LeaveStream: XGSP Streaming Gateway stops the stream specified when this message (Figure 4-5) is received.

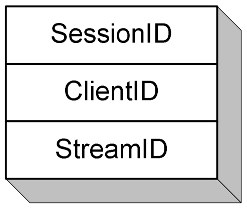


Figure 4-5: LeaveStream message fields

LeaveStreamReply: This is the reply message for LeaveStream. If somehow the stream is stopped successfully, it returns OK, otherwise it returns FAIL to indicate that an error occurred during the stop operation.

RealStreamEvent: It has two modes, NewRealStream and ByeRealStream. This event is generated by Session Server. Streaming lists are updated when this message is received. The stream is added to the list if the mode is NewRealStream, it is removed if the mode is ByeRealStream.

RealStreams: When a client/administrator joins to the session it requests the list of the RealStreams available in the session from Session Server by sending a RealStreams message.

RealStreamsReply: This is a reply message for RealStreams message. RealStreams are expressed in RealStreamEvent format and the list is included to RealStreamsReply message.

4.4 Interfaces Added to XGSP Session Server to Enable Communication

In order to support XGSP Streaming Gateway, XGSP Session Server provides some interfaces. These interfaces are RealStream_Join_Service, RealStream_Leave_Service, RealStream_List_Service and RealStream_Gateway_Service. XGSP Session Server also keeps a database for the streams being converted. The description of these services is as follows:

RealStream_Join_Service: This service handles JoinStream messages. XGSP Session Server includes additional information to JoinStream message and sends it to XGSP Streaming Gateway. If Session Server receives OK reply from XGSP Streaming Gateway, it sends RealStreamEvent message with NewRealStream mode to administrator and clients in that session.

RealStream_Leave_Service: This service handles LeaveStream messages. XGSP Session Server forwards the message to XGSP Streaming Gateway. XGSP Streaming Gateway sends LeaveStreamReply message to XGSP Session Server after processing the request. Then XGSP Session Server generates a RealStreamEvent with ByeRealStream mode.

RealStream_List_Service: When the streaming client/administrator first joins a session, it sends a RealStreams message to request a list of the available RealStreams on the XGSP Streaming Gateway. XGSP Session Server replies back with RealStreamsReply listing all the available RealStreams in the session.

RealStream_Gateway_Service: Streaming Admin can send InitializeRealGateway message to initialize the session for XGSP Streaming Gateway. XGSP Session Server forwards the message to XGSP Streaming Gateway and generates a RealStreamEvent with ByeRealStream mode for each RealStream in that session.

4.5 Signaling Between XGSP Session Server and XGSP Streaming Gateway

Figure 4-6 shows a signaling scenario among a streaming administrator, a streaming client, a XGSP Session Server and a XGSP Streaming Gateway. When the streaming administrator first connects to the XGSP Session Server, it requests a list of the available streams and RealStream streams in the session. Streaming administrator sends RequestStreamList, to request all of the available audio/video streams and RealStreams, consecutively. In the reply, the Session Server replies with RequestAllStreamsReply and RealStreamsReply. The streaming client only sends RealStreams message to receive available RealStream streams list. Next the administrator sends JoinStream for the chosen stream. XGSP Session Server adds some other fields to the same message and forwards it to XGSP Streaming Gateway. XGSP Streaming Gateway replies with JoinStreamReply and as a result, XGSP Session Server generates RealStreamEvent messages with NewRealStream mode and sends them to streaming administrator and streaming clients. LeaveStream has a similar scenario. Only instead of sending RealStreamEvent with NewRealStream mode, XGSP Session Server sends RealStreamEvent with ByeRealStream mode. In InitializeRealGateway case, XGSP Session Server forwards the message to XGSP Streaming Gateway and also sends RealStreamEvent with ByeRealStream mode for each of the streams removed.

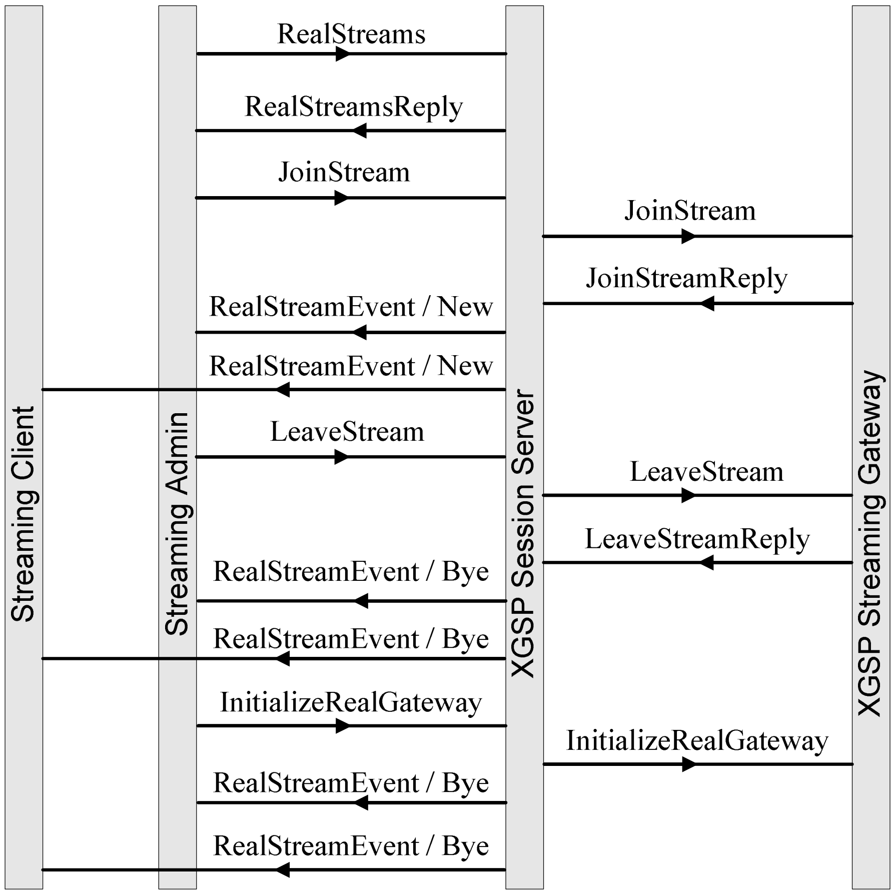


Figure 4-6: A sample signaling case among Session Server, Streaming Gateway and Streaming Client/Administrator

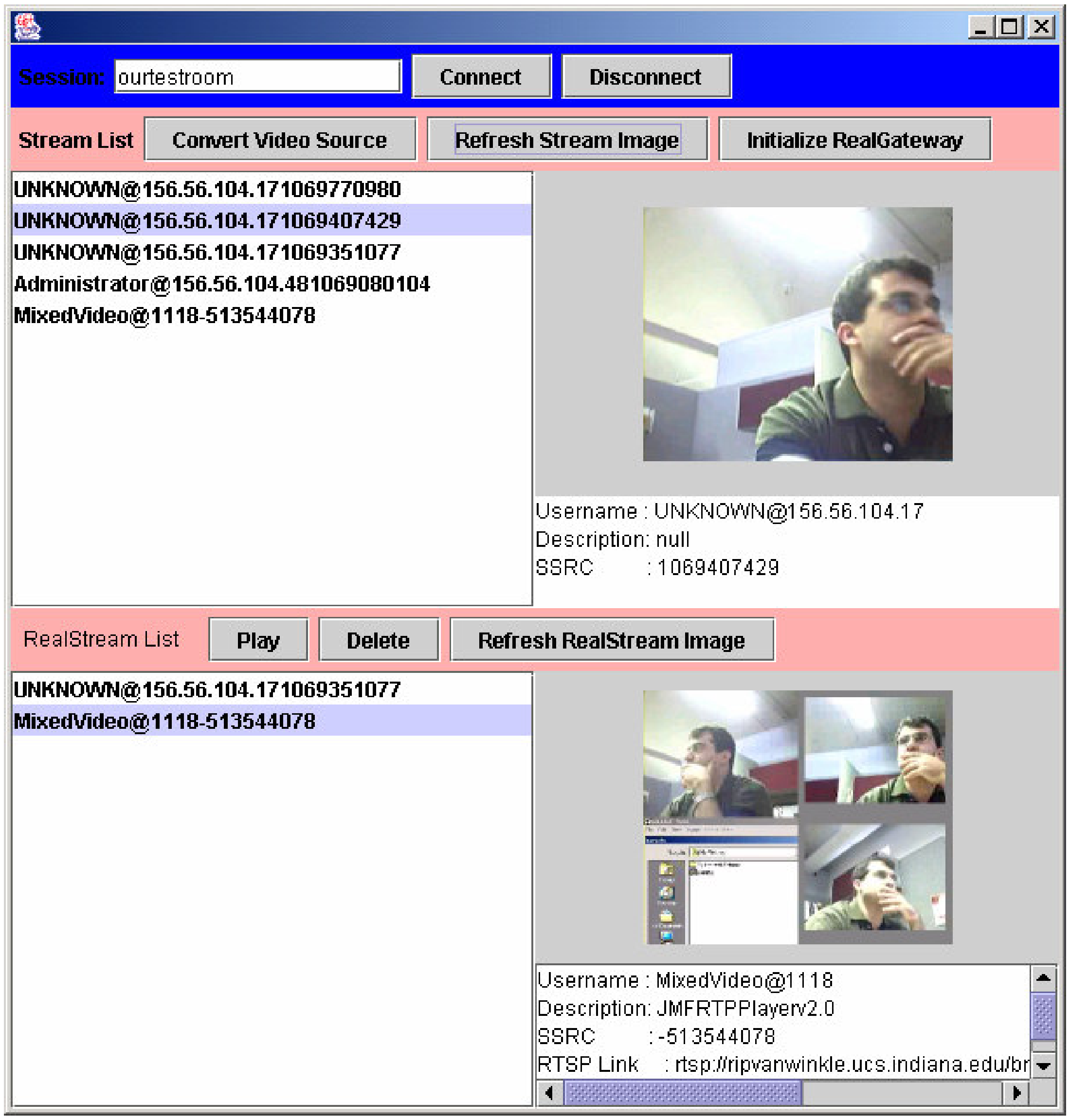
4.6 XGSP Streaming Gateway User Interface

Two types of interfaces are developed for XGSP Streaming Gateway. Streaming Admin and Streaming Client interfaces. As can be deduced from their names, Streaming Admin is implemented for administrative purposes and Streaming Client is implemented for accessing streams from an interface. Some part of the Streaming Admin interface is adapted from Streaming Client interface to enable the administrator with client capabilities.

4.6.1 Administrator Interface

Streaming Admin is designed and implemented to enable only the administrator to choose stream to be converted to RealStream. This conversion, especially for video streams, takes noticeable CPU percentage. Because of this and other administrative purposes, regular clients cannot be allowed to start and stop streaming jobs. The administrator can see the available streams in XGSP sessions. This enables the administrator to select streams to be converted.

As shown in Figure 4-7, the streams are listed by their unique identifiers (IDs). ID for each stream is constructed by appending the SSRC number to their usernames. The administrator can visualize the information for that stream by selecting it. Available image and other information; username, SSRC (a RTP field) and description, regarding a stream selected are also provided. After selecting the video source, the administrator can start the streaming job by clicking *Convert Video Source*. The south of the panel is adapted from the Streaming Client interface. In addition to playing streams, the administrator can stop a RealStream by selecting it and clicking *Delete* button.



UNKNOWN@156.56.104.171069770980
UNKNOWN@156.56.104.171069407429
UNKNOWN@156.56.104.171069351077
Administrator@156.56.104.481069080104
MixedVideo@1118-513544078

```
Username : UNKNOWN@156.56.104.17
Description: null
SSRC        : 1069407429
```

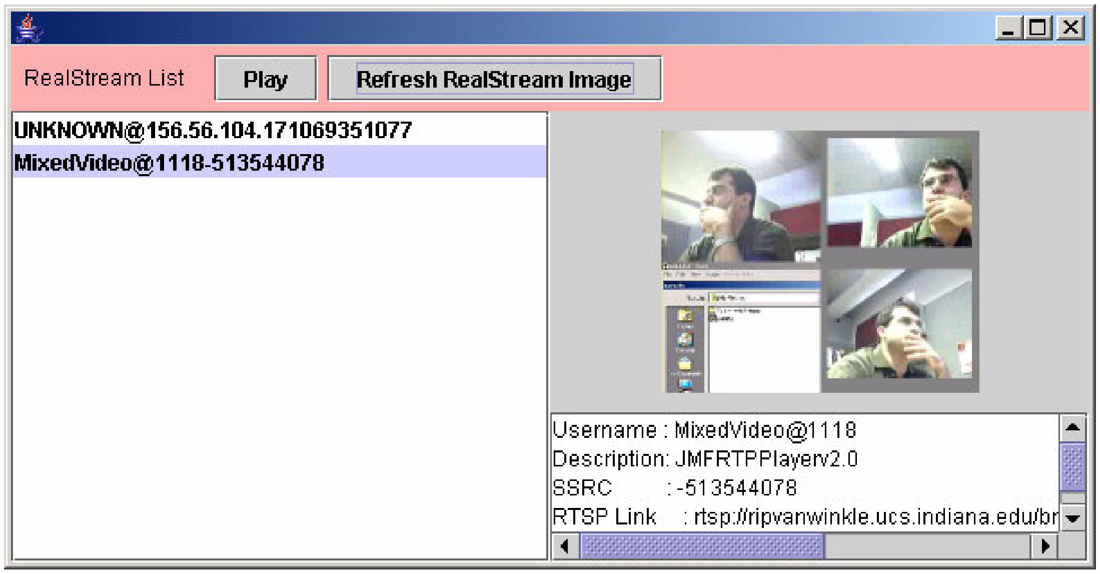
UNKNOWN@156.56.104.171069351077
MixedVideo@1118-513544078

```
Username : MixedVideo@1118
Description: JMFRTPPlayerv2.0
SSRC        : -513544078
RTSP Link    : rtsp://ripvanwinkle.ucs.indiana.edu/br
```

Figure 4-7: A screenshot of Streaming Admin interface

4.6.2 Client Interface

Streaming Client allows users to see the stream information and to play a stream from RealPlayer window. This interface does not allow clients to stop any stream conversion. A screenshot of the Streaming Client Interface is shown in Figure 4-8. The west of the panel shows the available streams, the east of the panel shows the image and information regarding to the stream chosen. When user selects a stream and clicks Play button, a RealPlayer windows pops up and plays the stream.



UNKNOWN@156.56.104.171069351077
MixedVideo@1118-513544078

Username : MixedVideo@1118
Description: JMFRTPPlayerv2.0
SSRC : -513544078
RTSP Link : rtsp://ripvanwinkle.ucs.indiana.edu/br

Figure 4-8: A screenshot of Streaming Client interface

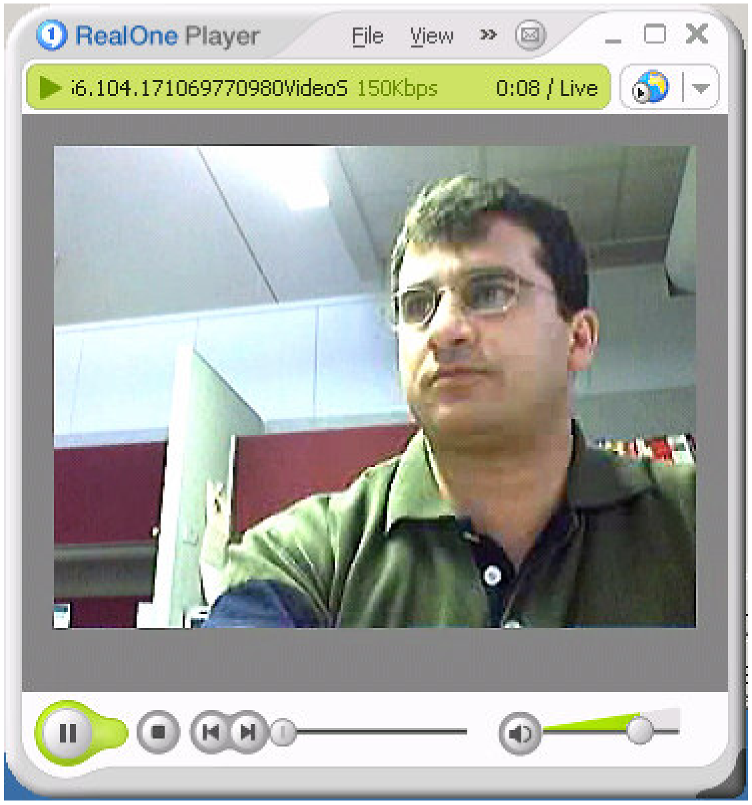


Figure 4-9: A screenshot of a RealStream played in RealPlayer window.

The stream lists on interfaces are dynamically updated when a stream leaves/joins the session. Figure 4-9 shows a stream played in a RealPlayer window.

4.7 Adapting Mobile Clients

Mobile devices have limited capabilities such as limited bandwidth, processing, memory capability and screen size. Due to this reason we cannot expect them to function like an Audio/Video (A/V) client on a desktop PC. Also the applications developed for mobile devices have limited features compared to their desktop PC correspondences. Although 3G [111] wireless network fully supporting multimedia application hasn't been widely used, limited multimedia services for cellular users are already available such as Real Mobile Streaming and the Microsoft Media Server (MMS).

In this section, we provide an architecture to enable cellular phone devices to receive streams from and send streams to videoconferencing sessions. Mobile devices show different characteristics when receiving streams from and sending streams to videoconferencing sessions due to the limitations we mentioned above. In receiving a stream, cellular phone device is an end-point where it only decodes the stream and displays it, while in sending the stream it is the source of the stream where it should be equipped with appropriate hardware and software to capture the images. Also note that encoding a stream is more CPU intensive compared to decoding a stream.

In order to enable cellular phone devices join a XGSP A/V session, we introduced a gateway (GobalMMCS Mobile Gateway) specific to mobile devices. We used Nokia 3650 to receive stream from Helix Streaming Server and send images to the GobalMMCS Mobile Gateway. Nokia 3650 has RealPlayer and MIDP 1.0 installed on it.

4.7.1 GlobalMMCS Mobile Gateway

GlobalMMCS Mobile Gateway provides two functionalities to enable mobile devices receive and send streams. In order to let mobile devices receive a stream it contains a web server to let mobile devices access XHTML pages where links to .ram files are stored. These .ram files reside in a directory shared with Helix Server. GlobalMMCS Mobile Gateway provides interfaces to XGSP Streaming Gateway to receive information regarding streams being encoded to mobile streaming format so that it can generate .ram files.

In order to enable mobile devices to send streams to GlobalMMCS sessions, GlobalMMCS Mobile Gateway also performs a similar functionality to XGSP Streaming Gateway in the sense that it transcodes the still images sent by a mobile device into video stream with H.261 format. We call this "Imgae-To-Stream Engine". GlobalMMCS Mobile Gateway is shown in Figure 4-10. We would like to explain receiving stream in (section 4.7.2) and sending image from (section 4.7.3) cellular phones next.

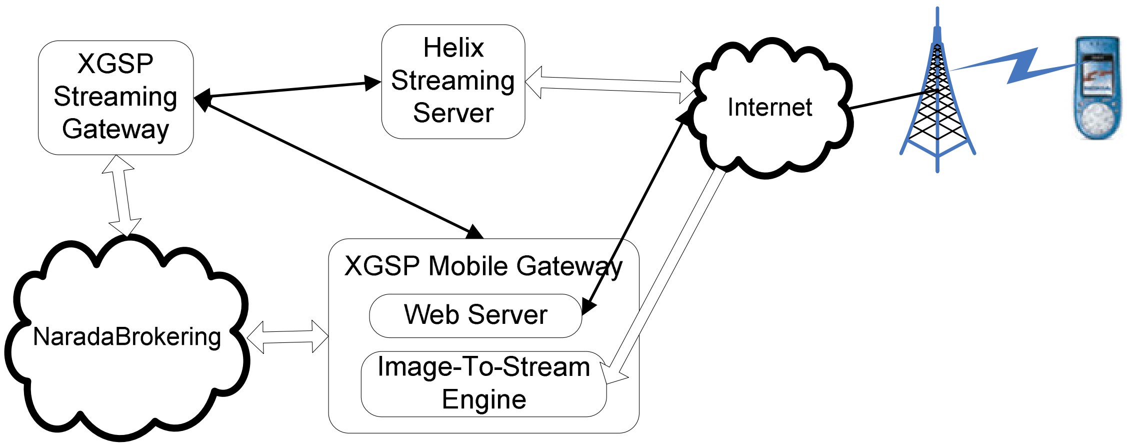


Figure 4-10: XGSP Mobile Gateway and interactions with other components in the system

4.7.2 Streaming to Cellular Phones

RealNetworks provides RealMedia formats for cellular phones as well. This RealMedia format is called as General Mobile Streaming which is 20 Kbps with 5 fps. The image size is also 160x120 pixels. RealPlayer on Nokia 3650 is capable of playing this streaming format.

The administration user interface of the Streaming Gateway can initiate streaming jobs for cellular phone clients as well. AdminUI specifies General Mobile Streaming format when he/she selects the video to be converted. Streaming Gateway receives the selected video stream and the session audio. Both audio and video are converted and combined into one RealMedia stream with General Mobile Streaming format. The generated streams have too long RTSP Uniform Resource Locator (URL). In order to let cellular phone clients access those streams from an interface, GlobalMMCS Mobile Gateway generates .ram files specific for RealPlayer and provides links to those files in a XHTML page. Cellular phone users can visit this page through browser and launch RealPlayers by clicking one of the links. Cellular phone clients can also access the whole session tree from the interface provided by GlobalMMCS Mobile Gateway. If the administrator stops mobile streaming jobs, the corresponding stream URLs are removed from the page. Figure 4-11 shows the stream played on Nokia 3650.

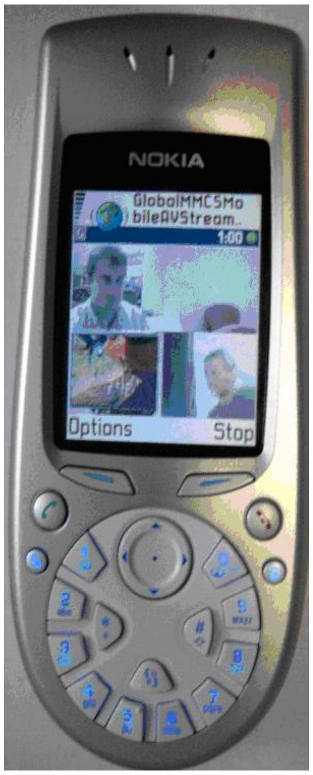


Figure 4-11: A GlobalMMCS session video converted into General Mobile Streaming format and viewed from Nokia 3650.

4.7.3 Streaming from Cellular Phones

A camera application developed with MIDP 1.0 for Nokia 3650 is used to achieve streaming from cellular phones to GlobalMMCS video session. This application is capable of capturing images and sending them to a web server. The images produced by that application are 160x120 pixels. In order to send them to a XGSP A/V session, we need to convert these still images into video streams. The transcoding module, which is Image-To-Stream engine, in the mobile gateway receives these images and produce a video stream constructed from them. Image-To-Stream module also resizes these images by a factor of 2 to produce a video stream with 320x240 pixels in size. The images are transported to this gateway over an HTTP connection with a interval of 7 ~8 seconds, encoded into a H.263 stream with 2 fps and then pushed to the XGSP A/V session. During the procedure, the conversion module reuses the same image until the next image is received. When the images received from cellular phone stops, Mobile Gateway simply sends an end-of-stream packet to the session and terminates.

Figure 4-12 and Figure 4-13 show the stream produced from the images captured by the camera application on the cellular phone. Figure 4-12 shows it in VIC panel which shows all of the streams in that Access Grid session. Figure 4-13 shows only the stream produced from the images received from the camera application in a VIC frame.

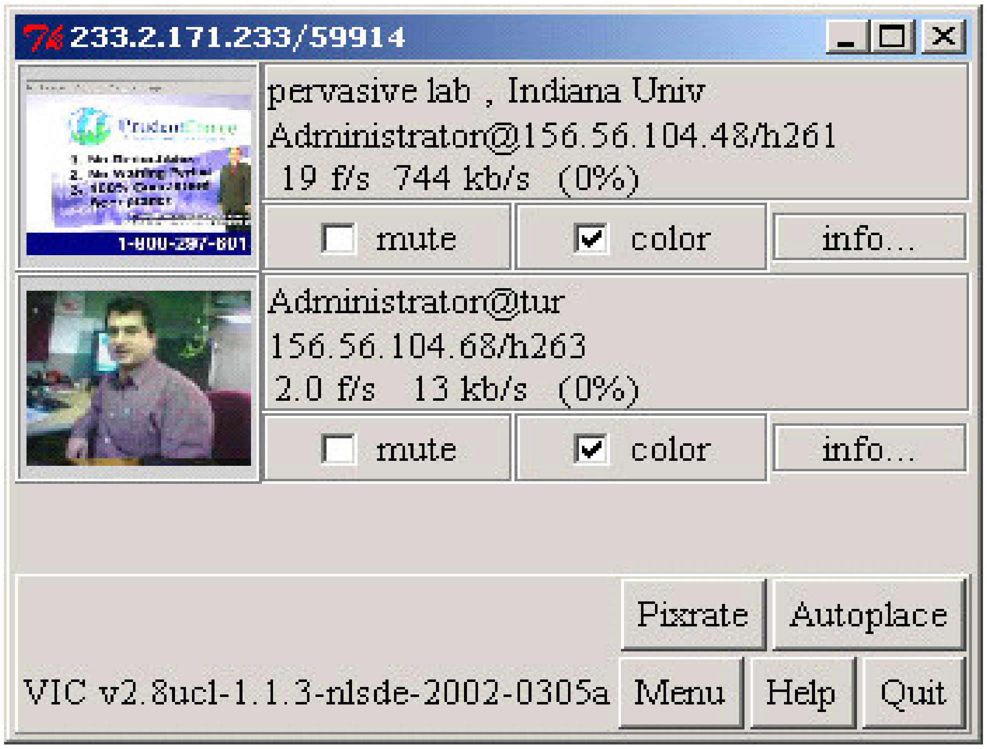


Figure 4-12: VIC panel that shows streams in GlobalMMCS Access Grid session

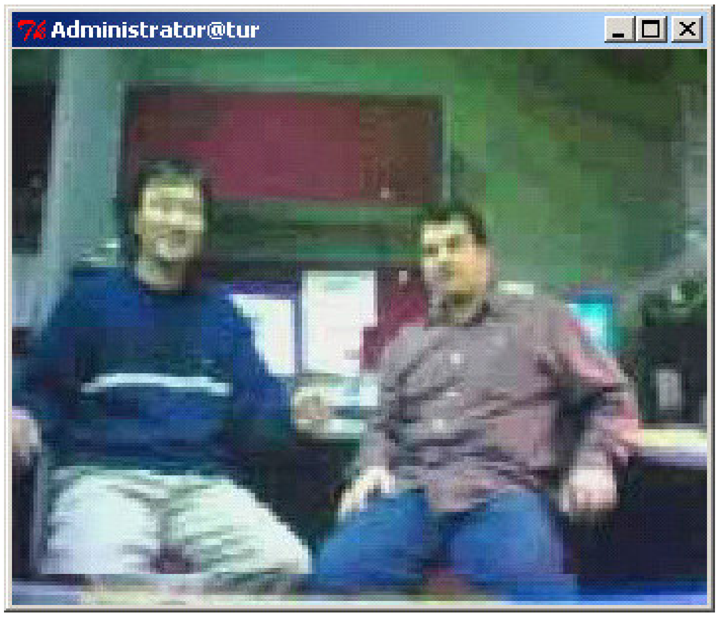


Figure 4-13: Stream received from Mobile Gateway

4.8 Performance Tests and Evaluation of Test Results

Stream conversion is a CPU intensive application. In order to see the CPU and memory usage of this conversion the effect of the number of streams converted on CPU and memory usage is observed. Streaming Gateway is running on the machine specified in Table 4-1.

OS	Windows XP
CPU	Intel® Pentium® 4 CPU 2.26 GHz
Memory	512 MB
JVM version	1.4.2_02

Table 4-1: Streaming Gateway machine specifications

Number of Streams	Frame rates of streams involved	CPU Usage Range	Memory Usage Range
1	23 fps	%10 - %25	29 MB
2	23 fps, 25 fps	%22- %40	34 MB
3	23 fps, 25 fps, 26 fps	%50 - %75	43 MB
4	23 fps, 25 fps, 26 fps, 16 fps	%65- %95	53 MB

Table 4-2: Approximate CPU and memory usage with respect to number of streams

Table 4-2 provides approximate CPU and memory usage of the streams. As the number of streams converted increases the CPU usage, the memory usage also increases. In this specific machine, 4 streams can successfully be converted without causing quality of service decrease. If the number of streams is increased, other streams are also affected and some time later the conversion performance reduces significantly. In this test the frame rate is kept high for most of the streams which is normal in an Access Grid session.

4.9 Summary

This chapter described the XGSP Streaming Gateway for integration of real-time videoconferencing and streaming media. XGSP Streaming Gateway has been developed as part of the GlobalMMCS prototype system to verify and refine this streaming framework. The XGSP Streaming Gateway integrates RealStream into videoconferencing systems within XGSP framework and, hence its prototype GlobalMMCS enables multiple communities to collaborate with each other. XGSP Streaming Gateway can easily be extended to other streaming formats rather than just RealStream. In order to do this, Stream Engine which is part of the XGSP Streaming Gateway needs to transcode the input stream into the streaming format desired.

Stream conversion requires much CPU and memory utilization. This limits the number of streams converted on one machine. In order to increase the number of streams converted, this research suggests a streaming job scheduler which will schedule and coordinate streaming jobs in a distributed environment.

Due to their capabilities and environment, mobile devices introduce different issues to be researched. Some of their limitations are low bandwidth, CPU and memory and limited power. Low bandwidth effects the bytes sent across the network. Low CPU limits the processing power which is crucial when encoding streams into different formats.

Chapter 5

Time Services within NaradaBrokering

In previous chapters, we stated that we use messaging environments to transport multimedia content. A collaborative system that uses events (messages) to transport its data needs to synchronize those events in order to synchronize streams generated at different clients who are residing at different machines. Those clients may also be distributed over a large geographical area which will introduce different network delay for different clients.

Time ordering of events generated by entities existing within a distributed infrastructure is far more difficult than time ordering of events generated by a group of entities having access to the same underlying clock. Because of the unsynchronized clocks, the events (messages) generated at different computers cannot be time-ordered at a given destination if local time is used for timestamping.

It is thus necessary to synchronize the clocks in a distributed system, because the time-based order of events matter. Lamport addressed this problem [78] by logical clocks. We have discussed other approaches [79-82] for time ordering events in section 2.4.2. But with global time servers fairly common now, time ordering can be had at lower overhead and more accurate results. One cannot rely on the underlying hardware or software clocks to provide synchronization. Hence, time ordering of events generated by entities existing within a distributed infrastructure is far more difficult than time ordering of events generated by a group of entities having access to the same underlying clock.

In this chapter, we will explain the Time Service we incorporated within NaradaBrokering messaging middleware to achieve synchronization of streams from different sources. This Time Service utilizes Network Time Protocol (NTP). Entities in NaradaBrokering environment can timestamp events with NTP timestamps which will provide more consistent timestamps than local clocks.

5.1 Using Network Time Protocol (NTP)

NTP can achieve to 1-30 milliseconds accuracy, where accuracy implies that by using NTP the underlying clock is within 30 milliseconds of time server clock (usually an atomic clock). There are atomic time servers provided by various organizations, i.e. National Institute of Standards and Technology (NIST) and U.S. Naval Observatory (USNO). NIST and USNO Internet Time Services use multiple stratum-1[2] time servers, which are open to public access. Using a NTP client, anyone can synchronize his clock with the atomic time server's time within the 1-30 milliseconds range. However, this accuracy also depends on the roundtrip delay between the machine and the time server

[2] Stratum number is an integer indicating the distance from the reference clock. Stratum-0 is the reference clock.

supplying the time service. The difference between the delay from the machine to the time server and the delay from the time server to machine also contributes to the accuracy of the offset computed. NTP achieves this accuracy by using filtering, selection and clustering, and combining algorithms to adjust the local time.

Real-time constraints for A/V conferencing applications can vary anytime between 30-100 milliseconds, depending on the jitter in inter-packet arrivals in these streams. Packets in these streams generated at different locations can be buffered (either during replay or real-time), and time-ordered to provide an efficient collaboration session. If time-ordering among these streams is lost, it would be very unpleasant during the replay/real-time-play of these streams. The range that NTP provides is sufficient for such a collaboration environment. To achieve time-ordering of events at the destination, NTP timestamps can be used instead of local time.

5.2 Time Service

The time service we provide as part of NaradaBrokering (0.95) currently implements NTP version 3. A configuration file, which contains time server addresses and other required NTP parameters, is provided to the time service. We impose no limit on the number of time servers that can be specified in the configuration file. In addition to these parameters, the interval that the time service should run is also specified in the configuration file. The value of this parameter affects the synchronization range of the computer clock. If it is too high, computer clock may be way of out of sync, and if it is too low it may utilize too much system and bandwidth resources.

It should be noted that time service does not change the system time. That is, unlike NTP daemons, it does not set system time to a new value. There are two reasons for this. First, in order to change the underlying system clock, one needs administrative privileges. This is not possible for clients without administrator privilege. The second reason is that the objective of this time service is to provide a mechanism to be able to time-order the events generated within NaradaBrokering without affecting the system and other applications running on the same machine. A call to the Time Service returns the adjusted time. It achieves this by keeping the offset in a separate variable. The `getTimestamp()` method returns the time obtained from the local system time adjusted with this offset in milliseconds. When time service starts, it computes the first value of the offset. After this initialization, time service updates the offset at regular intervals based on the parameter specified in the configuration file.

All events generated anywhere, by any entity, within the system utilize their Time Service to timestamp events.

5.2.1 Initialization

The initialization step is an important step in achieving synchronization. Because of this, the time service attempts to achieve a synchronization range within several seconds after being started. This initialization step is a blocking operation during the bootstrapping of NaradaBrokering services and should complete within a few seconds. The initialization step also uses NTP, but instead of using the interval specified in the configuration file, it waits 500 milliseconds between its attempts. The total attempt time is also limited for 5 seconds. 500 milliseconds is the upper limit to wait for replies from time servers which can be sufficient time to receive a packet in Wide Area Network (WAN). However, replies can be received in a shorter duration and the initial offset value can be computed from those replies.

5.2.2 NTP Implementation

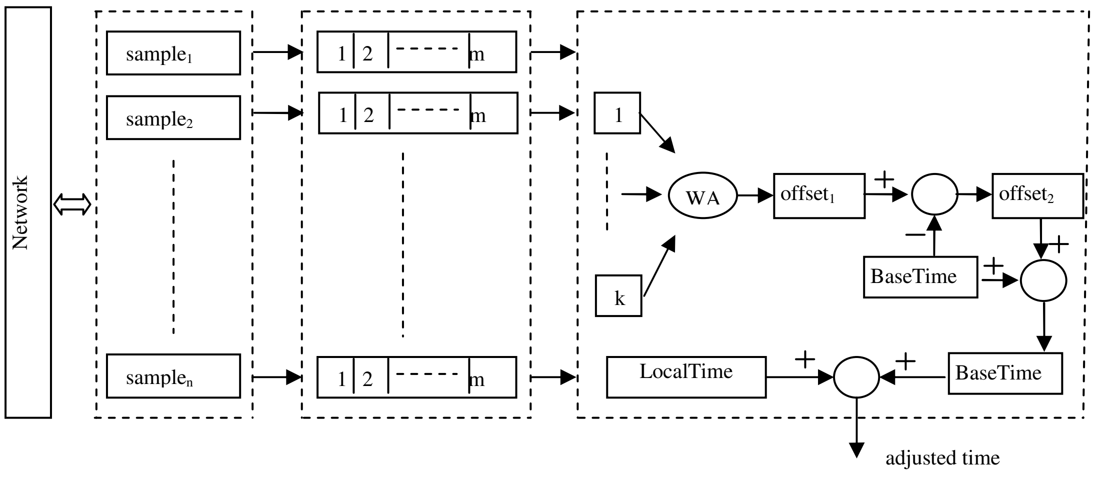
We use NTP version 3 as the basis for our NTP implementation. We have chosen NTP instead of SNTP, because NTP implements advanced algorithms to filter the NTP message obtained from the time server and implements selection algorithms to those received NTP messages. A NTP message can be received from any number of NTP time servers. Our NTP implementation is as follows:

The first step involves getting samples from the NTP time servers. A NTP client sends NTP messages to the servers specified in the configuration file one by one. That is, it sends a NTP message and waits for the response. The message is a datagram packet, and it may be lost in the network. Because of this, the NTP client sets a timeout to the UDP socket. The timeout chosen for our implementation is 500 milliseconds. This step requires that NTP replies be received from at least half of the servers. Offset and roundtrip delays are calculated at this step. Upon receiving a NTP packet, a NtpInfo object is generated which contains NTP parameter values, i.e. offset, roundtrip delay, dispersion, timestamps, etc.

After collecting NTP samples from the servers, the second step is to use the NTP filtering algorithm. The filtering algorithm checks timestamps to validate the received NTP message. It keeps a register dedicated for each time server and records the NTP samples received from that server. This register only keeps a specified amount of samples, and uses a First-In-First-Out (FIFO) scheme to accommodate new samples if the register is full. The window size of this register is specified in the configuration file and has an effect on the computed offset.

After the previous steps are successfully completed, the new offset is computed using selection and combine algorithms as explained in the NTP specification. A clustering algorithm explained in NTP specification is then used to find a candidate list from which the new offset would be computed. Clustering algorithm uses stratum value obtained from NTP message and synchronization distance computed from the NTP parameters of the related server to construct this candidate list. The result of the clustering algorithm provides a candidate list, which contains the synchronization distance and the offset obtained from each time server. The combining algorithm then computes the weighted average of this candidate list according to the synchronization distance. So a server with a small synchronization distance has more impact on the new offset.

The steps explained in this section can be depicted as in Figure 5-1. There are two offset values indicated in Figure 5-1, $offset_1$ and $offset_2$. $offset_1$ is the offset value computed with NTP. Since we do not change the system clock, we need to keep a variable named as *BaseTime*, which stores the offset computed with NTP in an earlier computation. $offset_2$ is the difference between the *BaseTime* and the offset computed by NTP. It can also be viewed as the change of the offset.



Phase 1:
collecting
samples

Phase 2:
filtering & adding to
the queue

Phase 3:
establishing candidate list &
computing offset

offset_1: Offset computed with regard to local computer time
offset_2: Offset computed with regard to adjusted time

Figure 5-1: Steps taken in computing offset using NTP algorithms

5.2.3 Updating Offset

Unfortunately, calculating offset as in previous steps is not sufficient to achieve synchronization. A newly computed offset may not be used as is. The order of the messages generated at the local computer should also be preserved. Let's say message m_1, the latest message before the new offset is calculated, has the timestamp of *t,* and message m_2, the first message to use the new adjusted time, has a timestamp of t_2. Then t_2 cannot be less than t_1. Because in that case, the ordering algorithms may conclude that m_2 was generated before m_1 which is not correct. So the offset is not updated because the new value would cause an inconsistency in the ordering of the messages. In this case, we do not update the timestamp so long as the discrepancy persists.

The pseudo code can be written as below;

```
long getTimestamp() {
     timestamp = LocalTime + BaseTime;
     if (timestamp >lastTimestamp)
            lastTimestamp = timestamp;
```

```
        return lastTimestamp;
    }
```

5.3 Benchmark Results

We have done tests on several linux and solaris machines. The interval for updating offset is about 30 seconds. There are 8 time servers specified in the configuration file and all of them are stratum-1 NIST time servers. In the test results, we also show the first offset value, standard deviation, the average offset change and the minimum and maximum values.

The tests mentioned in this section are done on computers at Community Grids Lab available to Indiana University researchers. Computer clocks are not modified or preset before the tests. Test cases are given below. In section 5.3.1 we give benchmark results and in section 5.3.2 we evaluate the results.

Initialization offset value indicates the first offset change needed when the Time Service runs. It also indicates how much the clock is out of sync. Other values are related to the change applied to the initialization offset value. So min and max values indicate the minimum and maximum values applied to the initialization offset values. Average value indicates the average of the change applied to the initialization offset value. Standard deviation is the deviation of the value of average offset change. Hence, the average offset value which is added to the system clock value can be formulated as below;

Average clock offset value = Initialization offset value + average offset change

5.3.1 Benchmark Results on Linux Machines

Test cases for Linux machines are shown in cases i – iv.

i) darya.ucs.indiana.edu

Name	darya.ucs.indiana.edu	initialization offset value	0 msec
OS	Red Hat Linux release 7.3 (Valhalla)	standard deviation	0.11
CPU	AMD Athlon(tm)MP 1800+, 1533.42 MHz	average offset change	-0.00018 msec
Memory	512 MB	min offset change	-2 msec
JVM version	1.4.1_03	max offset change	3 msec
		total offset change	-1 msec
		number of data	5690
		total test duration	172800 sec

Table 5-1: Machine specification and numeric values for darya.ucs.indiana.edu

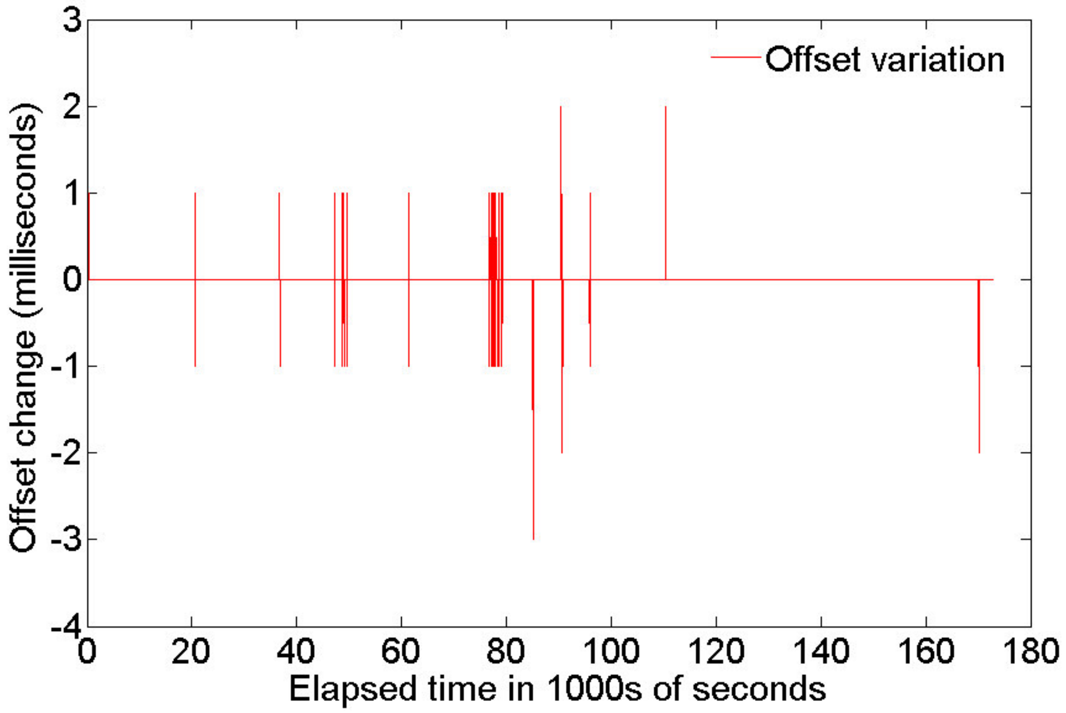


Figure 5-2: Change of offset with time for darya.ucs.indiana.edu

ii) kamet.ucs.indiana.edu

Name	kamet.ucs.indiana.edu	initialization offset value	1185869 msec
OS	Red Hat Linux release 9 (Shrike)	standard deviation	3.32
CPU	Intel(R) XEON(TM) CPU 1.80GHz	average offset change	5.21 msec
Memory	1 GB	min offset change	-1 msec
JVM version	1.4.1_02	max offset change	12 msec
		total offset change	29666 msec
		number of data	5690
		total test duration	172800 sec

Table 5-2: Machine specification and numeric values for kamet.ucs.indiana.edu

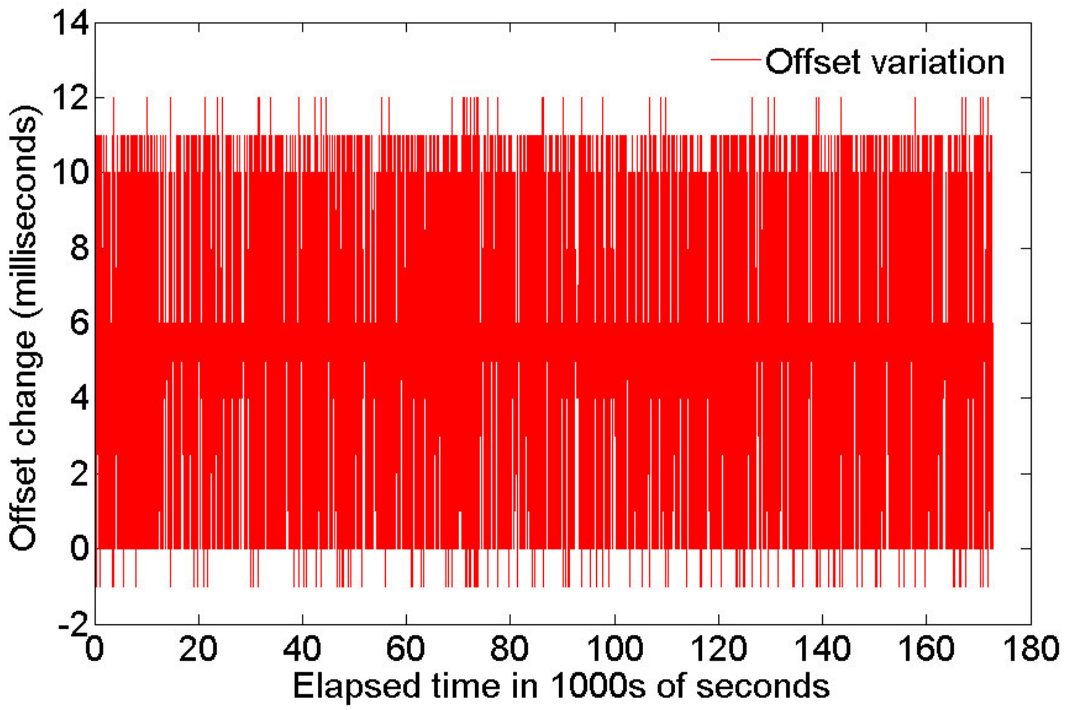


Figure 5-3: Change offset with time for kamet.ucs.indiana.edu

iii) murray.ucs.indiana.edu

Name	murray.ucs.indiana.edu	initialization offset value	-139895 msec
OS	Red Hat Linux release 7.2 (Enigma)	standard deviation	0.71
CPU	Intel Intel(R) Pentium(R) III CPU family 1266MHz	average offset change	-0.19 msec
Memory	1 GB	min offset change	-3 msec
JVM version	1.4.1-rc	max offset change	2 msec
		total offset change	-1060 msec
		number of data	5690
		total test duration	172800 sec

Table 5-3: Machine specification and numeric values for murray.ucs.indiana.edu

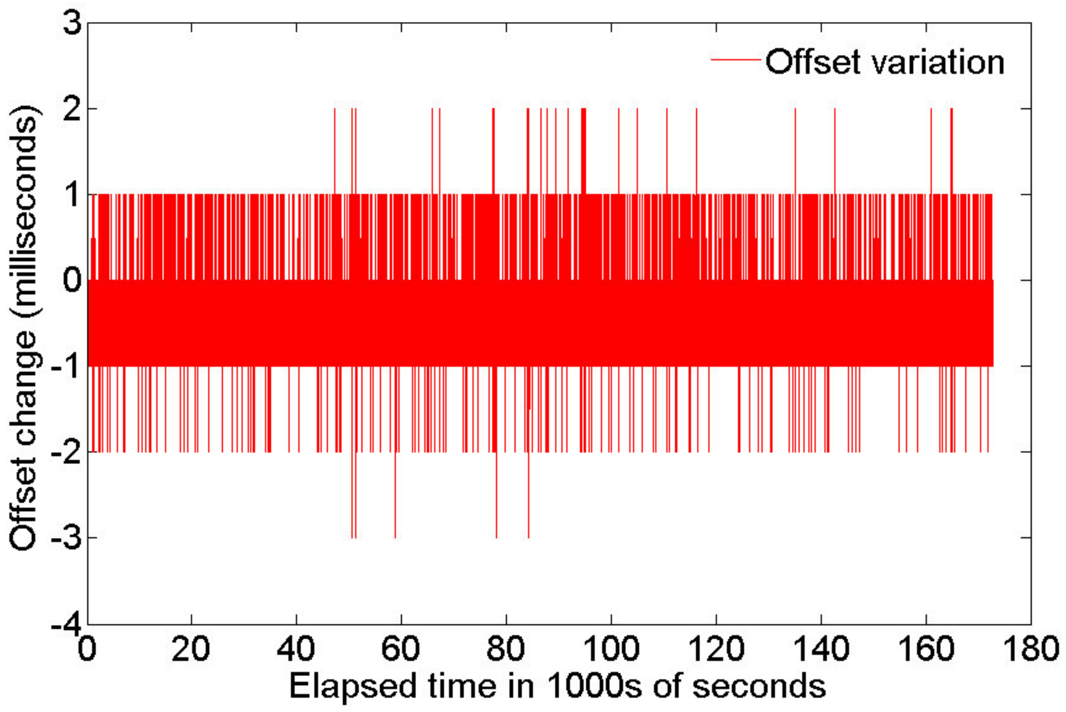


Figure 5-4: Change offset with time for murray.ucs.indiana.edu

iv) elkhart.ucs.indiana.edu

Name	elkhart.ucs.indiana.edu	initialization offset value	-4030278 msec
OS	Red Hat Linux release 8.0 (Psyche)	standard deviation	4.248092
CPU	Intel Intel(R) XEON(TM) CPU 2.20 GHz	average offset change	-6.35167
Memory	1 GB	min offset change	-18 msec
JVM version	1.4.1_02	max offset change	6 msec
		total offset change	-36141 msec
		number of data	5690
		total test duration	172800 sec

Table 5-4: Machine specification and numeric values for elkhart.ucs.indiana.edu

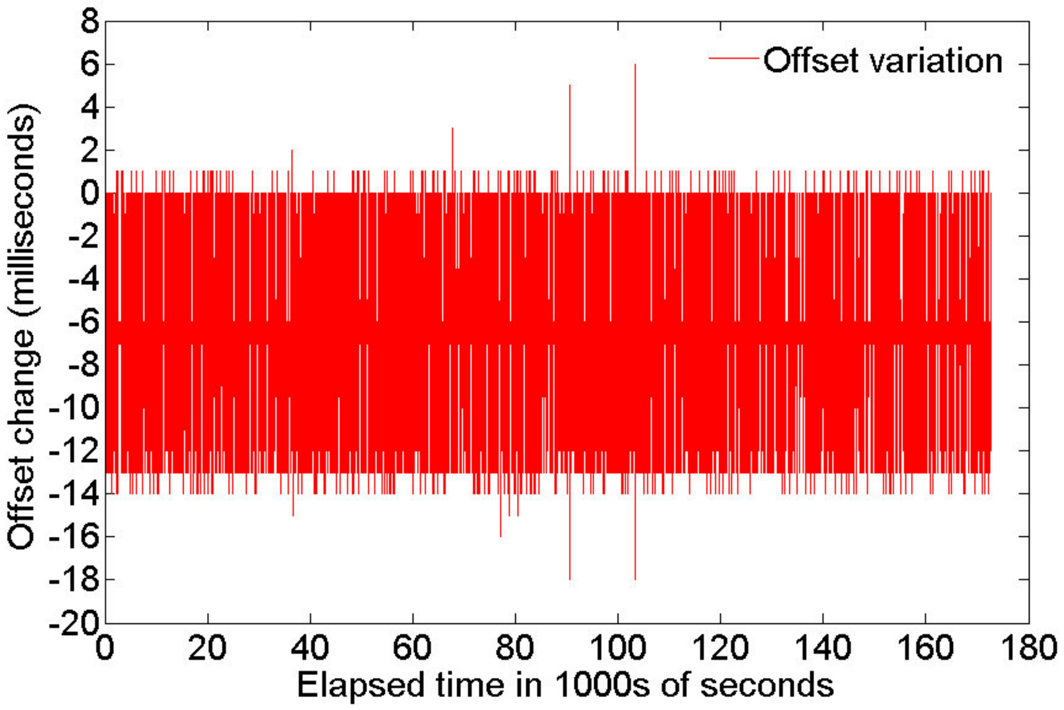


Figure 5-5: Change offset with time for elkhart.ucs.indiana.edu

5.3.2 Linux Machine Test Evaluation

Over a period of 48 hours, a total of 5690 offsets are computed and their changes are shown in Figure 5-2, Figure 5-3, Figure 5-4 and Figure 5-5. First offset values, standard deviations, averages, minimum values , maximum values and total changes in the offsets for test cases iiv- iv are shown in Table 5-1, Table 5-2, Table 5-3 and Table 5-4 to give us some numerical ideas.

In test case i, ntpd daemon is running. Among 5690 computed offset values only 24 of them are different than zero, which means that ntpd daemon running on the machine and our time service are very consistent with each other. The first value of the offset is also zero, because ntpd daemon is able to keep the machine synchronized. This ntpd daemon synchronizes its time with “time.nist.gov” time server.

In test case ii, ntpd daemon is also running. But time server is set to “clock.redhat.com” for this server, which was not reachable at that time. ntpd daemon cannot update locale time if it cannot connect to the time server. The change offset is between (-1) – (12) ms. The first offset value is 1185869 ms, which means that the clock is behind the real time by that many milliseconds.

In test case iii and iv, no ntpd daemon is running. The change of offsets in case iii is between (-3) - (2) ms while in test case iv it is between (-18) – (6) ms. The first offset value for case iii is -139895 ms and for case iv it is -4030278 ms, which shows how much the clocks in those machines are ahead of the real time.

If we pay attention to the average value of these offset changes, as indicated in Table 5-1, Table 5-2, Table 5-3 and Table 5-4, the machine in test case ii has a positive value and the machine in test case iii and iv have negative values. Also the total offset change for test case ii is positive and for test case iii is negative. From this, we can conclude that the clock running on machine for test case ii is a slow clock because positive adjustment is done to the underlying system clock and the clocks running on machine for test case iii and iv are fast clocks because negative adjustment is done to the underlying system clock. Note that the adjustments needed for cases ii - iv are different. This also shows that the clock rates on the machines are different. Since a ntpd daemon is running on the machine for test case i we avoid making such a conclusion regarding that clock. Clock on machine iv seems faster than the clock on machine iii.

5.3.3 Benchmark Results on Solaris Machines

Test cases for Solaris machines are shown in cases v – viii.

v) community.ucs.indiana.edu

Model	Sun Ultra60 Workstations	Initialization offset value	404646 msec
OS	Solaris 8	standard deviation	0.8
CPU	Dual 450Mhz UltraSparc CPU's	average	0.4
Memory	1 GB	min value	-2 msec
JVM version	1.4.2-beta	max value	3 msec
		total change	2026 msec
		number of data	5674
		total test duration	172744846 msec

Table 5-5: Machine specification and numeric values for community.ucs.indiana.edu

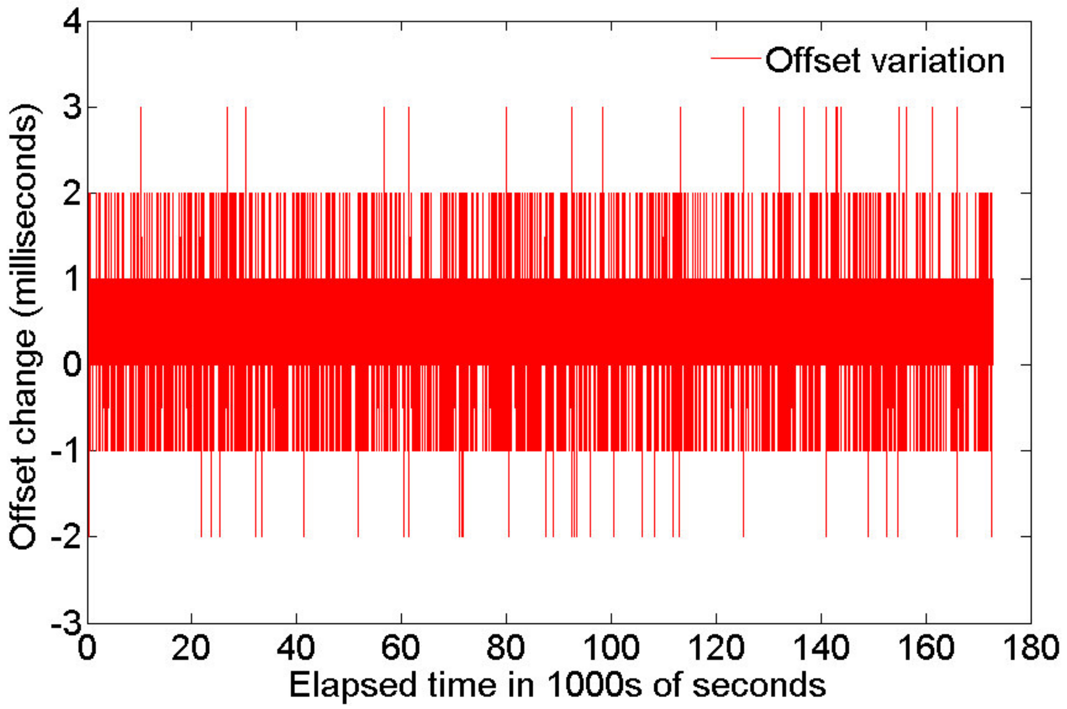


Figure 5-6: Change offset with time for community.ucs.indiana.edu

vi) grids.ucs.indiana.edu

Model	Sun Ultra60 Workstations	Initialization offset value	128395 msec
OS	Solaris 8	standard deviation	59.2
CPU	Dual 450Mhz UltraSparc CPU's	average	0.7
Memory	1 GB	min value	-1994 msec
JVM version	1.4.2_05	max value	9 msec
		total change	3721 msec
		number of data	5676
		total test duration	172744846 msec

Table 5-6: Machine specification and numeric values for grids.ucs.indiana.edu

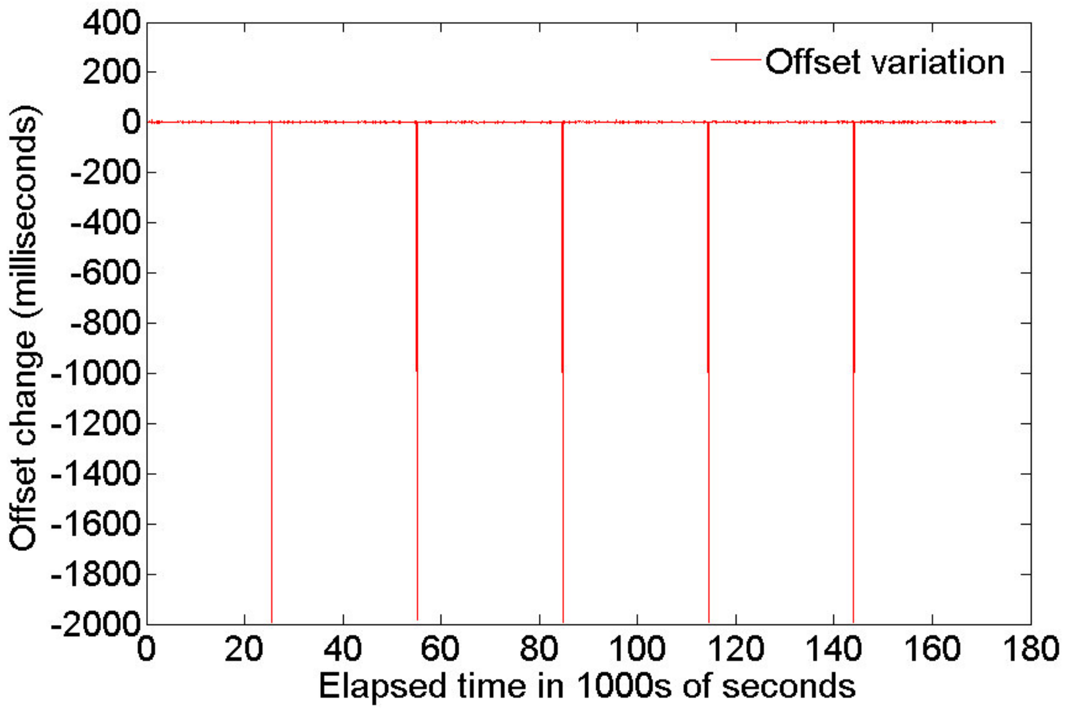


Figure 5-7: Change offset with time for grids.ucs.indiana.edu

vii) ripvanwinkle.ucs.indiana.edu

Model	Sun Fire V880	Initialization offset value	-672765 msec
OS	Solaris 9	standard deviation	37.5
CPU	8 1.2 GHz UltraSPARC III processors	average	-1.0
Memory	16 GB	min value	-9 msec
JVM version	1.4.2-beta	max value	1995 msec
		total change	-5487 msec
		number of data	5676
		total test duration	172744846 msec

Table 5-7: Machine specification and numeric values for ripvanwinkle.ucs.indiana.edu

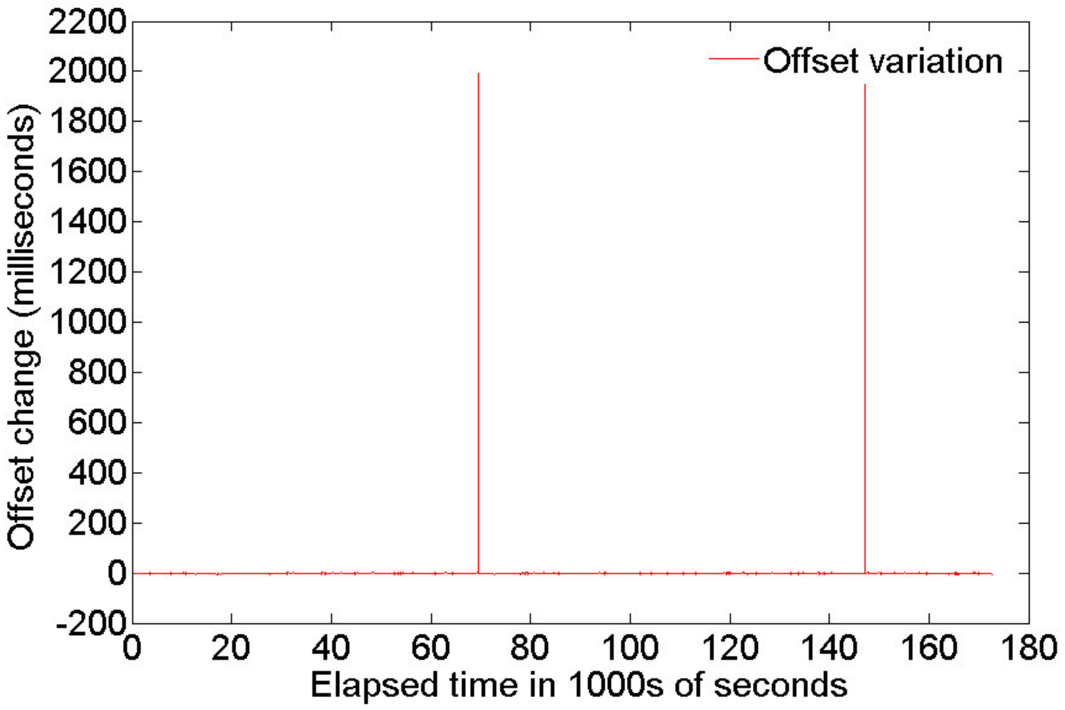


Figure 5-8: Change offset with time for ripvanwinkle.ucs.indiana.edu

viii) complexity.ucs.indiana.edu

Model	Sun Fire V880	Initialization offset value	-690120 msec
OS	Solaris 9	standard deviation	26.5
CPU	8 1.2 GHz UltraSPARC III processors	average	-1.0
Memory	16 GB	min value	-9 msec
JVM version	1.4.2-beta	max value	1994 msec
		total change	5668 msec
		number of data	5678
		total test duration	172750467 msec

Table 5-8: Machine specification and numeric values for complexity.ucs.indiana.edu

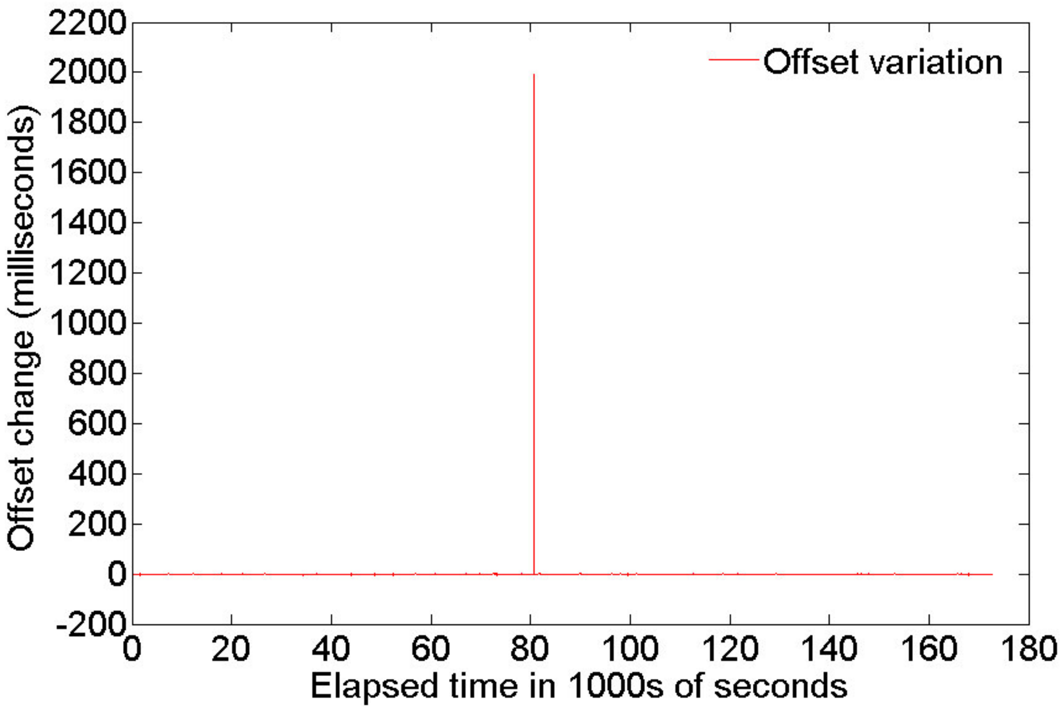


Figure 5-9: Change offset with time for complexity.ucs.indiana.edu

5.3.4 Solaris Machine Benchmark Results

Solaris machines' test results demonstrate a strange behavior of the underlying system clock. Three of the machines in test cases v - viii have jumps with some different intervals. These jumps are around +/-1990 ms and can have positive and negative signs.

Jump interval for test case vi is around 494 minutes and jump values are -1992 ms, -1995 ms, -1994 ms, -1992 ms and -1992 ms as shown in Figure 5-7. The jump interval in test case vii is around 1293 minutes and the jump values are 1992 ms and 1995 ms as shown in Figure 5-8. Figure 5-9 shows only one jump with a value 1994 ms. In order to see the jump interval, another test is done for a continuous 168 hours (1 week). We observed that the jump interval is around 1580 minutes.

Test case v has no jumps and the offset change is kept between -2 ms and 3 ms. The total changes and the sign of the average value show us that it is a slow clock.

ntpd daemons for machines in test cases vii and viii bare not running. Also ntpd daemons on machines in test cases v and vii have no effect on the system clock, because they have been misconfigured by the administrator.

The first offset value for case v is 404646 ms, for case vi it is 128395 ms, for case vii it is -672765 ms and for case viii it is -690120 ms. The differences in the first values show how much these computer clock values are apart from each other.

5.4 Inter Client Discrepancy

We have also tested the discrepancy between two of the machines that are running the NB Time Service. The test environment is established as shown in Figure 5-10. In order to receive requests from a remote client, we also implemented a NTP server. We also implemented a client that uses the NB Time Service time and is capable of sending NTP requests to the server.

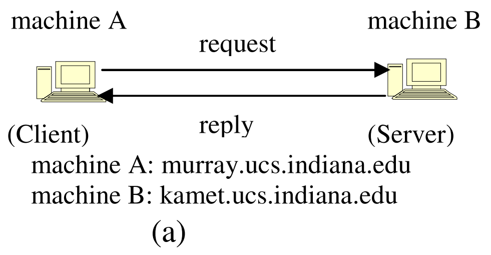
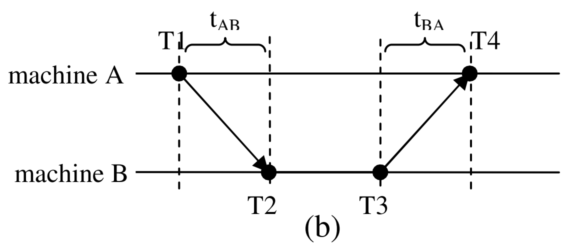

machine A
request
machine B
(Client)
reply
(Server)
machine A: murray.ucs.indiana.edu
machine B: kamet.ucs.indiana.edu
(a)


Figure 5-10: (a)Computers that run NTP server and NTP clients (b) Initiation and end of a time request between machine A and B.

standard deviation	2.94
absolute average discrepancy	5.6 msec
absolute minimum discrepancy	0 msec
absolute maximum discrepancy	17 msec

Table 5-9: Numeric values for discrepancy between machine A and machine B

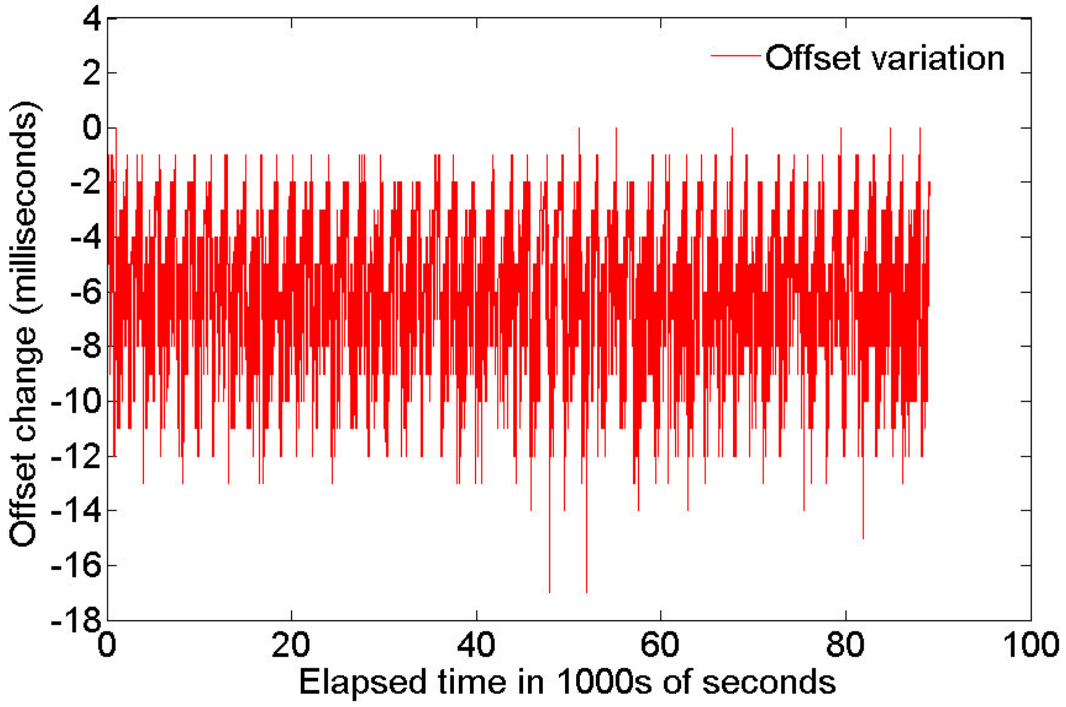


Figure 5-11: Discrepancy between machine A (murray.ucs.indiana.edu) and machine B (kamet.ucs.indiana.edu)

As shown in Figure 5-10.b, a request originates from the client and is received by the server. The server timestamps the request after receiving it and before sending it back to the client. The client then timestamps the reply. The delay involved in this process is the transmission time (t_{AB} and t_{BA}) and the process time on the server (T3-T2).

The discrepancy (ΔT) between these two machine clocks can be approximated as $\Delta T=0.5*(T2+T3-T1-T4)$, if t_{AB} and t_{BA} are ignored. Figure 5-11 shows this discrepancy computed. This approximated discrepancy is also found to be same with the offset computed by NTP. The discrepancy is measured every 30 seconds and test duration is about 25 hours.

5.5 High Resolution Timing Services

To ensure that messages are time-stamped as accurately as possible we have also incorporated a high resolution timer service into the substrate. This high resolution timer works on the Windows, Linux and Solaris operating systems; here we have leveraged native libraries available on these systems along with the Java Native Interface to enable high resolution timers. On Solaris and Linux the *gettimeofday()* function is used to retrieve the current time in microseconds. It returns time since 1/1/1970; the resolution is hardware dependent and is usually around 1 microsecond. The `QueryPerformanceCounter` on Windows is used to get number of ticks; the number of ticks in one second is 3759545, which is around one per 279 nanoseconds. It returns ticks from the start of the machine, but does not have limitations as in the *getTickCount()* function, which rolls-over every 49 days. Note that this gives us better results than relying only on the Java call. The resolution of the

System.currentTimeMillis() on Windows is around 15 milliseconds and 1 millisecond on Linux. The accuracy of this high resolution timer is around 3~4 microsecond.

High Resolution Clock (HRClock) has the following API;

```
long     getTimeMilliseconds()
long     getTimeMicroseconds()
void sleep(int sleepVal)          // millisecond sleep
void hrSleep(int sleepVal)        // microsecond sleep
```

Clients only need to call *HRClock.getClock()* function to get a reference to the clock. HRClock returns the appropriate clock implementation for the underlying operating system. Since we make calls from java to native side, also JNI call overhead can be introduced into these resolutions.

The above functions are wrapped in a java class called HRClock. Based on the platform that the program runs on, HRClock calls the corresponding methods implemented for that underlying platform.

5.6 Summary

While archiving and replaying collaborative sessions, in order to synchronize streams in the session, events in streams should contain accurate timestamp information with respect to the time they are generated and published. Since in a collaborative environment, we cannot expect clients to use the same underlying clock to timestamp their events, we need to synchronize their clocks as much accurate as possible.

Each machine has a different underlying clock which may run at different rates. Test results that we provided in this chapter demonstrate this issue. Since a synchronization range in order of 10s ms is sufficient for streams in a collaborative environment, NTP would be sufficient to achieve this range.

NTP allows us to synchronize the machine's clocks with atomic time servers available all over the world. Servers distributed over a wide geographical area would be synchronized with each other in a limited range. To achieve this, of course, time servers should use the closest time servers available to them. Since we know that atomic time servers are tightly synchronized with each other, we would also be able to synchronize computer clocks.

Chapter 6

Jitter Reduction Service within NaradaBrokering

Jitter is considered to be one of the most important quality of service measurements within collaborative systems such as audio/video systems. Multimedia applications used within audio/video systems handle real-time streams as well as archived streams. This multimedia data contains continuous audio and video content. Jitter values within these streams define the quality of the stream presented to the end user.

Multimedia applications mandate timely delivery of content and generally sustain loss of media packets very well. This makes UDP a very good choice for transporting media, since unlike TCP it does not incorporate an error detection/correction mechanism which adds delays associated with individual packets. Since UDP is point to point, it is best to use RTP over UDP when sending streams from one client to another directly. It is generally difficult to guarantee that all data traversing a packet-switched network will experience exactly the same delay. Packets encounter queues in switches or routers and the lengths of these queues vary with time, meaning that the delays tend to vary with time, and as a consequence, are potentially different for each packet in the A/V stream.

In the case of audio streams, high jitter values can cause voice breaks while in the case of video streams high jitters may cause degenerations in the image quality. In order to overcome the negative effects of high jitter, real-time audio/video clients typically have a buffer which buffers events up to 200 milliseconds and then proceeds to release them preserving the time spacing between packets. These buffering and time spacing services enhance jitter reduction.

The way to deal with this at the receiver end is to buffer up some amount of data in reserve, thereby always providing a store of packets waiting to be played back at the right time.

In this chapter, we will define the Jitter Reduction Service we developed for messaging systems. Jitter Reduction Service components are Buffering Service and Time Differential Service.

6.1 Jitter Reduction Service in Messaging Systems

As explained in previous chapters, messaging systems can be used to deliver multimedia content, while these brokers are distributed over a wide range of geographical area with different internet connections. Same high jitter reasons we defined above exist for messaging system too.

Messaging systems lack such a mechanism when they are used to deliver multimedia content for collaboration systems, such as transporting multimedia content within event to the client. This requires that a Jitter Reduction Service is necessary in messaging systems, especially if collaboration sessions contain replay of archived streams. This Jitter Reduction Service requires that events are timestamped using NTP timestamps explained in the previous chapter. Because only in that case events from different streams generated from different machines would be ordered correctly and timespacing between events preserved.

In order to achieve low jitter, we incorporated two services within Jitter Reduction Service, one is Buffering Service which orders events and the other is Time Differential Service, which preserve the timespacing between events when releasing them. Figure 6-1 shows this relationship between Buffering Service and Time Differential Service.

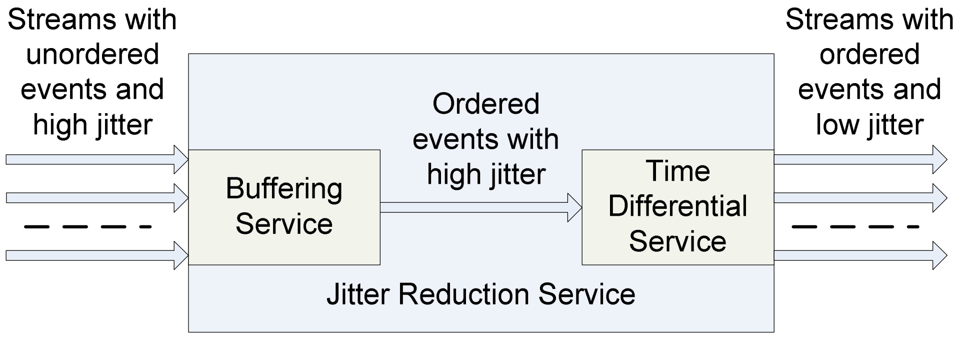

Streams with unordered events and high jitter
Buffering Service
Ordered events with high jitter
Time Differential Service
Streams with ordered events and low jitter
Jitter Reduction Service

Figure 6-1: Jitter Reduction Service

6.2 Time Buffering Service

When an entity receives messages, it may need a different buffer size based on the data type and application. For example, in real-time videoconferencing systems the buffer size is too low, and as a result packets are delayed for up to 200msec while in streaming applications which are not considered real-time can have delays around 10-15 seconds or more.

Buffer size also affects the jitter. Small buffer size may cause high jitter while larger buffer size can helps to reduce the jitter. Buffering service described in this section provides a customizable buffer size based on the entity's needs.

If a packet is delayed a long time, it goes into the buffer until its playback time arrives. If it gets delayed a long time, then it will not be stored for very long in the receiver's buffer before being played back. Thus, we have effectively added a constant offset to the playback time of all the packets as a form of insurance. The only time we run into trouble is when packets get delayed in the network for such a long time that they arrive after their playback time, causing the playback buffer to be drained.

The function of buffering service is to buffer some amount of events and release them in an orderly fashion to Time Differential Service (TDS). This will ensure that the events received within the buffer duration are ordered and events are delayed for buffer duration.

The design of the buffering service has incorporated four configurable parameters pertaining to the release of time-stamped messages:

i. The first criterion is the number of messages in the buffer maintained by the buffering service. If the number of messages reaches the maximum number of entries, it starts to release the time-ordered messages.

ii. The second criterion is the total size of the messages in the buffer. This along with the first criterion enables us to circumvent buffer overflows.

iii. The third criterion corresponds to the time spent by messages within the buffer. In some cases, the rate of messages arriving at an entity may be too slow and this may cause longer and unwanted delays within the buffer. The time-duration factor makes sure that the messages are released after a maximum specified duration if the first two criteria are not met.

iv. The final criterion is the release factor of the buffer. This typically has a value between 0.5 and 1.0. When any of the release criteria mentioned above is met, it releases at least `release_factor X total_buffer_length` messages. So that time-ordering for late coming events are achieved to some extend.

Figure 6-2 shows how these four criterions are utilized within Buffering Service.

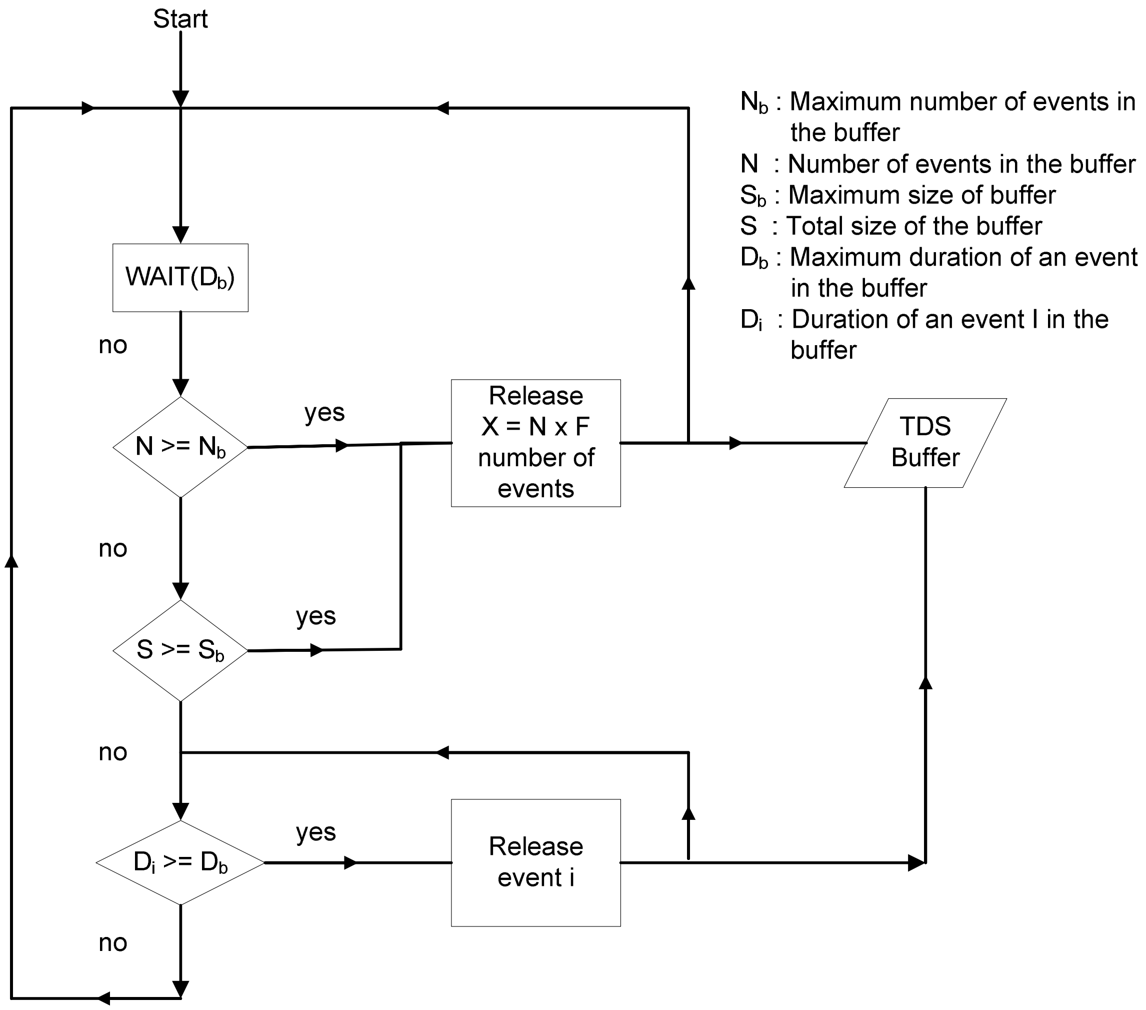

Start
N_b : Maximum number of events in the buffer
N : Number of events in the buffer
S_b : Maximum size of buffer
S : Total size of the buffer
D_b : Maximum duration of an event in the buffer
D_i : Duration of an event I in the buffer
WAIT(D_b)
no
N >= N_b
yes
Release X = N x F number of events
TDS Buffer
no
S >= S_b
yes
no
D_i >= D_b
yes
Release event i
no

Figure 6-2: Flowchart for releasing the events in the Buffering Service queue

6.3 Time Differential Service

In collaborative systems simply receiving messages in time-order may not be enough. An entity may also place constraints on the maximum jitter that it is willing to tolerate. The Time Differential Service (TDS) provides two very important functions. First, it reduces the jitter in messages caused by the network. Second, it releases messages while preserving the time spacing between consecutive messages. Preserving time spacing between messages is not an easy task primarily because most operating systems do not provide strict real-time capabilities. Depending on the operating system, the scheduling of processes and threads does not necessarily guarantee the CPU for that process or thread after a specified interval. For example, using Java on the Windows operating system, user-level threads can obtain the CPU back only after 10 milliseconds. Based on the scheduling configuration of Linux operating system this duration can vary from 1 millisecond to 10 milliseconds or more.

One of the main reasons that TDS uses threads rather than traditional polling to release events in the queue is to avoid high CPU utilizations. In the case of polling, in order to release events in the queue, their timestamps should be checked very frequently. This can lead to very high CPU utilizations. Furthermore, since the rate at which events are generated is not constant: the time spacing between consecutive events vary. Using threads ensures that the CPU utilizations are significantly lower. The reason that we have multiple threads instead of one to release the events in the queue is due to issues related to the underlying programming language (Java) and the operating system. For example, on Linux (Fedora 2), in order to check the timestamps every millisecond, we need to use at least three inter-leaving threads since each thread wakes up after a minimum of 3 milliseconds. On Windows, this value is 10 milliseconds; this high value may not be able to address jitter reduction adequately.

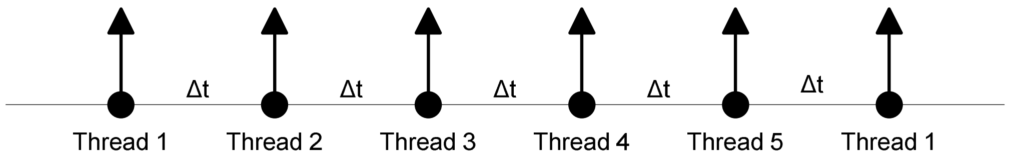


Figure 6-3: Ideal Time Differential Service thread invocation

TDS spawns five threads to process messages released by the buffering service, as shown in Figure 6-3. Threads are configured so that one thread runs for Δt time duration and wakes up after 4Δt time duration. Since threads are not strict real time (Java threads) we can only approximate the run time and sleep time duration values. Note that TDS itself maintains another buffer for processing. Each thread is initiated one after another with a specified time difference between consecutive initiations. Each thread sleeps for a specified time-slice. By interleaving the durations at which these threads wake-up, TDS can operate on the buffer at finer intervals while ensuring that CPU utilizations are low. The time-slice interval for individual threads impacts CPU utilization. We have observed that if the time interval between threads is 1 millisecond the CPU utilization stays around 5~6%, when this interval is decreased to 10 microseconds, it can reach about 20~25% on a Linux machine (1.5 GHz CPU 512 MB RAM). When a thread wakes up, it checks to see if any messages need to be released, and it does so if needed. It is done by comparing the message's timestamp, the local clock obtained from the high resolution timer and the time at which the last message was released. By preserving the time-spacing between messages, TDS reduces the jitter significantly.

6.4 Benchmark Results and Evaluation

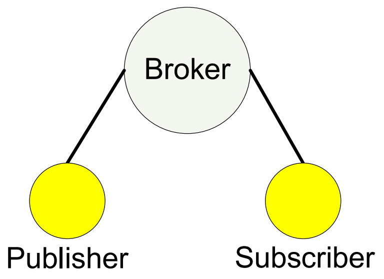


Figure 6-4: Experimental setup for TDS measurements

Experiment 1		Experiment 2	
Broker at Cardiff		Broker at Indiana	
OS	Red Hat Linux 7.1 2.96-79	OS	Red Hat Linux 3.2.3-6
CPU	Pentiun III 1 GHz	CPU	Intel (R) Xeon(TM) CPU 2.40 GHz
Memory	1 GB	Memory	2 GB
JVM version	1.4.1	JVM version	1.4.2_03
Clients at Indiana		Clients at Indiana	
OS	Red Hat Linux 3.4.2-6.fc3	OS	Red Hat Linux 3.4.2-6.fc3
CPU	Intel (R) Pentium (R) 4 CPU 1.5 GHz	CPU	Intel (R) Pentium (R) 4 CPU 1.5 GHz
Memory	1 GB	Memory	1 GB
JVM version	1.4.2_07	JVM version	1.4.2_07

Table 6-1: Broker and client machine specifications

Figure 6-4 and Table 6-1 depict the experimental setup for our Time Differential Service (TDS) related measurements. The transmitter (publisher) captures the input-video stream from a camera and publishes it using NaradaBrokering messages which are appropriately time-stamped. In the reported results, we ignored the first few messages that resulted in spikes due to media-player initializations.

For experiment setup (1) NaradaBroker is located at bouscat.cs.cf.ac.uk. We run transmitter and receiver clients on bozdag.ucs.indiana.edu. Using JMF libraries, transmitter client captures input video streams from camera and publishes them to a topic with native NB events. These events are timestamped before being published. Receiver client runs on the same machine as the transmitter client. The purpose of this is to eliminate the clock errors that are introduced by clock synchronization. In this case, both the receiver and the transmitter use the same underlying clock. So we do not have any clock synchronization problem. Events are published using NIOTCP connection in NaradaBrokering.

Published events reach the broker and from the broker they are streamed to the receiver client. Received events are first buffered in Buffering Service. Buffering Service time-orders events and releases them to Time Differential Service. Time Differential Service then releases them by preserving the time-spacing between consecutive events.

For experiment setup (2), NaradaBroker is located at gf3.ucs.indiana.edu. We run transmitter and receiver clients on bozdag.ucs.indiana.edu as in bouscat case. Gf3 and bozdag are on the same local area network (LAN). Events are also published using NIOTCP connection in NaradaBrokering.

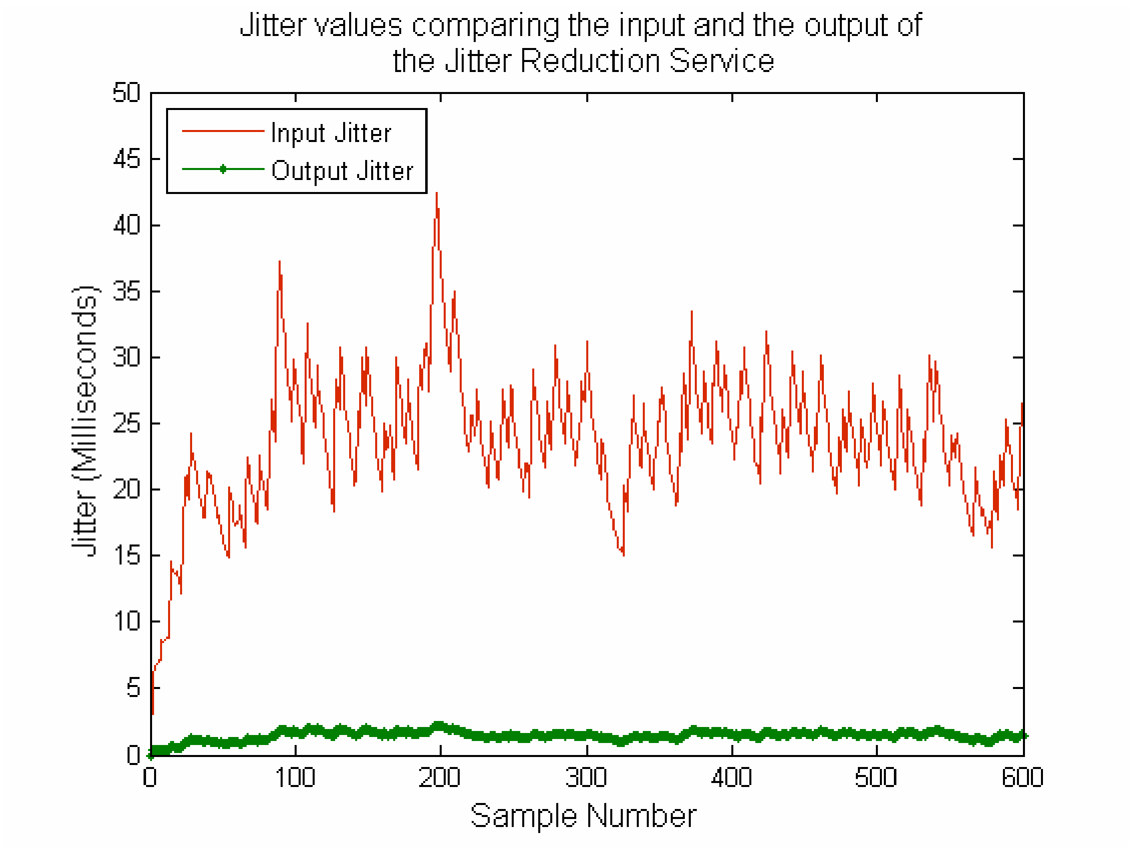


Figure 6-5: Jitter values comparing Buffering Service Input and TDS Output (Trans-Atlantic)

Figure 6-5 contrasts the jitters resulting from the experimental setup (1) involving machines at Cardiff and Indiana; the graph compares the jitters in the input to the buffering service and the output of the TDS. Please note that in the absence of the buffering service and TDS at a client, the jitters experienced at that node would be similar to that corresponding to the input of the buffering service.

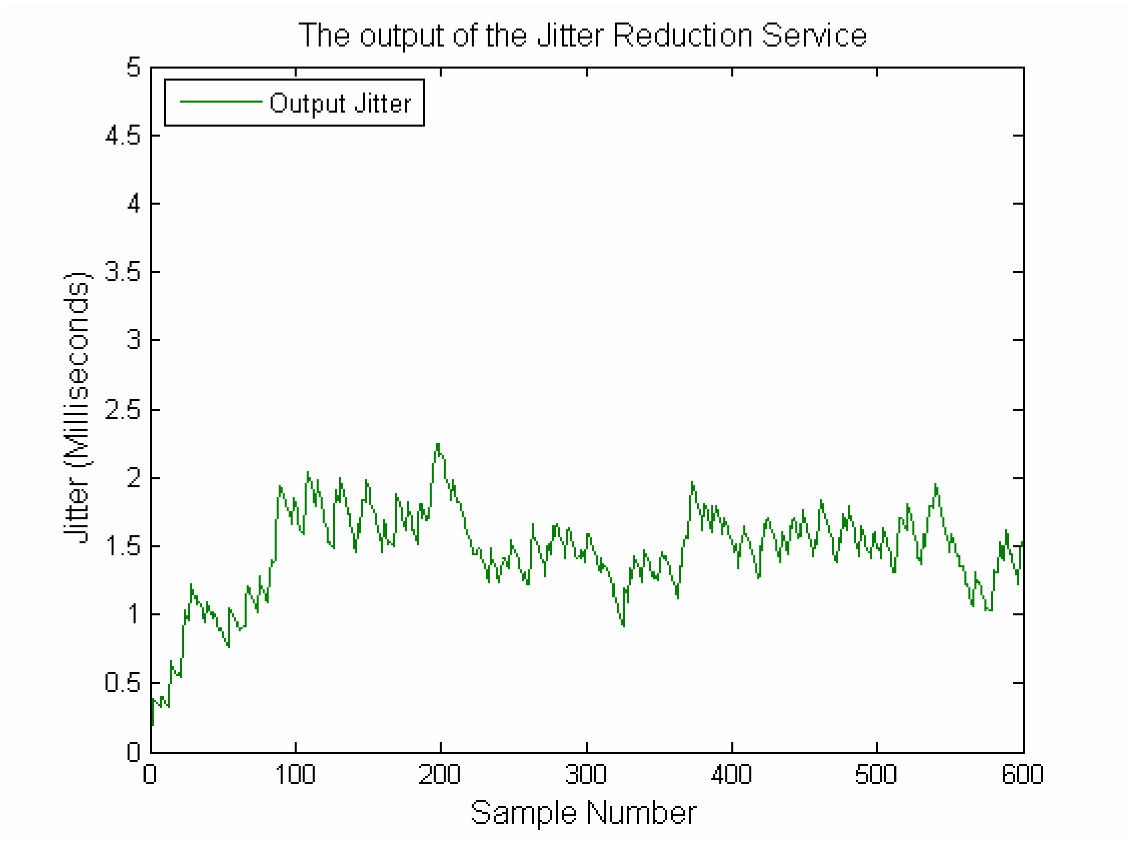


Figure 6-6: Jitter values from the TDS Output (Trans-Atlantic)

Figure 6-6 depicts only the jitters as a result of TDS for this experimental setup. The results demonstrate the significant reduction in jitter as a result of deploying the TDS. Table 6-2 summarizes mean value and the standard deviation of input and output jitter values for Trans-Atlantic benchmark.

Measured Parameter	Mean Value (milliseconds)	Standard Deviation
Input Jitter	23.71	5.00
Output Jitter	1.46	0.33

Table 6-2: Mean and standard deviation of input and output jitter values for Trans-Atlantic benchmark

We have also performed measurements within a Local Area Network (LAN) to profile the performance of TDS, the results reported here correspond to the experimental setup (2) depicted in Figure 6-4. Figure 6-7 contrasts the jitters in the input to the buffering service and the output of TDS. Once again, the results demonstrate jitter reduction even in cluster settings. Figure 6-8 shows only the output jitter for the LAN setting. We can achieve very low values of jitter. Table 6-3 summarizes mean value and the standard deviation of input and output jitter values for cluster benchmark.

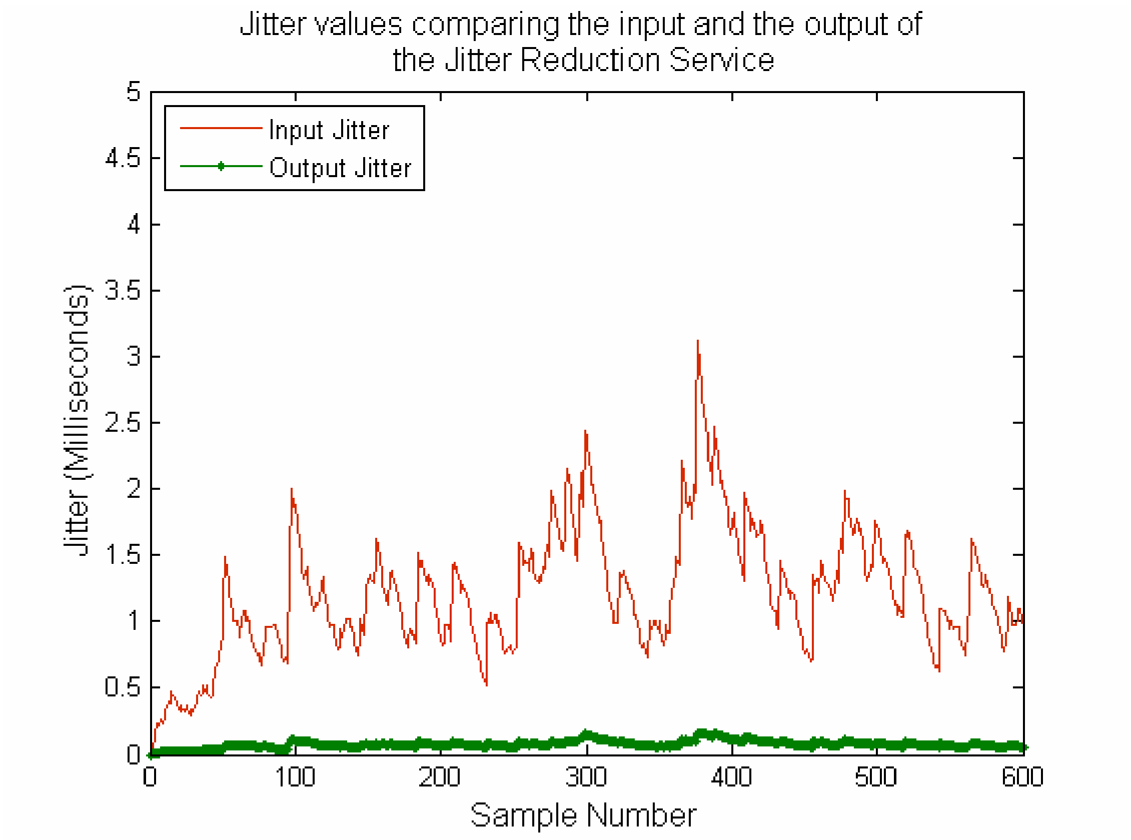


Figure 6-7: Jitter values comparing Buffering Service Input and TDS Output (Cluster setting)

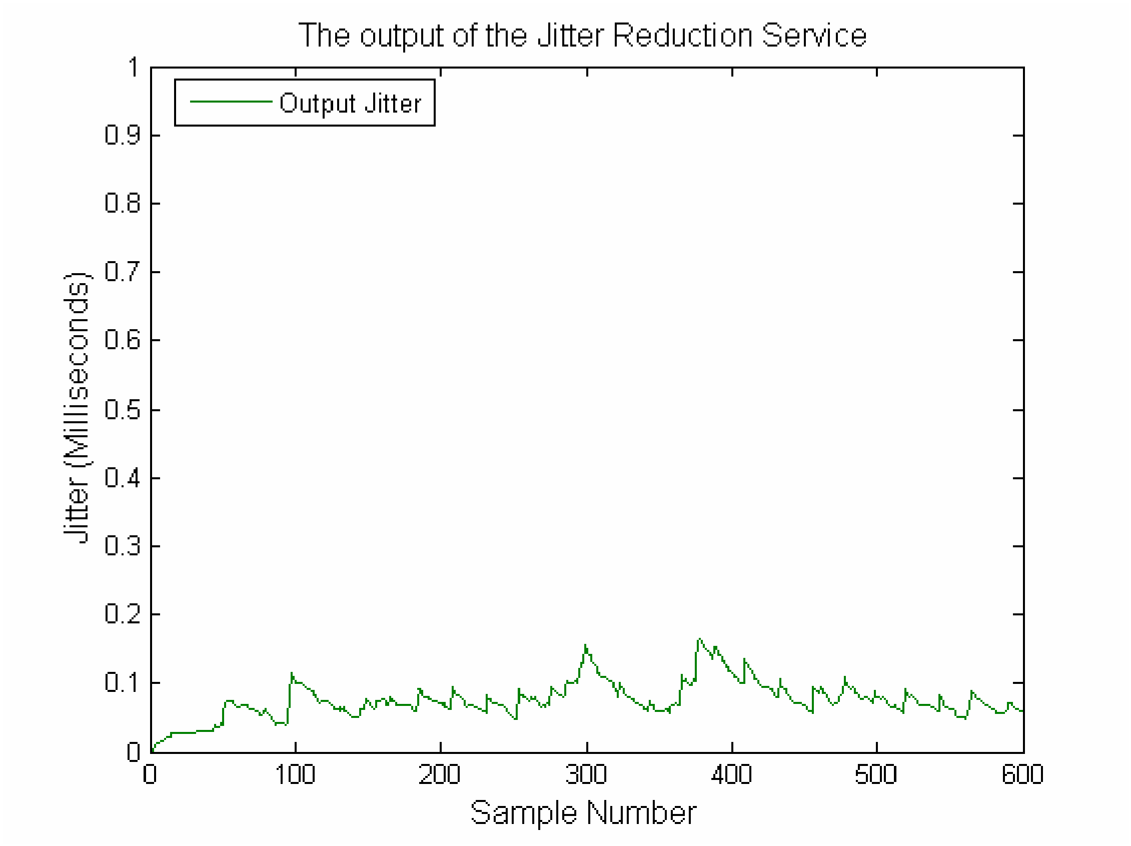


Figure 6-8: Jitter values from the TDS Output (Cluster setting)

Measured Parameter	Mean Value (milliseconds)	Standard Deviation
Input Jitter	1.22	0.47
Output Jitter	0.08	0.03

Table 6-3: Mean and standard deviation of input and output jitter values for Trans-Atlantic benchmark

6.5 Summary

In a distributed system, events are generated from multiple sources, and due to the underlying network, these events may reach the destination unordered. Receiving events time-ordered is also not enough for collaborative systems. The jitter reduction is important to reach a certain quality of service.

Buffering and Time Differential Services, together, provide a mechanism to improve the ordering of events and to reduce the jitter caused by the underlying network. By supplying a buffer, Buffering Service will ensure that the events received within the buffer duration are ordered, and that the events are delayed for buffer duration. Time Differential Service (TDS) reduces the jitter caused by the network by releasing the events and preserving the time spacing between consecutive events.

Using TDS in conjunction with buffering service will guarantee low jitter. Because benchmark results show that TDS can reduce the jitter values in a few milliseconds in a Trans-Atlantic setting and less than one millisecond in a cluster setting.

Chapter 7

Replicated and Fault Tolerant Repository

Reliable delivery of messages is an important problem that needs to be addressed in distributed systems. Topics over which authorized publishers and subscribers can have reliable communications are referred to as reliable-topics. The reliable delivery scheme in NaradaBrokering system enables reliable delivery of messages in the presence of link and node failures. The communication links within the system can also be unpredictable, with messages being lost, duplicated or re-ordered in transit over them, en route to the final destinations. This is facilitated by a specialized repository node. In this chapter, after we give an overview of the reliable delivery scheme in NaradaBrokering system, we present our strategy to make this scheme even more failure resilient, by incorporating support for repository redundancy. Each repository functions autonomously and makes decisions independently. The scheme enables updates to the redundancy scheme depending on the failure resiliency requirements. If there are N available repositories, reliable delivery guarantees will be met even if N-1 repositories fail.

7.1 Overview of NaradaBrokering Reliable Delivery Service

The scheme for reliable delivery of messages, issued over a reliable-topic, needs to facilitate error corrections, retransmissions and recovery from failures. In this system, a specialized *repository* node which *manages* this reliable-topic plays a crucial role in facilitating this. The repository facilitates reliable delivery from multiple publishers to multiple subscribers over its set of managed reliable-topics. The only requirement for the basic reliable delivery scheme is that if a repository fails, it should recover within a finite amount of time.

This reliable delivery guarantee holds true in the presence of four distinct conditions.

1. Broker and Link Failures: A broker network may have individual or multiple broker and link failures. Once the broker network recovers, the delivery guarantees should be met.
2. Prolonged Entity disconnects: If an entity is disconnected and misses events, the delivery guarantee will be met, with the entity receiving all the events missed in the interim once the entity reconnects.
3. Stable Storage Failures: It is possible that stable storages present in the system may fail. The delivery guarantees must be satisfied once the storage recovers.
4. Unpredictable Links: The events can be lost, duplicated or re-ordered in transit over individual links, en route to the final destinations.

To enable the management of reliable-topics, the repository facilitates the registration and de-registration of authorized clients (publishers & subscribers) for reliable communications over the reliable-topic. Publishers and subscribers that are not explicitly authorized (and registered) by the reliable-topic owner cannot avail reliable communications over that topic. To support error-corrections, retransmissions, and recovery from failures (including those of the repository itself) a repository also needs provision stable storage so that messages and other information pertinent to the reliable delivery algorithm can be stored.

Figure 7-1 summarizes the interactions between entities over a reliable topic. These interactions are explained in the following sub-sections.

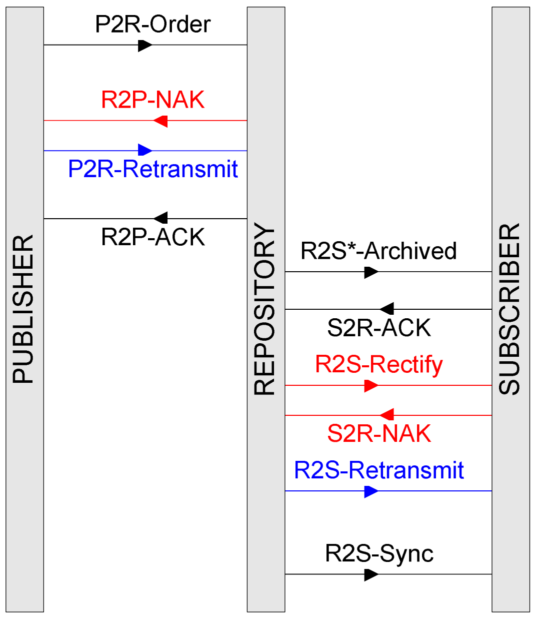


Figure 7-1: Summary of interactions between entities

7.1.1 Control-Events

The reliable delivery algorithm involves communications between various entities through the exchange of *control- events* (summarized in Figure 7-1). The control-events (simply events, for brevity, hereafter) relate to intermediate steps to facilitate reliable delivery, acknowledgements, error-corrections, retransmissions and recovery related operations. Our notation for events identifies the source, the destination(s) and the type of control-event: Source2Destination-ControlType. For purposes of brevity, we use only the starting alphabets of the entities involved in the exchange. Thus, an acknowledgement issued by the repository to the publisher is represented as R2P-ACK. The destination part is in **bold-face** if there are multiple destinations.

7.1.2 Publishing Messages

The messages issued by publishers to a topic will also be received at the repository. To ensure that the repository can know about and retrieve missed messages for every published message, the publisher also issues a P2R-Order event to the reliable-topic repository.

A publisher stores every message that it publishes, over a reliable-topic, in its local buffer (maintained in memory), which serves as a temporary storage. The catenation numbers contained within the P2R-Order event allows the reliable-topic repository to determine the order in which these messages were generated and to determine if messages were lost in transit.

7.1.3 Repository Processing of Published Message

Upon receipt of a message (issued over one of its managed reliable-topics), the repository queues the message in a temporary buffer, this message is not acted upon until the corresponding P2R-Order event is received. The repository also checks with the identifier contained in the event to see if this event has been previously received at the repository. If a message correlated with this duplicate event is in the repository's temporary buffer, that message is also discarded as a duplicate.

If the published message has been received, an acknowledgement – R2P-ACK event – is issued back to the publisher. If the published message corresponding to the P2R-Order event is missing, the repository issues a negative acknowledgement – R2P-NAK event –- to the publisher to retrieve the missing message. The repository also checks to see if there are any gaps in the P2R-Order events received from the publisher. When the repository stores the message it assigns a monotonically increasing *sequence number*.

7.1.4 Processing Repository Acknowledgements

A positive acknowledgement R2P-ACK event signifies successful receipt of the message and the corresponding P2R-Order event at the repository. The local buffer entry corresponding to this message can then be removed. Upon receipt of the negative acknowledgement R2P-NAK event the message(s) corresponding to the specified catenation number(s) are retrieved and prepared for retransmission. The retransmission occurs in the P2R-Retransmit event which contains both the original published message along with the catenation number for the message.

7.1.5 Message Storage and Persistence Notifications

Upon successful receipt of a published message at the repository, in addition to the operations outlined in section 7.1.3 the repository performs three additional functions.

First, depending on the topic type contained in the original published message, the repository loads the appropriate matching engine to compute destinations for the published message based on the registered subscriptions.

Second, the repository adds an entry to the dissemination table that it maintains. For a given sequence number, the dissemination table enables a repository to keep track of destinations that have not explicitly acknowledged the receipt of the corresponding published message.

Finally, the repository issues an event signifying the persistence of the published message. If S is the set of registered subscribers to a given reliable-topic, and if S* is the subset of subscribers whose subscription constraints are satisfied by the published message, then the R2S*-Persistent event signifies that it would be received only by that subset of subscribers.

The R2S*-Persistent event contains the sequence number assigned to the published message and also the identifier associated with the published message. A subscriber can then correlate a published message and its persistent event.

7.1.6 Processing Persistent Events at the Subscriber

A subscriber to a reliable-topic receives published messages from the publishers, and events from the repository. Upon receipt of a message from a publisher, a subscriber stores this message in its temporary local buffer. A subscriber releases a message only if both the message and the corresponding R2S*-Persistent event have been received.

If the subscriber has received both the message and the corresponding R2S*-Persistent event, it proceeds to issue an acknowledgement to the repository.

If the subscriber encounters a R2S*-Persistent event without the corresponding published message it concludes that the message was lost in transit. It issues a S2R-NAK event with the missing sequence number(s) for a given reliable-topic to retrieve the corresponding messages.

7.1.7 Processing Subscriber Acknowledgements

Upon receipt of an acknowledgement from the subscriber, the repository checks the dissemination table to see if there are any un-acknowledged messages within the range of sequence numbers contained in the S2R-ACK event.

On receipt of the S2R-ACK event from a subscriber, the repository updates the dissemination table entries corresponding to the sequence(s) contained in the event to reflect the fact that the subscriber received messages corresponding to those persistent event sequences.

The repository maintains a sync for every subscriber to the reliable-topics that it manages. The subscriber sync corresponds to the sequence number up until which the repository is sure that this subscriber has received all preceding messages. A subscriber maintains a local copy of this sync.

If the subscriber has received all the messages that it was supposed to receive, and if there were no missed messages between the subscriber’s current sync and the highest sequence number contained in the S2R-ACK event, the repository advances the sync point associated with this subscriber and issues a R2S-Sync event which notifies the subscriber about this sync advancement. Only upon receipt of this event is the subscriber allowed to advance its sync.

It is possible that the repository, based on the S2R-ACK event, detects that there are some persistent event sequences which were not explicitly acknowledged by the subscriber. The repository assumes that these un-acknowledged messages were lost in transit to the subscriber.

After the detection of missed sequences, the repository issues an R2S-Rectify event, which contains information pertaining to the client's sync advancement (if it is possible) and also the sequencing information and message-identifiers of the missed messages.

7.1.8 Processing Errors and Syncs Advances

Upon receipt of the R2S-Rectify event a subscriber performs three steps. First, the subscriber checks to see if any of the messages that it maintains in its temporary buffer has the identifier(s) corresponding to those listed in the R2S-Rectify; this accounts for the case where the R2S*-Persistent event was lost in transit to the subscriber, but the original published message was not. If the message exists in the temporary buffer, the message is delivered.

Second, the subscriber then proceeds to issue a S2R-NAK negative-acknowledgement event to the repository while excluding messages that were reliably delivered in the previous step. The S2R-NAK issued by the subscriber corresponds to the case where messages corresponding to the listed sequence numbers were lost in transit.

Finally, the subscriber advances its sync based on the advancement contained in the R2S-Rectify event. Note that this is also done in response to the R2S-Sync event.

7.1.9 Subscriber and Publisher Recovery

When a subscriber reconnects to the broker network after failures or a prolonged disconnect, it needs to retrieve the missed messages published over a reliable-topic. The recovering entity issues a recovery request S2R-Recovery for every reliable-topic that it had previously subscribed to.

Upon receipt of the recovery request, the repository scans the dissemination table starting at the sync associated with the client. The repository then generates an R2S-Rectify event, which is processed by the subscriber to advance its local sync and also to initiate retransmissions as described earlier in section 7.1.8.

In the case of publisher recovery, the repository's recovery response includes the last known catenation number for a given reliable-topic to which the publisher published.

7.2 Improvements to NaradaBrokering Reliable Delivery Service

We have extended the NaradaBrokering Reliable Delivery Service with a repository replication algorithm. This also impacts the reliable delivery protocol and can be summarized as follows:

i. A Publisher uses a monotonically increasing catenation number each topic it is publishing to. This is necessary for the repository to detect lost messages. Detecting gaps from a monotonically increasing sequence of catenation numbers is faster than detecting gaps from a sequence of <previous catenation number, catenation number> pair.

ii. To reduce the number of events exchanged between repository-publisher and repository-subscriber, we propose a lazy acknowledgment scheme. For this, the repository issues a R2P-NAK event to the publisher every few (configurable) seconds and the subscriber issues an S2R-ACK event to the repository every few (configurable) seconds. This allows us to include a list of lost message catenations into a single R2P-NAK event and a list of catenation numbers into a single S2R-ACK event, hence reducing the number of R2P-NAK and S2R-ACK events. This scheme can be viewed as lazy negative acknowledgement because repository issues NAK to the publisher including a list of catenation numbers of missed events and lazy positive acknowledgement because the subscriber issues ACK to the repository including a list of catenation numbers of received events.

iii. Upon the receipt of a P2R-Order event, the repository stores the corresponding message without waiting for missed messages. For lost messages, an entry is added to a storage table with a status flag to indicate that the message is being recovered and needs to be stored. A sequence number is also assigned for those missed events. This procedure avoids accumulation of messages. For example, suppose message *i* and all previous messages are stored successfully without any gaps between them. The messages from i+2 to i+20 arrive with the corresponding P2R-Order events which carry the catenation numbers of the corresponding messages. The repository assigns sequence numbers to missed events i+1 and messages from i+2 to i+20. An entry for message i+1 is added to the storage table with a flag indicating that it has not been received yet. Then repository stores messages from i+2 to i+20 successfully and publishes R2P-ACK event for them. For missed message i+1, the entry is overwritten as a message arrives. In the previous case, messages from i+2 to i+20 were not stored until message i+1 was received successfully. Waiting for message i+1 to store all messages afterwards also would cause burst in storing messages and publishing R2P-ACK events.

7.3 Repository Redundancy and Fault-tolerance

In the previous sections, we outlined our strategy to ensure a reliable delivery. In this scheme if there is a failure at the repository, the clients interested in reliable communication, over any of the managed reliable-topic, need to wait for this repository to recover prior to the reliable delivery guarantees being met. We now extend this scheme to ensure that the reliable delivery guarantees are satisfied in the presence of repository failures. To achieve this, we include support for multiple repositories –- constituting a *repository-bundle* –-for a given reliable-topic; it is not necessary that the topics managed by these repositories be identical. A repository may thus be a part of multiple repository-bundles at the same time.

We support a flexible redundancy scheme with easy addition and removal of repositories that manage a given reliable-topic. There are no limits on the number of repositories for a given reliable-topic. This scheme can sustain the loss of multiple repositories: in a system with **N** repositories for a given reliable-topic **N-1** of these repositories can fail, and reliable delivery guarantees are met so long as at least one repository is available.

The repositories that constitute the repository-bundle for a given reliable-topic function autonomously. At any given time, for a given reliable-topic, a client communicates with exactly one repository within the corresponding repository-bundle. This entity is also allowed to replace this repository with any other repository within the bundle at anytime.

Besides additional redundancy, and the accompanying fault-tolerance, a highly-available, distributed repository scheme enables clients to exploit geographical and network proximities. Using repositories that are "*closer*" ensures reduced latencies in the receipt of events from the repository. Packet loss-rates typically increase with the number of intermediate hops (for UDP and Multicast).

Besides the selection of the repository from a repository-bundle, as part of the bootstrap, operations at the clients are identical to those in place for a single repository.

7.3.1 Steering Repository

A publisher or a subscriber to a reliable-topic can interact with exactly one repository within the repository-bundle for that reliable-topic; this repository is referred to as the *steering repository* for that publisher/subscriber. At any time, a client is allowed to replace its steering repository with any other repository from the repository bundle.

Every repository within the bundle keeps track of a client's delivery sequences passively and actively. For a given entity, at any given time, there will be one steering repository operating in the *active* mode by initiating error-corrections and retransmissions. Other repositories operating in the *passive* mode do not initiate these actions.

At every repository, within the repository-bundle for a given reliable-topic, the list of registered clients is divided into two sets –- those that the repository steers and those that it does not. The repository operates in the *active* mode for the *steered* clients and in the *passive* mode for the clients that it does not steer. In the active mode, a repository performs all functions outlined in section 7.1. In the passive mode, a repository listens to all events initiated by the publishers and subscriber; however, the repository will not issue events – related to reliable communications – to the clients that it does not steer. Operating in the passive mode, allows a repository to take over as the steering repository for clients that it does not presently steer.

When a client is ready to initiate reliable communications, it has to designate a steering repository from the set of repositories within the repository-bundle associated with the reliable-topic. Selection of the steering repository is done based on network proximity using probes to compute network round-trip delays to the repositories. The client then issues an event over the repository's communications-topic designating it as the steering repository. Upon receipt of this event, the repository adds that client to its list of steered clients.

7.3.2 Ordered Storage of Published Messages

For every published message, the publisher issues a P2**R**-Order event (where **R** is the repository-bundle), which is received by all repositories within the repository-bundle. This allows all repositories within the repository-bundle to keep track of the published messages. However, only the steering repository (operating in active mode) for this publisher is allowed to issue the R2P-ACK and R2P-NAK events to acknowledge the receipt of messages and to initiate retransmissions respectively.

Retransmissions issued in response to the R2P-NAK event are sent to all repositories using the P2**R**-Retransmit event. The rationale for this is that if a message was lost in transit to the publisher's steering-repository, there is a good chance that the message (or the corresponding P2**R**-Order) event was also lost in transit to the other repositories.

7.3.3 Generation of Persistence Notification

Once a published message is ready for storage at the repository, the message is assigned a sequence number and is stored onto stable storage along with the published message. In this scheme each repository is autonomous, and thus maintains its own sequencing information. This implies that a message published by a publisher, may have different sequence numbers at different repositories. It follows naturally that the sync associated with a given subscriber can be different at different repositories. However, the catenation number associated with a publisher is identical at every repository within the repository-bundle.

A repository computes destinations associated with every published message. These destinations are computed based on the subscriptions registered by subscribers to this reliable-topic irrespective of whether subscribers they are steered by the repository or not. The repository then proceeds to issue a persistence notification. The topic associated with the R2***S****-Persistent event is such that it is routed only to the subset ***S**** of its steered subscribers with subscriptions that are satisfied by the topic contained in the original message.

7.3.4 Acknowledgements, Errors and Syncs

Upon receipt of R2***S****-Persistent events from its steering repository, a subscriber proceeds to issue acknowledgements. This acknowledgement, the S2**R**-ACK is issued over the repository-bundle communications topic. Since, the message is received by the repository-bundle, all repositories are aware of delivery sequences at different subscribers. The S2**R**-ACK event contains sequence numbers corresponding to its steering repository and also includes the identifier associated with the steering repository.

Error correction, and sync advancements, for a given subscriber is initiated by its steering repository through the R2S-Rectify event. Retransmission requests by a subscriber are targeted by its steering repository in the S2R-NAK event.

7.3.5 Gossips between Repositories

Repositories within a repository-bundle gossip with each other. Repositories within a repository-bundle need to exchange messages about the registration/de-registration of clients to a managed reliable-topic as well addition and removal of subscriptions by clients to a reliable-topic. A given repository stores each of these actions and assigns each action the next available sequence number.

7.3.6 Processing Stored Messages

A repository assigns monotonically increasing sequence numbers to each message that it stores. At regular intervals or after the storage of a certain number of messages, the repository issues a Gossip-ACK event, which contains an array of entries corresponding to its storage of published messages. Each entry contains the publisher identifier, the catenation number assigned by the publisher, the message identifier and the sequence number assigned by the repository. This gossip message plays a big role in allowing repositories to be aware of the sequence numbers associated with a specific message at different repositories.

Every repository also maintains a repository-table for each of the other repositories as shown in Figure 7-2. In these tables for every publisher-catenation number pair, the repository maintains a sequence number assigned by every repository (including itself) within the bundle. Upon receipt of the Gossip-ACK event from a repository, an entry is added to the corresponding table to reflect the sequence numbers assigned at different repositories.

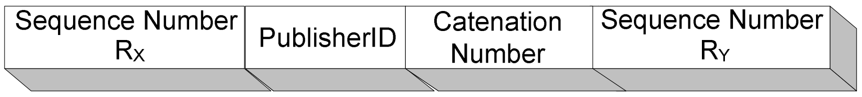


Figure 7-2: Widget with table maintained at repository Rx

The repository table thus allows a repository to correlate sequence numbers assigned to a given message at every other repository. The table also track messages not received by other repositories. This allows a repository to recover from any other repository after a failure. These repository tables also play an important role in processing acknowledgements from subscribers that it does not steer.

7.3.7 Processing Subscriber Acknowledgements

When a repository receives a S2**R**-ACK event from a subscriber, it checks to see if it steers the subscriber. If it does, the repository simply proceeds to update its dissemination table to reflect receipt of the message at the repository. If the repository does not steer the subscriber that issued the acknowledgement, the repository retrieves the sequence number corresponding to the original message from the repository-table. It then proceeds to update the dissemination-table for that sequence number to confirm the receipt from the subscriber in question. This scheme allows all repositories to be aware of delivery sequences at various subscribers irrespective of whether they are steered or not. Furthermore, based on the S2**R**-ACK acknowledgements from a subscriber, every repository computes its sync for that subscriber. Additionally, at regular intervals, a repository gossips about sync advancements for its steered subscribers.

Since all repositories process acknowledgements from all the subscribers any one of these repositories can take over as the steering repository for a given subscriber. Furthermore, since all messages issued by all publishers are stored at all repositories, and since the catenation numbers are identical on all repositories, a repository can take over as the steering repository for a given publisher at any time.

7.3.8 Dealing with Repository Failures

A publisher detects a failure in its steering repository, if it does not receive R2P-ACK events for published messages within a certain time duration. A subscriber detects a steering-repository failure if it receives published messages to reliable-topics, but no corresponding persistence notifications from its steering repository. These clients then proceed to discover a new steering repository. The publisher then exchanges information about its catenation number with the replacement steering repository. If there is a mismatch wherein the steering repository's catenation is lower than that at the publisher, the repository proceeds to retrieve this message from a repository within the bundle.

7.3.9 Recovery of a Repository

Upon recovery from a failure, it needs to discover an assisting-repository: this is a repository within the repository bundle that is willing to assist the repository in the recovery process. The recovering replica first checks to see if the list of registered clients and subscriptions have changed, and proceeds to retrieve updates to this list.

Next, the repository proceeds to retrieve the list of catenation numbers associated with the publishers. Based on these catenation numbers, the repository computes the number of missed messages and proceeds to set aside the corresponding number of sequences. For messages (missed and real-time) that it stores, a recovering-repository issues Gossip-ACK acknowledgements at regular intervals.

The recovering-repository proceeds to do two things in parallel. First, it proceeds to retrieve missed messages from the assisting repository. For every missed message the recovering-repository also retrieves a dissemination list associated with it. This allows a repository to keep track of the subscribers that have not acknowledged these messages. Additionally, the repository-table entries corresponding to each message are also retrieved. A repository cannot be the steering repository for any entity till all the missed messages have been retrieved.

Second, it subscribes to various communications topics so that it can start receiving messages published in real-time. The first time a repository receives a message from a publisher, it checks to see if the catenation number associated with that message indicates missed messages. This could happen because the missed message(s) would have been in transit to the assisting repository. Thus, during recovery if the assisting repository reported a catenation number of `2000`, and if the catenation number associated with the first real-time message received from the publisher is `2010` it implies that there are `9` additional messages from this publisher that are missed. The repository sets aside `9` sequence numbers, and issues a request to retrieve these messages. The repository also proceeds to store the published message based on the newly advanced sequence number.

7.3.10 Addition of a Repository

When a repository is added to the repository-bundle associated with a reliable-topic, the newly added repository takes the following steps. First, it needs to discover an assisting-repository: this is a repository which is present in the repository bundle and one which is willing to assist the repository in the addition process.

Second, the repository retrieves the list of registered clients, and the subscriptions registered by the registered subscribers. As described in section 7.3.9 the repository then proceeds to retrieve missed messages along with the corresponding dissemination lists and repository-table entries in addition to processing real-time messages.

7.3.11 Graceful Removal of a Repository

When a repository is ready to leave a repository bundle, it proceeds to issue an event to its active steered clients, requesting them to migrate to another repository. The departing-repository then operates in silent mode as far as the clients are concerned. The departing-repository also gossips with other repositories within the repository-bundle to check if the catenation numbers associated with previously steered publishers is greater than or equal to its last known value at the departing-repository.

Once a repository has confirmed that all messages published by its previously steered publisher have been received at one of the repositories within the bundle, it is ready to leave the repository-bundle. The departing repository then simply issues a Gossip-LEAVE event. Repository table entries corresponding to this repository will no longer be maintained at other repositories.

7.4 Experimental Results

We have measured several aspects of the reliable delivery framework to have an idea of the costs involved in reliable communications. All processes executed within JVM 1.6.0 with Linux as the OS were hosted on a 100 Mbps LAN. Machines involved in the benchmark have the following profile: Repositories run on machines with CPU (Xeon, 2.4 GHz processor) and 2GB RAM, while other components run on machines with CPU (2 Xeon - 4 core each, 2.3 GHz processor) and 8GB RAM

The repositories, brokers and sets of clients are all hosted on different machines. In all test cases, to obviate the need for clock synchronizations, the publisher and the *measuring* subscriber (which reports the results) were hosted on the same machine. We have benchmarked end-to-end delivery latency in order to understand the cost that the replication scheme causes.

In our first benchmarks, we used the topologies depicted in Figure 7-3 and Figure 7-4 for benchmarking in order to understand the cost of reliable delivery scheme. These two topologies are basic cases with one broker, one repository, one publisher and one subscriber.

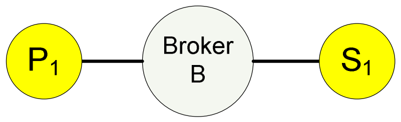


Figure 7-3: Topology A; 1 broker (B), 1 publisher (P1) and 1 measuring subscriber (S1)

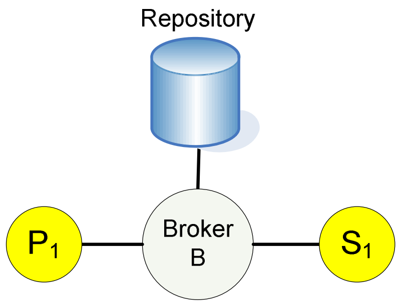


Figure 7-4: Topology B; 1 broker (B), 1 publisher (P1), 1 measuring subscriber (S1) and 1 repository

The results of delivery latency benchmarks for different payload sizes for topologies A and B are shown in Figure 7-5 and Figure 7-6. The cost of reliable delivery is acceptable because the cost increase from topology A to topology B is almost constant, particularly for message sizes less than 10 KB. The cost increase is due to control signaling taking place between the publisher and the repository and between the subscriber and the repository. Signaling between the publisher and the repository makes sure that the message published is stored at the repository. Signaling between the subscriber and the repository makes sure that the subscriber receives the message. Because of this signaling and because of the storage overhead there is an increase in reliable delivery scheme.

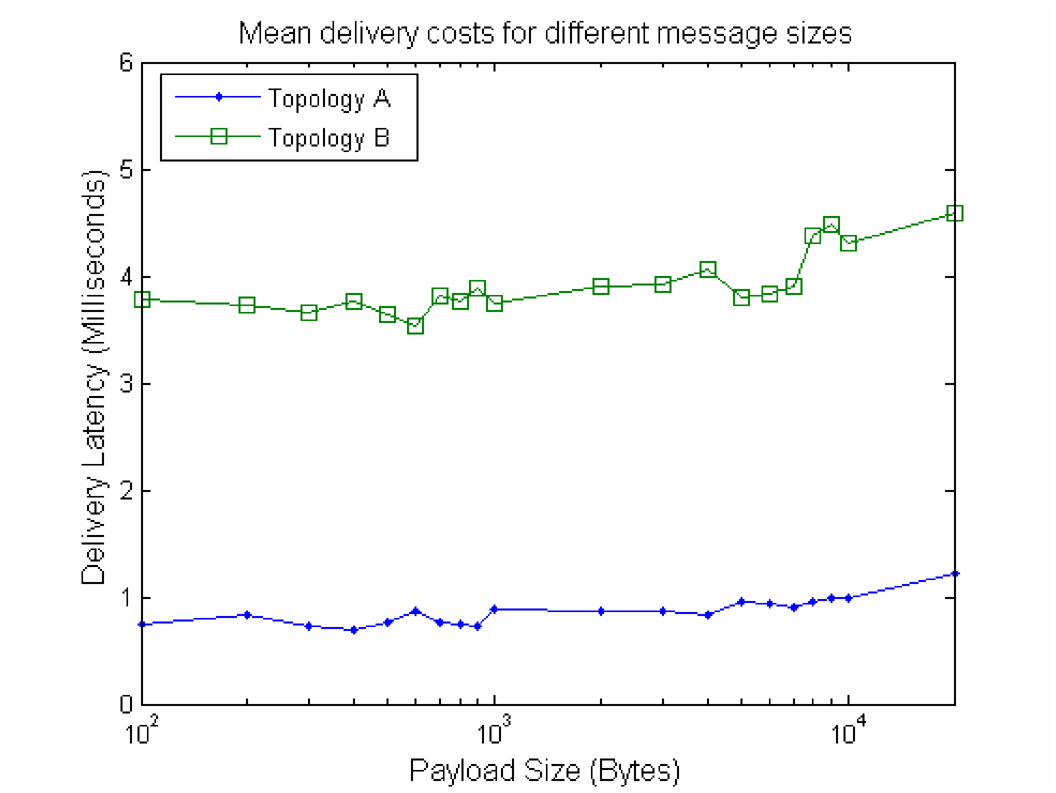


Figure 7-5: Mean delivery costs for topologies A and B.

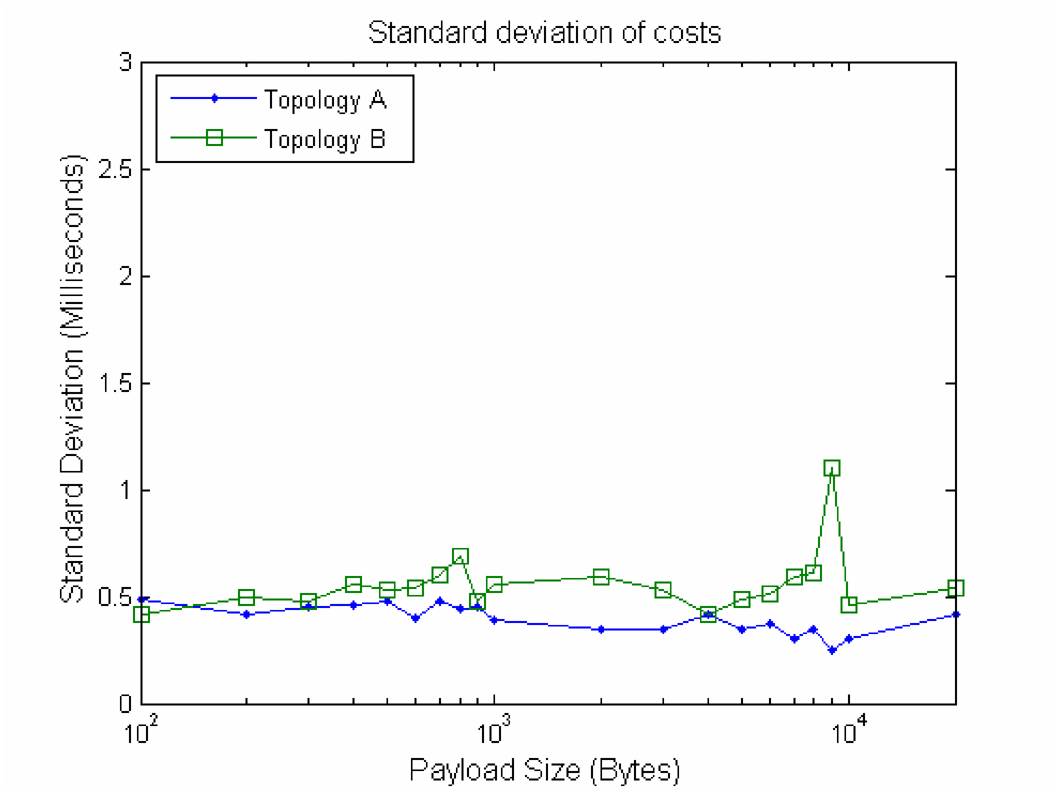


Figure 7-6: Standard deviation of reliable delivery costs for topologies A and B.

In the next topologies, from topologies C to F, as shown in Figure 7-7, Figure 7-8, Figure 7-9 and Figure 7-10, we would like to benchmark the cost of adding additional repositories to the system while we provide reliable delivery for a topic. We have increased the number of repositories from zero to three.

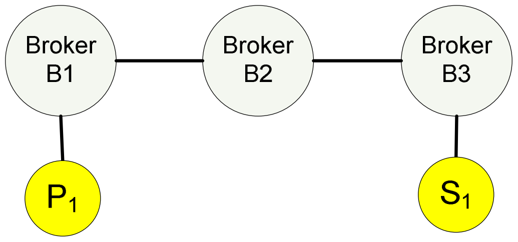


Figure 7-7: Topology C; 3 brokers (B1, B2, B3), 1 publisher (P1) and 1 measuring subscriber (S1)

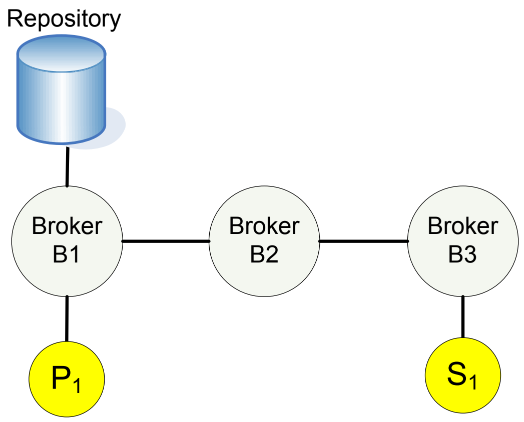


Figure 7-8: Topology D; 3 brokers (B1, B2, B3), 1 publisher (P1), 1 measuring subscriber (S1) and 1 repository

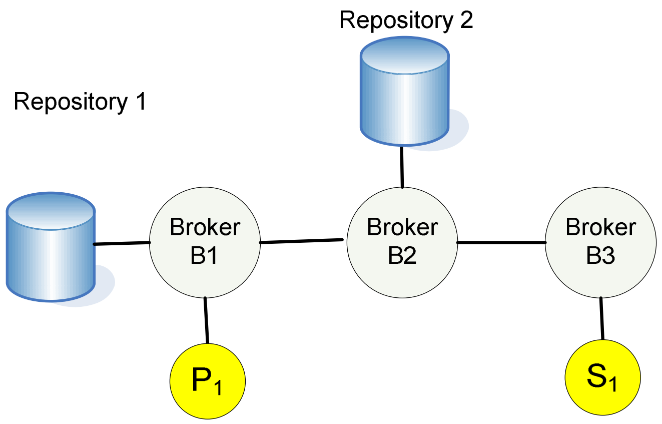


Figure 7-9: Topology E; 3 brokers (B1, B2, B3), 1 publisher (P1), 1 measuring subscriber (S1) and 2 repositories

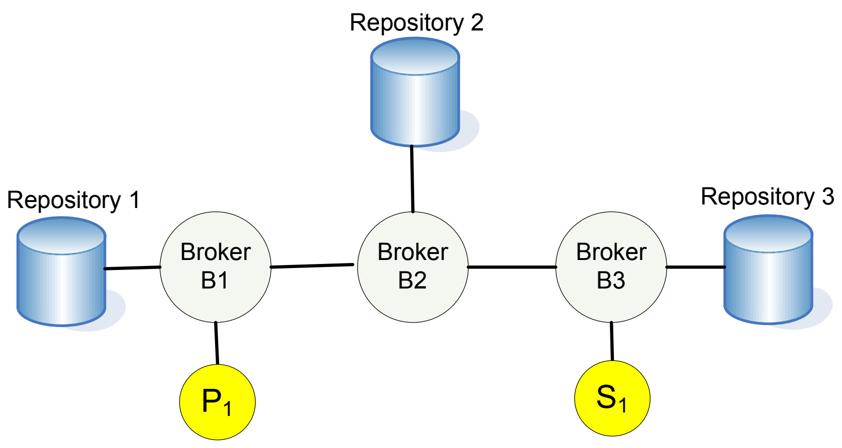


Figure 7-10: Topology F; 3 brokers (B1, B2, B3), 1 publisher (P1), 1 measuring subscriber (S1) and 3 repositories

The results of delivery latency benchmarks for different payload sizes for topologies C, D, E and F are shown in Figure 7-11 and Figure 7-12. The cost increase is acceptable for redundant repositories as well. For each repository, the mean latency increases for a few milliseconds. However, the cost increase is again constant for each repository addition.

For publisher the steering repository is Repository 1 in topologies D, E and F. Other repositories are storing the same topic that the publisher is publishing to. The reason for cost increase for repository addition is that the steering repository also handles messages from other repositories. Repository 2 and 3 publishes control messages as we described in section 7.3. So Repository 1 also maintains reliable delivery topic information at repositories 2 and 3. In addition to that, Repository 1 also publishes control messages so that other repositories maintain its topic information in their storage as well. Considering these control messages taking place to achieve redundant repository support for reliable delivery topics, the cost is acceptable.

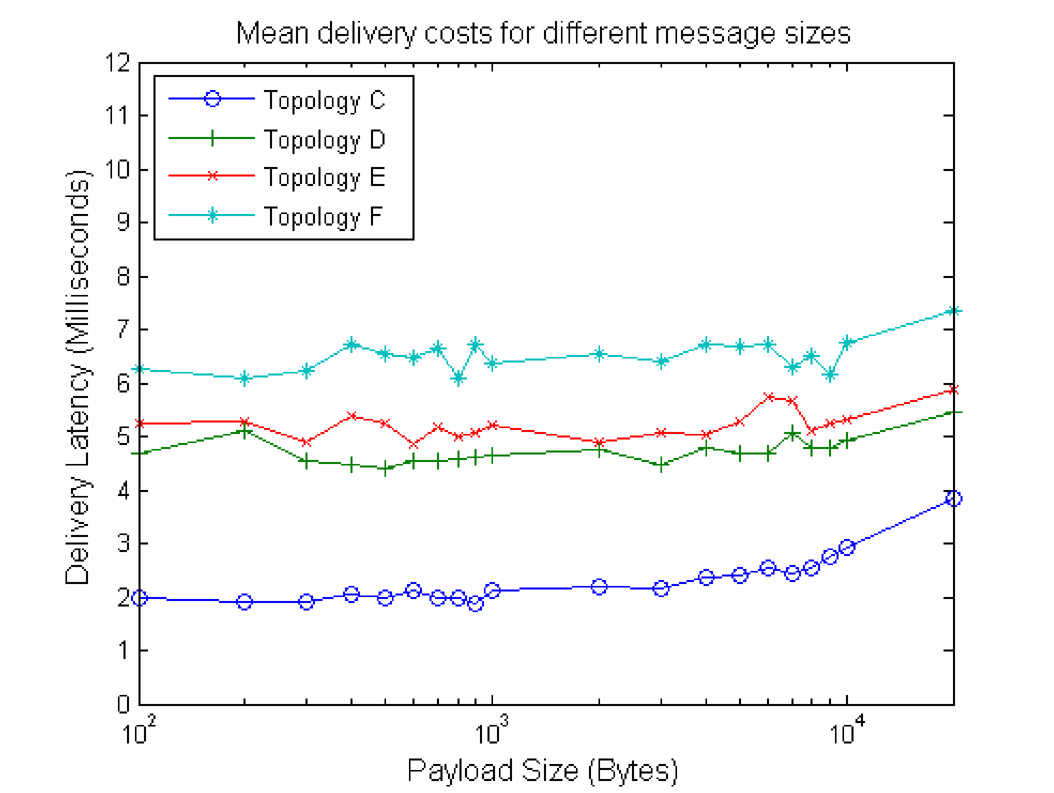


Figure 7-11: Mean delivery costs for topologies C, D, E and F.

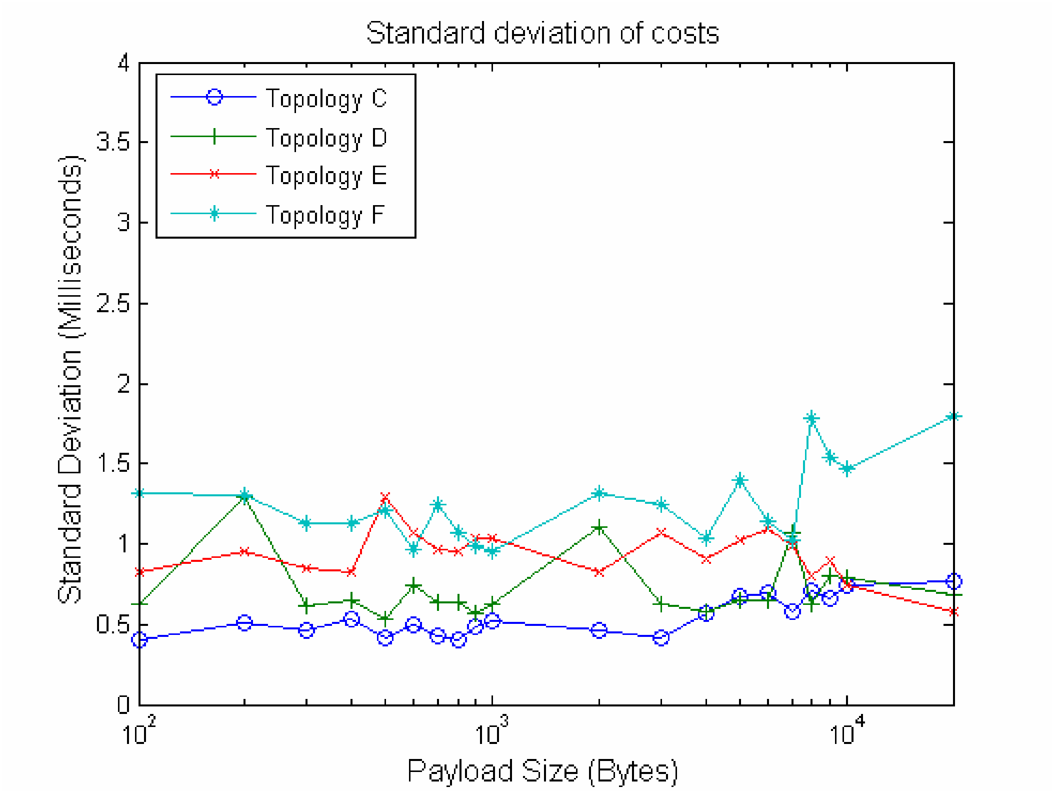


Figure 7-12: Standard deviation of latency measurements shown in Figure 7-11.

We also provided benchmark results of topologies A, B, C, D, E and F in a common figure (Figure 7-13 and Figure 7-14) so that we can compare these results. Topology A and B contain only one broker but other topologies (C – F) contain three brokers. Comparing the lines of topology A and C where none of these have repositories, we can have an idea of cost increase with increase in the number of brokers. This suggests that when we compare the effect of repositories added to the system, we need to keep in mind the number of brokers that the brokering network has. Because depending on where the client is connected, the number of brokers is also a factor in delivery latency.

In general, the costs increase as the number of brokers and the number of repositories for a given reliable-topic increase. The results also demonstrate that the costs for reliable delivery are acceptable

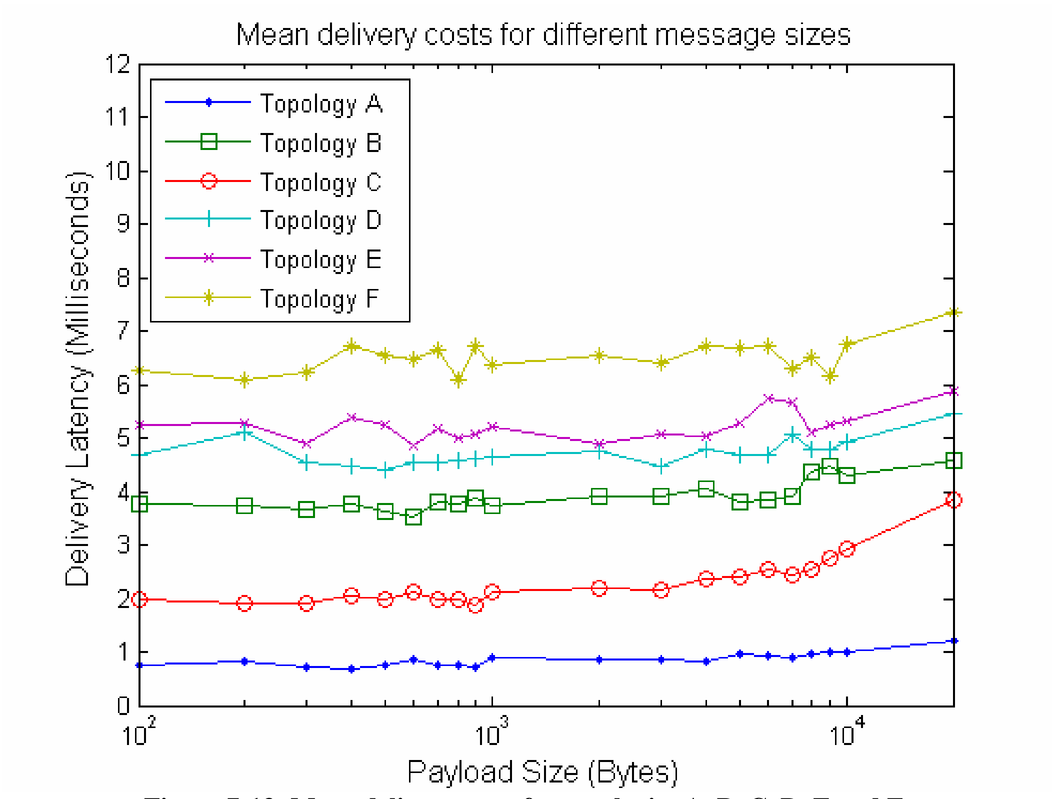


Figure 7-13: Mean delivery costs for topologies A, B, C, D, E and F.

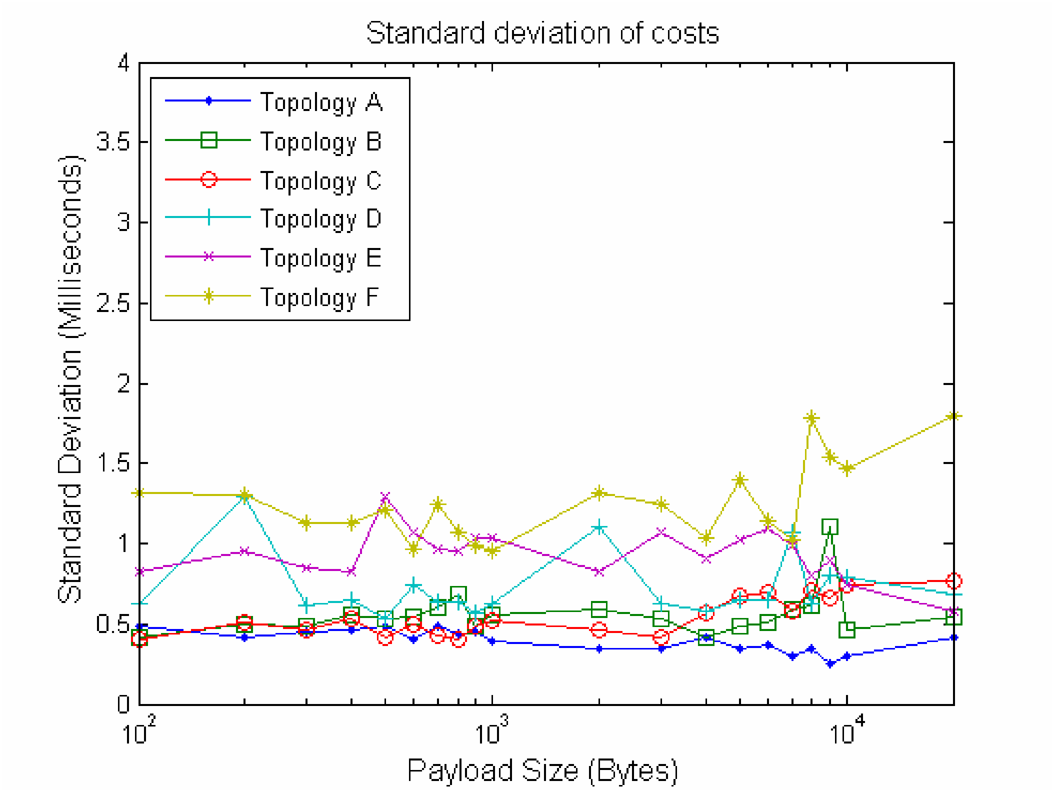


Figure 7-14: Standard deviation of latency measurements shown in Figure 7-13.

In addition to above benchmarks, we also benchmarked costs associated with various aspects of the framework. The results are summarized in Table 7-1. Some of these costs are reported in milliseconds (msec) and some in microseconds (µsec).

Operation		**Mean**	**Standard Deviation**	**Standard Error**
Storage Overheads				
	Message Storage	1408 µsec	141.71 µsec	31.69 µsec
	Message Retrieval	669 µsec	77.93 µsec	17.43 µsec
RDS recovery in a single repository system				
	Recovery time after a failure or scheduled downtime	85.7 msec	4.3 msec	958 µsec
Client Recovery				
	Time to generate Recovery response for a publisher	825 µsec	215 µsec	48 µsec
	Time to generate Recovery response for a subscriber	1613 µsec	588 µsec	131 µsec

Repository Recovery (1000 missed messages and 20 clients)				
	Recovery Response generation for Repository	59.28 msec	5.54 msec	2.77 msec
	Recovery Time at repository	11172 msec	1733 msec	867 msec
Repository Gossips				
	Generation of gossip	243 µsec	50 µsec	11 µsec
	Processing a gossip	241 µsec	16 µsec	3 µsec

Table 7-1: Reliable delivery costs within the framework. Values reported for a message size of 8KB.

7.5 Summary

In this chapter, we gave an overview of the NaradaBrokering reliable delivery scheme and the improvements we have made to this scheme. Then we extended the scheme by incorporating support for repository redundancy to make it more failure resilient. With this scheme, if there are N available repositories, reliable delivery guarantees will be met even if N-1 repositories fail. We also measured several aspects of the reliable delivery framework to have an idea of the costs involved in reliable communications for different topologies.

Chapter 8

Archived Streaming Service

Streaming of archived multimedia content is widely used in the today's Internet world. Usually the reason for this is to provide Media-On-Demand to users with high quality multimedia content. A client has some control over the stream, such as pausing, rewinding or forwarding depending on the choices provided to it. We have already explained streaming concept in chapter 4 while we explained XGSP Streaming Gateway. In this chapter, we focus on streaming session initiation and metadata support for describing archive and replay sessions including streams in these sessions. Our streaming architecture is based on messaging middleware and Web Services technology. An important aspect of this architecture is that it utilizes a fault tolerant distributed repository, WS-Context Service, to maintain session and stream metadata.

8.1 Archived Streaming Service Design

Archived Streaming Service provides metadata support and messaging middleware topic management to streaming sessions for archiving streams and replaying archived streams. There are several reasons for the need of archived streaming service. Some of these can be listed as follows;

i) Streams are archived with the topics they are published to. . That is, events store includes the original topic names they were published to. During a replay session, these archived streams need to be published to different topics, since there may be several replay sessions for the same archived streams. Although data in these streams are same, when they are published for a different replay session, they are treated as different streams since the control of each of these sessions is at a different client.

ii) As we explained in chapter 7, while replicating the repository to increase the fault tolerance of the system, we need unique templateIDs for every template that we store. The templateIDs used by the repository to map the topic names of the streams are assigned by the archiving service.

iii) There may be many repositories storing many different templates. Also one template may be stored (replicated) at many repositories. In this case, a component should decide which repositories should be used during recording or replay. This can be decided by the archived streaming service. In another sense, repositories are managed for load balancing.

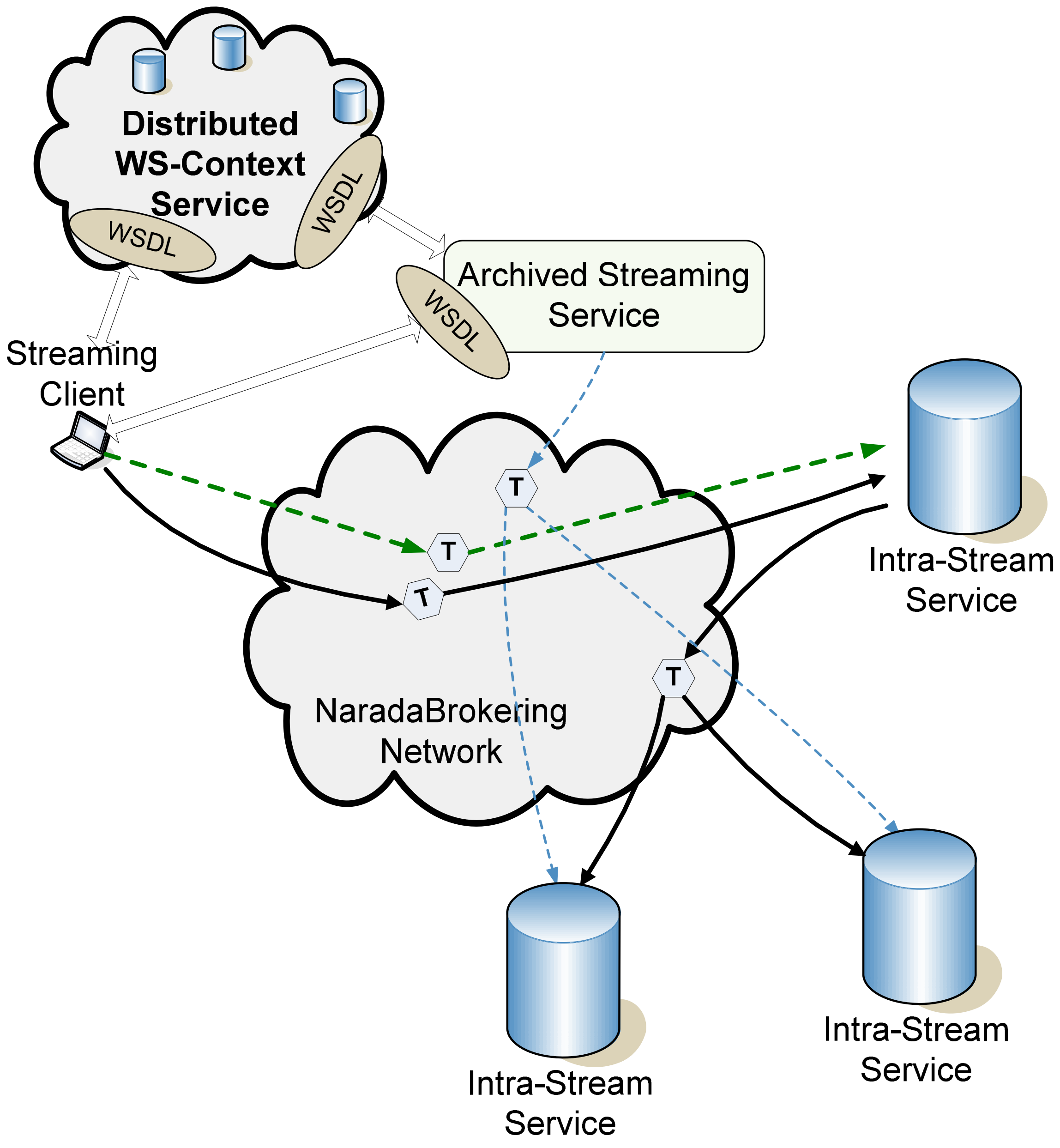
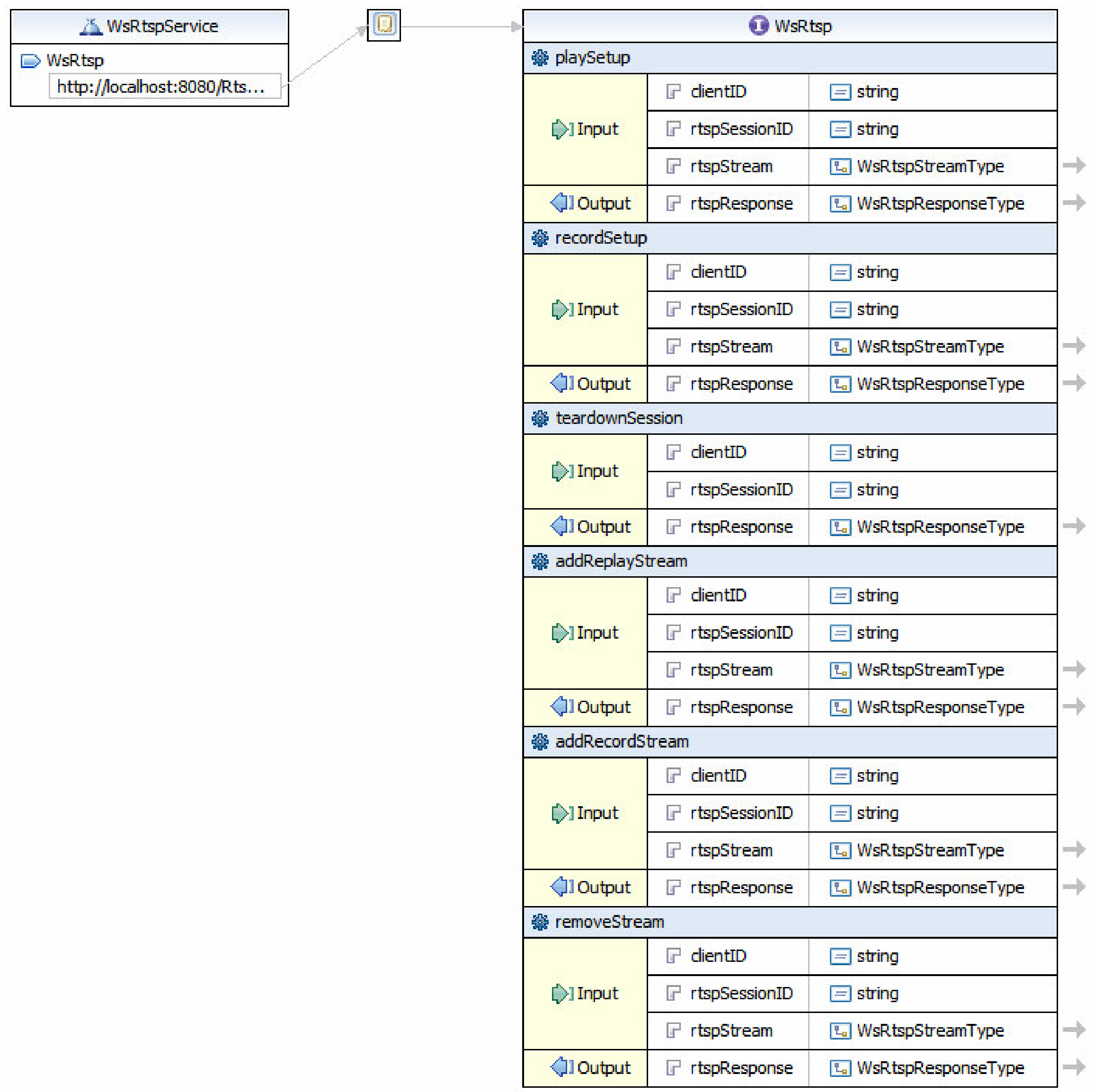


Figure 8-1: Archived Streaming Service and interaction with other components in the system

Figure 8-1 demonstrates Archived Streaming Service and interactions with other services in the system. Archived Streaming Service is a Web Service and can use publish/subscribe mechanism to interact with other components in the system. Archived Streaming Service can directly access WS-Context Service to update and retrieve metadata of sessions and streams. While doing that it also interacts with Generic

Streaming Service through messaging middleware to initialize the session for recording or replay.



WsRtspService

WsRtsp

http://localhost:8080/Rts…

WsRtsp

Operation	Message	Part	Type
playSetup	Input	clientID	string
		rtspSessionID	string
		rtspStream	WsRtspStreamType
	Output	rtspResponse	WsRtspResponseType
recordSetup	Input	clientID	string
		rtspSessionID	string
		rtspStream	WsRtspStreamType
	Output	rtspResponse	WsRtspResponseType
teardownSession	Input	clientID	string
		rtspSessionID	string
	Output	rtspResponse	WsRtspResponseType
addReplayStream	Input	clientID	string
		rtspSessionID	string
		rtspStream	WsRtspStreamType
	Output	rtspResponse	WsRtspResponseType
addRecordStream	Input	clientID	string
		rtspSessionID	string
		rtspStream	WsRtspStreamType
	Output	rtspResponse	WsRtspResponseType
removeStream	Input	clientID	string
		rtspSessionID	string
		rtspStream	WsRtspStreamType
	Output	rtspResponse	WsRtspResponseType

Figure 8-2: Archived Streaming Service operations

Each streaming session is assigned a Universally Unique Identifier (UUID) [112] which can be generated by the client as well. If it is not provided by the client, Archived Streaming Service generates one and provides it to the client during the session setup. This service has the operations shown in Figure 8-2.

All of these operations return *WsRtspResponseType* XMLmessage. The XML fragment for this message is shown in Figure 8-3. Record stream and replay stream information is passed via *WsRtspStream* field. WsRtspStream contains either a list of AtomicStreams or a list of ReplayStreams as shown in Figure 8-4.

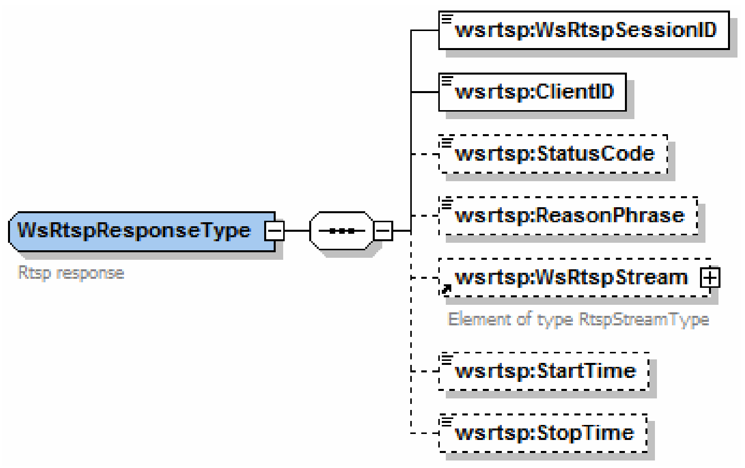


Figure 8-3a

```
<xs:complexType name="WsRtspResponseType">
  <xs:annotation>
    <xs:documentation>Rtsp response</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="WsRtspSessionID" type="xs:string"/>
    <xs:element name="ClientID" type="xs:string"/>
    <xs:element name="StatusCode" type="xs:int" minOccurs="0"/>
    <xs:element name="ReasonPhrase" type="xs:string" minOccurs="0"/>
    <xs:element ref="wsrtsp:WsRtspStream" minOccurs="0"/>
    <xs:element name="StartTime" type="xs:long" minOccurs="0"/>
    <xs:element name="StopTime" type="xs:long" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Figure 8-3b

Figure 8-3: WsRtspResponseType a) graphical representation and b) XML fragment

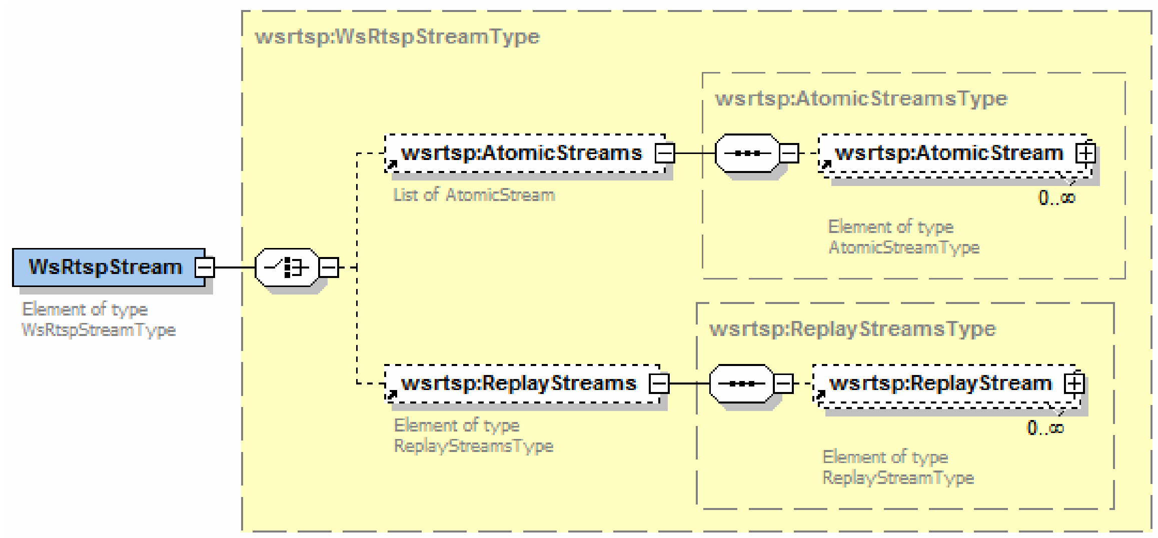


Figure 8-4: WsRtspStream definition as AtomicStreams and ReplayStreams

Sessions are initialized with recordSetup and playSetup operations. Each session is differentiated with a session ID and it is unique. Sessions are closed with teardownSession operation which only requires client ID and session ID.

8.1.1 Recording Sessions

If a client wants to record a session which may include one or more streams, it needs to know which topics it wants to record. It may retrieve the topic names either from WS-Context Service or it can be the source of the stream so that it already knows the topic it is publishing to.

Since we use publish/subscribe middleware, client initiating the archiving process does not need to be the one who is publishing the streams. It can be any separate component that can initiate the recording. Some of the operations shown in Figure 8-2 can be called in recording session while some of them are not allowed.

The first operation that should be called in a recording session is recordSetup operation. As an input, it also accepts an input of *WsRtspStreamType* which is a list of

*AtomicStream*s. When returning the *WsRtspResponseType* message, NaradaBrokering template information is included to each of the *AtomicStream* provided. *AtomicStream* representation is given in Figure 8-10. A client uses that information to initiate storing of streams (templates). In case the Archived Streaming Service cannot be accessed after the setup operation, the client just simply makes another recordSetup call to any available Archived Streaming Service. Since the new Archived Streaming Service already knows the session state from WS-Context Service, the session can be reestablished and recovered.

addRecordStream is to add more streams to a recording session while removeStream is used to remove streams from a recording session. Since *WsRtspStreamType* can contain multiple stream definitions, in all of these operations multiple streams can be added or removed.

8.1.2 Replay Sessions

A replay session can include multiple streams as well. As in recording case, client needs to know which streams it wants to replay. It can also retrieve this information from WS-Context Service.

When a client initiates a replay session, related information is stored in WS-Context Service. If another client wants to replay the same session, it can just retrieve replay topics from WS-Context Service and subscribe to these replay topics without interacting with Archived Streaming Service. In this case, a client is just a listener of the session and cannot have control over the replay session.

As in recording case, some of the operations shown in Figure 8-2 can be called in replay session while some of them are not allowed. The first step to initiate a replay session is to call playSetup operation. This will initialize the replay session. When a client passes *WsRtspStreamType* message, it sets the ReplayStreams field. ReplayStreamType message contains NBInfoType and AtomicStreamType fields. For each AtomicStream, Archived Streaming Service sets NBInfo field to provide the replay topic information to the client. After that the updated *WsRtspStreamType* message is returned to the client.

A client can add streams to the replay session with addReplayStream operation. *WsRtspStreamType* field is updated as in replaySetup operation and returned back to the client. If the client wants to remove a stream from the replay session, it simply calls removeStream operation and passes the streams it wants to remove.

8.2 Metadata Management

In order to manage collaboration and streaming sessions, we need to store metadata related to these sessions. Sessions may have both static metadata and dynamic metadata. Static metadata is the metadata that does not change during the lifetime of a session. Dynamic metadata is the changing (added, updated, deleted) part of the metadata of the session.

Static Metadata: Collaboration users need to know how many active sessions are available and their associated detailed information. The session information may contain meeting time, duration of the meeting or begin and end time of the meeting. From the meeting time, users can decide whether the session is available or not. In the streaming sessions, a session may record or replay sessions. Metadata regarding a record session may be considered as static metadata once the record session is over. Because during the record, streams can be added to the session, hence metadata of the session is updated. Once a stream is added to a record session, metadata of that stream becomes a static part of the session metadata.

Dynamic Metadata: Collaborative and streaming sessions also have changing metadata during their lifetime. Usually the changing part of the metadata is the part related to the stream definitions, for instance streams available in the session. In collaborative sessions metadata related to the users is dynamic, since users join and leave the sessions. In streaming sessions, record sessions may have dynamic metadata until the session is over, as we explained above. Replay sessions can be considered as dynamic sessions hence they also have dynamic metadata. Since the metadata regarding the replay session is deleted once the replay session is over.

For our system's metadata we have used WS-Context Service as a metadata repository. We also defined XML Schemas to define the session and stream metadata. In the next two sections we will explain some of the important XML Schemas we have defined and how we have used WS-Context Service to store these metadata. Details of the XML Schemas can be found in the Appendix B.

8.2.1 GlobalMMCS Metadata Management

We have developed a metadata management service to maintain GlobalMMCS sessions metadata from our streaming service perspective. We are only interested in the active sessions and streams available in these sessions. GlobalMMCS metadata management service subscribes to XGSP Session Servers and Media Servers topics to retrieve sessions and streams metadata. These metadata are stored in the WS-Context Service. Figure 8-5 shows this process.

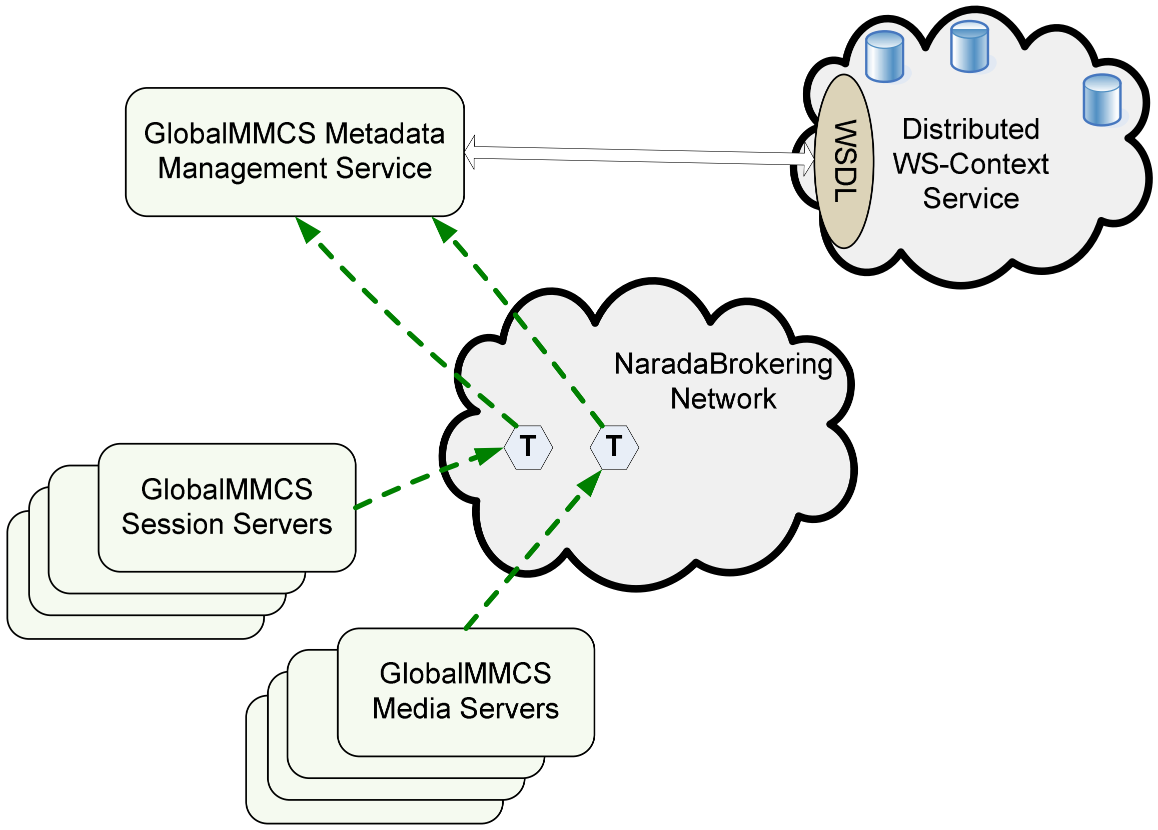

GlobalMMCS Metadata Management Service
Distributed WS-Context Service
WSDL
NaradaBrokering Network
T
T
GlobalMMCS Session Servers
GlobalMMCS Media Servers

Figure 8-5: Retrieving GlobalMMCS metadata and storing them to WS-Context Service

There are two levels of metadata management: sessions level and intra-session level. Sessions level management is to keep track of active sessions, that is, which sessions are available so that which intra-session level management should be started. Intra-session level management is to keep track of streams in the activated session.

WS-Context Service requires us to generate a session and store the context into that session. Context can be XML or non-XML text. We need to provide SessionUserKey while generating the session. Once we generate the session with this SessionUserKey, WS-Context Service provides us a system key, SessionSystemKey, corresponding to the session. System key provided by WS-Context Service is a unique UUID and its uniqueness is guaranteed by WS-Context Service. We use this system key to access the session. In order to store the context, we also need to provide a user key for each context we want to add to the session. We have used URI format while constructing the user keys for the sessions and contexts to be stored into those sessions.

8.2.1.1 GlobalMMCS Sessions Management

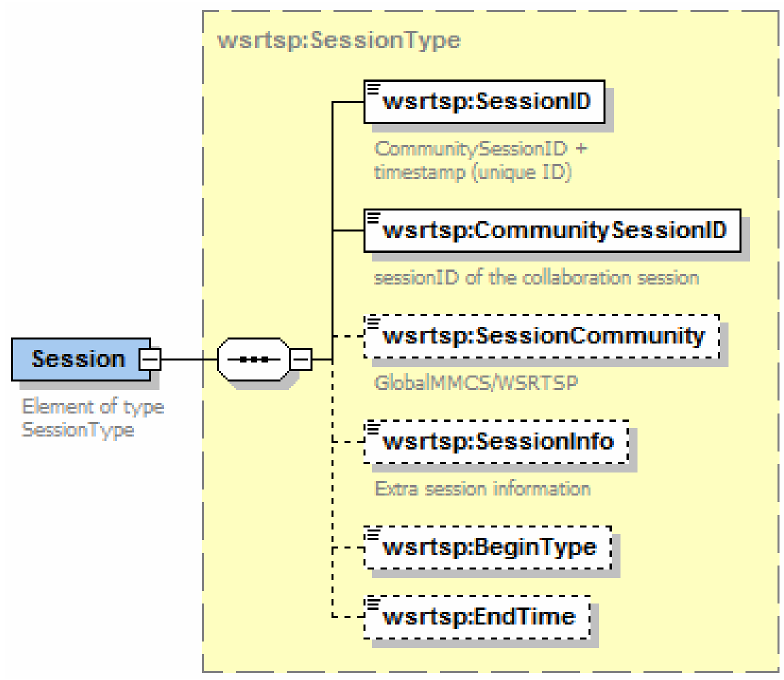

wsrtsp:SessionType
Session
Element of type SessionType
wsrtsp:SessionID
CommunitySessionID + timestamp (unique ID)
wsrtsp:CommunitySessionID
sessionID of the collaboration session
wsrtsp:SessionCommunity
GlobalMMCS/WSRTSP
wsrtsp:SessionInfo
Extra session information
wsrtsp:BeginType
wsrtsp:EndTime

Figure 8-6: Session representation

The Session representation is as in Figure 8-6. As can be seen from Figure 8-6, the important fields of this representation are the SessionID and CommunitySessionID. CommunitySessionID is unique within the community. Since a session within the community can be activated and deactivated any time, we append a timestamp to this ID to construct the new SessionID. Sessions level representation is just a list of these sessions as shown in Figure 8-7.

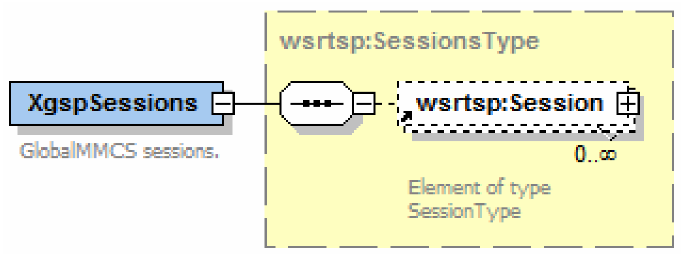


Figure 8-7: XgspSessions representation

There should be only one instance of this session in WS-context Service. So any component in the system can access this session using the SessionUserKey. The mapping between XML nodes and WS-Context User Keys are shown in Table 8-1. User key for first node is chosen as Session User Key. Others are context user keys used within the session.

XML Node Name	XML Path	User key
XgspSessions	XgspSessions	wsrtsp://xgspsessions
Session	XgspSessions\ Session	wsrtsp://xgspsessions/Session

Table 8-1: XML Node Name and WS-Context User Key mapping for XgspSessions representation

Each time a new session is activated, a new Session node should be added to the XgspSessions node as a child node. Also we add a context using the context user key (wsrtsp://xgspsessions/Session) to the WS-Context session that corresponds to this XML representation. Doing this, we avoid overwriting the whole XML structure and we only update a small part of it.

8.2.1.2 GlobalMMCS Intra-Session Management

At the intra-session level, the management of streams is the issue. In this section we are interested in audio and video streams, but the framework can be extended to other streams as well. GlobalMMCS session is defined as in Figure 8-8. In these sessions we have audio and video list in addition to the session information. Video and audio lists are dynamically changing because of streams joining and leaving the session. We need to update these lists.

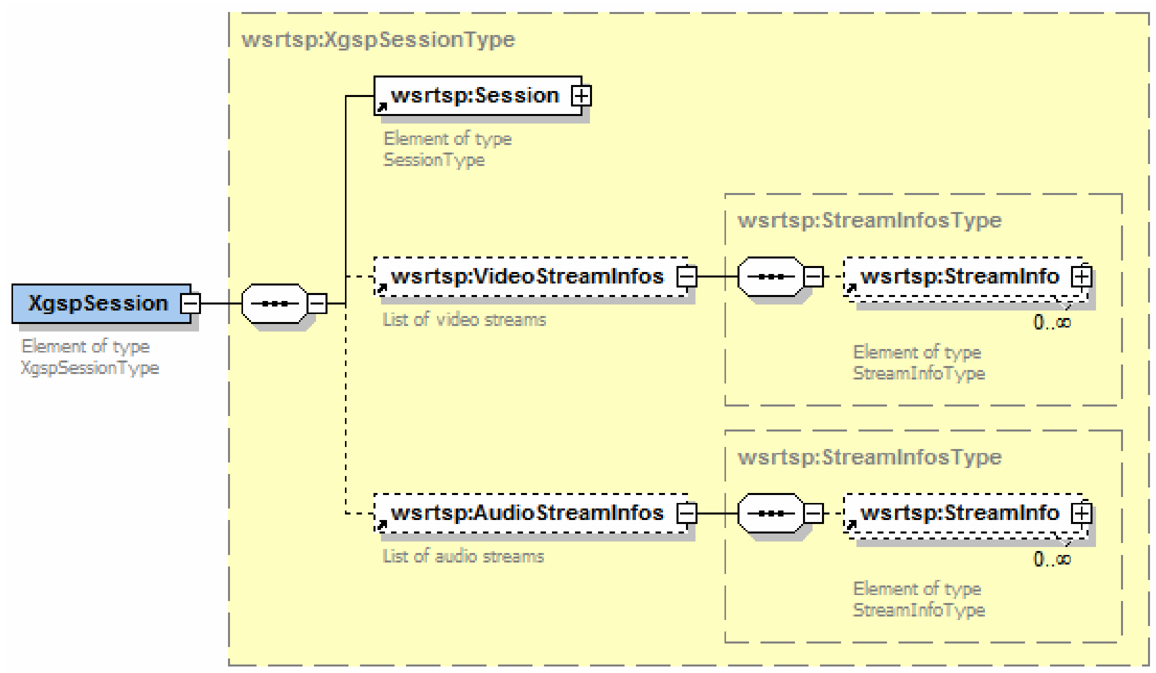


Figure 8-8: A GlobalMCS session (XgspSession) representation

StreamInfo filed describes a stream in the session and it is shown in Figure 8-9. StreamInfo can be used for any type of stream. In GlobalMMCS sessions, audio and video streams' event type is RTPEvent which is defined within NaradaBrokering system to transport audio and video data. Streams can also be in JMS event format or NB event (native NaradaBrokering event type) format.

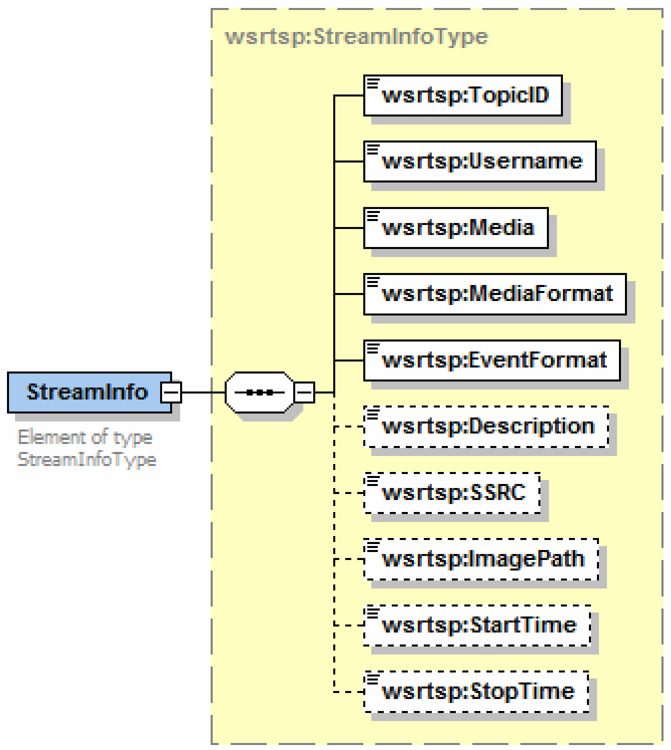


Figure 8-9: StreamInfo representation to describe any type of stream within messaging middleware

When a session is activated, after updating the XgspSessions session in WS-Context Service, we also need to generate a new session in WS-Context Service for the activated session. We follow a similar way as in XgspSessions case to store XgspSession fields to WS-Context Service.

XML Node Name	XML Path	User key
XgspSession	XgspSession	wsrtsp://xgspsessions/sessionID
Session	XgspSession\ Session	wsrtsp://xgspsessions/sessionID /Session
StreamInfos (for audio streams)	XgspSession\ AudioStreamInfos \StreamInfo	wsrtsp://xgspsessions/sessionID /AudioStreamInfo
StreamInfos (for video streams)	XgspSession\ VideoStreamInfos \StreamInfo	wsrtsp://xgspsessions/sessionID /VideoStreamInfo

Table 8-2: XML Node Name and WS-Context User Key mapping for XgspSession representation

When generating a session in WS-context Service for the activated GlobalMMCS session, the sessionID of the session is used in the user key. Session user key of this session is the user key of the parent node which is defined for XgspSession (wsrtsp://xgspsessions/sessionID). Since sessionID is constructed from a timestamp added to CommunitySessionID of GlobalMMCS session which can only be one session with that ID in GlobalMMCS environment. Appending sessionID to wsrtsp://xgspsessions will ensure that there is only one WS-Context session with that SessionUserKey. The sessionID is used for constructing user keys for audio and video streams as well. Table 8-2 shows the user keys for XML nodes defined in XgspSession representation.

8.2.2 Archive Metadata Management

Archive metadata management is similar to GlobalMMCS metadata management. We have sessions and intra-session level management. Sessions level management is the same as in GlobalMMCS sessions level management. XML structure is also similar; only the parent node name is different. However, session description is different since stream definition is different in archive sessions. In an archive session we have AtomicStream which representation is shown in Figure 8-10.

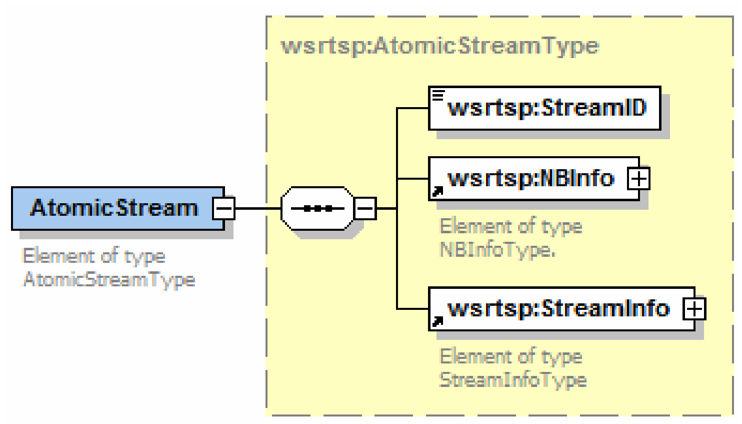

wsrtsp:AtomicStreamType
wsrtsp:StreamID
wsrtsp:NBInfo
Element of type NBInfoType.
AtomicStream
Element of type AtomicStreamType
wsrtsp:StreamInfo
Element of type StreamInfoType

Figure 8-10: AtomicStream representation for archive sessions.

AtomicStream contains not only the StreamInfo, which is the original stream definition based on the content of the stream but it also contains template and transport information of the stream, which are required to store the stream to the repository. This information is included in NBInfo field shown in Figure 8-10. The details of NBInfo are shown in Figure 8-11.

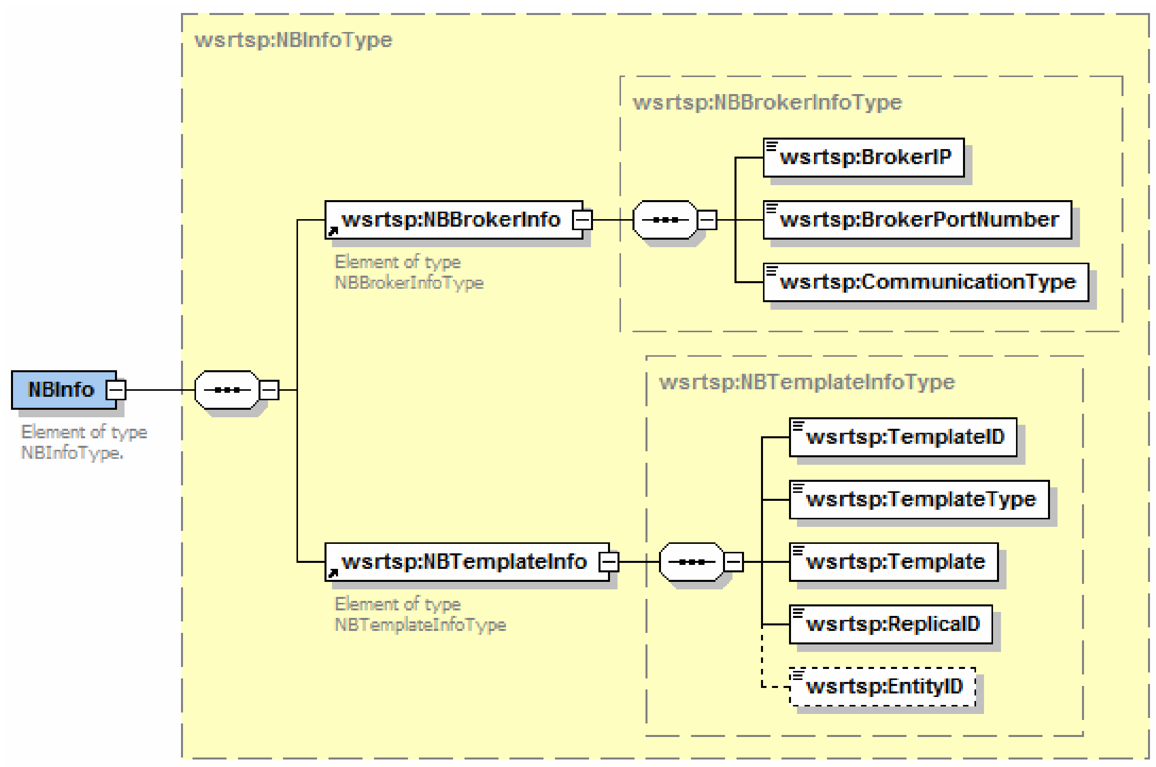


Figure 8-11: NBInfo representation, which is used to describe transport and template information regarding a stream.

8.2.2.1 Archive Sessions Metadata Management

Sessions level management is same as in GlobalMMCS sessions level management. The only difference is that the parent node name is ArchiveSessions (Figure 8-12) instead of XgspSessions (Figure 8-7). This also reflects to user key generation as can be seen from Table 8-3.

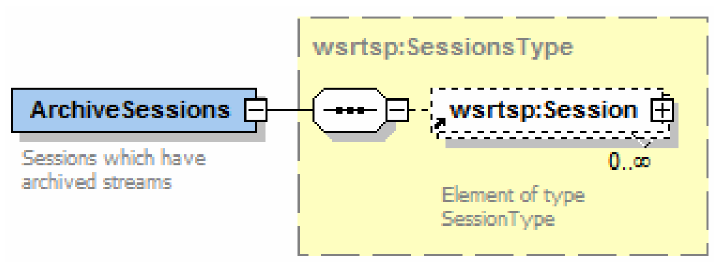

wsrtsp:SessionsType
ArchiveSessions
Sessions which have archived streams
wsrtsp:Session
0..∞
Element of type SessionType

Figure 8-12: ArchiveSessions representation.

XML Node Name	XML Path	User key
ArchiveSessions	ArchiveSessions	wsrtsp://archivesessions
Session	ArchiveSessions\Session	wsrtsp://archivesessions/Session

Table 8-3: XML Node Name and WS-Context User Key mapping for ArchiveSessions representation

There should only be one instance of this ArchiveSessions session in WS-context Service. SessionUserKey of this session is the user key of the parent node which is wsrtsp://archivesessions.

When a session is activated, a new Session node is added to the ArchiveSessions node and the context of that Session is added to the ArchiveSessions session in WS-context Service using wsrtsp://archivesessions /Session as context key within that session.

8.2.2.2 Archive Intra-Session Metadata Management

Archive sessions have only one type of stream, AtomicStream, which is explained above. So whenever a stream is to be stored, a new node is added parent node which is

ArchiveSession. ArchiveSession is shown in Figure 8-13. In addition to session information it has the list of AtomicStreams.

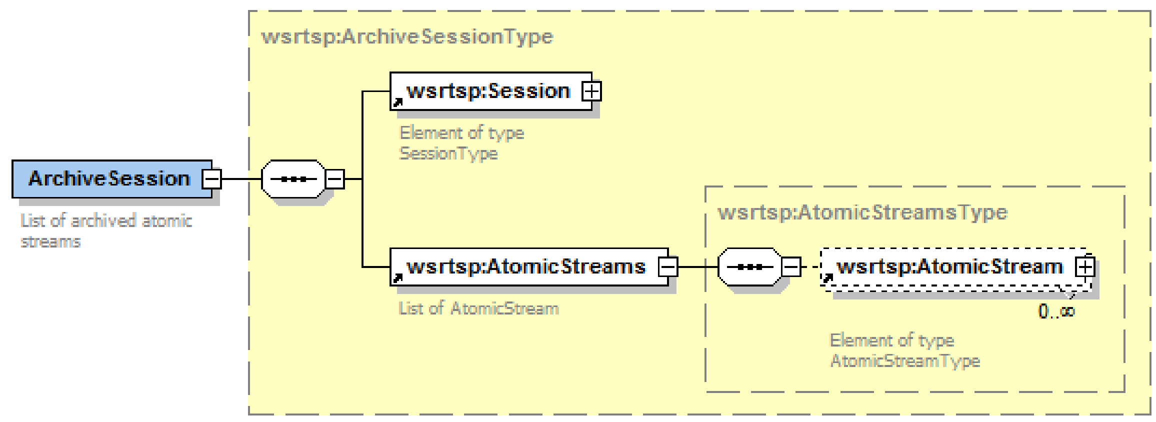

wsrtsp:ArchiveSessionType
wsrtsp:Session
Element of type SessionType
ArchiveSession
List of archived atomic streams
wsrtsp:AtomicStreamsType
wsrtsp:AtomicStreams
List of AtomicStream
wsrtsp:AtomicStream
0..∞
Element of type AtomicStreamType

Figure 8-13: ArchiveSession representation.

When streams are to be archived, an ArchiveSession is defined for that archive session. This also requires generating a new session in WS-Context Service. As in XgspSession case, we also use the SessionID of the ArchiveSession to generate user keys for WS-Context session. How user keys are generated is shown in Table 8-4.

XML Node Name	XML Path	User key
ArchiveSession	ArchiveSession	wsrtsp://archivesessions/sessionID
Session	ArchiveSession\Session	wsrtsp://archivesessions/sessionID /Session
AtomicStreams	ArchiveSession\AtomicStreams \AtomicStream	wsrtsp://archivesessions/sessionID /AtomicStream

Table 8-4: XML Node Name and WS-Context User Key mapping for ArchiveSession representation

8.2.3 Replay Metadata Management

As with GlobalMMCS and archive metadata management, replay metadata management also has two levels of management; sessions level and intra-session level. XML structure is the same except that the parent node name is ReplaySessions. As opposed to archive session, a replay session contains ReplayStream, as shown in Figure

8-14. ReplayStream contains not only the Atomicstream to be replayed but also a new NBTemplateInfo which contains the template information to which the stream will be published.

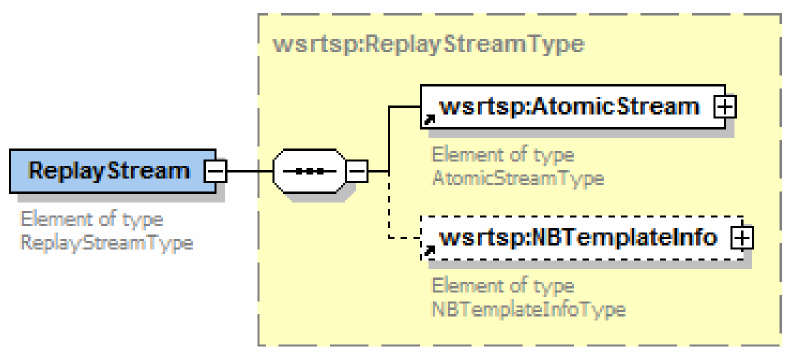

wsrtsp:ReplayStreamType
wsrtsp:AtomicStream
Element of type AtomicStreamType
ReplayStream
Element of type ReplayStreamType
wsrtsp:NBTemplateInfo
Element of type NBTemplateInfoType

Figure 8-14: ReplayStream representation

One of the reasons that a replay stream has different template information is that the original stream might still be in archive process to enable real-time live replay of the stream. If both streams, original and replay, have same template information this would result in archiving the replay stream which is not desired. Although the content of the replay stream is the same stream as the content of the original stream, they are different streams because the original source of the former stream is not the repository but the client publishing it.

Another reason is that, the same stream might be replayed in many replay sessions. In order to have separate controls in each session, replay stream in each session should be published to different topics and hence streams should have different templates. This does not avoid having multiple users in one session. In that case, users may have their own control sharing policies among each other as long as they use the same streaming session.

8.2.3.1 Replay Sessions Metadata Management

XML structure defined for replay sessions is the same as other session types, archives and GlobalMMCS. But parent node's name is ReplaySessions (shown in Figure 8-15), which should be different from others. But the session description is the same. Parent node's name is used to generate user keys for WS-Context sessions. User keys for sessions level metadata management is shown in Table 8-5.

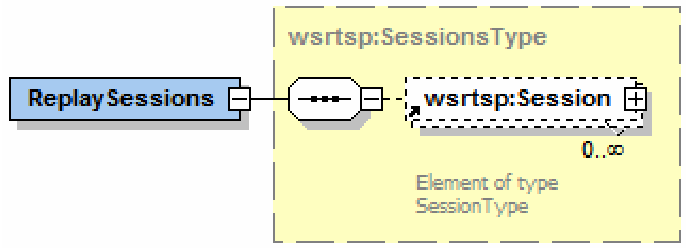


Figure 8-15: ReplaySessions representation.

XML Node Name	XML Path	User key
ReplaySessions	ReplaySessions	wsrtsp://replaysessions
Session	ReplaySessions\Session	wsrtsp://replaysessions/Session

Table 8-5: XML Node Name and WS-Context User Key mapping for ReplaySessions representation

There should be only one instance of WS-Context session that corresponds to the ReplaySessions session. SessionUserKey of this session is the user key of the parent node which is wsrtsp://replaysessions.

Each time a replay session is requested by the user, the context of the Session is added to ReplaySessions session in WS-Context Service using wsrtsp://replaysessions /Session as context key within that session.

8.2.3.2 Replay Intra-Session Metadata Management

Replay sessions have a list of ReplayStreams as shown in Figure 8-16. When a user adds/removes a stream to/from replay session, the session metadata is updated.

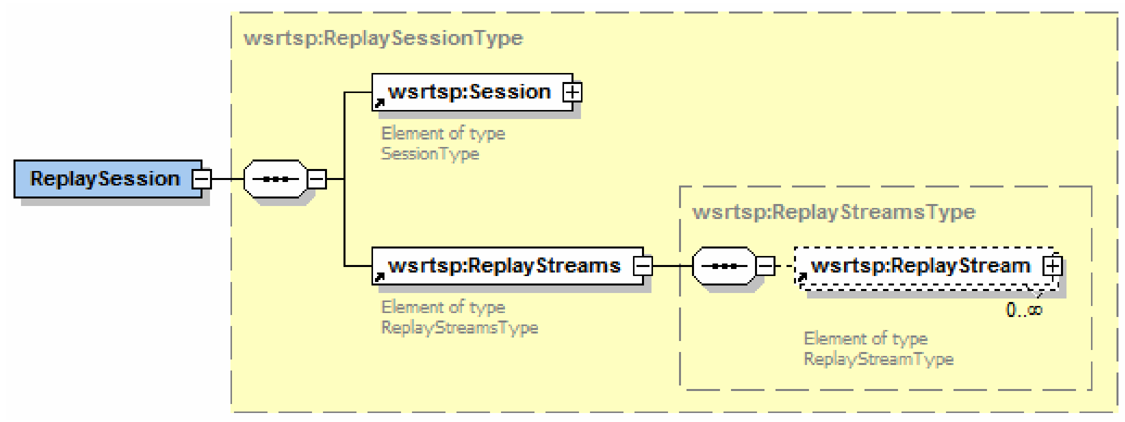


Figure 8-16: ReplaySession representation.

XML Node Name	XML Path	User key
ReplaySession	ReplaySession	wsrtsp://replaysessions/sessionID
Session	ReplaySession\Session	wsrtsp:// replaysessions/sessionID /Session
ReplayStreams	ReplaySession\ReplayStreams \ReplayStream	wsrtsp://replaysessions/sessionID /ReplayStream

Table 8-6: XML Node Name and WS-Context User Key mapping for ReplaySession representation

When a user wants to replay streams, a replay session is generated with the corresponding WS-Context session for this replay session. This replay session also has a sessionID like any other session. The sessionID is sessionID of the streaming session which is unique. Streams to be replayed are added to the replay session and corresponding WS-Context session. Table 8-6 shows the user keys, session user key and context user keys, for WS-context session.

8.3 GlobalMMCS Archive Manager

GlobalMMCS sessions may contain many video and audio streams. Recording of these streams is managed by GlobalMMCS Archive Manager which is specialized to start only the archiving of GlobalMMCS sessions. Although there are many audio sender clients in a GlobalMMCS session, usually there is only one speaker, which is the mix of all the audio streams in that session. So GlobalMMCS Archive Manager considers this and only initiates archiving of this speaker for session recording. Since every new audio client's stream is mixed into the speaker, initiating archiving of speaker only once should suffice. Figure 8-17 shows GlobalMMCS Archive Manager with other components required in the system to enable archiving of GlobalMMCS sessions.

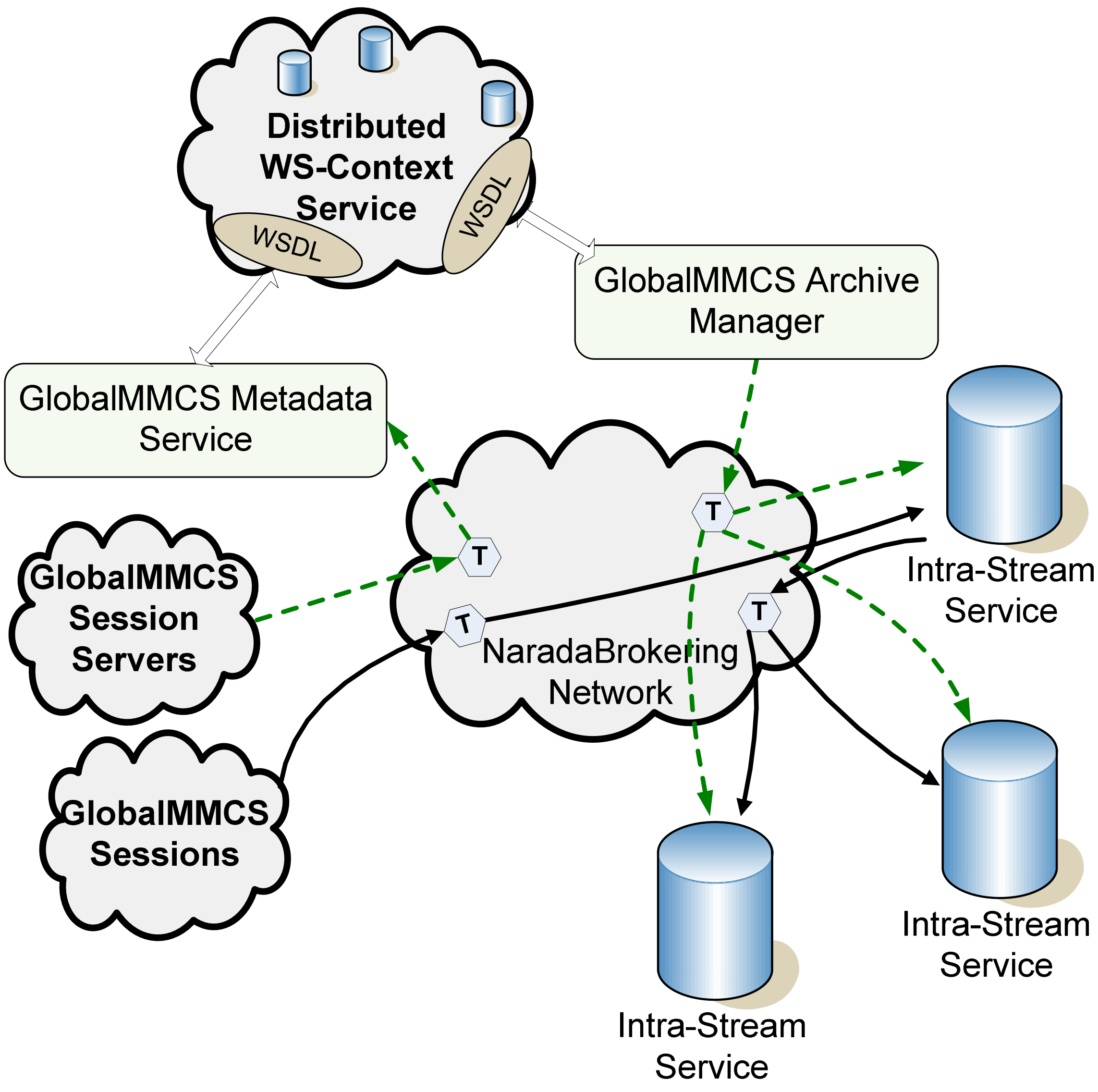


Figure 8-17: GlobalMMCS Archive Manager and interactions with other components in the system.

GlobalMMCS Archive Manager maintains some of the functionality of Archiving Streaming Service in order to archive sessions. These functionalities relate to archiving rather than replaying. That is, GlobalMMCS Archiving Manager does not provide any replay functionality for clients. Clients who want to replay GlobalMMCS streams should connect to the Archiving and Replay Services explained above. We can list GlobalMMCS archive manager’s functionalities as follows:

i. Decides which repository or repositories should be used to archive a session.

ii. Constructs AtomicStream metadata by adding NBTemplateInfo, which requires templateID and repository information, to the StreamInfo.

iii. Monitors GlobalMMCS streams and decides which streams should be archived or not. It can limit the number of streams to be archived in a session.

iv. When new streams join, it may add them to the session and when streams leave it removes them from the session.

v. Stops the archive session once the session is deactivated, that is ended.

vi. Accesses WS-Context Service and generates WS-Context sessions that correspond to archived GlobalMMCS sessions.

vii. Makes requests to Generic Streaming Service to generate archive sessions and add/remove streams from those sessions. This will be explained in the next chapter in detail.

8.4 Summary

In this chapter, we presented one of the components of streaming service – Archived Streaming Service which concerns streaming session initiation and metadata support for describing archiving and replay sessions including streams in these sessions. We have explained Web Service Definition Language (WSDL) interface of this service and messages used. We described XML descriptions of sessions and streams and how we store them in WS-Context Service which provides a fault tolerant and distributed metadata repository that can be shared among components and clients in the system. We also described GlobalMMCS Archive Manager.

Chapter 9

Generic Archiving and Replay

In the previous chapter (chapter 8), we explained streaming session initiation and metadata support for archiving and replay sessions where we utilized fault tolerant WS-Context Service to manage and access metadata. In addition to archive and replay session metadata, we also explained how we manage GlobalMMCS sessions and streams metadata which provides necessary information for real-time GlobalMMCS sessions.

In this chapter, we focus on initiating media delivery for streaming sessions, delivery of the stream. Our architecture is independent of media type and format. So it provides a generic delivery mechanism which can be used not just for multimedia content but also for any type of streaming content. It utilizes messaging middleware to achieve this generic framework. It supports both archiving and replay of streams with replay of real-time live streams.

9.1 Generic Archiving and Replay Framework

As we noted above, in traditional streaming servers, a server needs to be aware of the media format and encoding of the stream in order to start streaming to the client. Our architecture provides a generic framework for recording and replay of any type of streaming event or data. Because when we archive the stream, we do not archive it based on the format or encoding of the stream.

In order to start an archiving or a replay session, an initiator, a client or a component in the system needs to get session and stream metadata from distributed WS-Context service as described in chapter 8. The component we describe in this chapter provides streaming data capability to the overall architecture.

9.1.1 Uniform Event Type for Generic Framework

We receive the incoming streaming event or data as is and wrap it inside the NaradaBrokering native event (NBEvent) before it is archived. Figure 9-1 shows the fields we use when we wrap the received event or data inside the NBEvent. Events are published to a topic and when the repository receives those events it changes the topic name in order to avoid duplicate archiving. If the topic is not changed, the same event would be published to the same topic and would be received again. The generated NBEvent contains mapped topic information in TemplateInfo field. Timestamp field is also added to the event so that during the replay this timestamp is taken into account.

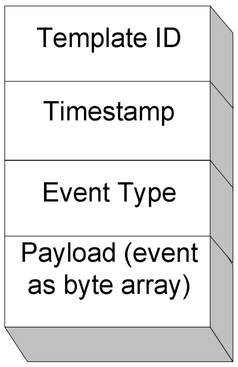


Figure 9-1: Fields used in NaradaBrokering native event

Timestamp information can be obtained from the client if the client uses NTP timestamp. We implemented NTP within NaradaBrokering for this purpose, which is explained in chapter 6. If the received event does not contain any timestamp information, we generate NTP timestamp and use that timestamp for the event.

We use another field to indicate what type of event is actually stored. Event types are JMS event, NaradaBrokering native event and NaradaBrokering RTPEvent which are used in GlobalMMCS sessions to transport audio and video data or RTP packets if the sender is using RTP when sending data.

9.1.2 Repository Changes

We use NaradaBrokering repository service to archive events. We added some tables to maintain some of the stream metadata in order to provide stream position seeking functionality to this generic framework. As part of the changes we maintain a table that keeps timestamp information of events. Repository assigns a sequence number to every event stored to database. Also a stream has a corresponding template ID which separates stream events from other streams' events. So timestamp table for streams can be constructed as shown in Figure 9-2.

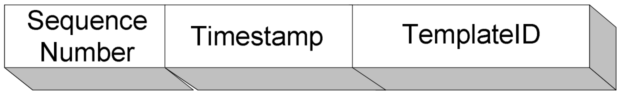


Figure 9-2: Event timestamp table

Running queries on this table can give minimum and maximum timestamp information. This table also provides timestamp information for any event archived, which enables exact positioning within stream.

We also maintain session and stream metadata because of topic mapping required during archiving. Another table is introduced for this purpose which has only one entry per stream. Fields are shown in Figure 9-3.

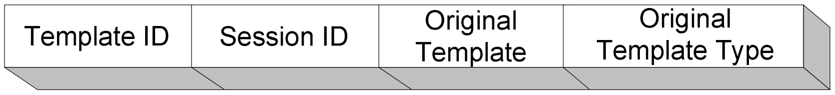


Figure 9-3: Table entries for stream original template information

9.2 Session Archiving

Archiving concept is session based. A session recorder is responsible for storing events of streams in a session. In order to start archiving, we defined a message object to be used within messaging middleware as shown in Figure 9-4. In addition to starting archiving of a session, we use this message to stop archiving, adding or removing streams from the archive session, which is indicated in ControlType. This message also indicates the event type of the stream, in EventType. It can be JMSEvent, RTPEvent or NBEvent.

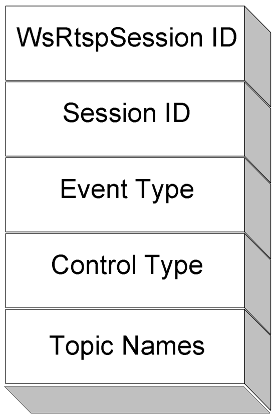


Figure 9-4: Session record request

Session recorder initiates a topic recorder for each stream. Topic recorder subscribes to the topic from which it will receive streaming events. Each topic recorder is assigned a topic name which is String type. While generating the new topic name, a prefix is added to the original topic. For JMS messages, the prefix is */JMS/Record/*, for NaradaBrokering RTPEvent subscribers the prefix is */RTPEvent/Record/*, and for NaradaBrokering native event subscribers it is */NBEvent/Record/*. Since topic names are unique within the session the mapped topics would also be unique. The templateIDs for streams are also received with this message.

Topic recorder assigns a NTP timestamp for generated NBEvent. So when an event is wrapped inside an NBEvent, it contains the timestamp in addition to the new topic name.

RTPEvent contains RTP packets which have media timestamp information. RTP timestamp starts from zero and it is incremented for the media. JMS event can start from a value or it can also start from zero. Based on whether they start from zero or from a preset timestamp, in order to preserve the synchronization among streams, they need to be replaced with NTP timestamps. In this case, offset is calculated from the timestamp of the first event and the other timestamps are updated by that timestamp. If no timestamp information is included in the events, then timestamps included in NBEvent will be obtained from NTP.

The original event is converted into byte array (marshaled) and put to the payload of the generated NB event. This will allow us to get the original when by simply unmarshaling the byte array based on the event type of the stream.

Topic recorder initiates the reliable delivery mechanism for the new topic name and corresponding template ID. This allows topic recorder to be located independent of the repository. But for performance reasons we implemented it within NaradaBrokering repository service, which has direct access to reliable delivery mechanism features. Since it uses the underlying reliable delivery mechanism, topic recorder also generates a CompanionEvent, which is explained in chapter 7. The relationship between topic recorder and NaradaBrokering reliable delivery service is shown in Figure 9-5.

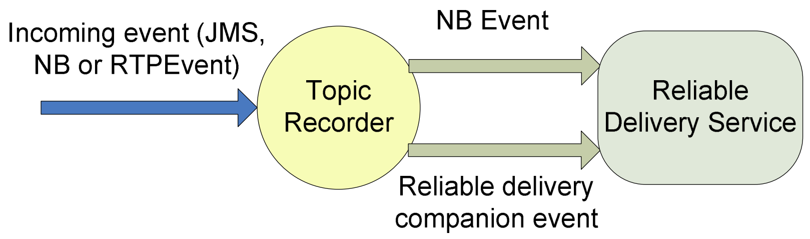

Incoming event (JMS, NB or RTPEvent)
Topic Recorder
NB Event
Reliable delivery companion event
Reliable Delivery Service

Figure 9-5: Topic recorder and NB event generation

Types of topic recorders change based on the transport type rather than media type. For example, for JMS subscription there needs to be a special topic recorder that can subscribe to JMS topics and receive JMS events. Since NaradaBrokering subscriptions is different from JMS subscriptions a topic recorder which is capable of subscribing to NaradaBrokering topics is needed to subscribe to NaradaBrokering topics in order to receive NB events. Likewise, RTPEvent subscription requires another topic recorder which is capable of subscribing to RTPEvent topics in order to receive GlobalMMCS audio and video data.

Although we have defined topic recorders for the event types specified above, this framework can be extended to other event types. Data types included in these event types are no concern, since the framework components does not operate based on the type of the data transported within these events.

Based on the event type, session recorder initiates appropriate topic recorder to handle the topic. This is demonstrated in Figure 9-6. The decision is based on the information received in session record request shown in Figure 9-4.

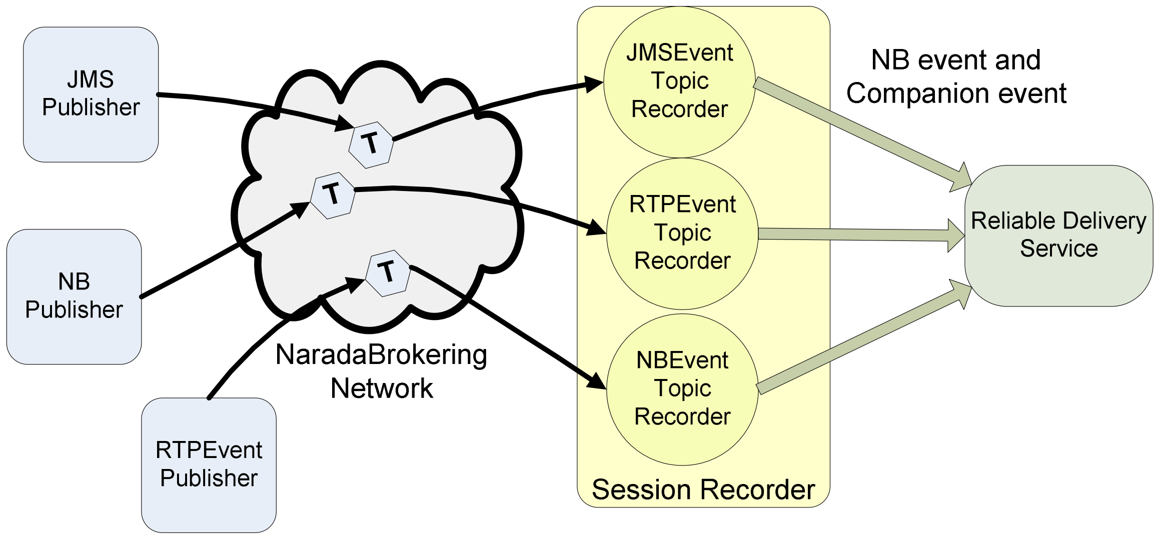


Figure 9-6: Session recorder

9.3 Session Replay

Similar to session recorder, replay is also session based. Session player is responsible for streams in that session. In order to start replay, we define a message object to be used within messaging middleware as shown in Figure 9-7. Session replay request object can be used to add streams to and remove streams from the session, and forward, rewind or pause the session. ControlType field defines the type of the request. The message also indicates the event type of the stream, in EventType field. It can be JMSEvent, RTPEvent or NBEvent.

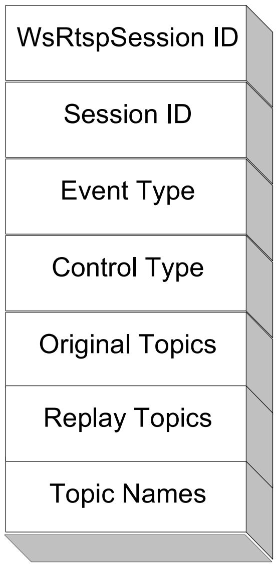


Figure 9-7: Session replay request

Session player contains timestamp provider to provide clients with timestamp information regarding streams in the replay session. Based on the client's request (Figure 9-8a), the session player can send timestamp range (Figure 9-8b) for each of the stream in the session. This range is updated if the stream is a real-time replay stream (will be explained in section 9.4). During the archiving of a stream, stream's end timestamp changes so the end timestamp information needs to be sent to the client in order to allow it to forward the stream. The timestamp range is fixed till the archiving of the stream has finished. The other timestamp request type is the list of timestamps (Figure 9-8c) of all the events in the stream. This is not practical for audio and video events, but it is useful if the stream contains only a small number of events. Using a timestamp list, a client can request an individual event. When sending timestamp information, either JMS events or NB events are used. RTPEvent is only capable of transporting audio and video data. So based on the event type of the stream, timestamp information is exchanged with JMS events or NB events. For streams with RTP event type, NB events are used for delivering the timestamp information. If the client is capable of only handling JMS events, then the timestamp information is send through a JMS connection and event. Otherwise, NB events are used. One timestamp provider for one session player is sufficient to handle timestamp information of all the streams in that replay session.

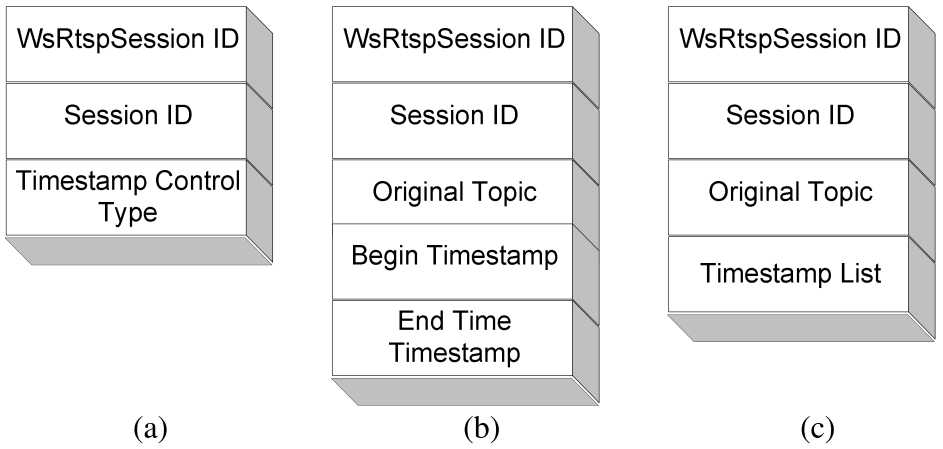


Figure 9-8: (a) Timestamp service request (b) Timestamp range reply (c) Timestamp list reply

Session player initiates a topic player for each stream. Replay topics are specified in the session replay request for each of the stream. Topic player makes a proper connection to the broker suitable with the event type of the stream. Replay topic type for RTPEvents is a type of long topic and for JMS events they are a type of string topics. NB events can be in various types supported in NaradaBrokering, such as String, XML.

In addition to topic player, the session player also contains an event retriever (EventRetriever) and Session Time Differential Service (SessionTDS) components. EventRetriever periodically reads events from repository. It maintains a rate control while reading events. So the session buffer will not overrun from excessive amount of events in the memory. Jitter Reduction Service can be located either at the repository or at the client side depending on the capability of the client and event type transported.

9.3.1 Running Jitter Reduction Services at the Repository

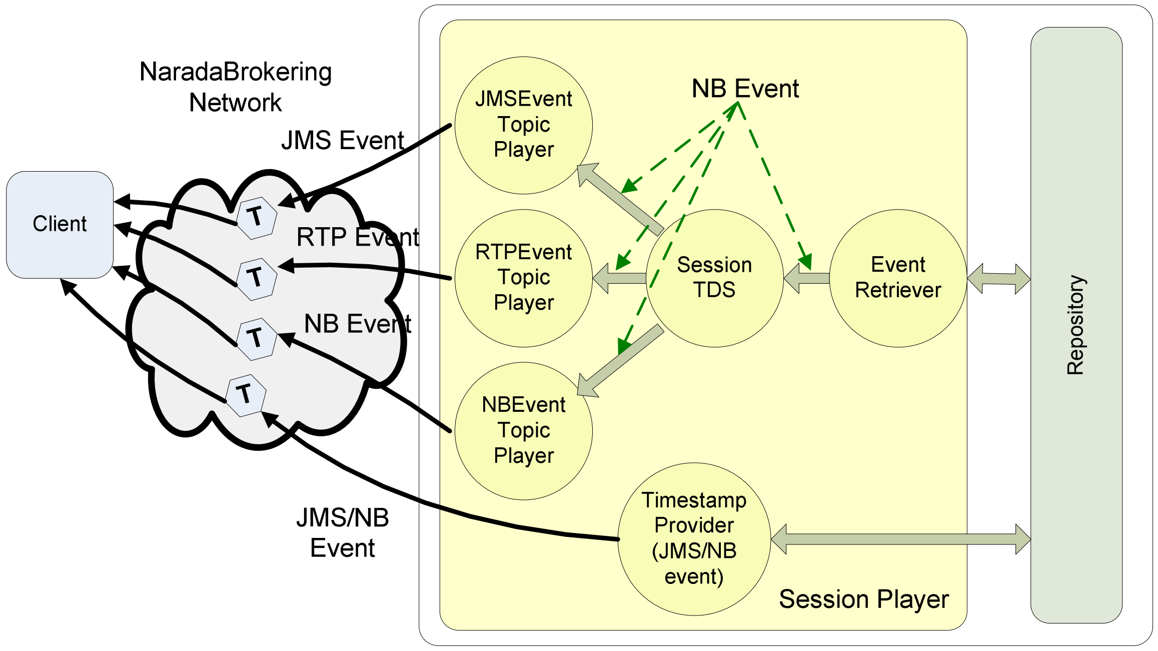


Figure 9-9: Session player with Jitter Reduction Service at the repository

In this case EventRetriever passes events to Jitter Reduction Service (SessionTDS) which orders events based on their timestamps and it also preserve the timespacing between consecutive events while releasing them. We explained Jitter Reduction Services (Buffering and Time Differential Services) in chapter 6. SessionTDS is a specialized component that provides Jitter Reduction Service for the replay session. Each session has its own Jitter Reduction Service (SessionTDS). Using a separate Jitter Reduction Service for each replay session will order and timespace only streams in that session. Components of session player with Jitter Reduction Service at the repository are shown in Figure 9-9. Jitter Reduction Service chooses the topic player that is responsible for publishing the stream. A topic player is responsible for only playing one stream. So there may be many topic players and they may be publishing the same type of event.

Each topic player modifies the topic name field before it publishes the archived event. For instance, when replaying RTPEvents, the topic name which is type of long is replaced with the replay topic name provided for that stream, which is a long value also. This is the only field modified in the replay event. The rest is unmodified so that the original stream is preserved. The original event may contain some other properties which are used by subscribers.

This scenario is suitable for clients that cannot handle NaradaBrokering native events (NB events). For example, a client that is capable of only receiving JMS events cannot handle NB events, so repository should run Jitter Reduction Services. Jitter at the output of topic players is very small since it runs Jitter Reduction Service. This scenario is suitable for any type of clients whether they are JMS event and RTP event subscribers or legacy RTP clients. It does not require any modification on the client side to run Jitter Reduction Service.

9.3.2 Running Jitter Reduction Services at the Client

In this case, EventRetriever reads events from repository and passes them to topic players according to the topic name the events contain. Each topic player also modifies the topic name of the event before it publishes. A client subscribes to these topics and receives events from all the topics passed to the Jitter Reduction Service. Jitter Reduction Service orders and time spaces events before it passes them to specialized topic players inside the client. These topic players then handle the event, either they process them or they pass them to further handle the data processing. This is shown in Figure 9-10.

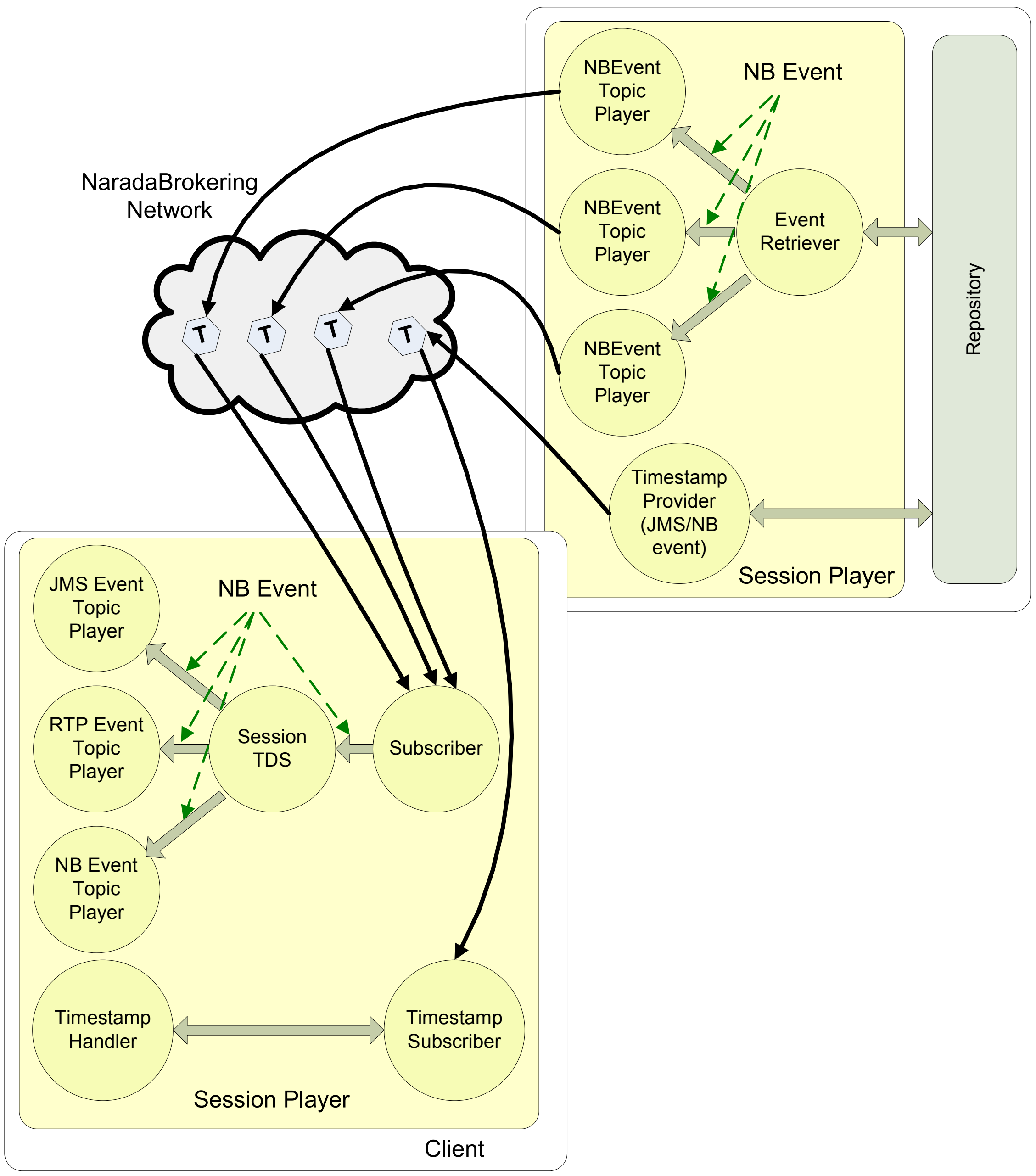


Figure 9-10: Session player with Jitter Reduction Service at the client

Running Jitter Reduction Services improve performance and also the jitter is reduced more than the case running Jitter Reduction Service at the repository. Because events released from Jitter Reduction Service do not go through network which could increase the jitter. Depending on the network bandwidth and proximity of the client to the repository, jitter can be high in the case where Jitter Reduction Service is located at the repository. This also brings performance degradation problems if there are many session players. Because each session has its own buffer and runs separate time differential service. Because of these reasons, client side Jitter Reduction Service would result in better performance and lower jitter.

9.4 Active Replay of Real-time Streams

Streaming control over real-time live streams is usually limited in today's architecture. Although RTSP leaves seeking (rewinding, forwarding, etc.) of live streams as optional functionality, RTSP based streaming servers do not implement it. This is because of the streaming architecture they posses. Archive units in those systems do not make the archive session available until the session is over. Clients can only play the live stream without having control of it, besides starting and stopping it.

Our architecture provides the replay of real-time live streams with seeking capability. Clients are allowed to rewind or forward the part of the stream archived. The archiving process does not need to finalize in order to make the session available.

Metadata of the archive sessions are made available through WS-Context Service so that any client interested in the archived sessions is notified and can access archived session metadata to initiate replays of all archived sessions including replay of archived real-time live streams. Also archived parts of the stream are not modified and it is not locked by session recorders and topic recorders. This allows session players to run full replay functionalities, including seeking in real-time live streams, over the archived data. Hence, archived parts of the stream are available for instant replay.

9.5 Session Synchronization

In the traditional streaming environment, when there are multiple streams in a RTSP session, streaming players use Synchronized Multimedia Integration Language (SMIL) description to play them in a timeline. SMIL is a layout language that allows generation of multimedia presentations consisting of multiple streams in a synchronized timeline. SMIL can also define relative timing among streams. In this case, streams are initiated according to a reference stream. In SMIL, absolute timing can also be used. In that case, streams are initiated when the time for the stream is reached. While presenting a stream, it uses the media time specified in the media data.

In our architecture, there are two important components that allow us to synchronize streams in a session.

- NTP timestamps: Events are either time stamped at the client with NTP timestamp when they are generated or at the repository when they are received. These timestamps are real-time timestamp. Since NTP provides an accuracy of 0-30 ms depending on the proximity to atomic servers, if every stream archived is time stamped with NTP, then streams are synchronized within this range. This range is sufficient for collaboration systems.
- Jitter Reduction Service: This service contains a buffering service and a time differential service, as explained in chapter 5. Buffering service orders events while time differential service releases events according to the time

differences between consecutive events, hence minimizing jitter. All of the streaming events in a session go through the same buffer and are handled by the same time differential service. Since all of them are in the same buffer, they are ordered and released according to timespaces maintained among events from different streams. As a consequence of this operation, events reach players in an ordered and synchronized fashion. Hence, streams in a replay session are synchronized without defining any relative or absolute timing for them. This is depicted in Figure 9-11.

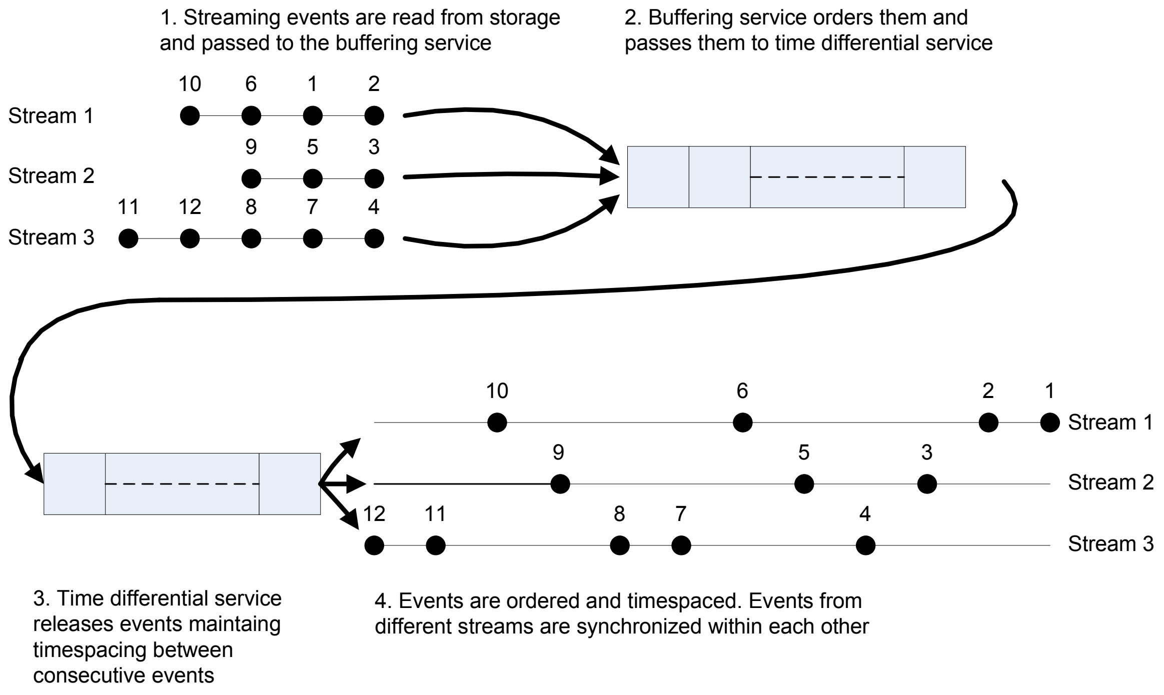

1. Streaming events are read from storage and passed to the buffering service
2. Buffering service orders them and passes them to time differential service
Stream 1
10 6 1 2
Stream 2
9 5 3
Stream 3
11 12 8 7 4
3. Time differential service releases events maintaing timespacing between consecutive events
4. Events are ordered and timespaced. Events from different streams are synchronized within each other
10 6 2 1 Stream 1
9 5 3 Stream 2
12 11 8 7 4 Stream 3

Figure 9-11: Synchronizing streams in a session using Jitter Reduction Service

9.6 GlobalMMCS Videoconferencing Sessions Archiving and Replay

As we mentioned before, GlobalMMCS sessions utilize RTPEvents for transporting audio and video data. RTPEvents contain RTP packet in their payload and contain some header information to be used within messaging middleware for routing the event to clients.

GlobalMMCS components use NBEvent (NaradaBrokering native event) to communicate with each other. Since RTPEvents are specifically designed to transport audio and video data, they cannot be used for message exchange for signaling and other negotiation purposes.

Because of the above reasons for archiving and replay of GlobalMMCS sessions we use NBEvents for signaling and communication, and for data transport we use RTPEvents.

9.6.1 Archiving GlobalMMCS Sessions

Topic recorders for GlobalMMCS sessions handle RTPEvents. They need to be RTPEvent compatible clients in order to receive those events (Figure 9-12). Each stream is handled by a different RTPEvent topic recorder.

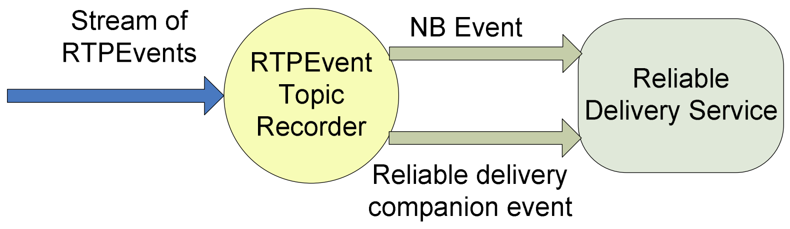


Figure 9-12: RTPEvent topic recorder

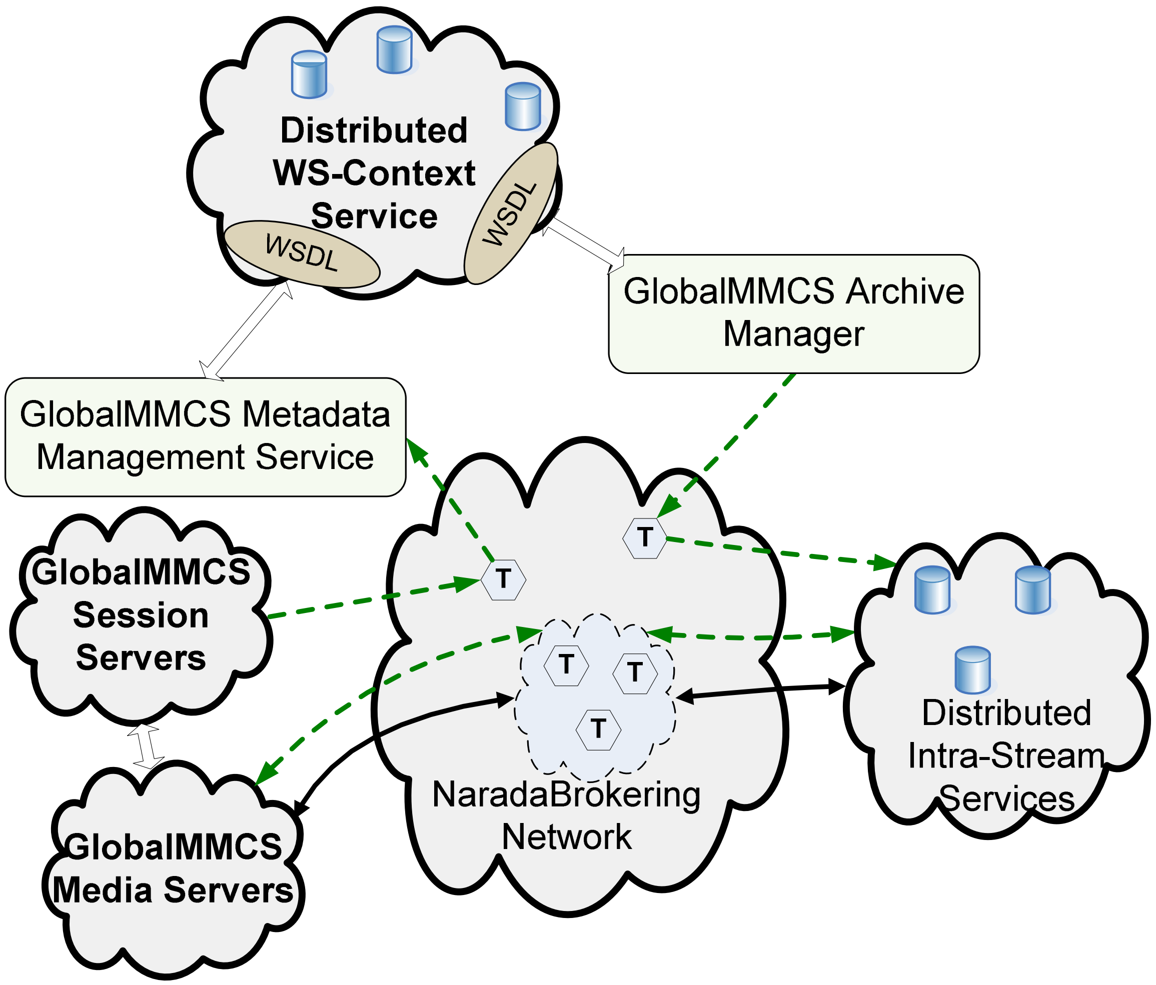

Distributed WS-Context Service
WSDL
WSDL
GlobalMMCS Archive Manager
GlobalMMCS Metadata Management Service
GlobalMMCS Session Servers
T
T
T
T
T
NaradaBrokering Network
Distributed Intra-Stream Services
GlobalMMCS Media Servers

Figure 9-13: Archiving GlobalMMCS sessions

In GlobalMMCS sessions, the number of streams is dynamically changing because clients join and leave the sessions. This requires adding/removing RTPEvent topic recorders to/from archiving sessions. So when streams join GlobalMMCS session, a corresponding topic recorder is initiated in the session recorder and when they leave the session the corresponding topic recorder is removed from the session recorder. Session recorder decides whether it should remove the topic recorder or not from SessionRecordRequest message sent by GlobalMMCS Archive Manager. Archive manager receives session and stream metadata, which is managed by GlobalMMCS metadata management service, from WS-context service. Archive manager locates repositories and communicates with Generic Streaming Service to initiate archiving. It decides which streams should be archived or not. As shown in Figure 9-13, topic recorders can directly communicate with GlobalMMCS media servers to establish streaming between media servers and topic recorders.

9.6.2 GlobalMMCS Stream Replay

Similar to RTPEvent topic recorders, topic players for GlobalMMCS streams are RTPEvent players, clients capable of publishing RTPEvents, as shown in Figure 9-14. Each topic player is responsible for a single stream.

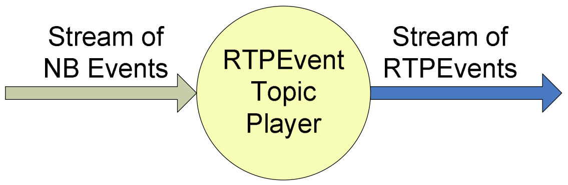


Figure 9-14: RTPEvent topic player

During a GlobalMMCS replay session, if a replay client can only play RTPEvents, then Jitter Reduction Service should be located at the repository as shown in Figure 9-15. If replay is capable of playing multiple event types including RTPEvents, then Jitter Reduction Service (SessionTDS) can be located at the client. Any number of GlobalMMCS streams can be selected to form a replay session. Streams can be removed or added by the client requesting the replay. Different replay sessions may replay the same GlobalMMCS stream. Replay client sends SessionReplayRequest to session player in order to initiate replay of GlobalMMCS streams.

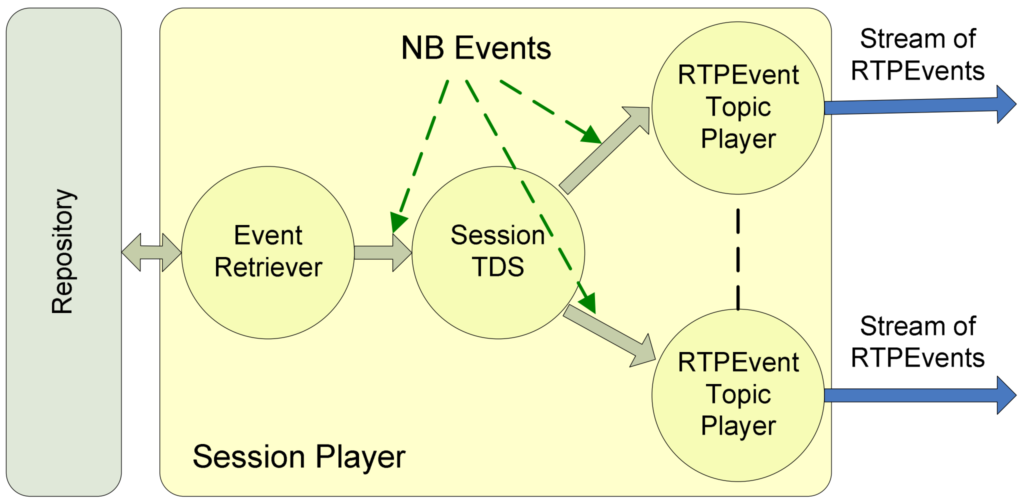


Figure 9-15: Session player for GlobalMMCS replay session

9.7 Anabas Session Archiving and Replay

Anabas eLearning and Collaboration system is an event based environment which utilizes JMS events to transport data among its components. JMS events are used to transport shared display events, text messaging chat, audio chat, whiteboard event, and many others. Our architecture has been used to archive and replay Anabas session.

9.7.1 Anabas Session Archiving

Topic recorders for Anabas sessions handle JMS events. They need to be JMS event compatible clients in order to receive those events (Figure 9-16). Similar to GlobalMMCS topic recorders, each topic is handled by a different JMS event topic recorder.

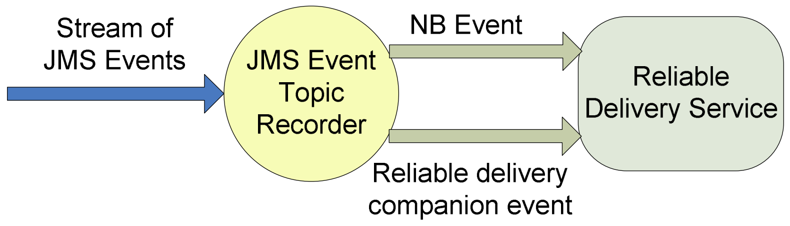


Figure 9-16: JMS event topic recorder

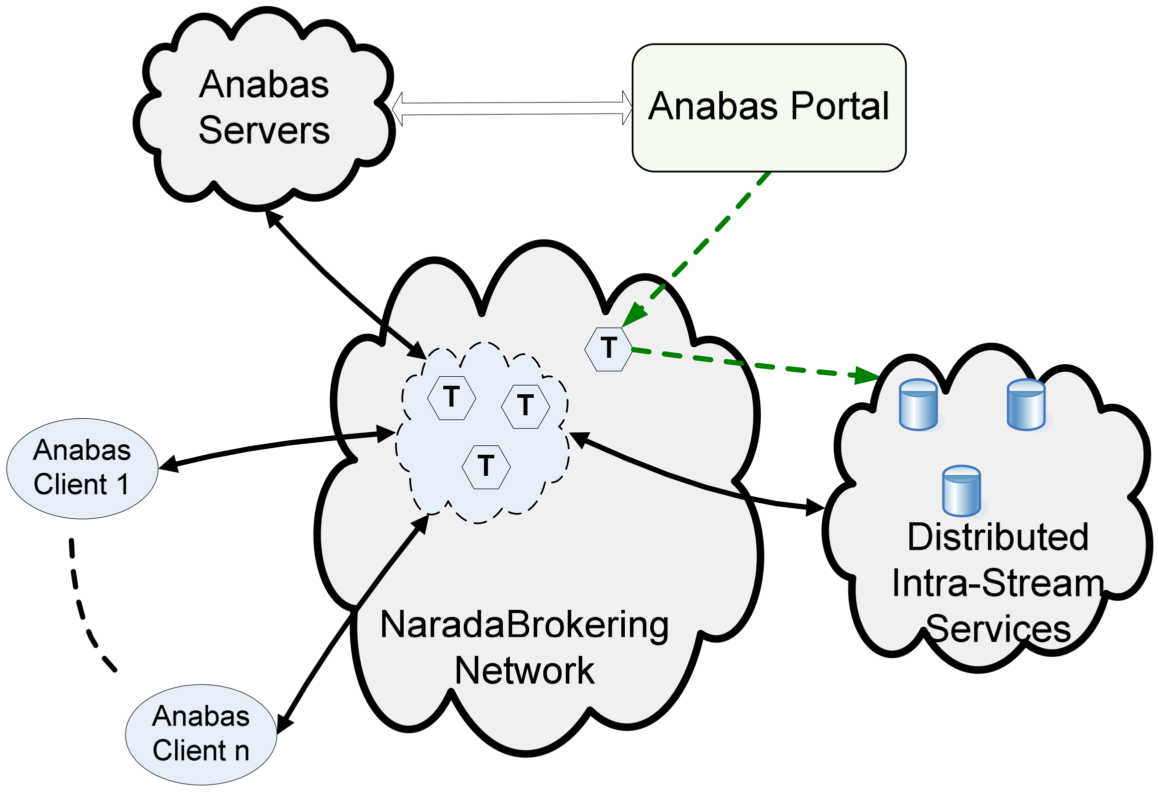


Figure 9-17: Archiving Anabas sessions

In Anabas sessions, there is fixed number of topics. Topics are known when the session is initiated. Although topic addition and removal functionality is provided to JMS recording sessions, Anabas archive manager only needs to send one SessionRecordRequest message to specify all topics to be archived. Session recorder initiates all JMS event topic recorders during session initialization. When the session is over, all of the topic recorders are removed and session recorder is closed. Archived session metadata is maintained by Anabas servers. Figure 9-17 depicts a scenario for archiving an Anabas session.

When a session recorder is initiated for archiving an Anabas session, JMS event topic recorders also subscribe to anabas session topics and start receiving events published to those topics. As with RTPevent topic recorders, JMS event topic recorders also generate NB events and corresponding companion events and pass them to NaradaBrokering reliable delivery mechanism.

9.7.2 Anabas Session Replay

Topic players for Anabas session topics are JMS event publishers as shown in Figure 9-18. Each topic player is responsible for a single stream and hence for a single topic.

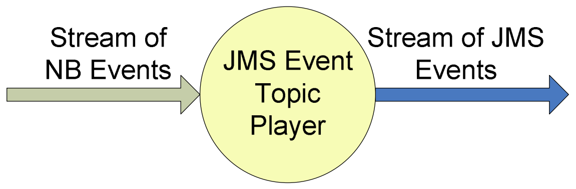


Figure 9-18: JMS event topic player

Since replay can be initiated by any component in the system, SessionReplayRequest message can be sent by Anabas portal. Clients can only connect to anabas servers to receive the replay session topics and subscribe to them, as shown in Figure 9-17. Anabas clients are capable of handling only JMS events, so Jitter Reduction Service needs to be located at the repository during replay session as shown Figure 9-19.

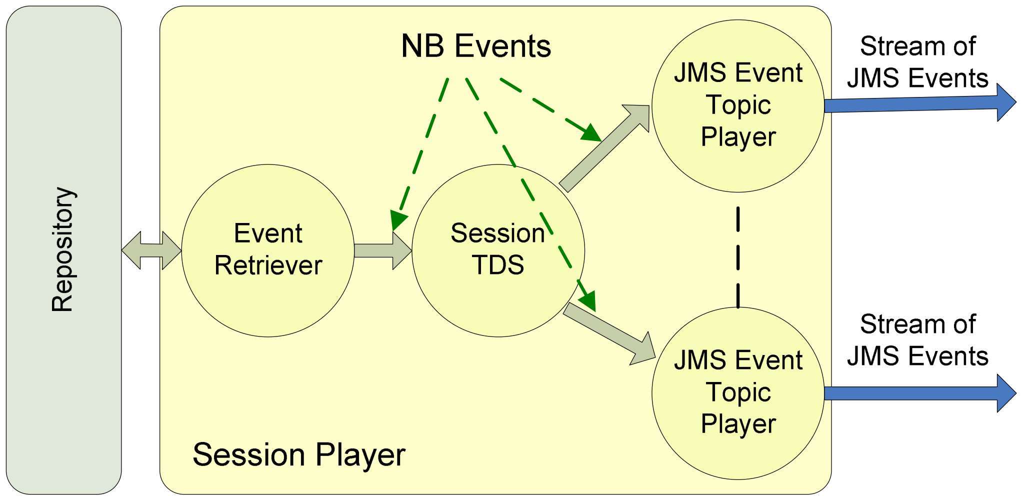


Figure 9-19: Session player for Anabas replay session

9.8 Summary

In this chapter, we presented Generic Streaming Framework which can be used not just for multimedia content but also for any type of streaming content. We have explained session archiving and replay concepts where a session may contain several streams. We then gave two examples where this framework has been used: archiving and replay of GlobalMMCs sessions and archiving and replay of Anabas sessions.

Chapter 10

eSports Collaborative Multimedia Streaming and Annotation System

In chapter 9, we explained how GlobalMMCS and Anabas sessions can be archived and replayed. GlobalMMCS archive manager and anabas portal are both used to start and stop archived sessions when these sessions are not the source of the streams but some other management components in their systems is. Archived sessions contain only one type of stream, RTPEvent based GlobalMMCS streams or JMS event based Anabas session topics. In addition to that, the streams being archived are real-time live streams. The session topics do not contain replay (archived) streams together with real-time live streams. Although streams can be instantly replayed while being archived, they are not played together in the same session.

In this chapter, we would like to present eSports Collaborative Multimedia Streaming and Annotation System which is built on top of the streaming architecture we have explained in this thesis. eSports system can provide new features with this streaming architecture. In eSports system, clients can collaboratively and synchronously play and archive real-time live video, take snapshots, and annotate those snapshots using whiteboard application. Annotated stream can be replayed synchronously with the original video stream. eSports system can be used in virtual education environments for distance sports coaching and distance learning.

10.1 eSports System Design

eSports system is a collaborative multimedia annotation system over heterogeneous grid computing environments. eSports system also supports not only the archived video streams but also the real time live ones. Annotated streams can be replayed synchronously together with the annotation streams. Figure 10-1 shows the interaction of eSports portal and clients with NaradaBrokering messaging middleware and our streaming architecture.

eSports system is built on top of the streaming architecture to provide new and innovative features as an annotation system. Hence, it can access the streams in GlobalMMCS sessions. As explained, GlobalMMCS integrates various videoconferencing systems. With this, eSports can annotate a stream from any video conferencing system with the help of Archive Service and Generic Streaming Framework.

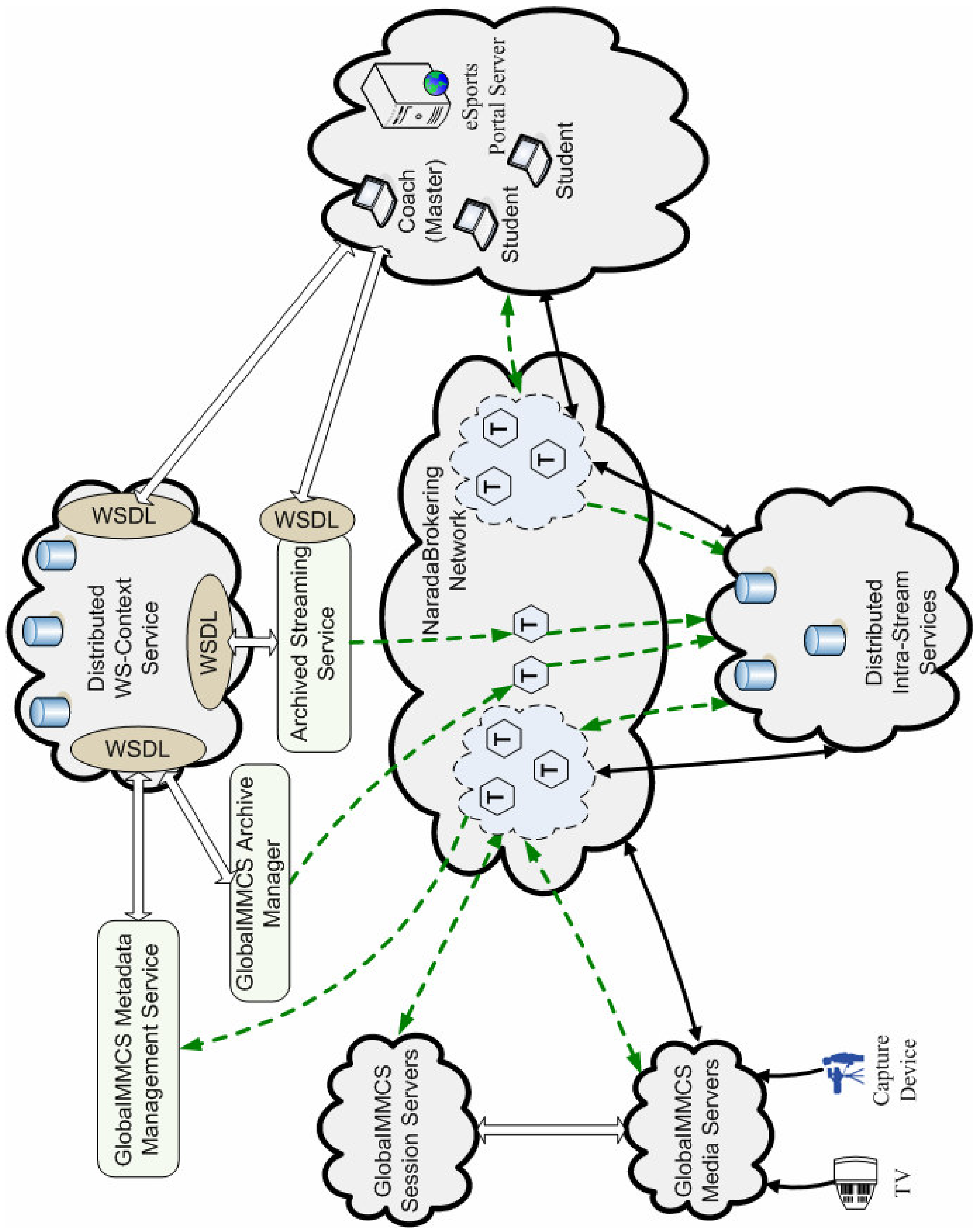

Coach (Master)
Student
Student
eSports Portal Server
Distributed WS-Context Service
WSDL
WSDL
WSDL
WSDL
Archived Streaming Service
NaradaBrokering Network
T
Distributed Intra-Stream Services
GlobalMMCS Archive Manager
GlobalMMCS Metadata Management Service
GlobalMMCS Session Servers
GlobalMMCS Media Servers
Capture Device
TV

Figure 10-1: Overall system including eSports system

10.1.1 Capabilities Provided By Archive Service and Generic Streaming Framework

The capabilities provided by streaming architecture to eSports system are as follows:

- **Archive and replay of NaradaBrokering native events**; NB events can be used for transporting any type of data in their payload. So they can be used to transport multimedia content such as audio, video and images as well as text messages. Since replay services are built around NB events they can be located at the client side (such as Jitter Reduction Service) and any type of data can be archived and replayed. Replay clients for different data types can be built. This enables eSports system to stream any type of data which enriches the streaming application types within the system.

- **Archive and replay of GlobalMCSS sessions**; GlobalMMCS is a conferencing system that integrates heterogeneous videoconferencing environments such as H.323 based system, the Access Grid and SIP based system. It also allows web cam users to join and leave these sessions through XGSP clients. eSports system can retrieve and play these streams in real-time while the session is going on. Since GlobalMMCS sessions use RTPEvents to transport multimedia content, eSports clients needs RTPEvent players. A camera viewing a sports game can be used by GlobalMMCS clients to stream the sports game to videoconferencing sessions. Since we can archive and replay GlobalMMCS streams, eSports can access the real-time live games. Similarly, TV programs can be streamed into

GlobalMMCS sessions. Hence, it enables eSports system to replay those programs through our streaming architecture.

- **Instant replay**; Streaming architecture enables eSports clients to control the real-time live streams. Capabilities include pausing, rewinding and forwarding (until the end of the archived stream) the streams.

- **Utilizing WS-Context Service**; Every session and stream metadata is stored in WS-Context service. It provides a distributed and fault tolerant metadata repository which can be accessed through WSDL interfaces. Metadata of archived and replay sessions stored in WS-Context service is also accessible by eSports system. This will let eSports clients share the same session metadata by a single URI.

- **Transporting messages through NaradaBrokering messaging middleware**; We use NaradaBrokering messaging middleware for data transport. Since NaradaBrokering messaging middleware is capable of software multicasting, one replay session can be received by many clients subscribed to the same replay session topics. The scalability of NaradaBrokering for RTPEvents can be seen in ref. thesis [106].

10.1.2 An Example eSports Session Scenario

The overall system including eSports system is shown in Figure 10-1. TV and capture devices are shown in Figure 10-1 to indicate that the system is capable of receiving those streams in a real-time live streaming scenario. In this scenario, a sports game is streamed with these devices. These streams are archived and made available through Archived Streaming Service. A coach and participants attend an eSports session through eSports portal to discuss the live game. All of the participants of this session share the same applications such as eSprot Player, eSports Whiteboard and Instant Messenger which we will explain in the section 10.2.

Activities involved in this session are:

- Participants obtain eSports session metadata from WS-Context service. Coach obtains archived session metadata as well.
- Coach initiates an archived session with TV or capture device video included in the session. Participants receive replay streams whose control might be shared among participants.
- Replayed real-time live stream is also being archived with new NTP timestamps along with other streams in eSports session. So it can be replayed synchronously with them.
- Coach takes snapshots from the replay stream and annotates it. Annotations are seen by every participant in the session.
- Coach saves annotations. Annotations are streamed using NB events. That is, an annotation stream is a NB event based stream. These annotations are considered

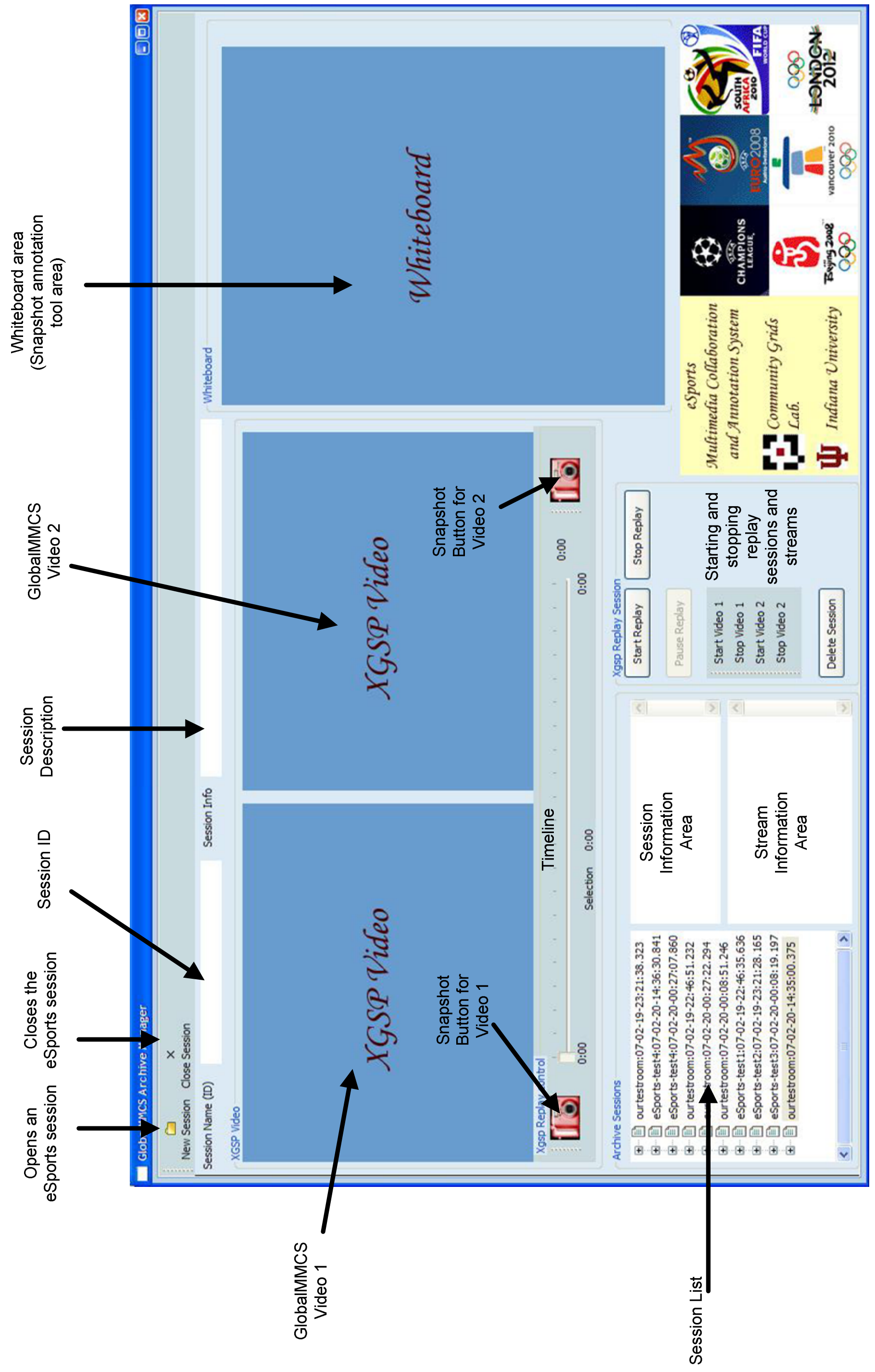
as another stream where snapshots carry the same NTP timestamps of the video frame it is taken from.

- Real-time live stream and annotation stream are also available for immediate replay in a synchronized fashion in another session as well.

10.2 eSports Collaboration System Interface

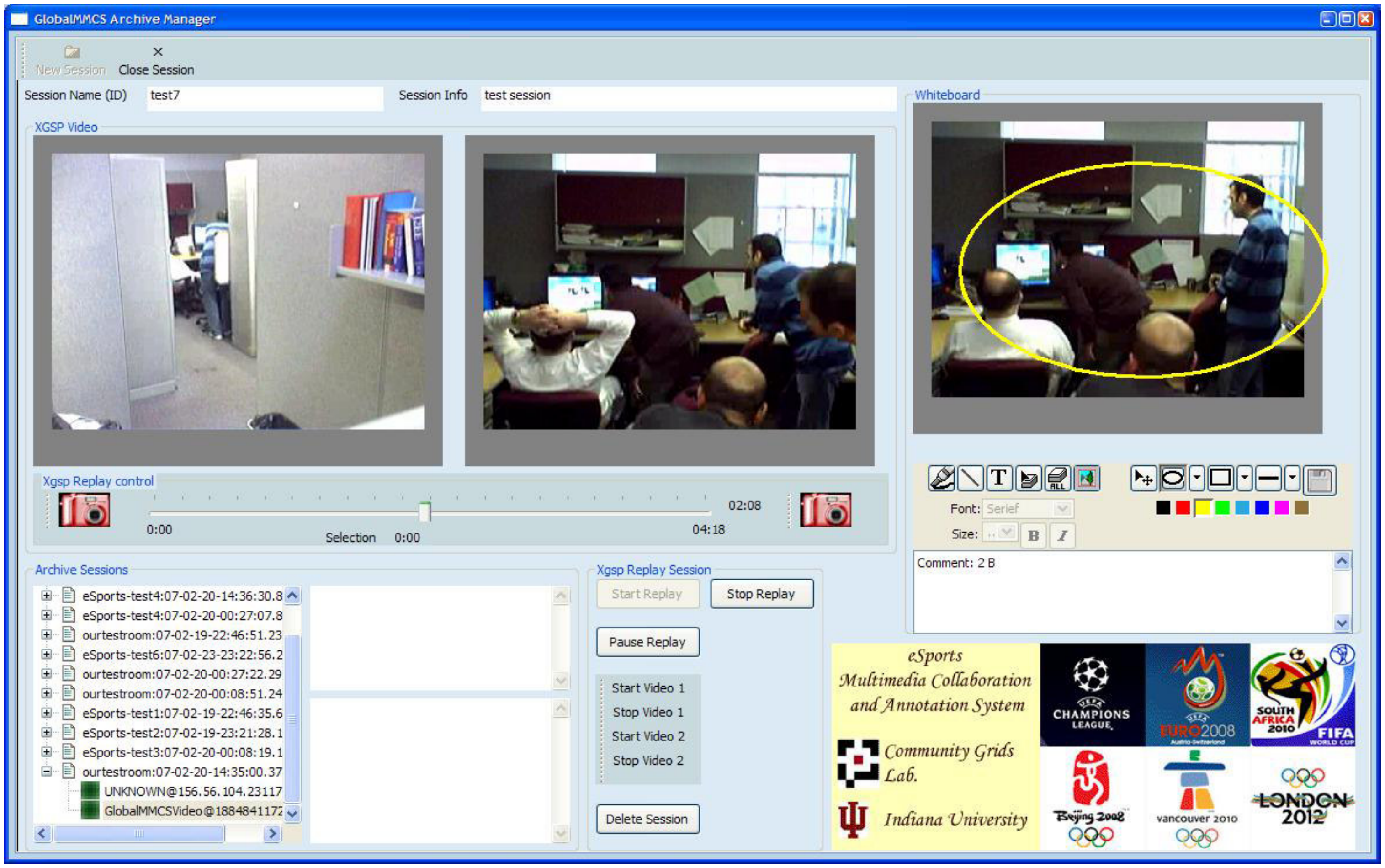
eSports Collaboration System interface is shown in **Figure 10-2**. A snapshot of a live eSports session is also shown in **Figure 10-3**. It includes two RTPEvent based video players and whiteboard client that also enables text comments for whiteboard annotations. In order to provide available sessions and streams to those sessions, it includes the list of the current archived GlobalMMCS sessions list. Selected session and stream information are displayed on fields indicated in **Figure 10-2**. Opening and closing of a typical eSports session can be described as follows;

In order to initialize a new eSports session, a user needs to click "New Session" button after providing sessionID and information. Upon clicking that button a timestamp is appended to the sessionID and session XML message is sent to WS-Context service with 3 streams information included; of which two of them are for video streams and the other one is for whiteboard stream to be recorded.



ourtestroom:07-02-19-23:21:38.323
eSports-test4:07-02-20-14:36:30.841
eSports-test4:07-02-20-00:27:07.860
ourtestroom:07-02-19-22:46:51.232
ourtestroom:07-02-20-00:27:22.294
ourtestroom:07-02-20-00:08:51.246
eSports-test1:07-02-19-22:46:35.636
eSports-test2:07-02-19-23:21:28.165
eSports-test3:07-02-20-00:08:19.197
ourtestroom:07-02-20-14:35:00.375

Figure 10-2: eSports Collaboration System Interface (eSports Recording)



Session Name (ID): test7

Session Info: test session

0:00 Selection 0:00 02:08 04:18

Font: Serif

Comment: 2 B

```
eSports-test4:07-02-20-14:36:30.8
eSports-test4:07-02-20-00:27:07.8
ourtestroom:07-02-19-22:46:51.23
eSports-test6:07-02-23-23:22:56.2
ourtestroom:07-02-20-00:27:22.29
ourtestroom:07-02-20-00:08:51.24
eSports-test1:07-02-19-22:46:35.6
eSports-test2:07-02-19-23:21:28.1
eSports-test3:07-02-20-00:08:19.1
ourtestroom:07-02-20-14:35:00.37
    UNKNOWN@156.56.104.23117
    GlobalMMCSVideo@1884841172
```

Figure 10-3: Snapshot taken from a live eSports session (eSports Recording)

Once eSports session is initialized, a GlobalMMCS replay session can be started. For this, a session on the list should be chosen and the user should click “Start replay” button. This will send a message to Generic Streaming Service to start a replay session for the archived GlobalMMCS session and for the streams in that session. Next step is to select a stream from the replayed session and click either “Start Video 1” or “Start Video 2” to play the selected stream in one of the players. Streams replayed are also being published as NBEvents and recorded.

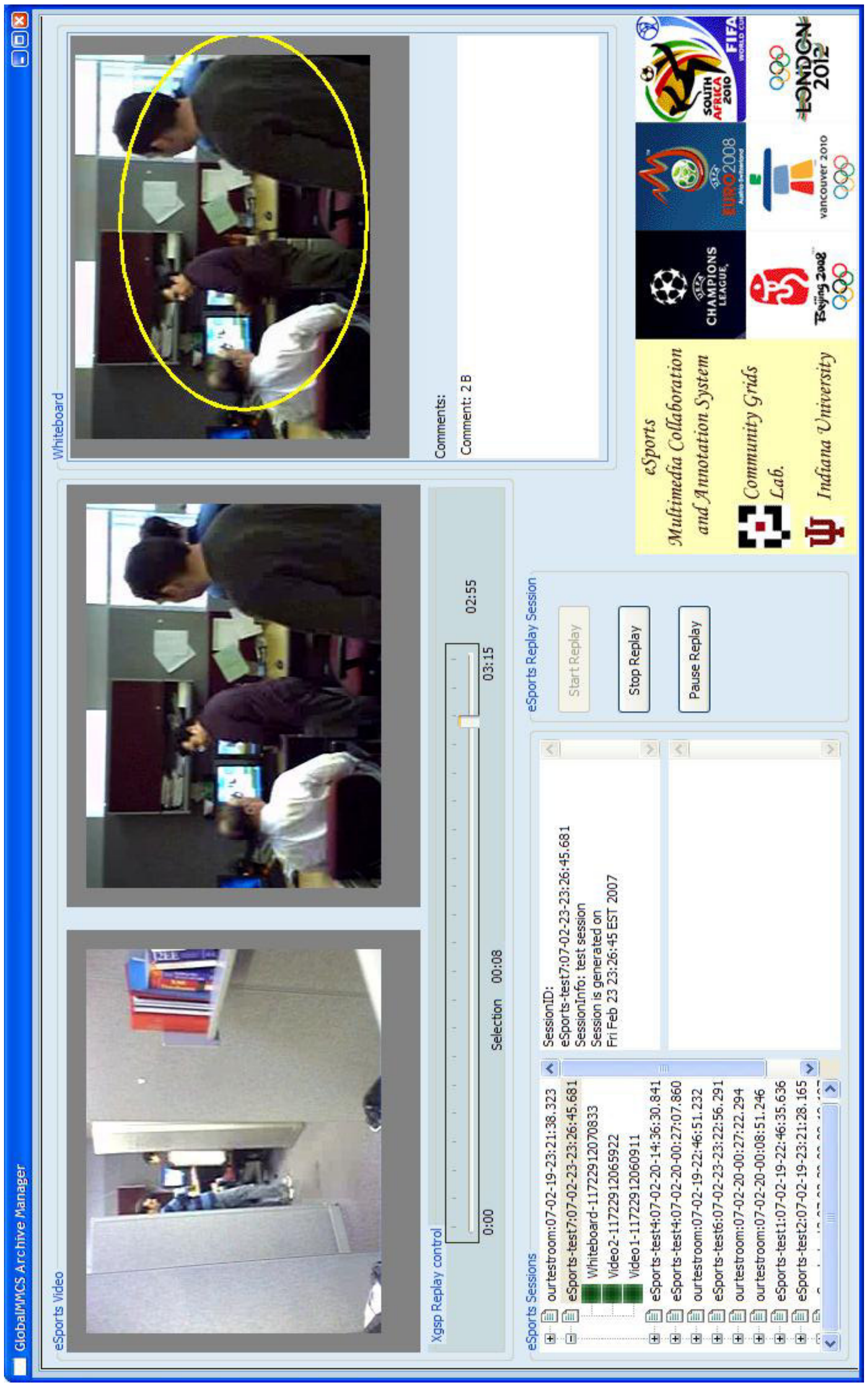
Each player has its own snapshot button. This button enables the user to take snapshots from the corresponding players and send them to the whiteboard player.

The stream play time is shown in the “Timeline”. The end time of this timeline is updated if the replay session is a real-time live session. The user is able to pause the replay session with “Pause Replay” button or he can go forward or backward over the timeline by dragging the timeline cursor to the desired location.

“Close Session” terminates all the subscriptions and closes all the players and it initializes the eSports interface for another session.

An eSports session as shown in Figure 10-3 can be instantly replayed while it is active (not closed). The eSports replay client is shown in Figure 10-4. This client only provides replay capability for an eSports session. The user can select the eSports session from the session list and start the replay of that session. Pause, rewind and forward functionalities are also provided to the user of that session.

The active eSports session can instantly be replayed or it can be replayed after the session is closed.



Comments:
Comment: 2 B

SessionID:
eSports-test7:07-02-23-23:26:45.681
SessionInfo: test session
Session is generated on
Fri Feb 23 23:26:45 EST 2007

ourtestroom:07-02-19-23:21:38.323
eSports-test7:07-02-23-23:26:45.681
Whiteboard-11722912070833
Video2-11722912065922
Video1-11722912060911
eSports-test4:07-02-20-14:36:30.841
eSports-test4:07-02-20-00:27:07.860
ourtestroom:07-02-19-22:46:51.232
eSports-test6:07-02-23-23:22:56.291
ourtestroom:07-02-20-00:27:22.294
ourtestroom:07-02-20-00:08:51.246
eSports-test1:07-02-19-22:46:35.636
eSports-test2:07-02-19-23:21:28.165

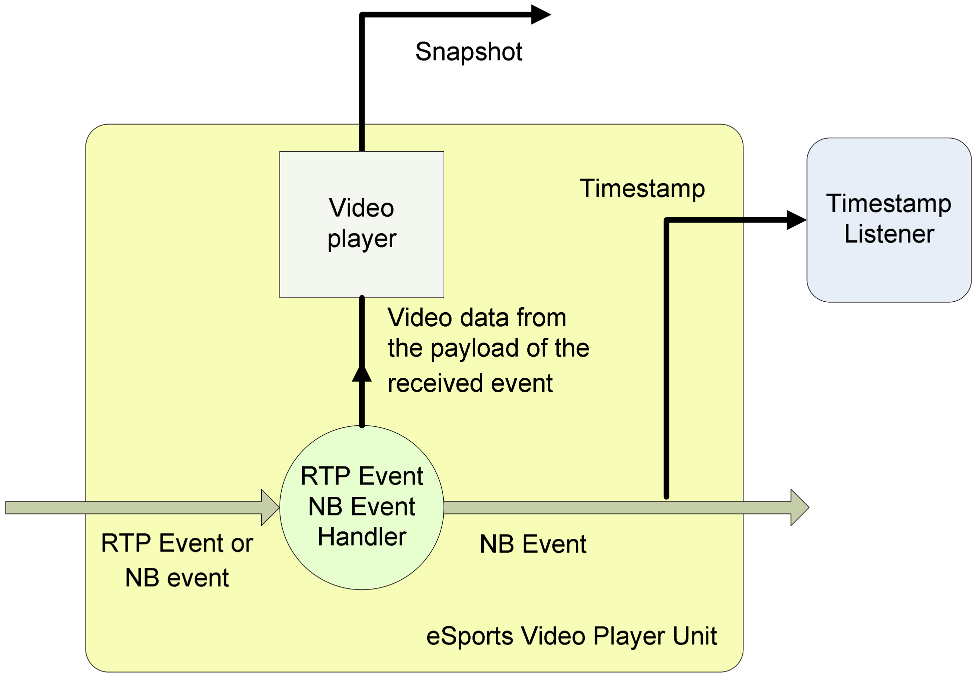
Figure 10-4: Snapshot taken from eSports replay session (eSports Replay)

10.3 eSports Collaboration Tools

eSports collaboration applications are eSports player, whiteboard and instant messenger. In eSports sessions, there are two roles, a coach and the student. According to the role eSports collaboration tools can provide different capability. We will examine these tools from streaming point of view.

10.3.1 eSports Player

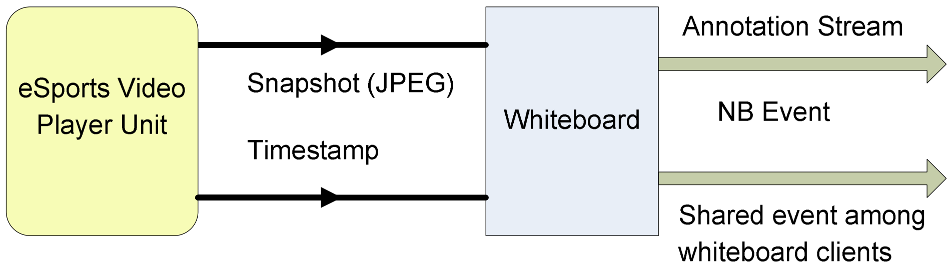
eSports player is composed of one to two eSports video player units where each of them is dedicated to one incoming video stream. eSports video player unit is shown in Figure 10-5. They can play stream type of either RTPEvent or NB event. RTPEvent handler is activated if the received stream is an archived GlobalMMCS video stream. Event handler reads the payload of the event from RTPEvent or NB event and then passes it to the video player. Video player displays the video data.


Snapshot
Video player
Timestamp
Timestamp Listener
Video data from the payload of the received event
RTP Event NB Event Handler
RTP Event or NB event
NB Event
eSports Video Player Unit
Figure 10-5: eSports Video Player Unit

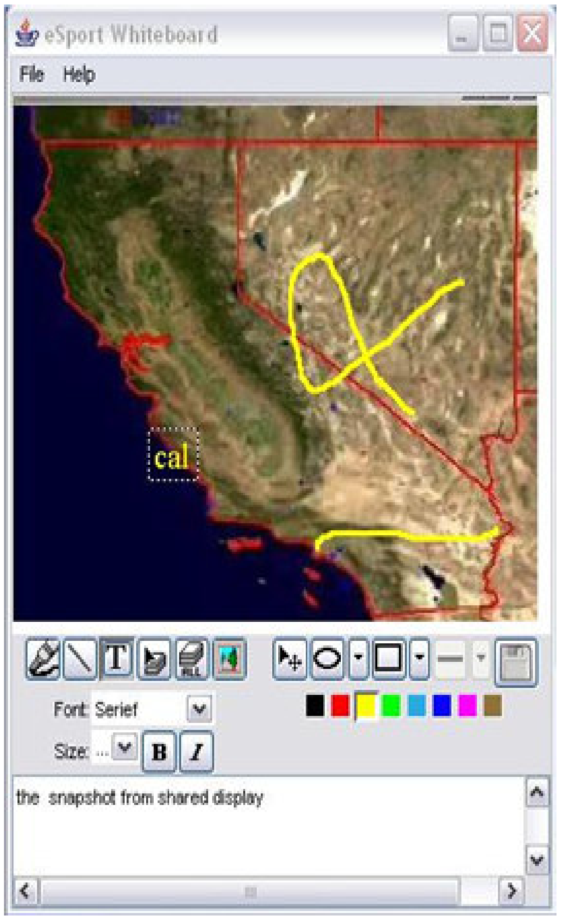
A video player provides interface to take snapshots of the played video. This feature is only allowed for clients with coach role. In addition to passing the video data, it also passes the NTP timestamp information of the data packet passed to the player of a listener, which is a client whiteboard application with coach role. Whiteboard application assigns timestamp information to the snapshot it has taken from the video player. If the eSports session is archived then it is necessary to publish the video played. The stream type is NB event. Even the played video is an archived GlobalMMCS video. Since it is played in a new archive session, it is considered as a different stream. The master of the session can rewind or forward the replayed stream, which would make it different from the original stream. In this case, the timestamps passed to the timestamp listener are generated at the eSports video player unit. If the received stream is RTPevent type stream the NTP timestamp is again generated at the eSports video player unit.

If the eSports session is only a replay session which does not require archiving of the session, then the event handler does not need to publish the stream. Because the input stream of the event handler can be received by any other eSports client connected to the same session.

10.3.2 Whiteboard


eSports Video Player Unit
Snapshot (JPEG)
Timestamp
Whiteboard
Annotation Stream
NB Event
Shared event among whiteboard clients
Figure 10-6: Relationship between eSports video player unit and whiteboard application

Whiteboard takes snapshots and timestamps of the snapshots from eSports video player unit as shown in Figure 10-6. Whiteboard content is shared among clients. So when a snapshot is taken it is shared with other clients. Although any whiteboard client can make annotations on the snapshot taken from video player, only clients with coach role can produce the annotation stream which is archived in eSports session. Annotation erasure is also done by clients with coach role. Other clients can only add comments such as text, drawings, pictures, etc. One user's comments whether text or drawing are propagated to other users and can be viewed immediately, as shown in Figure 10-7. Hence, all clients get a consistent and synchronized view of the shared whiteboard content.



the snapshot from shared display

Figure 10-7: Shared whiteboard content

When a client with coach role saves the annotated snapshot, NTP timestamp received during snapshot process is used to timestamp the annotated snapshot. When annotated snapshot is saved, it is saved as a JPEG image. Saving operation also produces an NB event, where the annotated snapshot is the payload of the event and NTP timestamp recorded during snapshot overwrites the event's timestamp. The annotation process will be explained in detail in section 10.4.

10.3.3Instant Messenger

This is to support text based messaging among eSports clients to provide chat capability in a session. Each client can post its message and receive messages from other clients in the same session. When a message is posted, an NB event is generated with the current NTP timestamp. The result of instant messaging is a text based NB event stream as shown in Figure 10-8. This stream is archived along with other streams in the session.

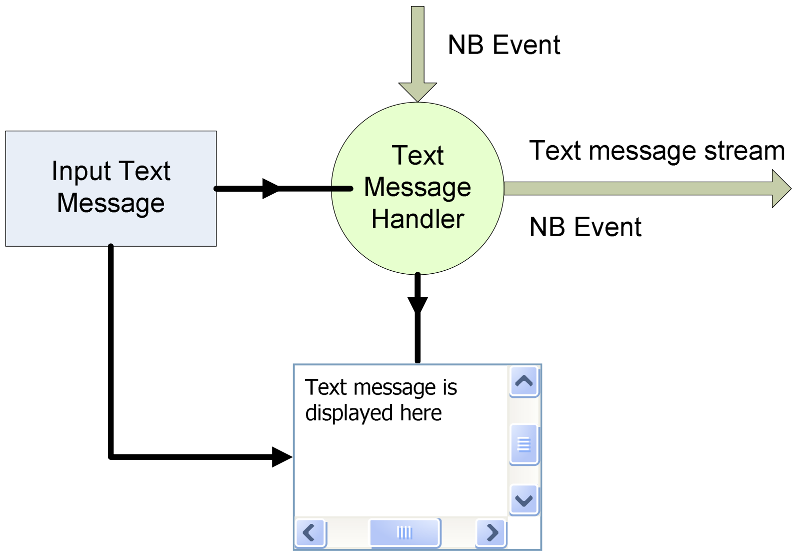


Figure 10-8: Instant message streaming

10.4 Stream Annotation

Figure 10-9 demonstrates the data flow between video stream replayed and the whiteboard application which takes snapshots from the player for annotation. The example shows only for two GlobalMMCS streams. But there may be more streams depending on the design of the client.

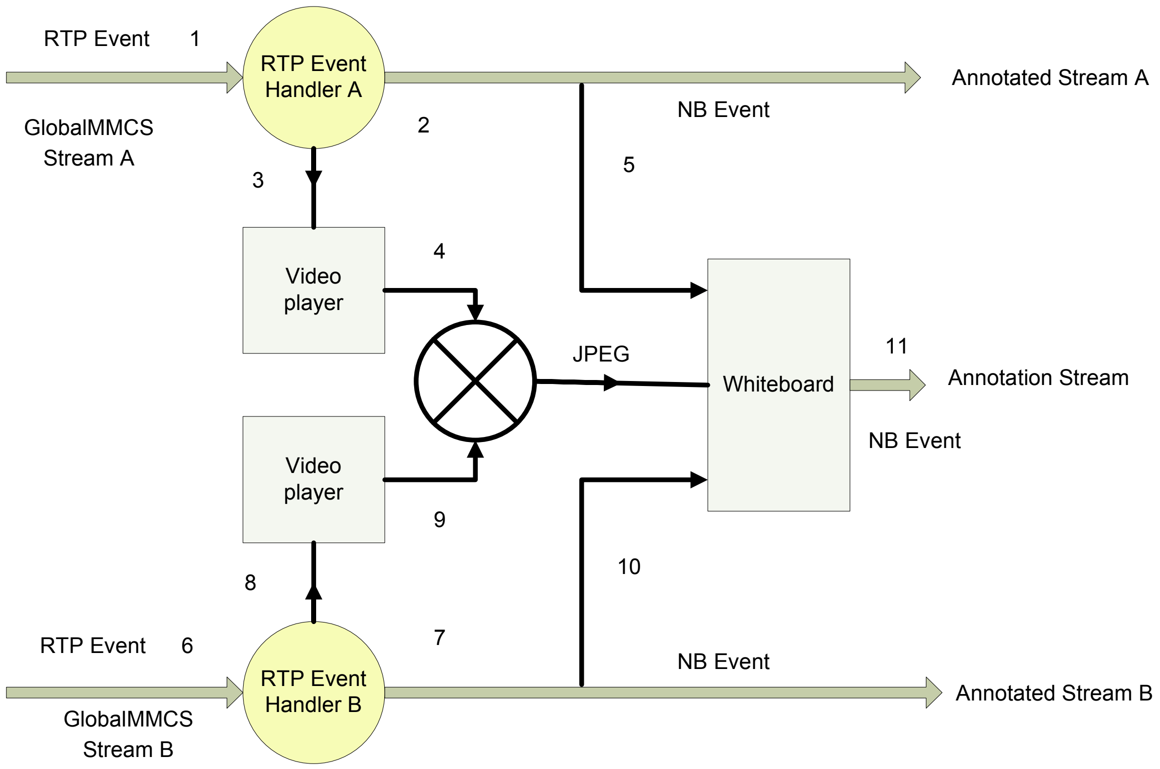


Figure 10-9: Data flow between two video streams and whiteboard application

The data flow is as follows:

1. RTPEvent handler A receives GlobalMMCS video stream A.
2. The same stream is published as a different stream. Because replay stream is now part of the eSports session and in order to replay the stream with other streams in the eSports session, it needs to be published as a new stream containing consistent timestamp with other stream.

3. RTPEvent handler A feeds the video player with the video data received. Video player is capable of playing the stream format received in RTPEvent.
4. The Whiteboard application takes snapshots from the video display. These snapshots are images taken from the video player, which is in JPEG format.
5. Whiteboard also registers itself with the RTPEvent handler A handling the incoming stream. RTPEvent handler A broadcasts the timestamps used in generating a NB event. When whiteboard takes snapshots from a video player, it also records this timestamp. When publishing the whiteboard event, which is an annotated snapshot, it timestamps the generated NB event with the recorded timestamp.
6. RTPEvent handler B receives GlobalMMCS video stream B.
7. RTPEvent handler B also publishes the stream in NBevent type with timestamps recorded at that time.
8. The stream B events are also passed to a video player for replay.
9. Whiteboard can switch to stream B to take snapshots.
10. Whiteboard receives timestamp information of stream B events.
11. Whiteboard can select between streams and annotate them. Annotated snapshots are wrapped inside NB events and these events are published to a topic which is considered an annotation stream.

10.5 Performance Tests

We have measured the delay from source (stream publisher) to destination (stream receiver); $t_4 - t_1$, storage overhead; $t_3 - t_2$ and jitter at the output of the storage.

We conducted our tests in Local Area Network (LAN) and in Wide Area Network (WAN).

10.5.1 Test Setup

The test setup is shown in Figure 10-10. We located the stream publisher to a machine in LAN and some machines in WAN. Machine configurations are shown in Table 10-1.

In LAN setup, we located the machine to gf4.ucs.indiana.edu. In the first WAN setup, we located the stream publisher to synseis.geongrid.org (a remote machine in University of California at San Diego - UCSD) and in the second WAN setup we located the stream publisher to vlab2.scs.fsu.edu (a remote machine in Florida State University – FSU).

In our tests, we have used NTP timestamps to timestamp packets. We have collected data at eSports Test Client and computed parameters as will be presented in this section. RTPEvents do not contain any timestamp information, so we included four long fields to RTPEvent to record t_1, t_2, t_3 and t_4.

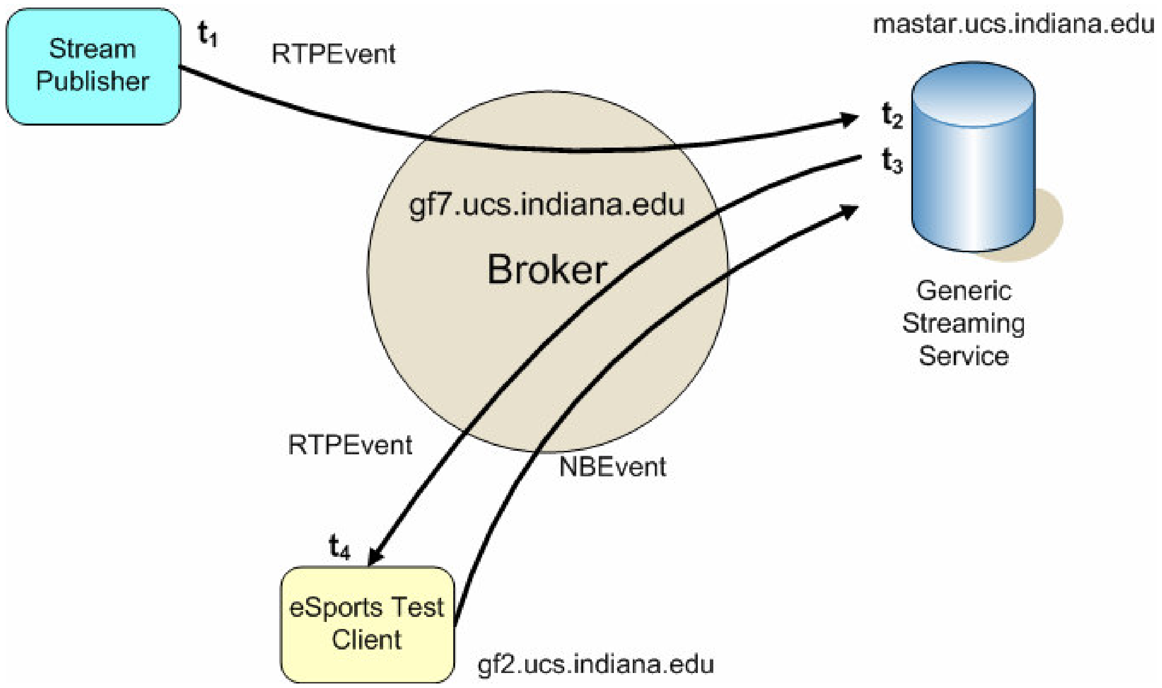


Figure 10-10: Test setup for measuring the delay and jitter

Component	Machine specifications		JVM version
Stream Publisher LAN test	OS	Linux (Fedora Core release 1 (Yarrow))	1.4.2_03
	CPU	4 Intel ® Xeon™ CPU 2.40GHz	
	Memory	2 GB	
Stream Publisher WAN (FSU) test	OS	Windows XP	1.4.2_13
	CPU	4 Dual Core AMD Opteron(tm) Processor 270 2 GHz	
	Memory	2 GB	
Stream Publisher WAN (UCSD) test	OS	Windows XP	1.4.2_04
	CPU	2 Intel® Itanium 2 IA-64 1.40 GHz	
	Memory	4 GB	
Broker	OS	Linux (Fedora Core release 1 (Yarrow))	1.4.2_03
	CPU	4 Intel ® Xeon™ CPU 2.40GHz	
	Memory	2 GB	
Generic Streaming Service	OS	Windows XP	1.6.0-b105
	CPU	Intel® Pentium® 4 CPU 3.40 GHz	
	Memory	2 GB	
eSports Test client	OS	Linux (Fedora Core release 1 (Yarrow))	1.4.2_03
	CPU	4 Intel ® Xeon™ CPU 2.40GHz	
	Memory	2 GB	

Table 10-1: Machine configurations for LAN and WAN test setup

10.5.1.1 Local Area Network Test

Stream duration for LAN test is 181435 milliseconds (~ 3 min. 1 sec.). Jitter at the input of the Generic Streaming Service is shown in Figure 10-11 while the output jitter is shown in Figure 10-12. Measured values are shown in Table 10-2.

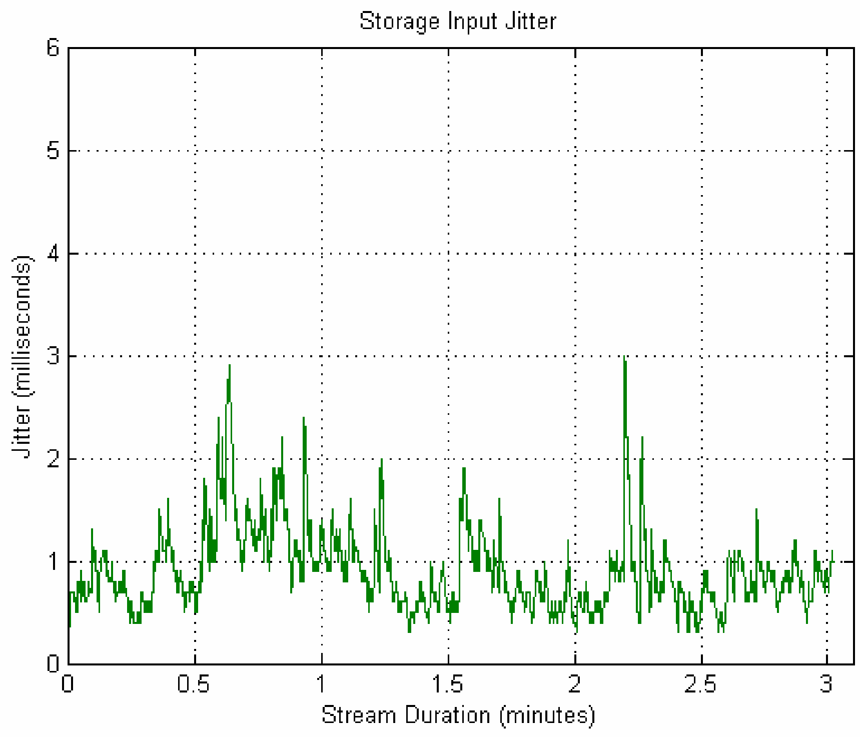


Figure 10-11: Storage input jitter for LAN setup

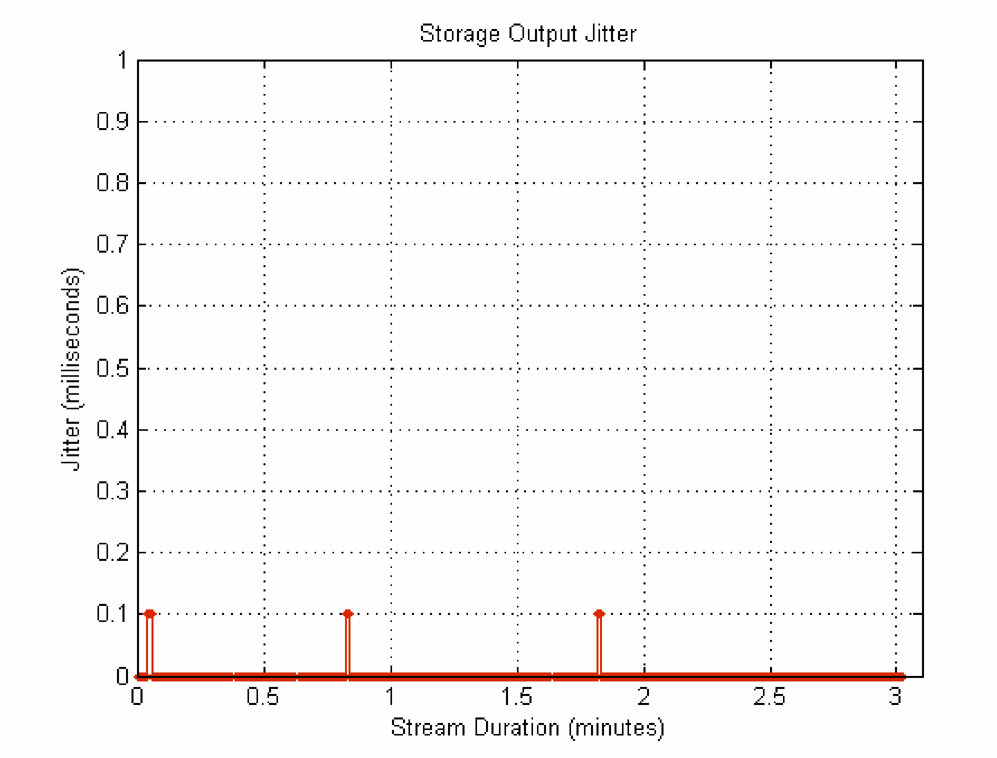


Figure 10-12: Storage output jitter for LAN setup

Parameter Measured	Mean	Standard Deviation	Standard Error
Overall delay	177.9 msec	21.2	0.406
Generic streaming overhead	174.8 msec	21.0	0.403
Jitter	0.0 msec	0.0032	0.00006

Table 10-2: Measured values for LAN setup

10.5.1.2 Wide Area Network Test (FSU)

Stream duration for WAN test (FSU) is 188321 milliseconds (~ 3 min. 8 sec). Jitter at the input of the Generic Streaming Service is shown in Figure 10-13 while the output jitter is shown in Figure 10-14. Measured values are shown in Table 10-3.

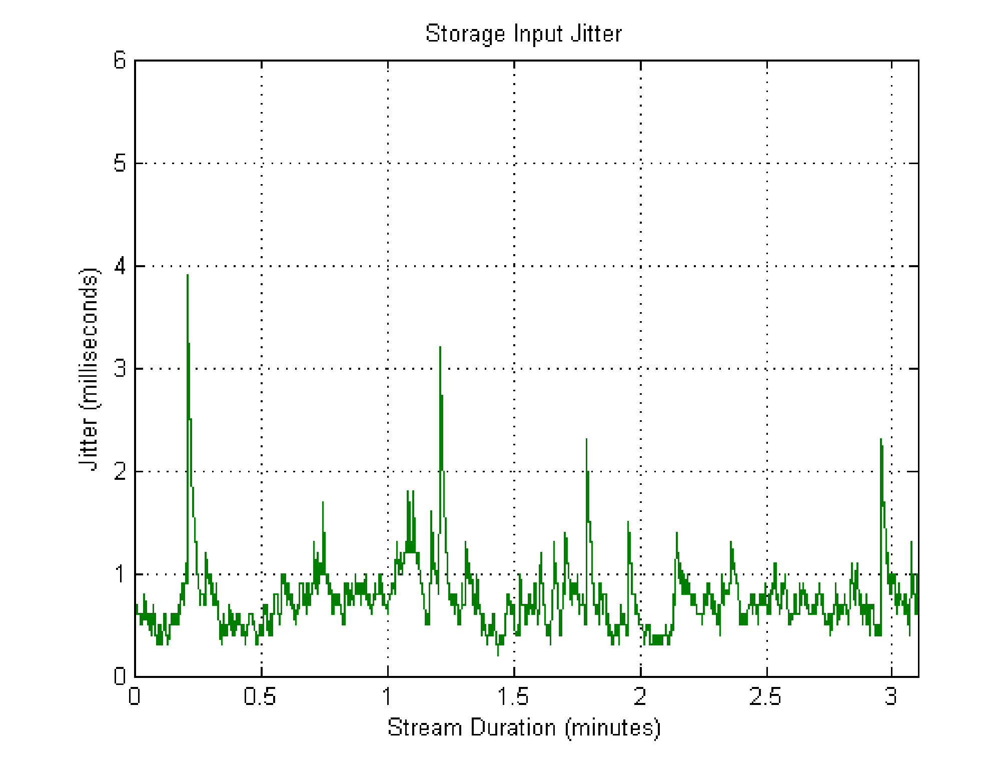


Figure 10-13: Storage input jitter for WAN-FSU setup

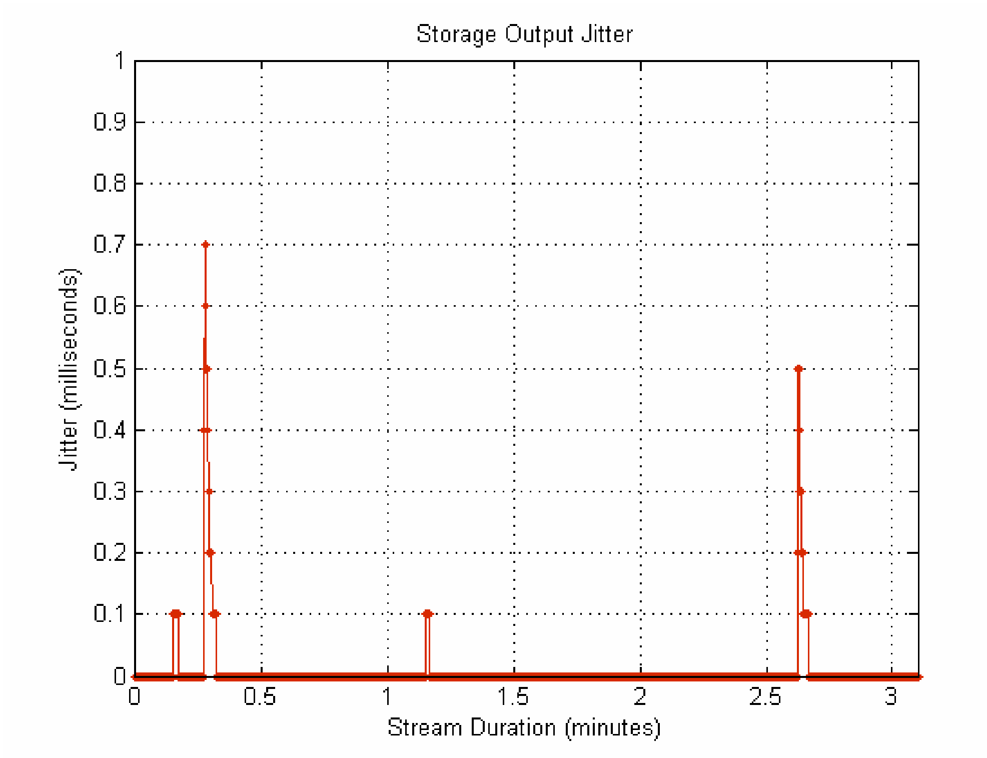


Figure 10-14: Storage output jitter for WAN-FSU setup

Parameter Measured	Mean	Standard Deviation	Standard Error
Overall delay	200.6 msec	26.8	0.503
Generic streaming overhead	174.9 msec	26.8	0.502
Jitter	0.0 msec	0.0145	0.0003

Table 10-3: Measured values for WAN-FSU setup

10.5.1.3 Wide Area Network Test (UCSD)

Stream duration for WAN test (UCSD) is 178360 milliseconds (~ 2 min. 58 sec). Jitter at the input of the Generic Streaming Service is shown in Figure 10-15 while the output jitter is shown in Figure 10-16. Measured values are shown in Table 10-4.

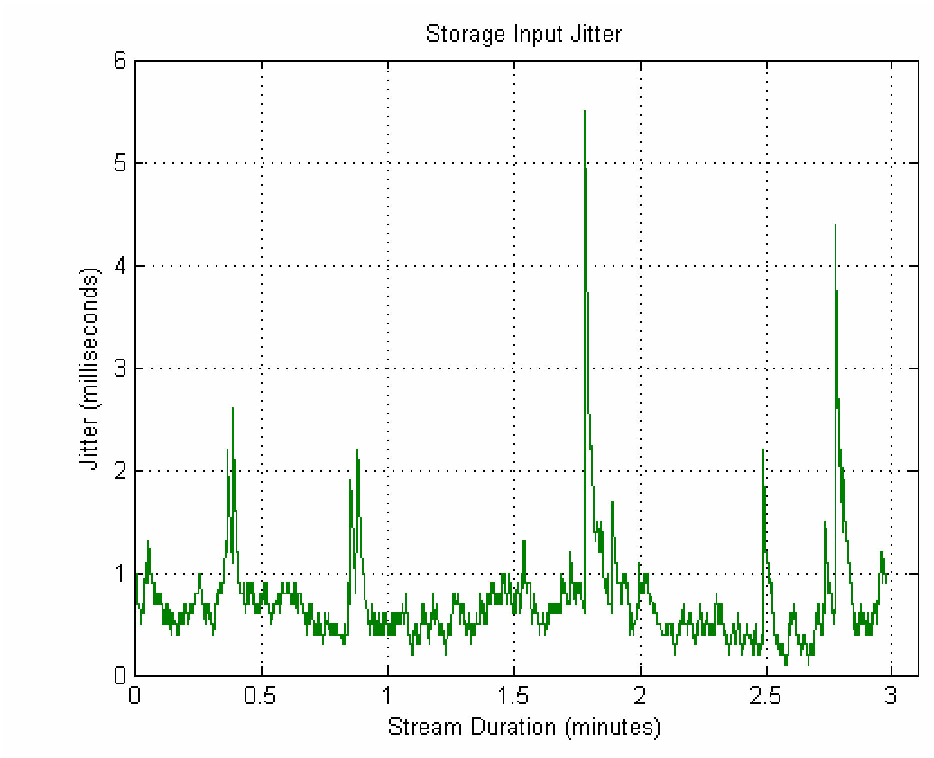


Figure 10-15: Storage input jitter for WAN-UCSD

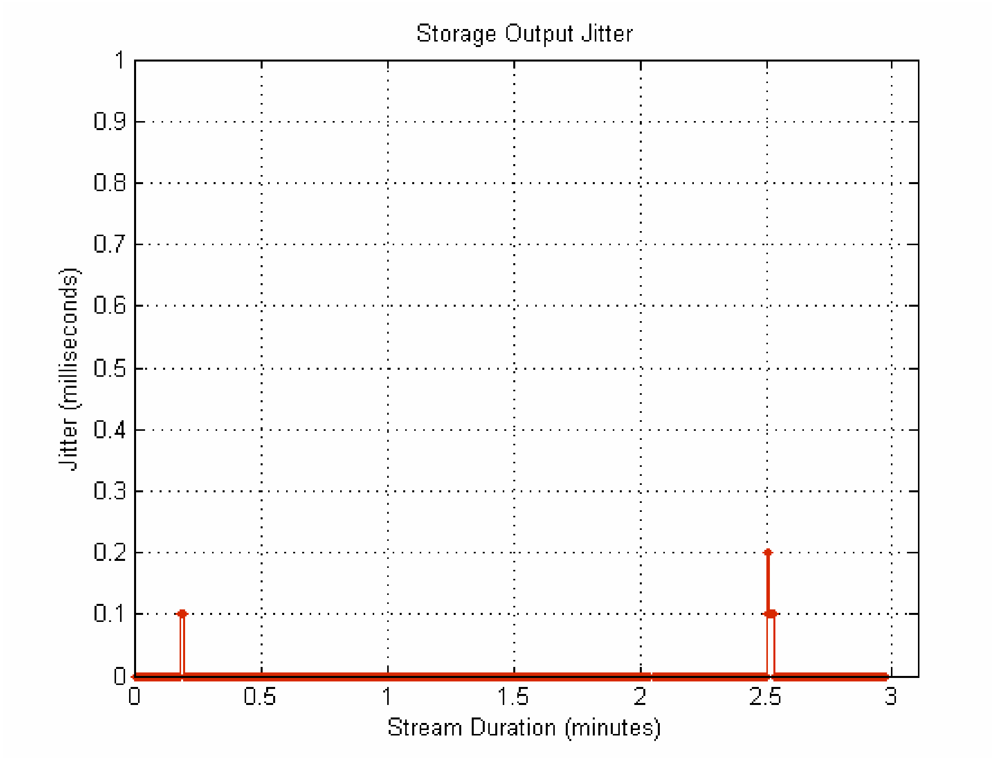


Figure 10-16: Storage output jitter for WAN-UCSD

Parameter Measured	Mean	Standard Deviation	Standard Error
Overall delay	229.3 msec	18.2	0.350
Generic streaming overhead	194.1 msec	18.2	0.350
Jitter	0.0 msec	0.0034	0.00007

Table 10-4: Measured values for WAN-UCSD setup

10.5.2 Evaluation of Test Results

For each case, a test stream is published for a duration of approximately 3 min. and results are recorded at eSports test client. In order to take timestamps at points indicated in Figure 10-10, we added 4 fields to RTPEvent and set the timestamps at each point.

We have demonstrated how jitter reduction service reduces the jitter of a stream generated from a client located in a Local Area Network and Wide Area Network. We have also measured the overall delay and the overhead of jitter reduction service into the overall delay. In tables we showed this as Storage Overhead.

Buffer of jitter reduction service has the duration of 200 msec which is a typical buffer size of a video player. This buffer is required to lower the jitter of the stream. The longer the buffer duration the lower the jitter is expected.

As seen from the results, the dominant factor for stream delay is caused by Generic Streaming Service overhead especially from the buffer of the jitter reduction service. The rest of the delay is caused by the transport delay which depends on the location of the client. The difference between “overall mean delay” and “generic streaming overhead” demonstrates that the rest of the delay is caused by the distance between the location of the publisher stream and Generic Streaming Service. Results are as follows:

LAN Test:

177.9 - 174.8 = 3.1 msec

WAN- FSU Test:

200.6 - 174.9 = 25.7 msec

WAN- UCSD Test:

229.3 - 194.1 = 35.2 msec

We have successfully reduced the jitter to zero while we kept the overall delay around 200 msec and 250 msec. There are spikes however in all cases, which are caused by thread switching and resource allocation by the underlying operating system.

10.6 Summary

In this chapter, we explained eSports Collaborative Multimedia Streaming and Annotation System which is built on Archive Service and Generic Streaming Framework (Archived Streaming Service) we have explained in this thesis. We explained how eSports system interacts with our streaming architecture to provide new features in the annotation systems.

Through this streaming service, instant replay of real-time live streams is made possible. It is able to support annotation of real-time live streams which can be captured from TV programs or capture devices. It can also access streams in other communities such as H.323 or the Access Grid, through GlobalMMCS.

Archived Streaming Service provides stream control functions such as pause, rewind, forward for NB event based streams. This enables eSports clients to stream any type of data with NB events. So, video data, whiteboard events and text messages can be encapsulated inside NB events and those events are provided with the control functions mentioned above.

Chapter 11

Conclusion, Contribution and Future Research Directions

11.1 Conclusion

In this dissertation, we have presented a fault tolerant, services-based generic streaming framework based on publish/subscribe systems. We separated session control and management services from data services.

We introduced XML Based General Session Protocol (XGSP) for integration of heterogeneous videoconferencing systems and demonstrated that this protocol is flexible and easy to extend to other multimedia streaming environments such as RealMedia clients (RealPlayer) which can receive streams in videoconferencing sessions. This also allows supporting heterogeneous client types; H.323 terminals, SIP clients, multicast clients, RealPlayers, and cellular phone media players in one collaboration session sharing same streams in the session. We also extended XGSP framework to include RTSP semantics (record, replay) to support archiving and replay of streams in this integrated environment. Using XML format for signaling and metadata description provides an easy integration of different collaborative environments to a common collaborative environment.

We used NaradaBrokering native events (NB Event) to transport any type of data such as audio, video, text and images. We encapsulated data inside NB Events and transported them across brokering network. We presented NTP based Time Service and Jitter Reduction Service within the broker system and concluded that these two services are essential in order to replay streams at their original rate and in a synchronous fashion. In our case, we implemented these services within NaradaBrokering system, but any messaging system can be used for such streaming as long as they provide these services within the messaging system.

We proposed a replication scheme within the brokering system. We introduced control topics and new event types to implement this scheme within the brokering system. Same guidelines can be followed for other messaging systems to implement this replication scheme. We conducted tests for different topologies to measure the overhead of this scheme. We investigated the effect of increasing the number of repositories on the latency and jitter.

Generic Streaming Framework that we proposed works for every event type such as NB Event, RTPEvent, JMS Event and can be used to stream any data type. It is independent of data type of the stream.

Using a shared, fault tolerant and distributed metadata repository service – WS-Context Service provides easy management and distribution of metadata among services and components in the system. Services and other components in the system can access the session metadata by using a URL pointing to that metadata. Since WS-Context Service is fault tolerant, sessions can easily be recovered from the metadata stored in the WS-Context Service.

We used the services and components developed within this architecture to develop an annotation system: eSports Collaborative Multimedia Streaming and Annotation System. With the help of the streaming architecture we proposed in this dissertation, eSports system can access streams in videoconferencing sessions and annotate real-time live streams.

11.2 Summary of Answers for Research Questions

Here we summarize the answers for the research questions presented in this dissertation;

1. **How can we build gateways for the XGSP based videoconferencing system so as to improve the heterogeneity of the system?**

   There are two types of gateways. First type of gateway just transforms the client signals into XGSP signals and XGSP signals into client signals as described in section 3.3 while explaining H.323 and SIP gateways. XGSP Session Server needs to receive client information including its media capability through JoinSession message. When a client leaves the session gateway it needs to send

LeaveSession for the client leaving the session. Rest of the communication between the gateway and the client connecting to it is hidden from XGSP Session Server. Another type of gateway is described in chapter 4 while explaining XGSP Streaming Gateway. This type of gateway does more than signal transformation. While some of its components convert streams into another format, other components of the gateway provide signaling support for the client.

2. **How can we provide streams to mobile clients particularly cellular phone clients?**

   As explained in the previous question, for this type of clients there needs to be stream conversion. We explained in chapter 4 that streams in XGSP sessions need to be converted into other streaming formats before being sent to these types of clients. Cellular phone clients are limited in screen size and bandwidth and utilize cellular network. Also a client provides limited API to be used in a flexible way. Therefore, as explained in section 4.7, we need to have a gateway (GlobalMMCS Mobile Gateway) similar to XGSP Streaming Gateway but one that provides services for mobile clients to stream cellular clients.

3. **What is the time discrepancy between two entities located on different machines and updating their time with Time Service timestamps?**

   As described in chapter 5, Time Service provides NTP timestamps. NTP can achieve 1-30 ms accuracy, where accuracy implies that using by NTP the underlying clock is within 30 ms of time server clock (usually an atomic clock).

However, this accuracy also depends on the roundtrip delay between the machine and the time server supplying the time service. The difference between the delay from the machine to the time server and the delay from the time server to the machine also contributes to the accuracy of the offset computed.

We have tested the discrepancy of two Time Services running on two different machines on the same LAN. The average discrepancy between them is around 6 msec. However, the same roundtrip principle mentioned above also applies to this test. The difference is that, the roundtrip in LAN is around 2-3 ms which is much less than the roundtrip in WAN. We also need to take into account the clock read accuracy which is 1 ms on Linux machines. Hence the error in average discrepancy is bounded with this roundtrip delay and the clock read accuracy.

4. How much can Jitter Reduction Service reduce jitter?

As explained in chapter 6, before releasing events, Buffering Service buffers those events. This will delay events but this delay is required in order to achieve a minimum jitter. For Time Differential Service (TDS) takes the first event's timestamp as a reference and releases events preserving the timestamp between consecutive events. As seen from section 6.4, we can achieve a jitter less than a millisecond.

5. **How much overhead is introduced because of the addition of repositories to the messaging system?**

   For reliable delivery we need to have at least one repository. In reliable delivery scheme, the publisher and the repository exchange control messages to make sure that the repository stores the published event. Similarly, the subscriber and the repository exchange control messages to make sure that the subscriber receives the message. There is an increase in the cost compared to non-reliable delivery system. But as seen in section 7.4, cost increase from one repository case to two repositories or three repositories case is acceptable. This is because repository redundancy scheme does not affect the underlying reliable delivery mechanism. Repositories exchange messages among themselves without interfering with the reliable delivery mechanism.

6. **What is the difference between videoconferencing session metadata and archived streaming session metadata? How can we categorize the metadata in this streaming architecture?**

   In videoconferencing sessions, clients join and leave the session. So the stream information in the session dynamically changes. Moreover, when the session is over, these metadata have no meaning, since the session is no longer available. Therefore, we only need to keep the metadata of videoconferencing session for the lifetime of the session. This type of metadata is temporary and is considered dynamic metadata.

In archived streaming sessions, if a session is an archived session, the information of streams archived in that session may be updated. Once a stream is archived, its metadata is permanently entered to the session stream metadata even if the stream is stopped, since some duration of the stream is already archived and it is available even after the session is closed. This type of metadata is static metadata. But if it is a replay session, the session metadata is dynamic, since replay sessions are temporary. Although streams are archived, and thus their metadata is static, these streams are replayed under different topics in the replay session and are no longer available when the replay session is terminated.

7. **How can we build a data format independent streaming architecture in order to archive and replay streams?**

Since client will replay the stream, it should be client's concern to know the media format and encoding. A client can obtain it from metadata of the stream as explained in chapter 8. Thus, it should choose streams that it is capable of playing. Generic Streaming Framework does not need to know the data format. When it archives the streams, it needs to connect to a topic, if it is NaradaBrokering topic it expects NB Events, if it is JMS topic it expects JMS Events. Therefore, it only needs to preserve the original event format rather than the format of the data in the payload of the event. NB Event and JMS Event can be viewed as transport packets in messaging systems. In traditional streaming architectures, data is transported with RTP packets. The payload of the RTP packet is the stream data. Traditional streaming servers reconstruct RTP packets from the streams they archived. Therefore, they need to support the data format of the stream and be capable of processing it. In our architecture, we receive events whether they are NB Event, JMS Event or RTPEvent, and we encapsulate them inside NB Event. We timestamp the NB Event (we generate the timestamp if the client did not timestamp the published event) and archive it. During replay, we replay it based on the timestamp they carry. Replay service only needs to know the event type to make the required connection to transport the event and to overwrite the topic information of the event. We demonstrated this framework for JMS Events, NB Events and RTPEvents in chapter 9.

8. **Which services should a messaging middleware possess to provide major RTSP functionalities on streams?**

   Messaging system should have a Time Service which provides NTP timestamps as described in chapter 5 and Jitter Reduction Service which can release events preserving the timespacing between consecutive events as described in chapter 6.

   Timestamps are required so that we can provide play, rewind and forward functionalities. Rewind is actually a play command where the time passed to the play command refers to the time in the past compared to the current playing time. Similar to rewind, forward is a play command where the time passed to play command is the timestamp in the future compared to the current playing time. Since these functionalities take timestamp as a parameter and since we need to synchronize the streams generated at different sources we need a NTP based Time Service.

Jitter Reduction Service is needed since we need to replay events in their original rate where the timespace between consecutive events is preserved. Without such mechanism we would not know when to release an event to the client.

9. How can we synchronize streams in a session? Do we need SMIL to synchronize streams?

As we explained in section 9.5, SMIL is a layout language that tells the player whether streams will be synchronized based on the absolute timing of the streams or the relative timing among streams. In our streaming architecture, we use NTP timestamps to timestamp events and replay events based on those timestamps. That is, the timing is absolute timing of the streams. In addition to that, we need Jitter Reduction Service to synchronize events from different topics. Therefore, only one Jitter Reduction Service should be used for one replay session. Because events from different topics will go through the same buffer and twill be ordered based on their timestamps. This will make sure that an event that carries a higher timestamp than another event from another topic will not be replayed before that event.

10. Where should we locate Jitter Reduction Service? What is the requirement of this service in order to benefit from its functionality?

Jitter Reduction Service runs on NB Events. It only uses timestamps that those events carry. As we explained in section 9.3.1 and 9.3.2, it can be located at the repository or at the client side depending on the client's capability. If the client is capable of playing NB Events, it can be located at the client side. This will also result in better jitter value, because if we run Jitter Reduction Service at the repository, events released from TDS need to go through network which may introduce jitter. However, running while locating it at the client makes sure that the jitter of the input events is minimal, since events released from Jitter Reduction Service do not go through the network.

11. How can we achieve instant replay of real-time live videoconferencing streams?

The instant replay of real-time live videoconferencing streams is possible in an integrated environment where the archived parts of the streams are provided to streaming clients immediately. The client should be allowed to replay, rewind and forward (until the archive point) the streams as we explained in section 9.4.

12. How much delay does Generic Streaming Framework introduce during instant replay and how successful is it to reduce the jitter?

Generic Streaming Framework utilizes Jitter Reduction Service to reduce the jitter of the incoming stream. From test results explained in section 10.5, the delay introduced during instant replay of streams can be around 200 msec depending on the location of the published stream. Jitter Reduction Service uses a buffer of 200 msec. The rest of the delay is introduced from the transport delay caused by the location of the streams. Jitter Reduction Service can successfully reduce jitter to less than 1 msec as shown in section 10.5.

13. What kind of functionalities can generic streaming framework provide to annotation systems?

Through Archived Streaming Service explained in chapters 8 and 9, annotation systems can display the following features;

- Videoconferencing streams are provided to annotation systems, which can display those streams.
- They can also instantly replay real-time live streams which can be streams captured either from TV programs or at remote events.
- They can annotate streams mentioned above as explained in section 10.3 and the annotated streams can be replayed synchronously with the annotations.
- Any data format, i.e. text or image, can be archived and replayed like video streams and can be replayed in the same sessions as explained in section 10.2.

11.3 Contribution

The main contribution of this dissertation is the development of a fault tolerant and services-based streaming architecture based on messaging middleware systems to integrate various video conferencing, streaming and annotation systems into an interoperable collaborative environment.

This research suggests using XML format in collaboration control framework, metadata description of the sessions and streams, and interactions between components of the collaboration system. It is easier to develop an environment that integrates videoconferencing and streaming systems with the help of service oriented architecture and Web Services technologies.

This architecture allows mobile clients; particularly cellular phone clients to receive streams from real-time videoconferencing sessions.

It uses a distributed fault tolerant metadata repository that can be shared among various services and clients in the system-WS-Context Service. Components in the system do not interact with each other to obtain metadata. They directly obtain metadata from WS-Context Service through a unique URL. This makes the system simpler and easier to manage. The collaboration environment is more tolerant, since the session metadata can be recovered from WS-Context Service.

This research demonstrates that a data format independent generic streaming framework for archiving and replay of streams can be built on top messaging middleware systems with the help of service oriented architecture and Web Services technologies. This study offers new functionalities for annotation systems such as providing replay of real-time live videoconferencing systems so that they can annotate them. Via this system annotation system can provide whiteboards or text messages as streams like video streams and replay all of them together in a synchronous fashion with RTSP functionalities (rewind, forward, pause, etc.).

Another contribution of this research is the services provided to messaging middleware. Major services are Time Service and Jitter Reduction Service. These two services are essential for a messaging middleware in order to provide RTSP functionalities for event based streams. Furthermore, our research also introduced a replication scheme in order to increase the fault tolerant of the messaging system in presence of repository failures.

11.4 Future Research Directions

We can list several research directives to extend this work:

First, XGSP Streaming Gateway converts streams into RealMedia format which can be played in RealPlayers. As seen from test results the number of streams is limited. In order to increase the number of streams converted, a streaming job scheduler, which will schedule and coordinate streaming jobs in a distributed environment can be developed. Also other streaming clients can be supported through this gateway or similar gateways can be developed for them.

Second, we have used MySQL database system to store events. More tests can be conducted on other databases or file-based storages to see if there would be performance improvements. A hybrid-scheme such as using MySQL for querying events and using file-based event storing and retrieval can be tested.

Moreover, a caching mechanism can be introduced to storage service. This can be used to decrease the number of storage access while replaying streams.

Replication scheme tests could be conducted in an intercontinental environment to see the behavior in Wide Area Network (WAN). Since the latency introduced and number of packets lost in WAN is higher than Local Area Network (LAN), it might be helpful to improve the replication scheme.

Appendix A

XGSP Audio/Video Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- by Hasan Bulut  -->
<!-- XGSPAV.xsd (XGSP Audio/Video Metadata Schema) -->
<xs:schema targetNamespace="http://globalmmcs.cgl/xgsp"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xgsp="http://globalmmcs.cgl/xgsp">
    <xs:complexType name="ActiveSessionType">
        <xs:sequence>
            <xs:element name="SessionID" type="xs:string"/>
            <xs:element name="Active" type="xs:boolean"/>
            <xs:element ref="xgsp:SessionMediaDescription" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CommunityType">
        <xs:sequence>
            <xs:element name="CommunityName" type="xs:string"/>
            <xs:element name="CommunitySessionID" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CommunitiesType">
        <xs:sequence>
            <xs:element ref="xgsp:Community" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Community" type="xgsp:CommunityType"/>
    <xs:element name="Media" type="xgsp:CommunitiesType"/>
    <xs:complexType name="SessionDescriptionType">
        <xs:sequence>
            <xs:element name="SessionID" type="xs:string"/>
            <xs:element name="SessionTime" type="xs:string"/>
            <xs:element name="URI" type="xs:anyURI"/>
```

```xml
            <xs:element name="SessionName" type="xs:string" minOccurs="0"/>
            <xs:element name="SessionOwner" type="xs:string" minOccurs="0"/>
            <xs:element name="SessionInfo" type="xs:string" minOccurs="0"/>
            <xs:element ref="xgsp:Contact" minOccurs="0"/>
            <xs:element ref="xgsp:SessionMediaDescription"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ContactType">
        <xs:sequence>
            <xs:element name="Email" type="xs:string"/>
            <xs:element name="PhoneNumber" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Contact" type="xgsp:ContactType"/>
    <xs:element name="SessionDescription" type="xgsp:SessionDescriptionType"/>
    <xs:complexType name="MediaDescriptionType">
        <xs:sequence>
            <xs:element name="MediaType" type="xgsp:MediaType"/>
            <xs:element name="MediaFormat" type="xgsp:MediaFormatType"/>
            <xs:element ref="xgsp:Transport"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TransportType">
        <xs:sequence>
            <xs:element name="IPAddress">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="7"/>
                        <xs:maxLength value="15"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="PortNumber" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:positiveInteger">
                        <xs:minInclusive value="1025"/>
                        <xs:maxInclusive value="65365"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="TopicNumber" type="xs:long" minOccurs="0"/>
            <xs:element name="SSRC" type="xs:positiveInteger" minOccurs="0"/>
            <xs:element name="TTL" type="xs:long" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Transport" type="xgsp:TransportType"/>
    <xs:element name="MediaDescription" type="xgsp:MediaDescriptionType"/>
    <xs:element name="AudioDescription" type="xgsp:MediaDescriptionType"/>
    <xs:element name="VideoDescription" type="xgsp:MediaDescriptionType"/>
    <xs:complexType name="JoinSessionType">
        <xs:sequence>
            <xs:element name="ClientID" type="xs:string"/>
            <xs:element name="SessionID" type="xs:string"/>
            <xs:element ref="xgsp:MediaDescription" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
```

```
<xs:element name="SessionMediaDescription" type="xgsp:SessionMediaDescriptionType"/>
<xs:complexType name="SessionMediaDescriptionType">
    <xs:sequence>
        <xs:element name="SessionID" type="xs:string"/>
        <xs:element ref="xgsp:AudioDescription"/>
        <xs:element ref="xgsp:VideoDescription"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="JoinSession" type="xgsp:JoinSessionType"/>
<xs:complexType name="LeaveSessionType">
    <xs:sequence>
        <xs:element name="ClientID" type="xs:string"/>
        <xs:element name="SessionID" type="xs:string"/>
        <xs:element name="LeaveReason" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="LeaveSession" type="xgsp:LeaveSessionType"/>
<xs:complexType name="SessionSignalResponseType">
    <xs:sequence>
        <xs:element name="ClientID" type="xs:string"/>
        <xs:element name="SessionID" type="xs:string"/>
        <xs:element name="ResponseType" type="xs:string"/>
        <xs:element name="Result">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="OK"/>
                    <xs:enumeration value="FAIL"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element ref="xgsp:MediaDescription" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="VideoSourceSelectionType">
    <xs:sequence>
        <xs:element name="ClientID" type="xs:string"/>
        <xs:element name="SessionID" type="xs:string"/>
        <xs:element name="Active" type="xs:boolean"/>
        <xs:element ref="xgsp:VideoSource"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ClientType">
    <xs:sequence>
        <xs:element name="Username" type="xs:string"/>
        <xs:element name="SSRC" type="xs:string"/>
        <xs:element name="Description" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="VideoSource" type="xgsp:ClientType"/>
<xs:element name="VideoSourceSelection" type="xgsp:VideoSourceSelectionType"/>
<xs:complexType name="VideoMixerType">
    <xs:sequence>
        <xs:element name="SessionID" type="xs:string"/>
        <xs:element ref="xgsp:VideoSource" maxOccurs="4"/>
    </xs:sequence>
</xs:complexType>
```

```
<xs:element name="VideoMixer" type="xgsp:VideoMixerType"/>
<xs:complexType name="VideoMixerReplyType">
    <xs:sequence>
        <xs:element name="SessionID" type="xs:string"/>
        <xs:element name="MixerID" type="xs:string"/>
        <xs:element name="Result" type="xs:string"/>
        <xs:element name="Reason" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="VideoMixerReply" type="xgsp:VideoMixerReplyType"/>
<xs:element name="Client" type="xgsp:ClientType"/>
<xs:complexType name="StreamEventType">
    <xs:sequence>
        <xs:element name="SessionID" type="xs:string"/>
        <xs:element name="EventType" type="xgsp:EventType"/>
        <xs:element name="MediaType" type="xs:string"/>
        <xs:element ref="xgsp:Client"/>
        <xs:element name="ImagePath" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="MediaType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="AUDIO"/>
        <xs:enumeration value="VIDEO"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="MediaFormatType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="H261"/>
        <xs:enumeration value="H263"/>
        <xs:enumeration value="G711"/>
        <xs:enumeration value="GSM"/>
        <xs:enumeration value="MSGSM"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="EventType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="NEWSTREAM"/>
        <xs:enumeration value="BYESTREAM"/>
        <xs:enumeration value="ACTIVETOPASSIVE"/>
        <xs:enumeration value="PASSIVETOACTIVE"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="RequestAllStreamsType">
    <xs:sequence>
        <xs:element name="ClientID" type="xs:string"/>
        <xs:element name="SessionID" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="RequestAllStreams" type="xgsp:RequestAllStreamsType"/>
<xs:complexType name="RequestAllStreamsReplyType">
    <xs:sequence>
        <xs:element name="SessionID" type="xs:string"/>
        <xs:element name="AudioStreamList" type="xgsp:StreamListType"/>
        <xs:element name="VideoStreamList" type="xgsp:StreamListType"/>
    </xs:sequence>
    </xs:complexType>
    <xs:complexType name="StreamListType">
        <xs:sequence>
            <xs:element ref="xgsp:StreamEvent" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="StreamEvent" type="xgsp:StreamEventType"/>
    <xs:element name="StreamList" type="xgsp:StreamListType"/>
    <xs:element name="ActiveSession" type="xgsp:ActiveSessionType"/>
    <xs:element name="Communities" type="xgsp:CommunitiesType"/>
    <xs:element name="SessionSignalResponse" type="xgsp:SessionSignalResponseType"/>
    <xs:element name="StreamSource" type="xgsp:ClientType"/>
    <xs:element name="RequestAllStreamsReply" type="xgsp:RequestAllStreamsReplyType"/>
    <xs:complexType name="VideoSwitchType">
        <xs:sequence>
            <xs:element name="ClientID" type="xs:string"/>
            <xs:element name="SessionID" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="VideoSwitch" type="xgsp:VideoSwitchType"/>
</xs:schema>
```

Appendix B

WsRtsp Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- by Hasan Bulut  -->
<!-- metadataschema.xsd (WsRtsp Metadata Schema) -->
<xs:schema targetNamespace="http://streaming.cgl/wsrtsp/schema"
xmlns:wsrtsp="http://streaming.cgl/wsrtsp/schema"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:complexType name="AtomicStreamType">
        <xs:annotation>
            <xs:documentation>Stream information with broker transport template
information</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="StreamID" type="xs:string"/>
            <xs:element ref="wsrtsp:NBInfo"/>
            <xs:element ref="wsrtsp:StreamInfo"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="AtomicStream" type="wsrtsp:AtomicStreamType">
        <xs:annotation>
            <xs:documentation>Element of type AtomicStreamType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="AtomicStreamsType">
        <xs:annotation>
            <xs:documentation>List of AtomicStream</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="wsrtsp:AtomicStream" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="AtomicStreams" type="wsrtsp:AtomicStreamsType">
        <xs:annotation>
            <xs:documentation>List of AtomicStream</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="CompositeStreamType">
        <xs:annotation>
            <xs:documentation>Includes zero or more template IDs of the atomic streams
</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="StreamID" type="xs:string"/>
            <xs:element ref="wsrtsp:NBTemplateInfo" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="CompositeStream" type="wsrtsp:CompositeStreamType">
        <xs:annotation>
            <xs:documentation>Element of type CompositeStreamType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="CompositeStreamsType">
        <xs:annotation>
            <xs:documentation>List of CompositeStream</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="wsrtsp:CompositeStream" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="CompositeStreams" type="wsrtsp:CompositeStreamsType">
        <xs:annotation>
            <xs:documentation>List of CompositeStream</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="NBBrokerInfoType">
        <xs:annotation>
            <xs:documentation>Broker IP address and port number
information</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="BrokerIP" type="xs:string"/>
            <xs:element name="BrokerPortNumber" type="xs:int"/>
            <xs:element name="CommunicationType" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="NBBrokerInfo" type="wsrtsp:NBBrokerInfoType">
        <xs:annotation>
            <xs:documentation>Element of type NBBrokerInfoType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="NBTemplateInfoType">
        <xs:annotation>
            <xs:documentation>Broker template information</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="TemplateID" type="xs:int"/>
            <xs:element name="TemplateType" type="xs:int"/>
            <xs:element name="Template" type="xs:string"/>
            <xs:element name="ReplicaID" type="xs:int"/>
            <xs:element name="EntityID" type="xs:int" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="NBTemplateInfo" type="wsrtsp:NBTemplateInfoType">
        <xs:annotation>
            <xs:documentation>Element of type NBTemplateInfoType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="NBInfoType">
        <xs:annotation>
            <xs:documentation>Includes NBBrokerInfo and NBTemplateInfo</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="wsrtsp:NBBrokerInfo"/>
            <xs:element ref="wsrtsp:NBTemplateInfo"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="NBInfo" type="wsrtsp:NBInfoType">
        <xs:annotation>
            <xs:documentation>Element of type NBInfoType.</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="SessionType">
        <xs:annotation>
            <xs:documentation>Session related information.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="SessionID" type="xs:string">
                <xs:annotation>
                    <xs:documentation>CommunitySessionID + timestamp (unique
ID)</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="CommunitySessionID" type="xs:string"
default="GlobalMMCS/AG">
                <xs:annotation>
                    <xs:documentation>sessionID of the collaboration session</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="SessionCommunity" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>GlobalMMCS/WSRTSP</xs:documentation>
                </xs:annotation>
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="GlobalMMCS"/>
                        <xs:enumeration value="GlobalMMCS/AG"/>
                        <xs:enumeration value="WSRTSP/GENERIC"/>
                        <xs:enumeration value="WSRTSP/GLOBALMMCS"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="SessionInfo" type="xs:string" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>Extra session information   </xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="BeginType" type="xs:long" minOccurs="0"/>
            <xs:element name="EndTime" type="xs:long" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Session" type="wsrtsp:SessionType">
        <xs:annotation>
            <xs:documentation>Element of type SessionType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="SessionsType">
        <xs:annotation>
            <xs:documentation>List of sessions</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="wsrtsp:Session" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="XgspSessionType">
        <xs:annotation>
            <xs:documentation>GlobalMMCS session information including audio and video
streams list.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="wsrtsp:Session"/>
            <xs:element ref="wsrtsp:VideoStreamInfos" minOccurs="0"/>
            <xs:element ref="wsrtsp:AudioStreamInfos" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="XgspSession" type="wsrtsp:XgspSessionType">
        <xs:annotation>
            <xs:documentation>Element of type XgspSessionType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="XgspSessions" type="wsrtsp:SessionsType">
        <xs:annotation>
            <xs:documentation>GlobalMMCS sessions.</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="ArchiveSessionType">
        <xs:annotation>
            <xs:documentation>Session information and list of AtomicStreams</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="wsrtsp:Session"/>
            <xs:element ref="wsrtsp:AtomicStreams"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ArchiveSession" type="wsrtsp:ArchiveSessionType">
        <xs:annotation>
            <xs:documentation>List of archived atomic streams</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="ArchiveSessions" type="wsrtsp:SessionsType">
        <xs:annotation>
            <xs:documentation>Sessions which have archived streams</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="StreamInfoType">
        <xs:annotation>
            <xs:documentation>Stream related information. Although specifically designed for
multimedia streams, it can also be used for any kind of stream.</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="TopicID" type="xs:string"/>
            <xs:element name="Username" type="xs:string"/>
            <xs:element name="Media">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="AUDIO"/>
                        <xs:enumeration value="VIDEO"/>
                        <xs:enumeration value="TEXT"/>
                        <xs:enumeration value="BINARY"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="MediaFormat">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="H261/RTP"/>
                        <xs:enumeration value="ULAW"/>
                        <xs:enumeration value="OTHER"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="EventFormat">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="RTPEvent"/>
                        <xs:enumeration value="NBEvent"/>
                        <xs:enumeration value="JMSEvent"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="Description" type="xs:string" default="Not available"
minOccurs="0"/>
            <xs:element name="SSRC" type="xs:long" default="0" minOccurs="0"/>
            <xs:element name="ImagePath" type="xs:string" minOccurs="0"/>
            <xs:element name="StartTime" type="xs:long" minOccurs="0"/>
            <xs:element name="StopTime" type="xs:long" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="StreamInfo" type="wsrtsp:StreamInfoType">
        <xs:annotation>
            <xs:documentation>Element of type StreamInfoType</xs:documentation>
        </xs:annotation>
    </xs:element>
```

```
<xs:complexType name="StreamInfosType">
    <xs:annotation>
        <xs:documentation>List of streams</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element ref="wsrtsp:StreamInfo" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="StreamInfos" type="wsrtsp:StreamInfosType">
    <xs:annotation>
        <xs:documentation>Element of type StreamInfosType</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="AudioStreamInfos" type="wsrtsp:StreamInfosType">
    <xs:annotation>
        <xs:documentation>List of audio streams</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="VideoStreamInfos" type="wsrtsp:StreamInfosType">
    <xs:annotation>
        <xs:documentation>List of video streams</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="WsRtspSessionType">
    <xs:annotation>
        <xs:documentation>Rtsp session inforation</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="WsRtspSessionID" type="xs:string"/>
        <xs:element name="ClientID" type="xs:string"/>
        <xs:element name="WsRtspMode" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="WsRtspSession" type="wsrtsp:WsRtspSessionType">
    <xs:annotation>
        <xs:documentation>Element of type RtspSessionType</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="WsRtspRequestType">
    <xs:annotation>
        <xs:documentation>Rtsp request</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="WsRtspSessionID" type="xs:string"/>
        <xs:element ref="wsrtsp:NBTemplateInfo" minOccurs="0"/>
        <xs:element ref="wsrtsp:ArchiveSession" minOccurs="0"/>
        <xs:element name="StartTime" type="xs:long" minOccurs="0"/>
        <xs:element name="StopTime" type="xs:long" minOccurs="0"/>
        <xs:element name="Speed" type="xs:int" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="WsRtspRequest" type="wsrtsp:WsRtspRequestType">
    <xs:annotation>
        <xs:documentation>Element of type RtspRequestType</xs:documentation>
    </xs:annotation>
</xs:element>
```

```
    <xs:complexType name="WsRtspResponseType">
        <xs:annotation>
            <xs:documentation>Rtsp response</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="WsRtspSessionID" type="xs:string"/>
            <xs:element name="ClientID" type="xs:string"/>
            <xs:element name="StatusCode" type="xs:int" minOccurs="0"/>
            <xs:element name="ReasonPhrase" type="xs:string" minOccurs="0"/>
            <xs:element ref="wsrtsp:WsRtspStream" minOccurs="0"/>
            <xs:element name="StartTime" type="xs:long" minOccurs="0"/>
            <xs:element name="StopTime" type="xs:long" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="WsRtspResponse" type="wsrtsp:WsRtspResponseType">
        <xs:annotation>
            <xs:documentation>Element of type RtspResponseType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="WsRtspStreamType">
        <xs:annotation>
            <xs:documentation>Either an atomic stream or a composite
stream.</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:element ref="wsrtsp:AtomicStreams" minOccurs="0"/>
            <xs:element ref="wsrtsp:ReplayStreams" minOccurs="0"/>
        </xs:choice>
    </xs:complexType>
    <xs:element name="WsRtspStream" type="wsrtsp:WsRtspStreamType">
        <xs:annotation>
            <xs:documentation>Element of type WsRtspStreamType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="StreamTopicType">
        <xs:annotation>
            <xs:documentation>Stream topic information</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:element name="TopicName" type="xs:string"/>
            <xs:element name="TopicType" type="xs:int"/>
            <xs:element name="TemplateID" type="xs:int"/>
        </xs:choice>
    </xs:complexType>
    <xs:element name="StreamTopic" type="wsrtsp:StreamTopicType">
        <xs:annotation>
            <xs:documentation>Element of type StreamTopicType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="ReplayStreamType">
        <xs:annotation>
            <xs:documentation>AtomicStream with NBInfo</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="wsrtsp:AtomicStream"/>
            <xs:element ref="wsrtsp:NBTemplateInfo" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ReplayStream" type="wsrtsp:ReplayStreamType">
        <xs:annotation>
            <xs:documentation>Element of type ReplayStreamType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="ReplayStreamsType">
        <xs:annotation>
            <xs:documentation>List of ReplayStream</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element ref="wsrtsp:ReplayStream" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ReplayStreams" type="wsrtsp:ReplayStreamsType">
        <xs:annotation>
            <xs:documentation>Element of type ReplayStreamsType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="UniqueIDType">
        <xs:annotation>
            <xs:documentation>Request and respond message for ID
assignment</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="MessageType">
                <xs:annotation>
                    <xs:documentation>Request / Response</xs:documentation>
                </xs:annotation>
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="REQUEST"/>
                        <xs:enumeration value="RESPONSE"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="IPAddress" type="xs:string">
                <xs:annotation>
                    <xs:documentation>IP address of the requesting client's
machine</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="PortNumber" type="xs:string" minOccurs="0">
                <xs:annotation>
                    <xs:documentation>0 if no port number is available</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="EntityID" type="xs:int" minOccurs="0"/>
            <xs:element ref="wsrtsp:StreamTopic" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="UniqueID" type="wsrtsp:UniqueIDType">
        <xs:annotation>
            <xs:documentation>Element of type UniqueIDType</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="ReplaySessions" type="wsrtsp:SessionsType"/>
    <xs:complexType name="ReplaySessionType">
        <xs:sequence>
            <xs:element ref="wsrtsp:Session"/>
            <xs:element ref="wsrtsp:ReplayStreams"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ReplaySession" type="wsrtsp:ReplaySessionType"/>
</xs:schema>
```

Appendix C

WsRtsp WSDL Interface

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- by Hasan Bulut  -->
<!—WsRtsp.wsdl (WsRtsp Interface) -->
<wsdl:definitions name="WsRtsp" targetNamespace="http://streaming.cgl/wsrtsp/rtsp"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://streaming.cgl/wsrtsp/rtsp"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsd1="http://streaming.cgl/wsrtsp/schema">
    <wsdl:types>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://streaming.cgl/wsrtsp/schema"
schemaLocation="metadataschema.xsd"/>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="playSetupRequest">
        <wsdl:part name="clientID" type="xsd:string"/>
        <wsdl:part name="rtspSessionID" type="xsd:string"/>
        <wsdl:part name="rtspStream" type="xsd1:WsRtspStreamType"/>
    </wsdl:message>
    <wsdl:message name="playSetupResponse">
        <wsdl:part name="rtspResponse" type="xsd1:WsRtspResponseType"/>
    </wsdl:message>
    <wsdl:message name="recordSetupRequest">
        <wsdl:part name="clientID" type="xsd:string"/>
        <wsdl:part name="rtspSessionID" type="xsd:string"/>
        <wsdl:part name="rtspStream" type="xsd1:WsRtspStreamType"/>
    </wsdl:message>
    <wsdl:message name="recordSetupResponse">
        <wsdl:part name="rtspResponse" type="xsd1:WsRtspResponseType"/>
    </wsdl:message>
    <wsdl:message name="removeStreamRequest">
        <wsdl:part name="clientID" type="xsd:string"/>
        <wsdl:part name="rtspSessionID" type="xsd:string"/>
        <wsdl:part name="rtspStream" type="xsd1:WsRtspStreamType"/>
    </wsdl:message>
    <wsdl:message name="removeStreamResponse">
        <wsdl:part name="rtspResponse" type="xsd1:WsRtspResponseType"/>
    </wsdl:message>
    <wsdl:message name="teardownSessionRequest">
        <wsdl:part name="clientID" type="xsd:string"/>
        <wsdl:part name="rtspSessionID" type="xsd:string"/>
    </wsdl:message>
    <wsdl:message name="teardownSessionResponse">
        <wsdl:part name="rtspResponse" type="xsd1:WsRtspResponseType"/>
    </wsdl:message>
    <wsdl:message name="addReplayStreamRequest">
        <wsdl:part name="clientID" type="xsd:string"/>
        <wsdl:part name="rtspSessionID" type="xsd:string"/>
        <wsdl:part name="rtspStream" type="xsd1:WsRtspStreamType"/>
    </wsdl:message>
    <wsdl:message name="addReplayStreamResponse">
        <wsdl:part name="rtspResponse" type="xsd1:WsRtspResponseType"/>
    </wsdl:message>
    <wsdl:message name="addRecordStreamRequest">
        <wsdl:part name="clientID" type="xsd:string"/>
        <wsdl:part name="rtspSessionID" type="xsd:string"/>
        <wsdl:part name="rtspStream" type="xsd1:WsRtspStreamType"/>
    </wsdl:message>
    <wsdl:message name="addRecordStreamResponse">
        <wsdl:part name="rtspResponse" type="xsd1:WsRtspResponseType"/>
    </wsdl:message>
    <wsdl:portType name="WsRtsp">
        <wsdl:operation name="playSetup">
            <wsdl:input message="tns:playSetupRequest"/>
            <wsdl:output message="tns:playSetupResponse"/>
        </wsdl:operation>
        <wsdl:operation name="recordSetup">
            <wsdl:input message="tns:recordSetupRequest"/>
            <wsdl:output message="tns:recordSetupResponse"/>
        </wsdl:operation>
        <wsdl:operation name="teardownSession">
            <wsdl:input message="tns:teardownSessionRequest"/>
            <wsdl:output message="tns:teardownSessionResponse"/>
        </wsdl:operation>
        <wsdl:operation name="addReplayStream">
            <wsdl:input message="tns:addReplayStreamRequest"/>
            <wsdl:output message="tns:addReplayStreamResponse"/>
        </wsdl:operation>
        <wsdl:operation name="addRecordStream">
            <wsdl:input message="tns:addRecordStreamRequest"/>
            <wsdl:output message="tns:addRecordStreamResponse"/>
        </wsdl:operation>
        <wsdl:operation name="removeStream">
            <wsdl:input message="tns:removeStreamRequest"/>
            <wsdl:output message="tns:removeStreamResponse"/>
        </wsdl:operation>
    </wsdl:portType>
```

```
<wsdl:binding name="WsRtsp" type="tns:WsRtsp">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="playSetup">
        <soap:operation soapAction="http://streaming.cgl/wsrtsp/rtsp/playSetup"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="recordSetup">
        <soap:operation soapAction="http://streaming.cgl/wsrtsp/rtsp/recordSetup"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="teardownSession">
        <soap:operation soapAction="http://streaming.cgl/wsrtsp/rtsp/teardownSession"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="addReplayStream">
        <soap:operation soapAction="http://streaming.cgl/wsrtsp/rtsp/addReplayStream"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="addRecordStream">
        <soap:operation soapAction="http://streaming.cgl/wsrtsp/rtsp/addRecordStream"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="removeStream">
        <soap:operation soapAction="http://streaming.cgl/wsrtsp/rtsp/removeStream"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="WsRtspService">
        <wsdl:port binding="tns:WsRtsp" name="WsRtsp">
            <soap:address location="http://localhost:8080/Rtsp/services/Rtsp"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Bibliography

[1] Videoconferencing Cookbook Version 3.0, Video Development Initiative, Advanced Videoconferencing Components and Management, http://www.videnet.gatech.edu/cookbook*,* April, 2002.

[2] ITU-T Recommendation H.323, "Packet based multimedia communication systems", Feb. 1998.

[3] K. Almeroth, "The evolution of multicast: From the MBone to inter-domain multicast to {Internet2} deployment," *IEEE Network,* vol. 14, pp. 10-20, 2000.

[4] Polycom Inc., http://www.polycom.com.

[5] Radvision Ltd., http://www.radvision.com.

[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, Internet Engineering Task Force, June 2002, http://www.ietf.org/rfc/rfc3261.txt.

[7] The IP Telecommunications Portal , http://www.iptel.org/.

[8] The Access Grid Project, http://www.accessgrid.org/

[9] H. Eriksson, "MBONE: the multicast backbone," *Communications of the ACM,* vol. 37, pp. 54-60, 1994.

[10] O. Hodson and C. Perkins, "Robust audio tool (RAT)," http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/.

[11] V. Jacobson and S. McCanne, "VIC: A video conferencing tool," http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/.

[12] The Virtual Rooms VideoConferencing System, http://www.vrvs.org/.

[13] D. Adamczyk, D. Collados, G. Denis, J. Fernandes, P. Galvez, I. Legrand, H. Newman, and K. Wei, "Global Platform for Rich Media Conferencing and Collaboration," CHEP03, Ed. La Jolla, California, March 24-28, 2003

[14] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," RFC 2326, April 1998, http://www.ietf.org/rfc/rfc2326.txt.

[15] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RFC 3550: RTP: A Transport Protocol for Real-Time Applications " http://tools.ietf.org/html/rfc3550 2003.

[16] IG Recorder, http://www.agsc.ja.net/services/igrecorder.php.
[17] inSORS Integrated Communications, http://www.insors.com/main.htm.
[18] T. Disz, R. Olson, and R. Stevens, "Performance model of the Argonne Voyager multimedia server," in *IEEE International Conference on Application-specific Systems, Architectures, and Processors* Zurich, Switzerland, 1997.
[19] Argonne National Laboratory, http://www-fp.mcs.anl.gov/fl/accessgrid/, Ed.
[20] T. Dorcey, "CU-SeeMe Desktop Video Conferencing Software," *Connexions,* vol. 9, 3, March 1995.
[21] J. Han and B. Smith, "CU-SeeMe VR immersive desktop teleconferencing," ACM Press New York, NY, USA, 1997, pp. 199-207.
[22] QuickTime, http://www.apple.com/quicktime/.
[23] RealNetworks, http://www.realnetworks.com/.
[24] M. Claypool and J. Tanner, "The effects of jitter on the peceptual quality of video," in *MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 2)*, 1999, pp. 115-118.
[25] J. R. Smith and B. Lugeon, "A Visual Annotation Tool for Multimedia Content Description," in *Proc. SPIE Photonics East, Internet Multimedia Management Systems*, November, 2000, pp. 49–59.
[26] D. Bargeron, A. Gupta, J. Grudin, E. Sanocki, and F. Li, "Asynchronous collaboration around multimedia and its application to on-demand training," in *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii September, 2000.
[27] G. D. Abowd, "Classroom 2000: An experiment with the instrumentation of a living educational environment," http://www.research.ibm.com/journal/sj/384/abowd.html.
[28] D. Yamamoto and K. Nagao, "iVAS: Web-based Video Annotation System and its Applications," in *3rd International Semantic Web Conference(ISWC2004)* Hiroshima, Japan, 7-11 November 2004.
[29] M. P. Steves, M. Ranganathan, and E. Morse, "SMAT: Synchronous Multimedia and Annotation Tool," in *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, September, 2000.
[30] The Gryphon Project, http://researchweb.watson.ibm.com/distributedmessaging/gryphon.html
[31] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Trans. Comput. Syst.,* vol. 19, 3, pp. 332–383, 2003.
[32] SonicMQ, http://www.sonicsoftware.com/products/sonicmq/index.ssp.
[33] M. Hapner, R. Burridge, and R. Sharma, "Java Message Service Specification", Sun Microsystems, http://java. sun. com/products/jms, 2000.
[34] Sun Java System Application Server, http://www.sun.com/software/products/appsrvr/index.xml
[35] The IBM WebSphere MQ Family, http://www-3.ibm.com/software/integration/mqfamily/
[36] BEA WebLogic Server, http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/server/.

[37] K. Singh and H. Schulzrinne, "Interworking between SIP/SDP and H. 323," in *Proceedings of the 1st IP-Telephony Workshop (IPTel'2000)*, April 2000.
[38] S. Cisco, " H.323 and SIP Integration," Whitepaper, http://www.sipcenter.com/sip.nsf/html/WEBB5YP4SU/$FILE/Cisco_sh23g_wp.pdf.
[39] RealAudio, http://www.realnetworks.com/products/codecs/realaudio.html.
[40] RealVideo, http://www.realnetworks.com/products/codecs/realvideo.html.
[41] RealMedia, https://datatype.helixcommunity.org/.
[42] Community Grids Lab, Fault Tolerant High Performance Information System (FTHPIS), http://www.opengrids.org/extendeduddi/index.html.
[43] Global Multimedia Collaboration System (GLOBALMMCS), http://www.globalmmcs.org.
[44] G. Fox, W. Wu, A. Uyar, H. Bulut, and S. Pallickara, "Global multimedia collaboration system," *Concurrency and Computation: Practice & Experience,* vol. 16, pp. 441-447, 2004.
[45] G. Fox, W. Wu, A. Uyar, and H. Bulut, "A Web Services Framework for Collaboration and Audio/Videoconferencing," in *The 2002 International Multiconference in Computer Science and Computer Engineering, Internet Computing(IC'02).* vol. 2 Las Vegas, NV, June 2002, pp. 24-27.
[46] W. Wu, A. Uyar, H. Bulut, and G. Fox, "Integration of SIP VoIP and Messaging with the AccessGrid and H. 323 Systems," in *The 2003 International Conference on Web Services (ICWS'03)* Las Vegas, NV, USA, June 2003.
[47] W. Wu, H. Bulut, A. Uyar, and G. C. Fox, "Adapting H. 323 terminals in a service-oriented collaboration system," *IEEE Internet Computing,* vol. 9, pp. 43-50, July/August 2005.
[48] W. Wu, G. Fox, H. Bulut, A. Uyar, and H. Altay, "Design and Implementation of A Collaboration Web-services system," *Journal of Neural, Parallel & Scientific Computations,* vol. 12, pp. 391–406, 2004.
[49] W. Wu, H. Bulut, A. Uyar, and G. C. Fox, "A Web-Services Based Conference Control Framework for Heterogenous A/V Collaboration," in *7th IASTED International Conference on Internet and Multimedia Systems and Applications* Honolulu, Hawaii, USA, August 13-15, 2003, pp. 13-15.
[50] Anabas, Inc. eLearning and Collaboration, http://www.anabas.com.
[51] M. Handley, J. Crowcroft, C. Bormann, and J. Ott, "Very Large Conferences on the Internet: The Internet Multimedia Conferencing Architecture." vol. 31, 1999, pp. 191-204.
[52] C. Bormann, D. Kutscher, J. Ott, and D. Trossen, "Simple conference control protocol service specification," 2001.
[53] ITU Recommendation H.225, "Calling Signaling Protocols and Media Stream Packetization for Packet-based Multimedia Communication Systems," Feb., 2000.
[54] ITU Recommendation H.245, "Control Protocols for Multimedia Communication," Feb., 2000.
[55] ITU Recommendation H.243, "Terminal for low bit-rate multimedia communication," Feb., 1998.

[56] ITU Recommendation G.711, "Pulse Code Modulation (PCM) of Voice Frequencies," 1988.
[57] ITU Recommendation H.261, “Video Codec for Audiovisual Services at p x 64 kbit/s," 1991.
[58] ITU-T Recommendation H.263, “Video coding for low bit rate communication," 1998.
[59] ITU-T Recommendation T.120, “Data Protocols for Multimedia Conferencing," July 1996.
[60] P. Koskelainen, H. Schulzrinne, and X. Wu, "A SIP-based conference control framework," ACM Press New York, NY, USA, 2002, pp. 53-61.
[61] X. Wu, P. Koskelainen, H. Schulzrinne, and C. Chen, "Use SIP and SOAP for conference floor control," Internet Engineering Task Force, Feb. 2002.
[62] Implementation of the Globus Security Policy: v0.1 GLOBUS-SEC, http://archive.nsf-middleware.org/documentation/NMI-R1/0/GlobusToolkit/security/implementation.htm.
[63] R. Olson, "Certificate Management in AG 2.0," http://fl-cvs.mcs.anl.gov/viewcvs/viewcvs.cgi/AccessGrid/doc/, Ed., March 5, 2003.
[64] ITU-T Recommendation G.114, “One Way Transmission Time," 05/2003.
[65] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC2616: Hypertext Transfer Protocol--HTTP/1.1," RFC Editor United States, 1999.
[66] M. Handley and V. Jacobson, "SDP: Session Description Protocol," RFC 2327, April 1998, http://www.ietf.org/rfc/rfc2327.txt
[67] RealNetworks, "Using RTSP with Firewalls, Proxies, and Other Intermediary Network Devices," version 2.0/rev. 2, http://docs.real.com/docs/proxykit/rtspd.pdf, 1998.
[68] Microsoft, Windows Media Networking Protocol Kit, http://www.microsoft.com/windows/windowsmedia/licensing/netprokit.aspx.
[69] Microsoft, "Windows Media Format," http://msdn2.microsoft.com/en-us/library/aa387410.aspx.
[70] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar, "RTP Payload Format for MPEG1/MPEG2 Video," RFC 2250, Internet Engineering Task Force, Jan. 1998.
[71] Microsoft, "AVI File Format," http://msdn2.microsoft.com/en-us/library/ms779631.aspx.
[72] F. Nack and A. T. Lindsay, "Everything You Wanted to Know About MPEG-7: Part 1," *IEEE MultiMedia,* vol. 6, pp. 65-77, July 1999.
[73] F. Nack and A. Lindsay, "Everything You Wanted to Know About MPEG-7: Part 2," *IEEE Multimedia,* vol. 6, pp. 64-73, October 1999.
[74] J. M. Martinez, R. Koenen, and F. Pereira, "MPEG-7: the generic multimedia content description standard, part 1," *IEEE MultiMedia,* vol. 9, pp. 78-87, April-June 2002.
[75] MPEG, MPEG-7 Overview, http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.
[76] D. Deeths, "Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP. Sun BluePrints™ OnLine," 2001.

[77] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*: Prentice Hall PTR Upper Saddle River, NJ, USA, 2002.

[78] L. Lamport, "Time, clocks, and the ordering of events in a distributed system." vol. 21: ACM Press New York, NY, USA, 1978, pp. 558-565.

[79] F. Mattern, "Virtual time and global states of distributed systems," 1989, pp. 215–226.

[80] C. J. Fidge, "Logical time in distributed computing systems." vol. 24, 1991, pp. 28-33.

[81] F. Cristian, "Probabilistic clock synchronization." vol. 3: Springer, 1989, pp. 146-158.

[82] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by TEMPO inBerkeley UNIX 4.3 BSD." vol. 15, 1989, pp. 847-853.

[83] P. B. Danzig and S. Melvin, "High resolution timing with low resolution clocks and microsecond resolution timer for Sun workstations." vol. 24: ACM Press New York, NY, USA, 1990, pp. 23-26.

[84] P. Ramanathan, D. D. Kandlur, and K. G. Shin, "Hardware-assisted software clock synchronization for homogeneous distributed systems," *IEEE Trans. Computers,* vol. 39, pp. 514-524, April 1990.

[85] P. H. Dana, "Global Positioning System (GPS) Time Dissemination for Real-Time Applications," *Real-Time Systems Journal,* vol. 12, pp. 9-40, 1997.

[86] M. Horauer, U. Schmid, and K. Schossmaier, "NTI: A Network Time Interface M-Module for High-Accuracy Clock Synchronization," in *Proceedings of the 6th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, Orlando Florida USA, March 30 - April 3 1998, pp. 1067-1076.

[87] D. L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305 March, 1992.

[88] D. L. Mills, "Internet TimeSynchronization: the Network Time Protocol." vol. 39, 1991, pp. 1482-1493.

[89] D. Mills, "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," RFC 2030, October 1996.

[90] NTP: The Network Time Protocol, http://www.ntp.org/.

[91] K. P. Birman, "Replication and fault-tolerance in the ISIS system," in *Proceedings of the10th ACM Symposium on Operating Systems Principles*, 1985, pp. 79-86.

[92] R. Renesse, K. P. Birman, and S. Maffeis, "Horus: A Flexible Group Communication System," *Communications of the ACM,* vol. 39, pp. 76-83, April 1996.

[93] D. Dolev and D. Malki, "The Transis approach to high availability cluster communication," *Communications of the ACM,* vol. 39, pp. 64-70, 1996.

[94] K. P. Birman, R. van Renesse, and W. Vogels, "Spinglass: Secure and scalable communications tools for mission-critical computing," in *International Survivability Conference and Exposition, DARPA DISCEX-2001* CA, June 2001.

[95] P. T. Eugster, R. Boichat, R. Guerraoui, and J. Sventek, "Effective multicast programming in large scale distributed systems," *Concurrency and Computation: Practice and Experience,* vol. 13, pp. 421-447, April 2001.

[96] IBM, BEA Systems, Microsoft, and TIBCO Software, "Web Services Reliable Messaging Protocol (WS-ReliableMessaging)," ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200502.pdf 2005.
[97] A. Chervenak, B. Schwartzkopf, H. Stockinger, B. Tierney, E. Deelman, I. Foster, W. Hoschek, A. Iamnitchi, C. Kesselman, and M. Ripeanu, "Giggle: a framework for constructing scalable replica location services," in *Proceedings of ACM/IEEE Supercomputing, SC2002*, 2002, pp. 1-17.
[98] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker," in *Procs. of CASCON'98*, Toronto, Canada 1998.
[99] The NaradaBrokering Project , http://www.naradabrokering.org.
[100] S. Pallickara and G. Fox, "NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids," in *Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware*, 2003.
[101] S. Pallickara and G. Fox, "On the Matching Of Events in Distributed Brokering Systems," in *Proceedings of IEEE ITCC Conference on Information Technology*. vol. 2, April 2004, pp. 68-76.
[102] S. Pallickara and G. Fox, "A scheme for reliable delivery of events in distributed middleware systems," in *Proceedings of the IEEE International Conference on Autonomic Computing*, 2004, pp. 328-329.
[103] G. Fox, S. Lim, S. Pallickara, and M. Pierce, "Message-based cellular peer-to-peer grids: foundations for secure federation and autonomic services," *Journal of Future Generation Computer Systems,* vol. 21, pp. 401-415, March 2005.
[104] G. Fox, S. Pallickara, and S. Parastatidis, "Towards Flexible Messaging for SOAP-Based Services," in *Proceedings of the IEEE/ACM Supercomputing Conference*, Pittsburgh, PA, 2004.
[105] S. Pallickara, M. Pierce, G. Fox, Y. Yan, and Y. Huang, "A Security Framework for Distributed Brokering Systems," Available from http://www.naradabrokering.org.
[106] A. Uyar, "Scalable Grid Architecture for Video/Audio Conferencing," in *EECS Department of Syracuse University* Syracuse, NY: Syracuse University, Spring 2005.
[107] Helix DNA Server, https://helix-server.helixcommunity.org/
[108] W3C, "Synchronized Multimedia Integration Language (SMIL 2.0)," http://www.w3.org/TR/2005/REC-SMIL2-20050107/ 2001.
[109] Sun Microsystems, "Java Media Framework 2.1," in http://java.sun.com/products/java-media/jmf/2.1.1/index.html, 2000.
[110] Helix Community, https://helixcommunity.org/.
[111] D. Wisely, P. Eardley, L. Burness, and D. Wisely, *IP for 3 G: Networking Technologies for Mobile Communications*: John Wiley & Sons, 2002.
[112] P. J. Leach and R. Salz, "UUIDs and GUIDs," IETF Internet Draft. Feb, 1998.

Vitae

NAME:	Hasan Bulut
DATE OF BIRTH:	January 1, 1973
PLACE OF BIRTH:	Salihli, Manisa, TURKEY

DEGREES AWARDED:

May 2007	Ph.D. in Computer Science, Indiana University Bloomington, IN, U.S.A
May 2000	M.S. in Computer and Information Science Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, U.S.A.
June 1996	B.S. in Electronics and Telecommunication Engineering Istanbul Technical University Istanbul, TURKEY