Application-Layer Group Communication Server for Extending Reliable Multicast Protocols Services*

Ehab Al-Shaer Hussein Abdel-Wahab Kurt Maly

Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162
email: (ehab, wahab, maly)@cs.odu.edu

Abstract

Reliable multicast protocols are becoming an essential element in distributed applications such as interactive distance learning applications. However, the existing implementations of reliable multicast protocols are not sufficient to satisfy the group communications requirements of some distributed applications. Our experience of using number of multicast protocols in IRI distance learning applications shows the necessity of providing an application-layer Reliable Multicast Server (RMS) to extend the primitive group communication services provided by such multicast protocols. Examples of such services include handling heterogeneous environments (such as LAN vs. WAN networks and different reliable multicast protocols), automatic fault recovery and simple application interface. In this paper, we motivate and describe the design and the implementation of RMS architecture which has been used in IRI learning sessions for two semesters. We also show how RMS improves the reliability, performance and flexibility of IRI sessions via supporting extended group communication services.

KEYWORDS: Reliable Multicast Transport Protocols; Fault Recovery; Distributed Applications.

1. Introduction

With the recent advances in network and computing technology, the demand of group ware distributed applications is increasing. Examples of such application include distance learning[9], group ware editing [7], and video conferencing [10]. A reliable multicast service is essential for distributed applications in order to deliver messages to a group of members simultaneously. Examples of available implementations of reliable multicast protocols include [6] , [11], [13] and [14].

Integrating and using the existing protocols in real applications offer a valuable experience to the research community as well as the end users. We use reliable multicast protocols in the Interactive Remote Instruction (IRI[1]) [9] system which is a collaborative distributed multimedia system for distance learning. Our experience in integrating, tuning and using reliable multicast protocols identifies the following limitations of the existing implementations in supporting group communication for some distributed applications such as IRI:

- The lack of group recovery mechanism that may protect the applications from any bugs or run-time failures that may exit in the current reliable multicast implementations.
- The lack of accommodating different environments (such as LAN and WAN) simultaneously without a significant degradation in the performance. Different protocols show different performance in LAN and WAN environments [2].
- The lack of supporting special group communication services such as *selective re-transmission* in which messages are sent semi-reliably [8] and *dynamic group masking* which enables the sender to block some clients in the group from receiving certain messages.
- The lack of supporting a *transparent* multi-session group communication service where the client can join multiple groups simultaneously and without managing each group explicitly [14].
- Some protocols offer an *imperative* Application Programming Interface (API) which requires the appli-

*This work is partially supported by the National Science Foundation, Sun Microsystems and COX Fibernet.

[1] http://www.cs.odu.edu/~tele/iri

0-8186-8061-X/97 $10.00 © 1997 IEEE

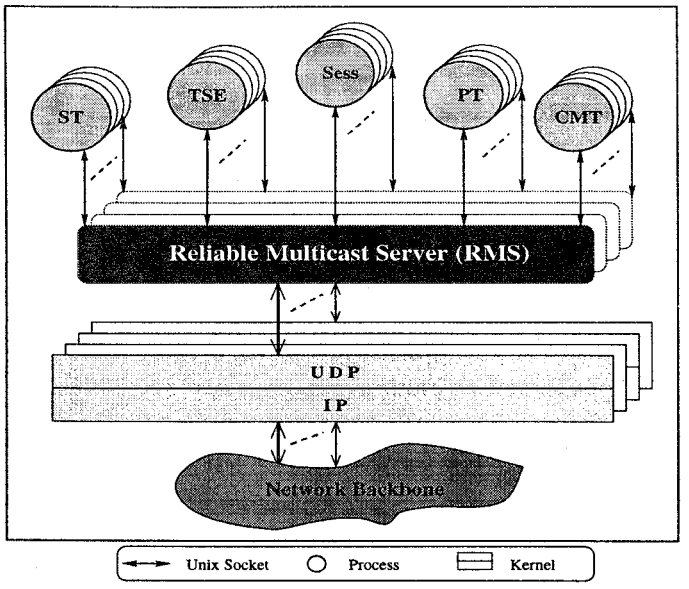

Figure 1. IRI Software Architecture

cation to be involved with protocol-specific operations [2].

- The lack of providing group and protocol information such as the group member list, the status of multicast groups and communication statistics (see Section 4).

For these reasons, we developed an application-layer *Reliable Multicast Server (RMS)* which acts as a service access point between the clients and the reliable multicast protocols to enhance and extend the group communication services provided by the existing multicast protocols. In this paper, we will motivate and describe the design and the implementation of the RMS which was developed to support the reliable multicast communication in IRI system. We also describe the RMS extended group communication services and their impact in improving the reliability, performance and flexibility of the group communication in IRI environment.

This paper is organized as follows: Section 2 describes IRI system and its software architecture; Section 3 discusses the design and the implementation of RMS in IRI system; Section 4 explains the extended group communication services supported by RMS and their contribution to improve the reliability, the performance and the flexibility of multicast sessions; and Section 5 presents the concluding remarks.

2. Overview of IRI System

The RMS was developed as part of IRI software architecture to provide reliable multicasting for IRI tools. However, RMS can, generally, be used in many distributed applications. In this section, we give a brief description of IRI system which is useful for our discussion throughout this paper.

IRI is a collaborative distributed multimedia system that has been developed at Old Dominion University to support distance learning [9]. A typical IRI session may include a teacher and tens to hundreds of students located in different classrooms and they all can participate in one single "virtual classroom" facilitated by IRI system. Each student, in the virtual classroom, is assigned a multimedia workstation that runs IRI software which consists of 9 processes that communicate with each other through UNIX socket messages [9]. Similarly, IRI applications in different machines communicate with each other through multicast messages.

The major software components of IRI are Session Control (Sess) and Reliable Multicast Server (RMS) which are used for resource management and group communication, respectively (See Figure 1). IRI provides full interaction via its Audio, Video, Presentation Tool (PT) and Tool Sharing Engine (TSE) [1] components. The PT is used for slide presentation and TSE is used to share X applications (e.g. Netscape) in the virtual classroom. IRI also provides Survey Tool (ST) for collecting feedback information from students and Class Management Tool (CMT) to diagnose the components status. The major components in IRI, which use RMS, are Sess, TSE, PT, ST and CMT (see Figure 1). Each component that requires a multicast service joins its own private multicast group [9].

3. Design and Implementation of RMS

The Reliable Multicast Server (RMS) is an application-layer process that acts as an interface server between the multicast clients and the reliable multicast protocol itself. RMS communicates with multicast clients by exchanging messages via UNIX sockets (UNIX-domain or TCP) [12]. Thus, group communication requests such as joining or leaving the group are sent to RMS which performs these requests by invoking the proper functions specific to the underlying reliable multicast protocols. Currently, in IRI sessions, RMS uses RMP version 1.3b to provide reliable multicasting. However, any other reliable multicast protocol can be integrated with RMS and without the client knowledge. In this section, we describe the design and the object-oriented implementation of RMS in IRI system.

The RMS Interface: RMS provides a simple application programming interface (API) for requesting group communication services which alleviates the difficulty inherited by using reliable multicast protocols API directly. RMS API provides three types of interfaces (the C++

API code is shown in [2]) *initialization, primary* and *miscellaneous* interfaces. The first one is used to initialize and activate the group communication service such as `ConnectToLocalRMS()` which is used to establish communication channels between clients and RMS. The primary interfaces are used to request the group communications services such as `SendJoinRequest()` to join the multicast group [2]. The miscellaneous interfaces are used for acquiring group and protocol information such as `SendMembersListRequest()` to get a list of group members and `SendConfigurationRequest()` to get the protocol configuration which are important for group management as described in Section 4.4.

RMS-Client Communication Protocol: To acquire group communication service, a client (e.g. an IRI component) has to connect to RMS through two socket connections called channels: Control Channel (CC) and Data Channel (DC). The CC is used to send control messages such as join request or membership status from the client to RMS. Also, the requests result code and replies are sent from RMS to the client over the CC. The DC is used to send and receive data to and from the multicast groups, respectively. RMS can serve two types of clients: *local clients* which reside in the same host of RMS and request group communication service via UNIX sockets connections, and *remote clients* which reside in remote hosts that does not have RMS and request the group communication services via TCP connections.

There are three advantages for using separate channels for control and data: (1) enabling sending the control information as out-of-band messages, (2) enabling the client and the RMS to communicate via CC, if a failure exists in the DC, and (3) avoiding encapsulating the data messages with headers which imposes an overhead in send and receive operations. As soon as the UNIX connection is established, the client (called IRIApp) starts sending the group communication requests. In [2], we show part of the client code (IRIApp) and its interaction with RMS.

RMS-RMS Communication Protocol: RMS can use any reliable multicast protocol to communicate with another RMS and thereby deliver the multicast messages to remote clients participating in the group. This is performed transparent to the RMS clients and users. Therefore, IRI components such as PT send and receive from their multicast groups with no concern of the underlying multicast protocols and its specific interface. In fact, the underlying multicast protocol had been upgraded and replaced with other protocols several times, in IRI, without notifying the clients or developers since the RMS-Client protocols and the API are preserved. Moreover, clients can use different underlying multicast protocols simultaneously utilizing the same

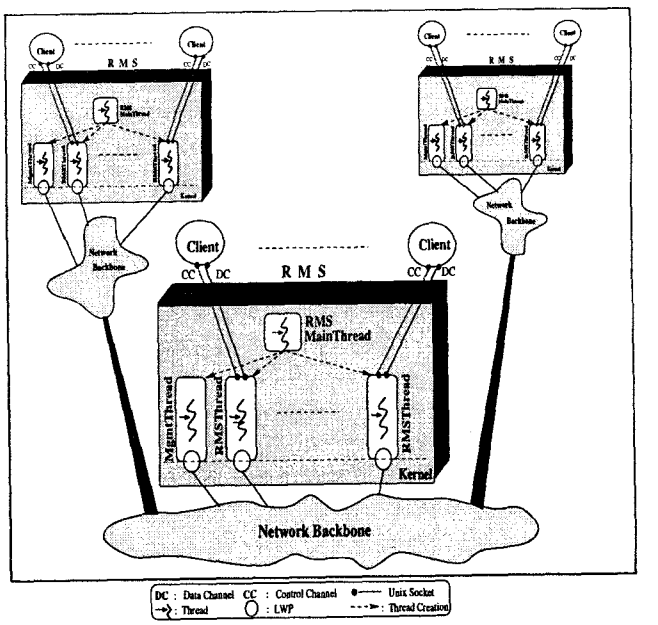

Figure 2. Architecture of the Reliable Multicast Server

RMS API (see Section 4.2).

RMS Objects: There are five objects used by RMS:

- *RMS* is the main server (MainThread in Figure 2) which receives the client connection and allocates a thread worker for each one [2].
- *MgmtThread* object is used for group management purposes such as collecting group information and sending group control requests to the multicast members. We will discuss the duty of MgmtThread in more details in Section 4.1.
- *RMSThread* object is used to serve the client and perform the protocol-specific operations such as joining the group, sending and receiving messages to and from the group, respectively (see Figure 2). RMSThread may represent any reliable multicast protocols integrated in RMS. The clients can have more than one RMSThread, however, in IRI sessions, there is typically one RMPThread per multicast client.
- *APIRMS* provides the application interface to interact with RMS. APIRMS is used by the client in order to request RMS services [2].
- *UnixSocket* object is used for UNIX-domain communications between RMS and its local clients.

Multithreaded Architecture: RMS is designed as a multithreaded application where each LWP (Light Weight Pro-

cess) or thread is completely independent from the others. This means no shared access exists and consequently no synchronization scheme is necessary, which is important to avoid any performance bottlenecks. The RMS MainThread starts by initializing the server via `RMS_Init()` [2] which creates the UNIX sockets files and the MgmtThread. Then, the RMS MainThread waits for the clients' connections through CC and DC. Clients such as IRI components connect to RMS through two UNIX socket connections (CC and DC), as described before. As soon as the UNIX connection is accepted by MainThread, RMS spawns a thread called *RMSThread* to serve the corresponding requester and it then goes back to listen to new connections ([2] shows the RMS main program). The RMSThread then establishes the control and the data connections and waits for clients requests. Figure 2 shows the design architecture of RMS in one machine. It also shows that one RMS can serve more than one client in the same machine by allocating RMPThread per client.

The RMSThread is created upon request and lasts for the duration of the requesting component. From this point, the multipoint communication activities between RMSThread and the IRI component go through the established channels. The multithreaded design of RMS has the following performance and reliability advantages: (1) it alleviates the overhead of managing multiple processes such as context switching that may degrade the performance, and (2) it isolates the RMP failures that may occur in an RMSThread from affecting other RMSThread(s) and their associated clients. Thus, even if one or more clients and their RMSThread have aborted or crashed, other clients can still using RMS safely. From our experience with the current version of RMP (RMP 1.3b), we found that RMP sometimes has run-time problems that could cease IRI session [2]. Thus, this feature is important to improve the reliability and the availability of IRI during the class session.

4. Extended Group Communication Services

The main objective of this work is to *extend* and *enhance* the services of the available implementations of reliable multicast protocols and thereby providing an efficient support for group communications. One of the major motivations of is its contribution to the existing protocol implementations which could be used in many other applications. Secondly, it provides important services without modifying the existing protocol implementation. Thirdly, it requires no involvement from the application itself in order to attain these services. In this section, we describe RMS extended group communications services which are not provided by the exiting implementations of reliable multicast protocols (RMP 1.3b, GSRM, RMTP and SCE). We will also illustrate the impact of these service on improving the reliability, the performance and the flexibility of the group communications in IRI session.

4.1. Group Communication Fault Recovery

Recovery from group communications failure is necessary in distributed applications specially in distance learning applications such as IRI [3, 4]. Faults result in isolating a client or group of clients from the rest of the group which cause a considerable disturbance to the session activity and dialog. In [3], we discussed some errors and failures that may be generated in group communications and terminate the entire IRI session. Providing means for efficient and rapid recovery is important in order to maintain the session with minimal destruction. RMS uses the following algorithm to support an automatic fault recovery mechanism in IRI group communication:

1. Each RMS join a special multicast group called *Management Group (MgmtGrp)*. MgmtGrp is served by an independent RMSThread, called MgmtThread, which consequently will not be effected by failures occur in other groups.
2. If RMSThread receives an indication of group failure:
   (a) The RMSThread deallocates the current object and re-allocates a new object of the reliable multicast protocol.
   (b) Then, the RMSThread informs the MgmtThread of this failure by:
      i. Depositing the failure information in a shared data structure that includes the failure type, the group name and the thread unique identification number.
      ii. Then, RMSThread sends a `SIGUSR1` UNIX signal to MgmtThread to indicate the failure occurrence.
   (c) The RMSThread re-joins the group name again, sets a *join-timer*[2] and waits for the group reformation to complete.
3. When the MgmtThread receives this signal, it will clear the failure entry from the data structure and multicast a notification of failure (GrpFailure message) to all MgmtThreads.
4. When the MgmtThread receives a notification of failure, it sends a `SIGUSER2` signal to the corresponding RMSThread assigned to this group that had the failure.
5. If the RMSThread receives a `SIGUSER2` signal, it leaves the group, allocates a new protocol object and re-join the same group again. It then wait for other RMSThread to join the group again.

[2] it is typically 15 seconds in IRI.

6. Whenever a new RMSThread joins the group, RMSThread members receives a list of all active clients in the group. If all RMSThreads (and clients) are successfully re-joined or the join-timer expires, the first RMSThread discovered the failure will send a multicast message to RMSThreads to resume working (sending and receiving) normally.
7. RMSThread will ignore any GrpFailure duplicates if they are received after the first one by time period t, such that $t \leq$ *join-timer* (seconds).
8. During the recovery time, RMSThread buffers the messages coming from the client. The RMSThread will request the client to pause and resume transmission via the CC if the buffers are full. If the group is reformed successfully, the buffered messages will be sent immediately.

Basically, the algorithm recovers the communication failures by reforming the group transparently without the clients involvement. In RMP version 1.3, the group communication failure are normally identified via an explicit event notifications (*Failure Detected*) from the protocol itself. However, some reliable multicast protocol may not convey an explicit message when a group failure occurs. For this reason, we use a *failure detection algorithm*, described in [2], for group diagnosis and fault detection. The group fault detection and recovery mechanisms improve the reliability and the robustness of IRI application [3].

4.2. Supporting Inter-protocol Multicast Communication

One of the major extended services that RMS contributes to group communications is offering a mechanism to bridge different reliable multicast protocols. Clients using different reliable multicast protocols can participate (send and receive) in one multicast group. In this case, the RMS act as a gateway between two or more protocols. This service is significantly important for the following reasons:
(1) It enables combining the features of different reliable multicast protocols into one multicast suite in the distributed application. Reliable multicast protocols have different design objectives and application domains. Therefore, combining different multicast protocols in one application could be useful to accommodate heterogeneous environments (LAN vs WAN environment).
(2) For large-scale applications, it is desirable to provide an open solution where users are not obligated to use one multicast protocol. So, it is possible to have different users using different reliable multicast protocols. (3) This service is useful for experimental purposes since it enables the developers as well as the users to compare and study the behavior of different reliable multicast protocols concurrently.

In IRI system, we chose to integrate RMP which is a token-ring based protocol and RMTP which is a tree based protocol [2]. There are two alternative RMS architectures to support inter-protocol communication: *centralized gateway* and *distributed proxies* architectures. The former gains simplicity but the later gains efficiency and flexibility. In the following, we describe both approaches:

Centralized Gateway Architecture: This architecture is typically used when each RMS has only one reliable multicast protocol but different RMS(s) in the application environment may have different protocols. This implies that clients of different RMS(s) may use various reliable multicast protocols to communication in one group. This can be achieved using the centralized gateway architecture shown in Figure 3. In this architecture, one machine acts as a gateway between different reliable multicast protocols. When the gateway starts, it creates a new thread (MgmtThread) to join a special group called management group (MgmtGrp). Every RMS member has a MgmtThread which joins the same MgmtGrp group. The MgmtGrp is used to exchange group information between the gateway and the MgmtThreads regardless what protocols their clients use. The gateway has a MgmtThread for each reliable multicast protocol used in the distributed application. For example, in IRI, there are one MgmtThread for RMP clients and another MgmtThread for RMTP clients (see Figure 3). If a client sends a *join* or *leave* request, the RMSThread performs join or leave operations and sends a multicast control message to the MgmtGrp in order to notify the gateway (as well as other MgmtThreads) of this action. The gateway, then, joins the groups that have members using different reliable multicast protocols and starts forwarding any received multicast message to the same group name through the RMSThread associated with the reliable multicast protocols (see Figure 3). The gateway stops forwarding multicast messages if no member uses different reliable multicast protocol in the group. If the gateway joins late and it does not have the group information, it requests this information from the other MgmtThreads in RMS. One MgmtThread will reply to avoid reply implosion as described in [2]. This request is repeated for every multicast protocol integrated in RMS. Similarly, if the gateway crashes, it will get the information from one MgmtThread of each protocol domain.

This architecture has the following limitations: (1) it involves one extra receive and send for each multicast message, (2) this architecture does not allow testing different protocols concurrently, and (3) it suffers a single-point of failure [2].

Distributed Proxies Architecture: This architecture is more efficient and flexible than the previous one. However, it requires integrating all different protocols which can po-

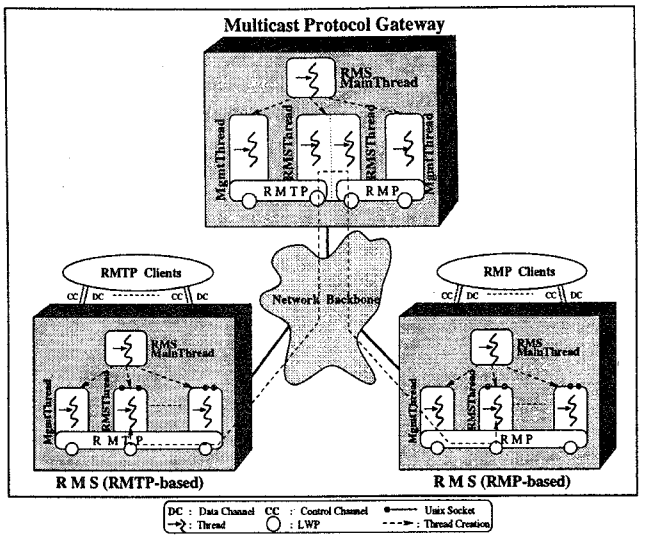

Figure 3. Centralized Gateway Architecture

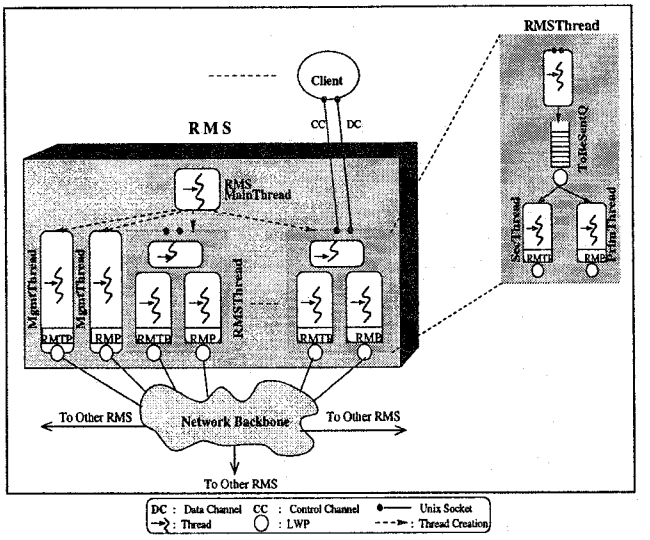

Figure 4. Distributed Proxies Architecture

tentially be used by the application in each RMS. This architecture enables the application to use a hybrid protocol configuration which could combine the features of various reliable multicast protocols. Thus, clients in the same application (e.g. LAN and WAN clients) may use a different multicast protocols in order to improve the overall communication performance.

Figure 4 shows that every RMSThread integrates both RMP and RMTP, for example. After the client performs `ConnectToLocalRMS()`, the RMS creates LWP thread called the primary thread or *PrimThread* that uses a primary multicast protocol and another LWP thread called a secondary thread or *SecThread* that uses the other reliable multicast protocol as shown in Figure 4. When a join request is sent by the client, the RMS performs join operation using both protocols. In particular, the PrimThread performs join operation and then it requests the SecThread to join the same group too. If join and leave operations are announced to the group by the underlying protocols, then RMSThread will know immediately if there exist member(s) that use different protocol. Otherwise, RMSThread uses the MgmtThread protocol as described before to distribute the group information [2]. In either case, each RMSThread knows if there exist clients which use different primary protocol than its own primary protocol in the same group. This information is important for sending data out to the group. When `SendData()` is invoked, the message is read from the UNIX socket and inserted in *ToBeSentQ* queue. This queue is checked by both PrimThread and SecThread simultaneously. If there exists a message in ToBeSentQ and there exist clients in the same group using different primary protocols, then both PrimThread and SecThread send the message to the group. Only PrimThread receives multicast messages and forwards them to the client over the UNIX socket. Thus, SecThread is used only for sending multicast messages to members in the same group using a different multicast protocol. It is important to notice that these protocol operations (joining, exchanging group information, sending and receiving) are completely transparent to the application clients and the users as well. Figure 4 shows the general design of this architecture. LWP threads are used to make the system calls in each thread independent from each other [2]. The length of ToBeSentQ can be adjusted to control the synchronization between different clients/protocols in the group [2].

This architecture is more efficient because it only involves one extra send for each multicast messages and the delay in the multicast path is negligible since each message is multicasted via independent LWP threads. It is flexible because (1) it enables users/developers to test and evaluate different multicast protocols concurrently, and (2) the client is allowed to use different multicast protocols simultaneously which is important to accommodate different environments such as LAN and WAN [2].

4.3. Extended Protocol Services

RMS is used to support new group communication services that are not defined in the reliable multicast protocol specification. In this section, we discuss two important reliable multicast protocol services which are supported by RMS, even though such services are not provided by the underlying protocol. These protocol services are:

Selective Re-transmission Mode: RMP version 1.3b does not support selective re-transmission mode (also called

```
void RMSThrReceiver(int RGSock, int UGSock)
{ /* ..... */
  CurrentSeq = LastSeq=0;
  FD_ZERO(&afds);
  FD_SET(RGSock, &afds); /* reliable group  */
  FD_SET(UGSock, &afds); /* unreliable group*/
  while(TRUE) {
    LastSeq = CurrentSeq;
    if (select(nfds,&rfds,(fd_set*)0,(fd_set*)0,
              (struct timeval *)0) < 0)
    {  perror("select"); exit(1);}
    if (FD_ISSET(RGSock, &rfds)) {
      ReceiveRGData(RGSock,MAX_SIZE,RelMsg);
      CurrentSeq = GetSeqNum(RelMsg);
      AddToReceiveQ(RelMsg); //put msg in Queue
    }
    else if (FD_ISSET(UGSock, &rfds)) {
      ReceiveURGData(UGSock,MAX_SIZE,UnRelMsg);
      CurrentSeq = GetSeqNum(UnRelMsg);
      if (CurrentSeq < LastSeq)
         continue; //ignore this one //  Case 1
      else if (CurrentSeq == LastSeq+1)
         AddToReceiveQ(UnRelMsg);     //  Case 2
      else if (GetRelSeqNum(UnRelMsg)<=LastSeq)
         AddToReceiveQ(UnRelMsg);      // Case 3
      else
         continue; // ignore this one // Case 4
    }
  } /* end of while */
} /* end of RMSThrReceiver() */
```

Figure 5. The Receiver Selective Re-transmission Algorithm

semi-reliable) which enables the client to send some messages reliably and others unreliably in the same stream. The selective re-transmission mode is significantly important in multicasting continuous streams such as video in distributed multimedia applications [8]. Using the RMS, the client can select the transmission mode (reliable or unreliable) for each message by setting the `QoS` parameter [5] in `SendData()` (0 means unreliable otherwise reliable). If the client requests the selective re-transmission service for a certain group, each RMSThread in this group establishes two different multicast groups for every client: *Reliable Group* (RG) using RMP and *Unreliable Group* (UG) using native IP multicasting. The RMSThread uses the RG to send messages designated as reliable and the UG to send other messages. Before sending, the RMSThreads assign a sequence number (RMS-sequence) for each message (reliable or unreliable).

In the receiver side, the RMSThreads uses the algorithm in Figure 5 to receive messages from the RG (RGSock) and UG (UGSock) groups. It is mandatory (1) to receive the message stream in same order it were sent, and (2) to deliver reliable messages to the client. In order to preserve these two properties, when the RMSThread receives a message via the UG, it checks its sequence number and discards the unreliable messages that has sequence number less than the last one received (Case 1 in Figure 5). On the other hand, if it has the next sequence number, the message will be delivered to the client immediately (Case 2). However, what if the sequence number of the UG-message is greater than the next sequence number (Case 3 and Case 4). To handle this case efficiently, the RMSThread puts also in the header of the UG-messages the sequence number of the last RG-message (reliable message) that has been sent (called SeqLRG-Msg). If SeqLRG-Msg found in the UG-message has been received by RMSThread, then the UG-message is delivered to the client (Case 3). Otherwise, the UG-message is discarded (Case 4). This algorithm has two main advantages: (1) it avoids infinite buffering of UG-messages, and (2) it avoids unnecessary dropping of UG-messages in the receiver which increases the throughput and the quality of the communication [2].

Rapid and Dynamic Sub-Group Formation: The ability to form transient sub-groups is a desirable service in many distributed applications. A member in a multicast group may desire to send "private" information to a subset of the group for a short time. For example, in IRI [9], the teacher may desire to have a short private communication (e.g. chatting) with one or group of students in the virtual classroom.

Forming new transient groups via joining and leaving may not be efficient because of the overhead involved in "join" and "leave" operations [2]. RMS provides a rapid and dynamic formation of sub-groups via filtering out the messages which do not belong to the receiver, based on a *masking identifier* included in the message header. The masking identifier is constructed by the sender prior to the sending process (`SendGroupMaskRequest()`) and consists of unique identifiers of all members in this sub-group who must receive this message [2]. A drawback of this approach is creating unnecessary traffic for non sub-group members. However, this could be acceptable for short lasting sub-groups where small volume of traffic is generated.

4.4. A Simple Programming Interface

Some reliable multicast protocols such as RMP has an imperative API interface. For example, in RMP 1.3b, the protocol event loop must be controlled from the the application event loop itself which makes the programming environment less manageable [2]. Also, the applications must be aware of some protocol-specific states such as "Flow-Control is Full" and behave accordingly to resolve this problem. The RMS provides *asynchronous*, *declarative* and *simple* programming interface (see RMS API in [2]). It is asynchronous since the applications using RMS API can send non-blocking requests. It is declarative since no protocols-

specific information is needed to request any communication service. It is also simple since the application event loop is independent from the protocol event loop. Moreover, RMS API enables the client to join multiple groups simultaneously and transparently (i.e. without managing each group explicitly) which is not the case in RMP version 1.3b [2].

The RMS API also provides new group communication requests which may not be supported by the underlying reliable multicast protocols. Examples of such requests include `SendMembersListRequest()` to request the group member information list and `SendStatisticsRequest()` to request some group communication statistics such as number of dropped, retransmitted and duplicated packets [2]. This information is useful for group management, problem diagnosis and performance tuning purposes.

5. Summary and Future Work

This paper motivates and describes our work of developing an application-layer Reliable Multicast Server (RMS) to enhance and extend the services provided by the current implementations of reliable multicast protocols. RMS supports the following extended group communication services which are not provided by the underlying multicast protocols:

- Providing an automatic recovery mechanism for group communication failures,
- Providing an architecture for inter-protocol communication between different reliable multicast protocols,
- Supporting rapid and dynamic sub-group multicasting,
- Supporting selective re-transmission mode, and
- Providing a simple declarative group communication programming interface that supports transparent multi-session communication and new group management requests.

These services are essential to improve the reliability, performance and flexibility of the group communications in distributed applications.

Our future work in this area includes conducting performance benchmarking experiments for the hybrid protocol configuration in LAN/WAN environment, developing an adaptive mechanism for supporting slow clients in multicast groups and designing a reliable multicast protocol for distance learning applications.

References

[1] H. Abdel-Wahab and K. Jeffay. Issues, Problems and Solutions in Sharing X Clients on Multiple Displays. *Journal of Internetworking Research & Experience*, pages 1–15, Vol. 5, No. 1, March 1994.

[2] Ehab Al-Shaer, Hussein Abdel-Wahab, and Kurt Maly. Application-Layer Group Communication Server for Extending Reliable Multicast Protocols Services. Technical report, Computer Science Department, Old Dominion University, August 1997.

[3] Ehab Al-Shaer, Alaa Youssef, Hussein Abdel-Wahab, Kurt Maly, , and C.Michael Overstreet. Reliability, Scalability and Robustness Issues in IRI. *WETICE'97:IEEE* 6^{th} *Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 1997.

[4] K. P. Birman and T. A. Joseph. Reliable Communication in the Presence of Failures. *ACM Transactions on Computer Systems*, 5(1), February 1987.

[5] D. Clark and D. L.Tennenhouse. Architectural Considerations for New Generation of Protocols. In *ACM SIGCOMM'90*, pages 200–208, Philadelphia, September 1990.

[6] S. Floyd, V. Jacobson, S. McCanne, C-G. Liu, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. pages 342–356, October 1995.

[7] Michael Koch. Design Issues and Model for a Distributed Multi-User Editor. *The International Journal of Computer Supported Cooperative Work*, 3(4):359–378, March 1995.

[8] A. Koifman and S. Zabele. RAMP: A Reliable Adaptive Multicast Protocol. In *Proceedings IEEE INFOCOM '96*, San Francisco, CA., March 1996.

[9] K. Maly, H. Abdel-Wahab, C. M. Overstreet, C. Wild, A. Gupta, A. Youssef, E. Stoica, and E. S. Al-Shaer. Interactive Distance Learning over Intranets. *Journal of IEEE Internet Computing*, pages 60–71, Feb. 1997.

[10] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. *ACM Multimedia*, pages 511–522, November 1995.

[11] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, April 1997.

[12] W. Richard Stevens. *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP and the UNIX Domain Protocols*. Addison-Wesley, Reading, Massachusetts, 1996.

[13] R. Talpade and M. Ammar. Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, Vancouver, BC, Canada, June 1995.

[14] B. Whetten, T. Montgomery, and S. Kaplan. A High Performance Totally Ordered Multicast Protocol. In *Theory and Practice in Distributed Systems*, Springer, Verlag LCNS 938, 1994.