J. eng. & appl. sci. Vol. 29 No. 2 July - December 2010 *ISSN 1023-862X*

RELIABLE MULTICAST IMPLEMENTATION IN JAVA

S.W. Shah[*], M.I. Babar[*], L. Khan[**], M.N. Arbab[*], H. Ullah[***] and R.A. Syed[***]

ABSTRACT

This paper describes the implementation of reliable group communication in Java. The underlying delivery mechanism for multicast is presently based on User Datagram Protocol (UDP) that provides a “best effort” delivery service. Best effort implies that IP packets are treated with essentially equal weight, and while IP makes an effort to deliver all packets to their destination, packets may occasionally be delayed, lost, duplicated, or delivered out of order. One of multicast’s weaknesses is its lack of reliability due to its use of UDP for data transmission. Reliable transmission means that there should be no packet loss, no disordering and no duplication of packets at the receiver side. The focus of this paper is to implement Reliable Multicast using Java, which is mainly used for one-to-many connections. This work focuses on reliable multicast in a local area network (LAN) environment. The reliability has been introduced at application layer and is receiver’s initiated, NACK (negative acknowledgement) based.

Key words: *Java, Multicast, MulticastSocket, Reliable Multicast, Ggroup Communication.*

INTRODUCTION

There are various ways to deliver data in a network which basically include:

Unicast Data can be delivered to one specific destination with unique IP address, this means only one sender sends data and another receives at a time.

Broadcast Data can be delivered to all hosts present in the network whether interested or not.

Multicast Data can be delivered to a group of interested users across the network.

Most of the traffic generated in the internet is unicast. To deliver a message to a particular group of users, unicast is inefficient because it consumes more bandwidth and will flood the network subsequently resulting in cogestion in the network. Broadcast is efficient than unicast in terms of network traffic problem because sender generates only one data stream to be delivered to all hosts whether interested or not, but its drawback is that it floods the entire network. While in multicast type of applications, e.g., videoconferencing and teleconferencing, instead of sending a separate copy to each of the receivers, the sender sends to the network a single copy, the network then sends it to all the receivers so it provides the best features as compared to unicast and broadcast because it neither carries the burden to deliver the data to all hosts separately nor it floods the entire network.

In unicast, there is one receiver and one sender so communication is easy, e.g., the File Tranfer Protocol (FTP) for file tranfer and the Hypertext Transfer Protocol (HTTP) for web access are unicast applications using TCP that is a unicast (point - to - point) protocol[1]. Whereas in multicast we deal with the group of users/processes which poses a challenging task to reliably communicate with the group of users/processes. Reliability means no loss, no disordering and no duplication of packet. Reliability mechanism is the best option for text data delivery. However, for real time applications like video conferencing, VoiceOverIP (VOIP), online games, etc, would be a bottle neck because such applications cannot afford delay at any cost. IP Class D addresses are used for multicast services only.

The idea of our work is to design a protocol to reliably multicast data to a group of users in LAN environment. We are striving for protocol which avoids packet loss, packet duplication, and delivers packets in order. Our final goal is to develop a protocol to be implemented in Java which helps in realization of reliable multicast in LAN environment.

[*] *Department of Electrical Engineering, University of Engineering and Technology, Peshawar, Pakistan*

[**] *COMSATS Institute of Information Technology, Abbottabad*

[***] *Department of Mechanical Engineering, University of Engineering and Technology, Peshawar, Pakistan*

BACKGROUND AND PREVIOUS WORK

A lot of work has been done in the area of data communication and still it is an active area of research. For reliable multicasting, techniques at different layers of TCP/IP model[2] have been proposed that include physical layer, network layer and application layer[3,4,5,6]. In Java API, there is only one class for multicasting i.e. *MulticastSocket*[7,8]. As this class extends *DatagramSocket*[8] which is UDP based, so it is unreliable. Until now there is no facility of reliable multicast data delivery available in Java. Various packages have been proposed, but none of them is included in the standard library of Java until now.

PROBLEM WITH IP MULTICAST AND ITS SOLUTION

The main problem with current IP based multicasting technique[9] is that it is unreliable and does not guarantee the data delivery to each member of the multicast group. Our work is focusing on developing a protocol with introduction of reliability where each member of the group will receive the data multicast to the group. This reliability has been achieved at application layer of TCP/IP model and is receiver initiated, NACK based.

We consider a distributed system that consists of static processes, locations and communication channels. A location is a logical place that provides an execution environment for the processes communicating via message exchange through communication channels.

Here we denote the set of all possible locations by 'L', i.e., L={l_1, l_2, l_3,........,l_n} and a process group by 'g' such that g={p_1, p_2, ...,p_n}. For a given process group g, the protocol performs the following four fundamental operations:

join (g, p_i) : issued by a process p_i for joing the group g;

leave (g, p_i) : issued by a process p_i for leaving the group g;

send (g, P_i, m) : issued by a process p_i in order to multicast a message m to the members of gorup g.

send (p_s, p_r, s) : issued by a receiving process p_r in the group g to send a string s wtih a NACK to a sending process p_s in wake of failure of message delivery. The sending process p_s may or may not be a member of the process group g.

Using Java programming language we have implemented protocol that ensures the data delivery reliably. The implementation consists of two main classes and two supporting classes. The main classes are ReliableMulticastServer (sender) and a ReliableMulticastClient (receiver), which communicates simple strings of text with a NACK (negative acknowledgement) implementation. In order to achieve reliability, we break the message (information) to be transmitted into small chunks or packets and give sequence numbers to them. The message segments (chunks) are stored in a buffer, which is a dynamic data structure like Linked List because if a packet is lost or corrupted during transmission, the receiver sends back the NACK for missing packet and the sender retransmits the lost packet only.

In addition to the main classes, we have two auxiliary/ supporting classes, *ReliableMulticast* and *MulticastPacket*. The job of *ReliableMulticastServer* is to get data from the user, disseminate data, add metadata to it and then at the end multicast it to the group.

The job of *ReliableMulticastClient* is to receive packet from the group and check for ordered delivery and deliver it to the end user. It is also the job of *ReliableMulticastClient* to send NACK back to the *ReliableMulticastServer* for missing packet. *ReliableMulticastClient* accepts up to three disorder packets and stores it to avoid packet duplication. If the missing packet is still absent then *ReliableMulticastClient* sends NACK for the missing packet to *ReliableMulticastServer*.

The auxiliary class *MulticastPacket* is the complex data structure. It contains field for message and required metadata.

The other auxiliary class is *ReliableMulticast* which contains methods for serializing the *MulticastPacket* sent and deserializing the *MulticastPacket* received on *MulticastSocket*.

OPERATION OF PROTOCOL

Whenever the user has data to send to the multicast group, it creates a *ReliableMulticastServer* instance as shown in Figure 1. It then calls *getOutputStream( )* method of this class which returns an output stream for user to put data in it. *ReliableMulticastServer* in turn receives the data from user, calls the *wrap( )* method of *ReliableMulticast* class to add metadata and serialize it. The *wrap( )* method returns serialized byte array of message and its metadata which is put in the *DatagramPacket* by *ReliableMulticastServer* and then multicasts on *MulticastSocket*. It also stores a copy of the sent message in a dynamic data structure i.e. Link List.

The *ReliableMulticastClient* receives the byte array from *MulticastSocket* and put it in *DatagramPacket* as shown in Figure 2. It then calls *unwrap( )* method of *ReliableMulticast* class to deserialize it into *MulticastPacket*. It then checks the *MulticastPacket* for ordered delivery. If it is in order, the *ReliableMulticastClient* extracts the message from *MulticastPacket* and delivers it to the end user. If it is not in order, then it stores it in buffer and waits for the specified number of disorder packets allowed. If specified number of disorder packets allowed limit is reached, then it sends NACK for the missing packet to the *ReliableMulticastServer*. In order to limit buffer in size on *ReliableMulticastServer side*, the *ReliableMulticastServer* sends check frame on *MulticastSocket* when buffer reaches its maximum size. In response to this, every *ReliableMulticastClient* sends its message number, for which it is waiting, back to the *ReliableMulticastServer*. The *ReliableMulticastServer* waits for specified amount of time (with Round Trip Time-RTT- equal to 1 second) and then cleans the buffer up to least message number received.

Table 1: Network Traffic vs. Multicast Clients

S. No	No. of Clients in Multicast Group	Unicast Traffic (Kbps)	Multicast Traffic (Kbps)
1.	3	198	66
2.	4	200	50
3.	5	215	43
4.	6	480	80
5.	7	455	65
6.	8	560	70
7.	9	675	75

PERFORMANCE ANALYSIS

An experiment was performed with the protocol that was implemented in java in parallel with unicast mode of communication. The experiment had seven parts and each part involved different number of receiving clients in multicast group. Different numbers of receivers having Windows installed were involved during each part of the experiment. Each part was run in two different modes, unicast and multicast, for 45 seconds each, in order to analyze the performance of different techniques. Network traffic was measured in each mode of each part as shown in Table 1 and to compare the performance in each part a graph was drawn as shown in Figure 3. It is clear from the graph that unicast traffic directly depends on the number of hosts in the network, while the multicast traffic is independent of the group size.

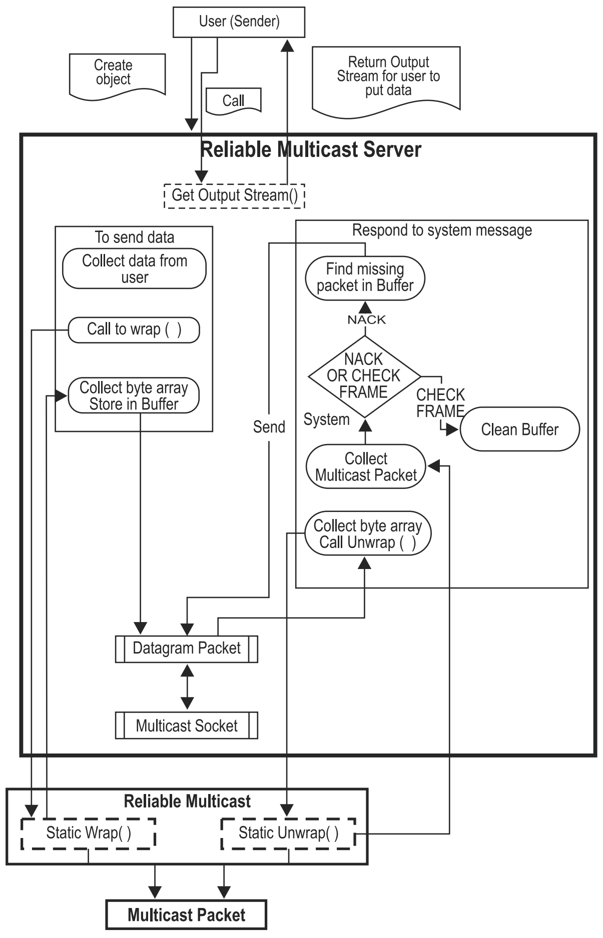

Figure 1: Server Operation

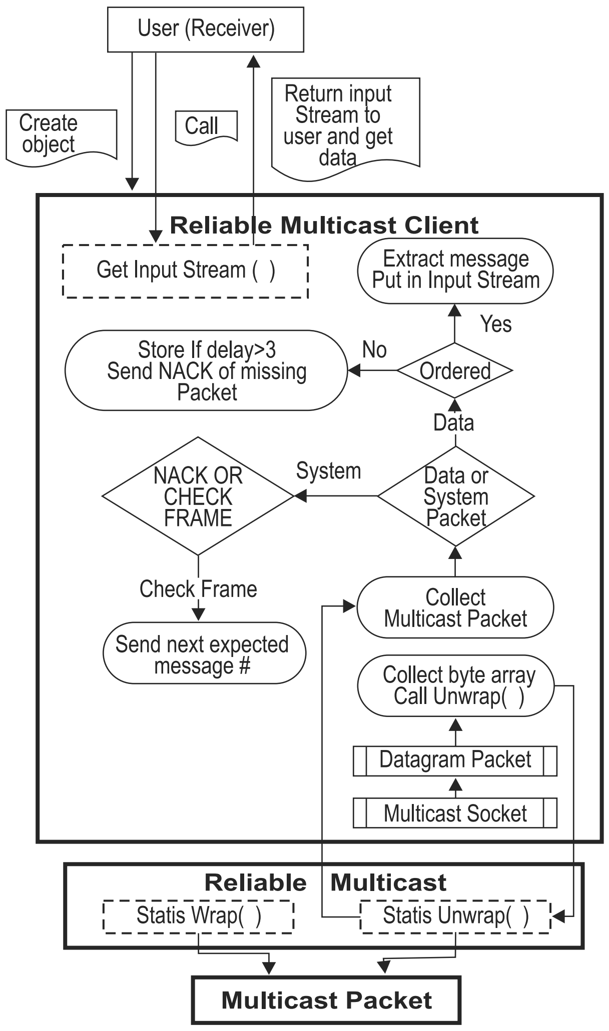

Figure 2: Client Operation

CONCLUSION

The experiments performed with the arrangement in LAN environment show that multicast saves a lot of bandwidth which is a scarce resource across the internet. With bandwidth saving, other applications can also send their data across the network.

FUTURE WORK

- NACKs received from multiple users simultaneously by the server will be tackled in future. This is a scaling issue related to the size of a multicast group. As the number of receivers increases, the amount of NACKs to the sender subsequently overwhelms its capacity to process them[9]. Further, the NACK messages from the receivers congest the network on the sender site.
- More sophisticated protocols are required for buffer cleaning, that is, if one client sends check frame reply having number less than other, then other clients should not send check frame response.
- The client must be intelligent if the NACK is sent by another on *MulticastSocke*. It should avoid sending this duplicate NACK, thus saving *ReliableMulticastServer* from NACK implosion.
- The number of group D addresses is limited and will limit the number of multicast applications, but this problem is solved in IPv6.

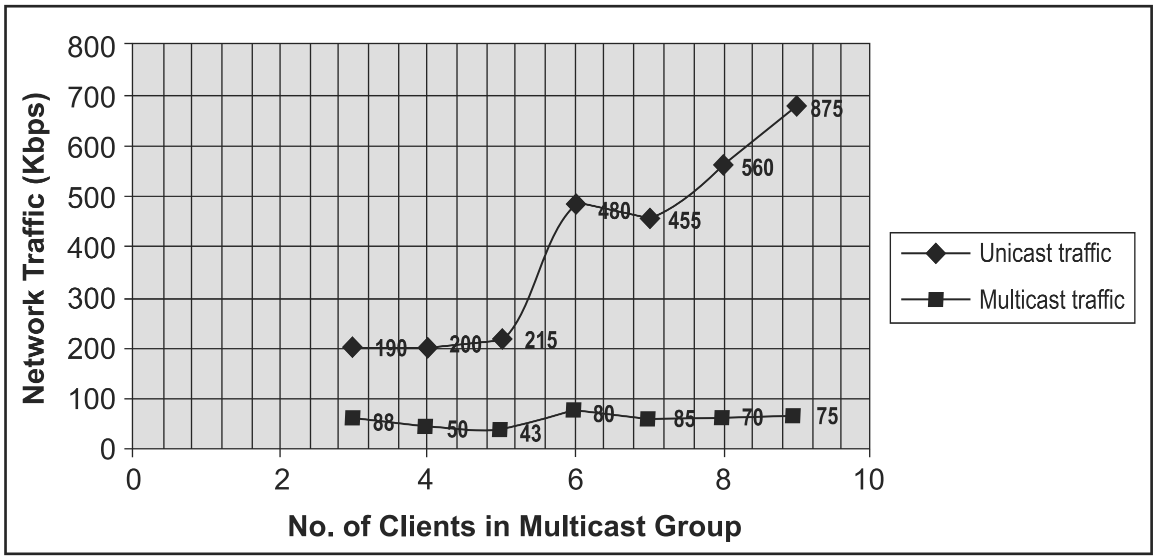

Figure 3: Network Traffic vs. Multicast Clients

REFERENCES

1. Kenneth Miller, C. StarBurst Communications: Reliable Multica Protocols and Applications, The Internet Protocol Journal - Vol. 1, (2).
2. Behrouz A Forouzan, 2002. TCP/IP Protocol Suite, 2nd edition, McGraw-Hill.
3. Ganjam A. and Zhang H., 2005. Internet Multicast Video Delivery, Proceedings of the IEEE, Vol. 93, (1).
4. Lao L., Cui J-H., Gerla M. and Maggiorini D., 2005. A Comparative Study of Multicast Protocols: Top, Bottom, or In the Middle?, IEEE Global Internet Symposium (GI 2005), Miami, FL, USA.
5. Matrawi A. and Lambadaris I., 2003. A Survey of Congestion Control Schemes for Multicast Video Applications, IEEE Communications Surveys and Tutorials, fourth quarter 2003, Vol. 5, No. 2.
6. Adrian Popescu, Doru, 2007. Constantinescu, David Erman, Dragos Ilie: A Survey of Reliable Multicast Communication pp. in proc. 3rd Euro-FGI Workshop on Next Generation Internet Networks NGI, Trondheim, Norway.
7. Sun Microsystems, Inc. http://java.sun.com/javase/reference/ api.jsp.
8. Elliotte Rusty Harold, 2005. Java Network Programming, 3rd Edition, O'Reilly.
9. Michael Henderson, Adam Ryan, Dr. Haibin Lu, Dr. Wenjun Zeng: Efficient reliable IP multicasting, http://www.cs.missouri.edu/~reu/REU07/IP%20Multicast/program_files/Project%20Report.pdf.