B.Tech. Project Report

A Reliable Multicast Framework for Applications

Raja Mukhopadhyay and **Vikas Gupta**
Department of Computer Science & Engineering
Indian Institute of Technology
Kanpur, INDIA

under the guidance of
Dr D. Manjunath and **Dr Dheeraj Sanghi**

August 1995 - April 1996

submitted in partial fulfillment of
the requirements of obtaining the degree of
Bachelor of Technology
in
Computer Science & Engineering

Certificate

This is to certify that this project entitled **A Reliable Multicast Framework for Applications** has been carried out under our supervision and, to the best of our knowledge, has not been submitted elsewhere as part of the process of obtaining a degree.

April 9, 1996 D. Manjunath Dheeraj Sanghi

Abstract

Our work involves developing a reliable multicast framework that can scale well to both very large networks and very large sessions. The scalability and the efficiency of the framework is estimated by using it as the backbone of 'WhiteBoard', a distributed teleseminaring tool which maintains a shared window supporting graphics and text. In order to avoid the associated overhead, the framework does not guarantee that packets will be delivered in any particular order. Through 'WhiteBoard', we show how applications can easily extend the framework to enforce any particular order which they might need to satisfy their quality of service requirements.

Acknowledgements

First and foremost, we would like to thank our guides, Dr Manjunath and Dr Sanghi, for having suggested the topic of our project and for their constant support and guidance, without which we would not have been able to attempt this project.

Thanks to Atul Jain and Sandhya Sule for being a constant help throughout the project. Thanks are also due to Suvrat Gupta for helping out with some of the protocol issues. We thank the Telematics Lab. for providing the necessary hardware and software for developing Whiteboard. We also thank Microsoft Inc. for providing Winsock 2.0 which made our project possible, and also being one of the most frustrating reasons of our numerous nights out.

Raja Mukhopadhyay Vikas Gupta

Contents

Chapter 1

Introduction

With the emergence of networks providing huge bandwidths and the availability of increased processing power at the desktop, multimedia applications no longer remain a distant dream. Of these multimedia applications, perhaps the highest demand is for the conferencing applications which allow a group to coordinate its work on a real-time basis, irrespective of the physical location of the members. Moreover, given the increasing trend toward distributing work and getting it done on a collaborative basis in industry and elsewhere, the need for such conferencing applications can only rise in the future.

However, the technology needed for these conferencing applications is still maturing. The major reason for this is that multimedia communication has given birth to a host of new problems that are still being looked into. Moreover, conferencing applications entail a lot of group dynamics and the traditional method of having point to point connections between hosts does not scale to these situations.

The switch to multicast communication - a communication method in which the group is treated as an immutable entity with all information being addressed to it rather than to the individual members - therefore becomes unavoidable. The problem is that most of the methods developed for unicast communication do not carry over to the multicast case.

In unicast communication, the requirements for reliable, sequenced delivery are fairly general. Therefore, it has been possible to come up with schemes that satisfy these needs for the whole spectrum of unicast applications. However, different multicast applications have have widely varying ordering and reliability requirements. Although generic multicast protocols that meet the worst-case requirements, *viz.* complete reliability and total order, have been reported in the literature, the overhead imposed by these protocols on applications with more modest requirements is substantial. This precludes the design of a single multicast delivery scheme that can simultaneously meet the functionality and efficiency requirements of all applications.

The important thing is to realise is that in the case of multicast communication, the best we can achieve is to have a framework which provides minimal functionality so far as reliability, ordering and latency are concerned. Different applications can then add structure to this framework to satisfy their quality of service requirements. Our work concerns the design and development of a multicast framework with the aforementioned properties. We also use this framework as the backbone for Whiteboard - a multicast application providing a shared window which supports the exchange of graphics and text. We have also looked into the issues involved in providing Whiteboard with audio and video capability so as to have a full-blown multimedia conferencing application.

Chapter 2

Background and Motivation

Talking of generic reliable multicast protocols, much as TCP is a generic reliable unicast transport protocol, several approaches have appeared in literature. It has long been recognized that protocols that achieve the reliability, scalability and efficiency requirements of all applications cannot possibly be designed. An approach for Application Level Framing (ALF) was proposed by Clark and Tennenhouse [CT90] which explicitly includes the application's semantics in the design of that application's protocol. Extensions of ALF involved light-weight rendezvous mechanisms based on IP group delivery models, and included a notion of receiver based adaptation for unreliable, real-time applications, such as video conferencing. This resulted in development of Light Weight Sessions(LWS), which have been very successful in the development of scalable wide-area conferencing applications.

ALF suggests that to achieve maximum flexibility, most of the functionality should be left to the application. On these lines, Van Jacobson *et al* [JAC95] developed a framework for scalable reliable multicast (SRM). The functionality provided by their framework is the eventual delivery of all data to all group members, without enforcing any particular order. Their framework has been prototyped in 'wb', a distributed teleseminaring application which maintains a shared window supporting graphics and text. This application, which runs under X-Windows, has become popular as 'whiteboard'.

Our framework borrows heavily from the SRM framework. Following ALF and SRM, we achieve a framework that achieves eventual delivery of all data to the entire group. The requirements of Whiteboard allowed us to pursue a design devoid of global ordering per se. Since Whiteboard was not destined to be a complete video conferencing package in itself, the requirements on ordering and real-time delivery were not very stringent. The design however still permits an application to use this framework and incorporate its own ordering on the packets exchanged in the group. Thus Whiteboard, while being able to operate on a stand-alone basis, can be extended to support multimedia and

real-time aspects of video conferencing.

One of the major motivating factors for Whiteboard was that similar applications do not exist on the most widely used platforms - the PCs. The 'wb' by Van Jacobson *et al* provided the application we were aiming at, but on X-Windows. Our implementation of 'WhiteBoard' is for the MS-Windows platform. We expect a greater reachability and use of Whiteboard on this platform simply because of its immense presence all over the world. We eyed Whiteboard as a product that can reach out to a lot of people, and this encouraged us throughout the project.

Chapter 3

Framework for Whiteboard

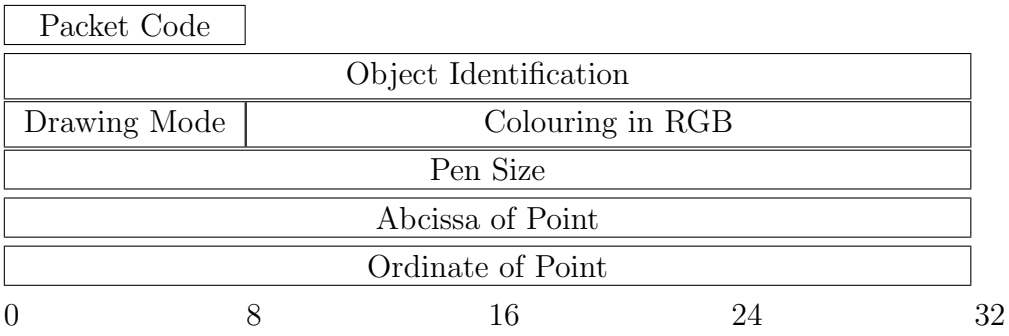
3.1 Building the Skeletal Application

To begin with, we have developed a rudimentary whiteboard which supports only graphics. To concentrate on the design of the application, rather than on the delivery scheme, we used TCP as the underlying transport protocol.

The communication model comprises a single server and multiple clients. A client communicates with the group by passing the message to the server which then 'relays' it to other members of the group. The packets are sequenced in the order in which they are received at the server and thus a total order is enforced.

The design is object-oriented and consists of classes like 'server', 'client', 'shape', 'frame' and 'socket'. The 'shape' class is further subdivided into classes like 'straight line', 'rectangle', 'ellipse' and 'curved line'. The application is structured in a layered manner : the Shape Layer, the Frame Layer and the ByteStream Layer. The Shape Layer interprets data in terms of 'shapes' and maintains the objects displayed in the window. The Frame Layer is concerned with the generation of application protocol frames and communicates with the peer layer to maintain a session. The ByteStream Layer operates at the socket level and interprets data in terms of bytes. Details on the implementations are provided in Chapter 5.

The application level protocol is used by the clients and the server to force a consistent interpretation of data. To ease the understanding of the protocol, specification of the object Shape is in order. A Shape can be thought of as an object with the attributes *Type*, *Identification*, *Pen Size*, *Colour* and *Position on Screen*. The protocol interface to the Shape Layer is in terms of objects with the above attributes. Following are the packet formats used by the protocol.



The protocol makes sure that each object drawn on each client window is communicated to all other client windows and is given a unique representation in the system. When a client starts a new object, it sends a request to the server for assigning a new object id. The server assigns the object a unique identification contained in the *Object Identification* field of the frames relayed to all clients regarding the start of the object. Subsequently, the originator may use this object id to add points to the shape. The other fields in the protocol frames carry information about the attributes of the shape.

3.2 Shifting to Multicast

After designing the application level skeleton for 'WhiteBoard', we changed the underlying delivery scheme from unicast to multicast. This necessitated a major change in the communication model. Whereas in the unicast case, the clients communicated via the server, this was not possible under multicasting where the model was necessarily serverless. Although we could have settled for a model in which one of the group members acted as the server, we did not do so because ensuring robustness would have involved substantial overhead.

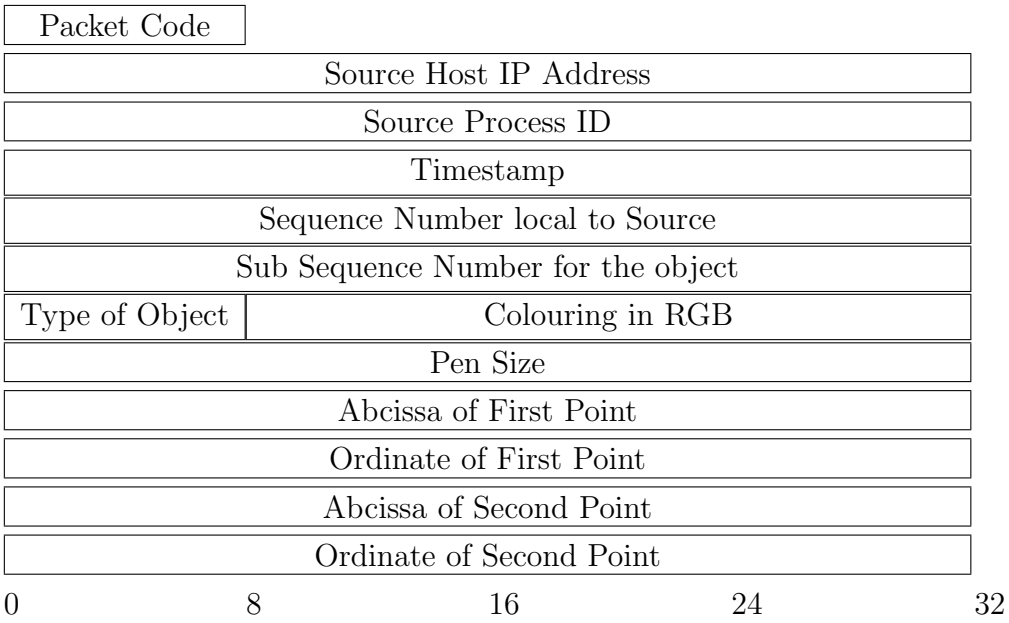
Our communication model follows the IP group delivery model which is the centrepiece of the IP multicast protocol. In IP multicast, corresponding to every active session, there is a group address. Data sources simply write to the group address and receivers gather the data by listening to the group address. Any individual member does not need to know the number of active senders at any instant or the IP addresses of other members. Further, any member can join or leave the group at any time without affecting the communication among the other members. Our model adds functionality to the IP model to ensure that the shared window is consistent among the members and that members exercise some kind of ownership over the data they create.

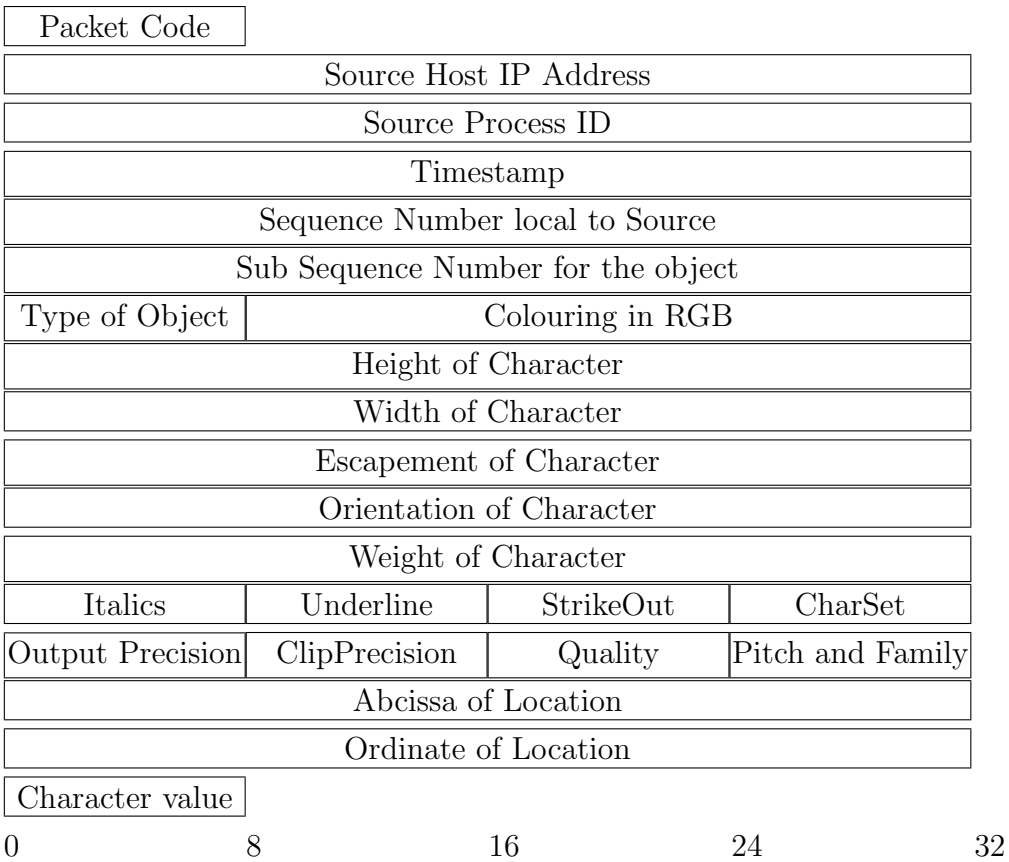
The move to multicast brought about some changes to the application level protocol. The ByteStream Layer had to be modified to work on top of UDP and use multicasting for communication. The use of UDP as the underlying transport protocol was necessary

because the IP group delivery model works only on top of UDP. Since we no more had a client-server mode of communication, no cenralized ordering of the objects was possible. This translated into a complete upheaval of the protocol and packet formats.

Since the IP group delivery model works on top of UDP, the protcol assumes lossy, connectionless services from the underlying network. Packets are multicast to the group whenever an active agent puts up a shape on his window. At the same time, all agents keep on listening to their 'group socket' and read up the data whenever it is available. Whenever, an agent detects the loss of a packet, it sends a request for the same to the whole group. Each host also calculates its 'time-distance' from all other active hosts. This is achieved by interfacing with Network Time Protocol (NTP) which provides for a globally synchronized time over the network. The host calculates the time-distance value of another host by taking the difference between the time of receipt of the packet and the timestamp marked on the packet. Corresponding to each request for a retransmission, each host starts a timer whose expiry value is proportional to the calculated time-distance value. Upon expiry of the timer, the host sends over the requested information to the group. Other hosts on receiving this information suppress their own retransmissions. This scheme ensures that only one host, the 'closest' one, retransmits and thus flooding of the network with duplicate information is avoided.

The packets are of fixed size (64 bytes) which makes the task of writing them onto and reading them from the network easier. Further, most of the packets contain data and control information that add upto almost 64 bytes, so making this choice does not waste too much bandwidth either. Whiteboard provides support for two forms of information - graphics and text, there is a separate packet format for each. These two packet formats are shown below.





Each user level object is uniquely identified by the user's Host IP address, his process id and a sequence number local to the user. The packets pertaining to the same object are distinguished and ordered by the subsequence numbers. The *Timestamp* field is used

to calculate delays, round trip time estimates and values for retransmission timeouts.

The construction of objects is controlled by passing some control information with the packets. In the case of a straight line, a rectangle or an ellipse, since the object needs to be displayed only when its position has been entirely decided, we need a single packet with a corresponding control code which has the two relevant points. However, in the case of a curved line, we would like to ensure that the process of drawing be also visible on all windows. To achieve this, we have a control code denoting the start of a curved line. The points comprising the curved line are tagged with subsequence numbers so that they can be indexed properly even if they arrive out of order. Finally, after the last point has been sent, a special packet denoting the end of the curved line is multicast to the group. Text is passed in packets on a per character basis along with the relevant font information.

We note that since we had developed the system in an object-oriented, layered manner, changes to the protocol affected only the Frame and ByteStream Layers.

Chapter 4

Whiteboard Design Issues

4.1 Providing Concurrency

One major problem encountered in the development of Whiteboard was to provide for concurrent update in any part of the window. Any method of overcoming the concurrency problem must necessarily identify the packets uniquely and provide for queueing of packets so that the objects are displayed onto the window at the appropriate time. Major hurdles incorporating this over multicast were the reflected packet problem and the simultaneous write problem. Due to the asynchronous manner of communication, each packet sent to the network was immediately received back. This led to shift of control to another message handler. This message handler then identifies the packets to be its own, and drops them silently and quickly. A similar problem appeared when two hosts try to write simultaneously. At the uppermost layer, this implies the presence of more than one Graphics device context at the same time. For each received message, Whiteboard allocates a device context, updates the window using the message and deallocates the device context. For the local drawing primitives, however, a device context is maintained throughout the creation of the shape.

Whiteboard uses the source IP address and the source process ID to uniquely identify a packet. The incoming packets are demultiplexed on the basis of these two parameters and attached to a list which we will refer to as the Host List. Each node on the host list has a source IP address, a source process ID and a pointer to a list of shapes which we will refer to as the Shape List. The Shape List serves as a container for the objects created by the particular host. Each node on this list contains the local sequence number for the object, the highest received subsequence number(only in the case of curved line) and a pointer to the actual shape. Once the membership of the incoming packet has been decided for the Host List, the Shape List is traversed to find a match for the corresponding sequence number. Thus an incoming packet is attached to its intended

object which is displayed when notification is received about its completion.

4.2 Achieving Local Ordering

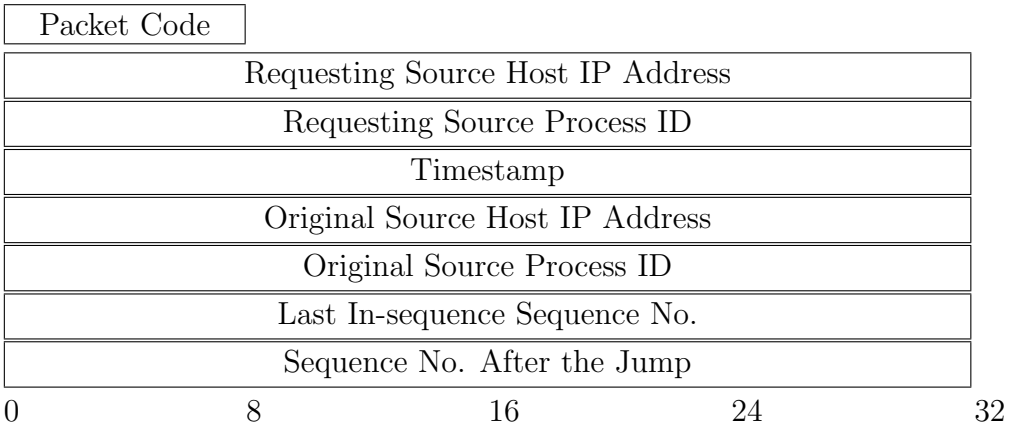
The ordering requirements for Whiteboard are not very stringent. We only require that the windows of different users be consistent more or less, it does not hurt if the order of placement of two shapes is different in two windows. However, some local ordering has to be done so as to allow for the unique identification of each object. This local ordering is achieved by providing objects with sequence numbers. Each host orders the objects for display depending on the order of their arrival. A chain of cross-links across the Host List and Shape List structures achieves this. In this cross-linked structure, we ensure that the objects corresponding to a particular host are in order, even if they had arrived out of order. Thus the only inconsistency that can be present among different windows is regarding the order of placement of objects created by different hosts.

4.3 Providing Reliability

We had discussed the mechanism by which the protocol achieves reliability in Chapter 3.2. Here we discuss the issues involved and the packet structures used. We work with a receiver-based notion of reliability where the onus of achieving reliability lies with the receiver and not with the sender. This is unlike the case in unicast communication where the sender is primarily responsible for ensuring reliability. The fundamental reason for this difference is that there is no simple way the sender can learn whether the packets are reaching all the receivers in the case of multicast communication. Further, sender-based notion of reliability would not allow us to exploit the distributed information present among the group members; data sought by a host can be more readily supplied by a member different from the sender. In Whiteboard, therefore, receivers send requests for retransmissions and the member with the lowest 'time-distance' value satisfies this request.

Whenever a host sends out a request for retransmission, it uses the following packet structure. Original source address indicate the source on whose packets the requesting node saw the jump in the sequence number. The packet includes both ends of the gap in the sequence number sequence. Thus only hosts that can fill the entire gap consider retransmitting. This further prevents flooding of unnecessary information on the network. Nodes that also have data missing from this gap, also use the responses to update their own information. When a host replies to a retransmission request, it uses

the usual packet structures except that the first bit of the *Packet Code* is set marking the packet as a reply to a retransmission.



4.4 User Interface

The user interface presented is very similar to any standard application running under MS Windows. We provide a menu bar with options to set drawing and text parameters like pen thickness, font and colour. Usual operations like opening and closing files have already been provided for. There is also a tool bar that presents an icon-driven interface to these operations.

Chapter 5

Details of Implementation

5.1 The ByteStream Layer

This layer communicates directly to the network. To make this layer an independent entity, we introduced a *SocketBase* class. This provides the upper layers with handles to start one end of a session without concerning itself about the myriad details about socket options, errors and maintenance. The *SocketBase* class provides support for both UDP and TCP based end sessions. However, no multicast support is directly provided. The class *MulticastSocketBase* derives from the *SocketBase* class, and extends the functionality to provide for multicast support. These classes encapsulate all error handling, and interface with the upper layers by means of handles that them to start or end unicast and multicast sessions and perform the read/write operations on the connection. Further the support for reading and writing is entirely asynchronous, and the parent window receives the messages that contain information regarding the connection that has message to be read.

5.2 The Frame Construction Unit

The entire design being object-oriented, Whiteboard has a separate frame construction unit that deciphers the packets received from the network. This unit is organised in a way to minimise the overhead on the other layers, and also to speed up the job. This unit comprises both encapsulation of outgoing data into frames as well as retrieval of data from incoming frames. Outside this unit, the treatment of frames is in terms of abstract entities that form the part of the frames. In essence, this means that the frame construction unit understands all packet formats (and only this unit needs to be aware of the packet structure), and the upper layers just ask for whatever information they may need from this frame. Upper layers only need to know *what* goes in a frame, without

bothering about *where* or *how* this data is organised in the frame. Also, this allows the unit to hide the conversion of data into network byte order within itself. The frame construction unit converts all outgoing data into network byte order before placing it into the frame, and converts data extracted from an incoming frame into the local byte order before giving it to the unit that needs the data. This conversion is necessary to communicate with Whiteboard applications running on platforms that support different byte orderings.

5.3 The Frame Layer

The Frame Layer primarily consists of the class *Client* which derives from *Multicast-SocketBase* and extends it with an end session functionality for Whiteboard. The term "Client" remains from the earlier developed skeletal application, though all nodes in the group do not differ at all. The Frame Layer provides a simple handle to the upper layer, which just specifies the Shape to write to the network. The Frame Layer calls the Frame Construction Unit to generate the frame, and calls the appropriate handle of the Bytestream layer to transmit the packet. The incoming packets are taken from the Bytesttream layer on being requested by the upper layer. The Frame Layer constructs the shapes from these packets and returns them to the upper layer.

The Frame Layer also does the work of session maintenance. All queues and lists of shapes are maintained here, and this layer does all memory management for Whiteboard. Details of list structures used are covered in Chapter 4.1. Frame layer also sends out retransmission requests and replies to such requests from others. A typical response involves sending out multiple packets over the network. This is not done in one block. After each packet sent out, the host checks if a message is waiting to be serviced. It continues sending out packets until it finds a pending message. If a pending message is encountered, it marks the retransmission response as pending and returns control to the window, to allow the message to be serviced. Each time thereafter a message is handled, the Frame Layer checks for any pending retransmission response. It begins retransmitting if this is the case. This process ensures lower delays for both local update as well as retransmission responses.

5.4 The Shape Layer

This is the layer that provides the highest level of abstraction and deals with shapes that are put up on the window and the operations pertaining to them. We have defined

an abstract class called Shape and derived specific classes like StraightLine, Rectangle, Ellipse, CurvedLine and Text from it.

The Shape Layer passes on the shape objects to the Frame Layer which packetizes them and sends them over the network through the ByteStream Layer. At the receivers' end, the Frame Layer passes pointers to the shapes to the Shape Layer which then displays them by calling the appropriate display function for the shape.

Chapter 6

Conclusions

The Whiteboard, in its present state of development uses a reliable multicast framework which provides minimal functionality so as to ensure low overhead. As noted earlier, different applications can add structure to this framework to satisfy their own quality of service requirements.

Currently, Whiteboard supports only the exchange of graphics and text. It can be supplemented with an audio capability which will allow the exchange of voice packets over the network. We have looked into some of the issues involved but the work is far from complete and left for the future. Another feature that can be added to Whiteboard is the ability to load files of certain formats, like ps and bmp. One immediate popular use of this feature will be the ability to edit conference papers in a shared fashion.

Though it is difficult to accept the current version of WhiteBoard as a complete video conferencing package, it certainly holds a lot of charm as a stand alone application for accompanying other multimedia conferencing tools available elsewhere. Since the network requirements of Whiteboard are never heavy, it is an ideal companion for other conferencing packages. Also, Whiteboard covers entirely a new ground as far as conferencing on MS-Windows is concerned. This in itself makes Whiteboard a unique application.

Bibliography

[JAC95] Floyd, S., Jacobson, V., McCanne, S., Liu, C., Zhang, L., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *SIGCOMM 1995*.

[CT90] Clark, D. and Tennenhouse, D., "Architectural Considerations for a New Generation of Protocols", *Proceedings of ACM SIGCOMM'90*, Sept 1990, pp 201-208.

[RFC1112] Deering, Steven, "Host Extensions for IP Multicast", *RFC 1112*.

[RFC1458] Braudes, R., Zabele, S., "Requirements for Multicast Protocols" *RFC 1458*.

[RFC1301] Armstrong, S., Freier, A., Marzullo, K., "Multicast Transport Protocol" *RFC 1301*.

[RFC1075] Waitzman, D., Partridge, S., Deering, S.E., "Distance Vector Multicast Routing Protocol" *RFC 1075*.

[COMER] Comer, Douglas E., "Internetworking with TCP/IP Vol I, II, III".

[WS2] Winsock 2.0 Specifications and Manual.