Dissertation Approval Sheet

This is to certify that the dissertation titled

Design and Deployment of a Reliable File Transfer Protocol over Asymmetric Satellite Networks

by

Anupam Goyal
(01329001)

is approved for the degree of **Master of Technology**

_____

Prof. Sridhar Iyer
(Guide)

_____

Internal Examiner

_____

External Examiner

_____

Chairperson

Date :_____

Acknowledgment

I express my sincere gratitude towards my guide *Dr Sridhar Iyer* for his constant support, encouragement and inspiration throughout the project. I also thank *Dr Kavi Arya* for his constant support to the project. I would also like to thank the members of the *Mobile Computing Research Group* at KReSIT, namely *Dr Leena Chandran Wadia*, *Srinath Perur*, *Vijay Rajsinghania*, *Vikram Jamwal*, *Deepanshu Shukla*, *Satyajit Rai* and *Abhishek Goliya* for their valuable suggestions and helpful discussions. I would also like to extend my gratitude towards the personnel of *HCL Comnet* for providing timely help and making invaluable suggestions regarding the issues related to the *Distance Education Program Satellite Network*.

Anupam Goyal

IIT Bombay.

January 10, 2003

Abstract

Satellite networks are the most widely deployed networks for audio and video transmissions. Though the initial installation cost is more, the benefit of reaching users spread over very large areas makes it a viable and very effective option for sending information in which some loss is tolerable. Some of the main drawbacks of satellite networks are limited bandwidth availability, large delay involved in transmission and their asymmetric nature. This leads to still higher delay in correcting the losses occurred during transmission making them difficult to use for reliable transmission. But when the need for sending data reliably to multiple users spread across a wide region arises, the long reach and inherent multicast nature of satellite networks make them an interesting option. Our work explores the possibility of using asymmetric networks like satellite networks for reliable data transmission to multiple users. An asymmetric network consists of a comparatively large bandwidth unreliable communication channel coupled with a low bandwidth reliable communication channel. We have designed a protocol which can be used to transmit data reliably over above defined asymmetric networks. We have implemented the protocol in developing a file transfer utility for Distance Education Program, IIT Bombay which uses a satellite network of the above kind. The algorithm is general enough to be employed for reliably transferring data over any asymmetric network though the change in network parameters will effect the protocol efficiency.

Contents

List of Figures

Chapter 1

Introduction

Satellite networks are the most widely deployed networks for audio and video transmissions. Though the initial installation cost is more, the benefit of reaching users spread over very large areas makes it a viable and very effective option for sending information in which some loss is tolerable. Some of the main drawbacks of satellite networks are limited bandwidth availability, large delay involved in transmission and their asymmetric nature. This leads to still higher delay in correcting the losses occurred during transmission making them difficult to use for reliable transmission.

Our work explores the possibility of using asymmetric networks for reliable data transmission to multiple users. An asymmetric network consists of a comparatively large bandwidth unreliable communication channel coupled with a low bandwidth reliable communication channel. When the need for sending data reliably to multiple users spread across a region arises, using such networks becomes an interesting possibility. Satellite networks are one such architecture. We have designed a protocol which can be used to transmit data reliably over above defined asymmetric networks. We have implemented the protocol in developing a file transfer utility for Distance Education Program, IIT Bombay which uses a satellite network of the above kind. The algorithm is general enough to be employed for reliably transferring data over any asymmetric network though the change in network parameters will effect the protocol efficiency.

1.1 The Distance Education Program Network Setup

The Distance Education Program (DEP) has the goal of the imparting quality education to a larger section of students spread across the country. The lectures are transmitted from IIT Bombay to various Remote Centres (RC). While the lecture is being transmitted, students at the various remote centres can ask questions which the instructor answers to his discretion.

The network consists of a 512 kbps half duplex multicast channel and a 16 kbps full duplex control channel on a satellite link. The various remote centres are connected in the network through one router at each remote centre. The 512 kbps multicast channel is a Demand Assigned Multiple Access (DAMA) channel and only one centre can use it for data transfer at a time. Other centres remain in receive mode when the station having control of the 512 kbps channel transmits. The 16 kbps full duplex channel is a Time Division Multiple Access (TDMA) channel and is available for data transfer at all times to all the centres in the network. Figure 1.1 gives an overview of the DEP satellite network. Figure 1.2 gives an overview of the setup at the main controlling station at IIT Bombay. The various remote centres also have a similar but a scaled down equipment setup at their end.

The video and audio of the on going lecture is transmitted on the 512 kbps multicast channel. The control information to maintain the session is exchanged on the 16 kbps control channel. The 512 kbps can be used by the one centre at a time. When a student at a remote centre has a query, the remote centre coordinator transmits a floor grant request to the lecture transmitting station. This message is carried on the 16 kbps control channel. The session coordinator at the transmitting station then grants the floor to the remote centre from where the request came. Granting the floor means relinquishing the control of the 512 kbps control channel so that the transmission of the student question from the remote centre can be done. After the student's question is over, the floor is reclaimed back by the transmitting station or the control of the 512 kbps multicast channel is seized by the transmitting station again.

IP multicast technology is used for transmitting the lecture audio and video on the satellite network. H323 protocol is used for accomplishing the video conferencing.

1.2 Requirement

It is of utmost importance to record the lectures and send them to the remote stations for later reference of the students. Also, various course related documents like tutorials, question papers, assignments etc need to be sent to the remote centres. The network remains free after the lecture sessions get over. If the available bandwidth of the satellite network could be used to reliably send the above mentioned documents as well then it would not only cut the costs but also serve as good use of the available bandwidth which gets wasted otherwise. Hence, the need to develop a reliable file transmission protocol.

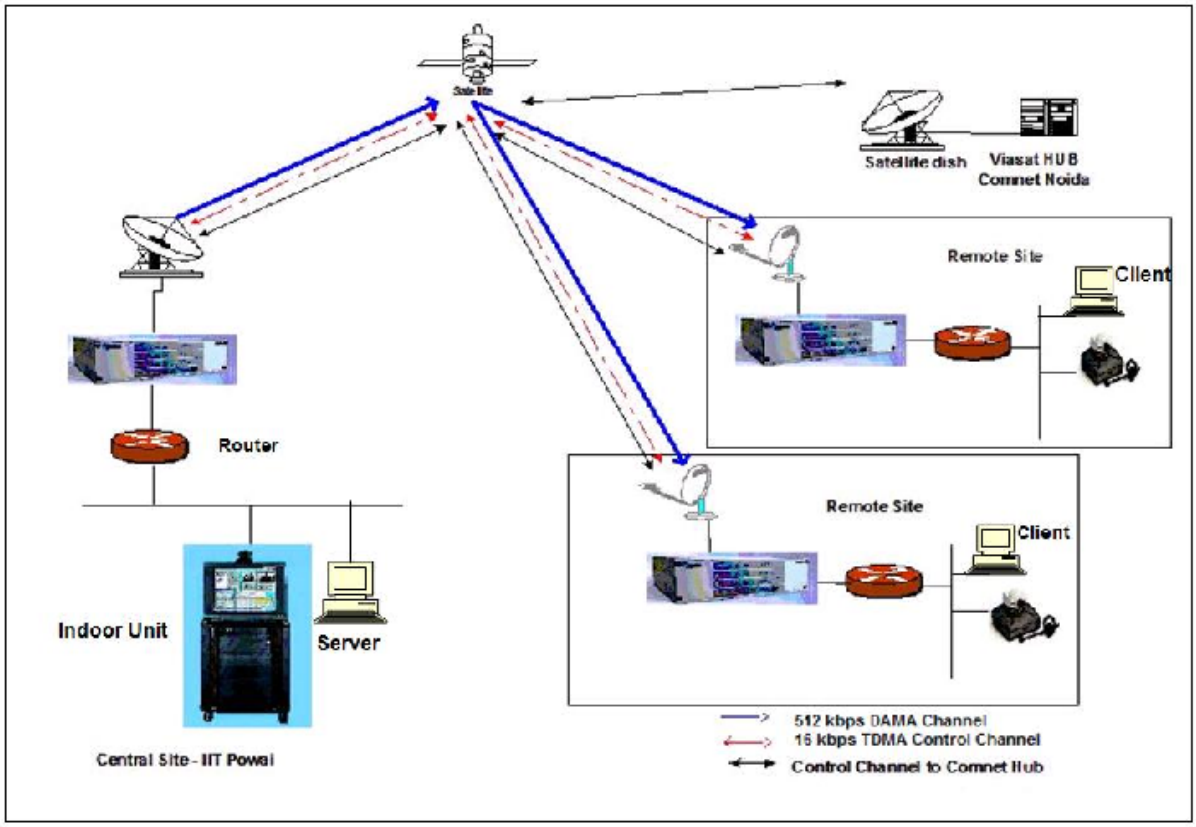

Figure 1.1: The DEP Satellite Network

1.3 Problem Definition

The requirement is to transfer files to various remote centres. Hence, TCP like reliability is essential. Since, the size of the files to be transmitted ranges from a kilobyte to approximately one gigabyte, the 16 kbps full duplex reliable channel is too inadequate for transferring the various lectures or other documents. The only option is to use the higher bandwidth 512 kbps multicast channel for data transmission. But the 512 kbps channel is a loss prone multicast channel. One solution is to use the loss prone 512 kbps channel for transferring data to the remote centres and use reliable 16 kbps control channel for exchanging control information to perform remulticast of data lost during the initial multicast. Also, using the 512 kbps channel is more efficient as all the remote centres can receive the files simultaneously. Hence, the need to develop a protocol to synchronize both the channels when sending data and increase the efficiency as far as possible.

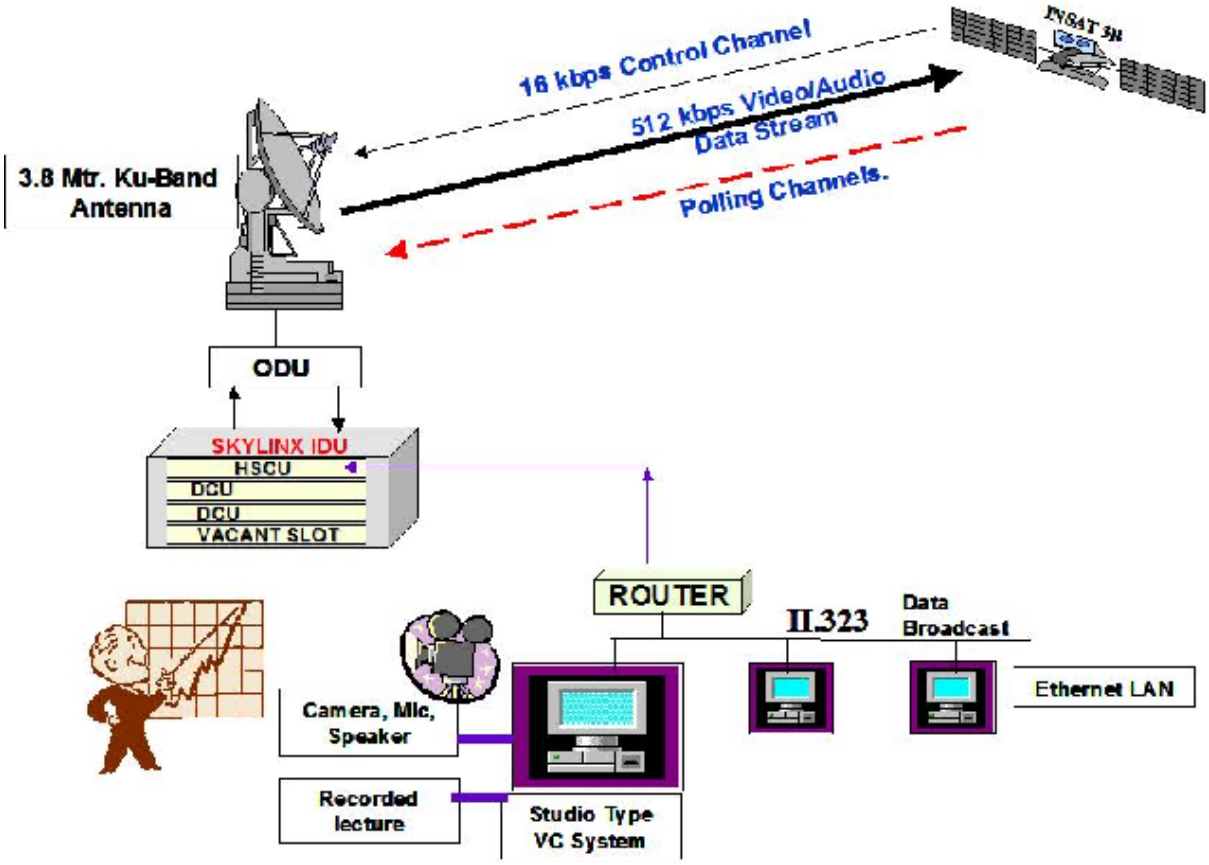

Figure 1.2: The DEP Main Centre (the hub)

Chapter 2

Literature Survey

This chapter provides an overview of satellite communication and work on reliable multicast protocols.

2.1 Satellite Communication

2.1.1 Communication Satellites

The principle of satellite communication is: a source is emitting data in form of electromagnetic (radio) waves up to the satellite in the sky which relays it back to the earth where it is picked up by the receiver's antenna or dish. The signal can be widespread over a large area, or focused to reach only a few recipients. Such focused beams are called spot beams and can be as wide as only about few hundred kilometers. Depending on the orbit, up to a third of the earth' surface can be covered by a single satellite. The link from source to satellite is called uplink, the link from satellite to receiver is called downlink.

A communication satellite contains several transponders. Each transponder listens to some portion of frequency spectrum, amplifies the signal and rebroadcasts at a different frequency to avoid interference with the incoming signal.

There are many beneficial features of satellite communication systems as against conventional communication over terrestrial networks. Some of these are robustness of satellite communication systems allowing a quick restoration even after natural disasters, ease of integration of new users as no new infrastructure is needed, low costs for installation of a terminal, inherent multicast capability etc. Especially, for applications with a high fraction of broadcast data, and for applications with bursty traffic and needs for high bandwidth, satellites seem to be an adequate alternative to conventional communication systems. Refer [**?**, **?**, **?**] for further details.

2.1.2 Classification of Satellites

Satellites are classified with regard to the orbit they are moving in. Hence, altitude, velocity, and duration of visibility are some important parameters. Satellites can be divided into following categories:

1. **Geostationary Satellites:** GEOs are launched in the geostationary orbit circling the earth in 24 hours of an altitude of nearly 36000 km. Placed over the equator and moving in the direction of Earth rotation, they can be seen permanently appearing as a fix point in the sky. Due to their permanent availability they are predestined to provide continuous communication services.

2. **Medium Earth Orbit Satellites:** These satellites are placed in an medium altitude between 9000 km and 19000 km describing oval paths around the Earth. They can be seen only for a certain period and have to be tracked by the ground stations.

3. **Low Earth Orbit Satellites:** These satellites orbit very near to the Earth (in an altitude down to 200 km). They have enormous speeds of more than 28000 km/h. They stay in sight for only a few minutes. Communication requires thus smart tracking mechanisms.

VSAT: Very Small Aperture Terminals or VSATs are relatively new developments in the communication satellite world. VSATs are low cost microstations having one meter antennas and capable of a power output of about one watt. The uplink is generally good for 19.2 kbps but the downlink is more, often 512 kbps. In many VSAT systems, the microstations do not enough power to communicate directly with each other via the satellite alone. So, a special ground station, the hub, with a large high gain antenna is used to relay traffic between VSATs. In this mode of operation, either the sender or the receiver has a large antenna and a powerful amplifier. The use of hub induces a larger delay but provides for cheaper end-user stations. A typical end to end delay value for a VSAT system with a hub is 540 milliseconds. Distance Education Program (DEP) at IIT Bombay employs a VSAT satellite network.

2.1.3 Techniques for Multiple Access

Three basic link capacity techniques exist:

1. **Frequency Division Multiple Access (FDMA):** FDMA assigns each user a subband of the available repeater bandwidth separated by small subbands to avoid interference, so that multiple signals can simultaneously and continuously access

the satellite amplifier. This is very simple to implement in current systems. The major drawback of this technique is the loss of capacity in order to avoid interference between the several channels.

2. **Time Division Multiple Access (TDMA):** TDMA provides each user the full available bandwidth, but only for a certain time period. All users transmit discontinuously using the same frequency. Data has to be sent in a way such that it fits into the time slots assigned by the satellites transponders. This requires synchronization of the users and an exact timing, but enables the satellite to operate at maximum output which leads to high efficiency compared with the FDMA scheme.

3. **Code Division Multiple Access (CDMA):** In CDMA systems all users can send continuously and simultaneously using the full capacity on the same frequency band of the channel. Signals from all the users are overlaid over each other.In order to distinguish overlaid signals of several users, a pseudorandom noise code is combined and transmitted with the useful data to allow the satellite to identify each sender. This requires a synchronization of transmit and receive codes, and limits the number of users. Advantages are the good efficiency, and the simple operation since no synchronization between users is necessary.

Demand Assigned Multiple Access (DAMA): A DAMA system is typically a single hop satellite transmission network which allows direct connection between any two nodes in the network among many users sharing a limited pool of satellite transponder space. DAMA supports full mesh, point-to-point or point-to-multipoint communications. In a DAMA system, the network allocates communication bandwidth on a demand-assigned basis. Communication bandwidth is provide from a central frequency pool. After a user has been served, the bandwidth is returned to the pool for reuse by others.

DEP Satellite Network: The DEP satellite network is structured as a combination of 2 channels - a 512 kbps half duplex multicast Demand Assigned Multiple Access (DAMA) channel and a 16 kbps full duplex TDMA channel. The 512 kbps DAMA channel is used for audio-video multicast and the 16 kbps TDMA channel is used as a control channel for exchanging control information.

Apart from multiple access, the satellite link quality is also an important issue. The quality of a link via satellite between two earth stations is mainly described by the ratio between the received signal to the noise disturbing a link. The two most ratios are -

- **Signal to noise power ratio (SNR):** SNR is expressed as the ratio of the amplitude of the desired signal to the amplitude of the noise signals at a given point in time.
- **Carrier power to noise spectral density (CNR):** CNR is a measure of the received carrier strength relative to the strength of the received noise.

2.2 Reliable Multicast Protocols

There are many applications which require reliable delivery of data to a group of users on the fly such as a whiteboard applications, editors etc. Availability of inherently multicast media such as satellite networks also open up possibilities of serving multiple users simultaneously. For achieving this, often, the users join a multicast group and the data is multicast to all the users of the group. But multicasting is inherently unreliable. Hence, a number of reliable multicast protocols have been developed to reliably transfer data to a multicast group.

The most important issue in designing reliable multicast protocols is deciding who initiates error recovery. There are basicly two different approaches to providing reliable, scalable multicast communication - the sender-initiated approach and the receiver initiated approach.

- Sender-initiated approach: The sender-initiated approach places the responsibility of error recovery on the sender, which maintains state information regarding all receivers that it is multicasting data to.
- Receiver-initiated approach: This approach shifts most of the responsibility for reliable data delivery to the receivers. Each receiver is responsible for detecting lost packets and informing the sender via negative acknowledgments (NAKs) when it requires the retransmission of a packet.
- There are other protocols like the tree based protocols and the ring based protocols which combine sender initiated and receiver initiated protocols.

Refer [**?**, **?**, **?**] for further details.

2.2.1 Sender-Initiated Protocol

In a sender-initiated protocol, the sender maintains a list (called the ACK list), for each packet, of the receivers from which it has received a positive acknowledgment (ACK). Each time a receiver correctly receives a packet, it returns an ACK. Upon receipt of the

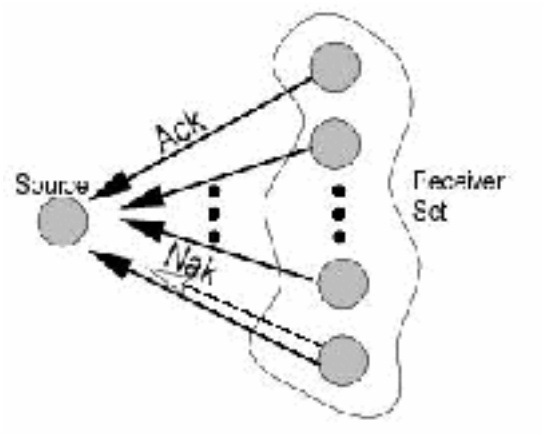

Figure 2.1: Sender-Initiated Protocol

ACK, the sender updates the ACK list for the corresponding packet. Timers are used to deal with the lost packets. If the timer expires prior to the sender having received ACKs for this packet from every receiver, the sender retransmits the packet and restarts the timer.

Although there are many variations, the Generic Sender-Initiated protocol exhibits the following behavior.

1. error recovery is selective repeat, i.e., only packets that are suspected to be lost or corrupted are retransmitted,

2. anytime that a sender transmits or retransmits a packet, it broadcasts to all receivers and starts a timer,

3. each time that a receiver receives a packet correctly, it transmits an ACK to the sender over a point-to-point connection,

4. whenever a timer expires, the sender rebroadcasts the corresponding packet, giving it priority over new packets.

Disadvantages of Sender-Initiated protocols:

1. ACK-implosion problem : As all receivers that are receiving the multicast send back ACKs for all received packets, there is a danger of sender getting flooded by the ACKs from receivers.

2. Receiver Set tracking problem :The need to know the whole receiver set and keep track of status for each receiver.

2.2.2 Receiver-Initiated Protocol

In a receiver-initiated protocol, the sender continues to transmits new data packets until it receives a negative acknowledgment (NAK) from a receiver. When this occurs, the sender then retransmits the packet required by that receiver. The receiver checks for lost packets. It transmits a NAK to the sender if it receives a packet with larger sequence number (or after a timeout if it is expecting a retransmission). In the case that the sender does not always have packets to send (i.e., must occasionally wait for data to be produced at a higher level), it may be necessary for the sender to multicast periodic state information (e.g., giving the sequence number of the last transmitted packet) while it is idle.

The generic receiver-initiated protocol that exhibits the following behavior

1. the sender broadcasts all packets,
2. the sender gives priority to retransmissions over transmissions of new packets,
3. whenever a receiver detects a lost packet, it transmits a NAK to the sender over a point-to-point channel and starts a timer,
4. the expiration of a timer without prior reception of the corresponding packet serves as the detection of a lost packet.

A further variation to the generic protocol can be the following:

1. the sender broadcasts all packets and state information
2. the sender gives priority to retransmissions over transmissions of new packets,
3. whenever a receiver detects a lost packet, it waits for a random period of time and then broadcasts a NAK to the sender and all other receivers, and starts a timer,
4. upon receipt of a NAK for a packet which a receiver has not received, but for which it initiated the random delay prior to sending a NAK, the receiver sets a timer and behaves as if it had sent that NAK,
5. the expiration of a timer without prior reception of the corresponding packet serves as the detection of a lost packet. receivers. Hence, the first NAK generated usually arrives to all other receivers prior to their generating additional NAKs.

Disadvantages of receiver initiated protocols:

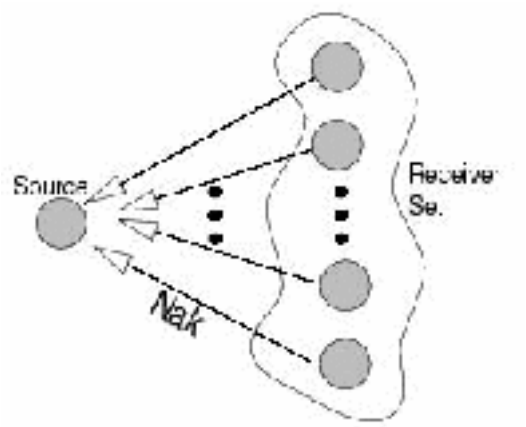

Figure 2.2: Receiver-Initiated Protocol

1. It has no mechanism for the source to know when it can safely release data from memory as it does not keep track of all the receivers in the multicast group. Since, the source receives feedback from the receivers only when packets are lost and not when they are delivered, it is unable to ascertain when it can safely release the data.

2. Practical implementations of the protocol fail to provide all the advantages. They need to keep track of receivers as well as process ACKs from each receiver.

Advantages of Receiver-initiated protocol over Sender-initiated protocol:

1. The source does not need to know the receiver set.

2. The source does not have to process ACKs from each receiver.

divided

2.2.3 Treebased protocols

The generic tree based protocols are characterized by the following:

1. Receiver set is divided into groups.

2. An acknowledgment tree (ACK tree) is built from the set of groups, with the source as the root of the tree. The ACK tree consists of the receivers and the source organized into local groups.

3. Each group has a group leader in charge of retransmissions within the local group. The source is the group leader in charge of retransmissions to its own local group.

Group leaders may be children of another local group, or minimally, may just be in contact with another local group. Each local group may have more than one group leader to handle multiple sources. Group leaders could also be chosen dynamically, e.g., through token passing within the local group. Hosts that are only children are at the bottom of the ACK tree, and are termed leaves. Obviously, an ACK tree consisting of the source as the only leader and leaf nodes corresponds to the senderinitiated scheme.

4. Each group leader other than the source communicates with another local group (to either a child or the group leader) closer to the source to request retransmissions of packets that are not received correctly.

5. Acknowledgments from children in a group, including the source's own group, are sent only to the group leader.

6. The children of a group send their acknowledgments to the group leader as soon as they receive correct packets. Such acknowledgments are called local ACKs or local NACKs, i.e., retransmissions are triggered by local ACKs and local NACKs unicast to group leaders by their children.

7. Treebased protocols can also delegate to leaders of subtrees the decision of when to delete packets from memory which is conditional upon receipt of aggregate ACKs from the children of the group. Aggregate ACKs start from the leaves of the ACK tree, and propagate toward the source, one local group at a time. A group leader cannot send an aggregate ACK until all its children have sent an aggregate ACK.

Advantages :

1. The ACK tree structure prevents receivers from directly contacting the source. Hence, these protocols have scalability with a large receiver set.

2. Tree based protocols eliminate ACK implosion problem.

3. The source does not need to know the whole receiver set.

4. The protocol operates solely on the basis of messages exchanged in local groups between a group leader and its children.

5. If aggregate ACKs are used, a tree based protocol can work correctly with finite memory even in the presence of receiver failures and network partitions.

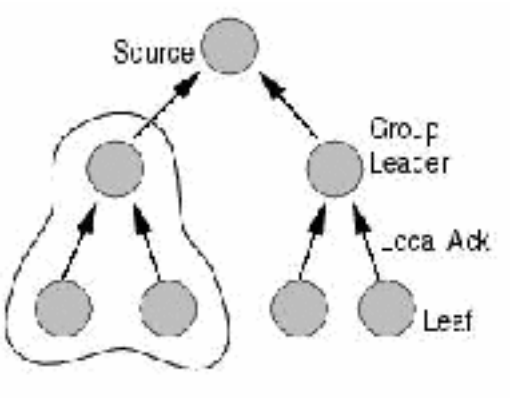

Figure 2.3: Tree-based Protocol

2.2.4 Ringbased protocols

Ringbased protocols for reliable multicast were originally developed to provide support for applications that require an atomic and total ordering of transmissions at all receivers. One of the first proposals for reliable multicasting is the token ring protocol. Its aim was to combine the throughput advantages of NACKs with the reliability of ACKs. The basic ring based protocols are characterized by the following:

1. Only one token site responsible for acknowledging packets back to the source.

2. The source times out and retransmits packets if it does not receive an ACK from the token site within a timeout period.

3. The ACK also serves to time-stamp packets, so that all receiver nodes have a global ordering of the packets for delivery to the application layer.

4. Receivers send NACKs to the token site for selective repeat of lost packets that were originally multicast from the source.

5. The ACK sent back to the source also serves as a token passing mechanism. If no transmissions from the source are available to piggyback the token, then a separate unicast message is sent. The token is not passed to the next member of the ring of receivers until the new site has correctly received all packets that the former site has received. Once the token is passed, a site may clear packets from memory. The token is passed exactly once per message.

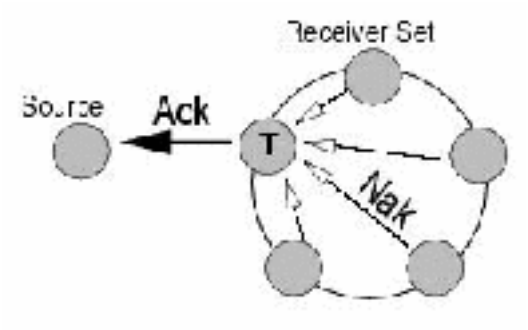

Figure 2.4: Ring-based Protocol

2.3 Multicasting over Satellite networks

2.3.1 Motivation

Satellites are excellently suited for distributing information simultaneously to multiple locations. As in nearly all communication systems, some sort of error control scheme is required in satellite multicasting to assure satisfactory fidelity of the information provided to each destination. Error control schemes may be broadly classified as forward error correction (FEC) or automatic-repeat-request (ARQ), and both can be applied for satellite communication.

FEC entails overhead in the form of many check symbols. Further, this overhead penalty is exacted even at times of good channel quality. This is particularly troubling since good channel conditions will be experienced a majority of the time when using a well-designed satellite link.

ARQ protocols adapt to different channel qualities by retransmitting data only as needed. Hence ARQ can provide high fidelity with less overhead than FEC during times of good channel quality. A drawback of ARQ not suffered by FEC is the need for a feedback channel, but this requirement is often an acceptable concession for achieving information transfer with excellent fidelity. Also, the presence of a feedback channel opens the possibility for reliable communication, in which the transmitter can know positively that the receiver has indeed received information which the transmitter had sought to deliver, in correct order, without duplicates or errors. Refer [**?**] for further details.

2.3.2 Multicast Automatic Repeat Request Protocols

The various ARQ protocols for reliably transferring data to multiple clients are described below.

2.3.2.1 Stop-and-Wait Multicast ARQ

This protocol is adapted from a standard stop-and-wait (SW) point-to-point ARQ protocol, for ARQ multicasting via satellite. In this protocol, the transmitter sends a block of N frames and then waits to collect acknowledgments. Each receivers acknowledgment comprises a field identifying the receiver and an N-bit field to acknowledge positively and negatively the individual frames of the block just received. After waiting enough time to collect acknowledgments from all receivers, the transmitter sends a block of those frames not yet acknowledged by all receivers. The receivers again send acknowledgments and this process continues until all N frames have been acknowledged by all receivers.

2.3.2.2 Go-back-N Multicast ARQ

There are two variations to this protocol - end-to-end protocol and a tandem protocol. The end-to-end scheme is a modified conventional multicast GBN ARQ protocol. Two features of such a conventional protocol are (1) every frame must be acknowledged, either positively or negatively, by all receivers; and (2) every frame transmitted must be positively acknowledged by all receivers in order to avoid a retransmission. The conventional protocol is modified by introducing a special retransmission indication block (RIB) to notify all receivers before a GBN retransmission commences. When a receiver having no outstanding frames and so not awaiting any retransmitted frames-receives such an RIB, it discards the next N frames and so avoids accepting duplicate frames. The tandem ARQ protocol assumes special supporting equipment aboard the satellite. It uses a point-to-point protocol for error control on the uplink between a transmitting earth station and the satellite, and a point-to-multipoint protocol for the downlink from the satellite to multiple receiving earth stations.

2.3.2.3 Improved GBN Multicast ARQ

There are three variations to this point to multipoint ARQ protocol. These three protocols differ in the degrees to which the transmitter regards past acknowledgments from receivers. In the first protocol, called memoryless, the transmitter ignores this history entirely. All receivers must acknowledge the same transmission of a frame, or else the frame will be retransmitted. In the third protocol, called full-memory, the transmitter regards the entire available history of acknowledgements from all receivers for all frames transmitted [within the ARQ window]. An intermediate approach is taken in the second protocol, called limited memory. Full-memory operation applies for only the first packet in a group of consecutive packets retransmitted by the GBN protocol, but no history is maintained for any packets following that first one. Thus the transmitter considers ac-

knowledgments of individual receivers but disregards the information of which receivers have acknowledged frames sent before the transmitter goes back for a retransmission.

2.3.2.4 Other Multicast ARQ Techniques

One suggestion is to send multiple copies of each ARQ frame to improve the likelihood of correct reception before requesting a retransmission. The optimal number of frame repetitions for this scheme is determined by a dynamic programming technique. Another protocol suggested an selective repeat ARQ protocol in which frames requiring retransmission are combined by modulo-2 addition into an XOR frame. If the frames are properly combined, the XOR frame will contain at most one frame not received correctly by any particular receiver. Since the list of frames used to compose the XOR frame is sent with the XOR frame, an individual receiver can extract from the XOR frame the single constituent frame which that receiver has not yet correctly received.

2.4 Reliable Multicast Protocols for Satellite Networks

The performance of TCP is very good over wired networks but its efficiency greatly decreases over satellite networks. The main reasons being the large round trip time and high bit error probability involved in satellite networks. Small TCP receiver window size and TCP's slow start and congestion control algorithms contribute to the performance degradation of TCP over satellite networks. Hence, the need for alternative protocols for reliable transfer of data over satellite networks arises. Refer [**?**, **?**] for further details.

2.4.1 Multicast File Transport Protocol (MFTP)

Multicast File Transport Protocol (MFTP) is a one-to-many reliable multicast protocol. MFTP uses a check-point based automatic repeat request (ARQ) reliability scheme. The protocol partitions a file into packets and builds equally-sized blocks of packets. Each block consists of as many packets as bits fit into an IP-packet of maximum size. The source initially multicasts all packets (equivalent to the whole file) in the first pass (or data transmission phase). Receivers start receiving data packets and note the missing packets. After each file transfer pass, receivers send a NACK-bitmap listing the status (received/missed) for each data packet sent within the block. The sender collects all the NACK packets and determines the set of data packets that were not correctly received and later retransmits in the subsequent file transfer pass. This procedure continues until

all receivers have completely received the file. Receivers leave the multicast group as soon as they complete file reception. Refer [?] for further details.

2.4.2 Multicast File Transport Protocol with Erasure Correction (MFTP/EC)

Multicast File Transport Protocol with Erasure Correction (MFTP/EC) is a continuous based ARQ reliable multicast protocol similar to MFTP. MFTP/EC uses erasure correction methods to perform loss recovery more efficiently. Both the sender and receivers partition the set of data packets into groups of k packets each and build blocks of n groups (where n ¿ 10,000). Like MFTP, receivers in MFTP/EC send back NACK-bitmaps after a complete file transmission indicating whether or not all packets of a group have been received successfully. In the second and subsequent passes, the sender multicasts one redundancy packet for each NACKed group. Ideally, this makes the recovery passes in MFTP/EC only last up to a fraction 1k of a full MFTP pass (although in general, MFTP/EC would need more passes than the basic MFTP protocol). Refer [?] for further details.

2.5 Reliable Multicast Protocols

This section explains some of the proposed multicast protocols for reliable transport briefly. Refer [?, ?, ?] for further details.

2.5.1 Reliable Broadcast Protocol

RBP provides multi-point communication between sites connected by a local-area broadcast network. RBP combines NACKs and positive acknowledgments (ACKs) to achieve reliability, ordering, and fault tolerance. Messages are multicast to the group through the token site. In other words, the token site multicasts an ACK for each message it receives. ACKs inform the sender that the token site received a message. Since ACKs carry a global timestamp, receivers use them to order and detect lost messages. When a receiver detects a lost message, it sends a NACK to the current token site, which replies with the missing message. For resilience and to limit the amount of state each site needs to keep, the role of the token site rotates among all group members. A member accepts being the next token site only if it has all the messages earlier than the timestamp of the token passing message. Thus, after a message is generated and the token site rotated K times, K sites can fail and the message will still be delivered to the sites that are active.

Furthermore, the current token site can purge all messages whose timestamps are earlier than the last time this site had the token. Refer [**?**] for further details.

2.5.2 Multicast Transport Protocol

MTP provides reliable and globally ordered delivery of client data among a group of communicating processes. MTP is built on top of IP multicasting, and provides strong consistency by multicasting messages under centralized control. One of the participating processes is assigned to be the group master. The group master ensures that data is delivered reliably and in order by giving out transmit tokens and controlling the status of tokens and data messages. It implements an implicit congestion control mechanism by not granting tokens if a certain number of messages are still in the pending state. A message is pending until the master sees all of its packets. The master is also responsible for supervising group membership. If a group member that is holding a token becomes disconnected, the master removes it from the group and rejects that member's outstanding messages. A process that wants to join the group sends a request to the master, who may accept or reject it. If the request is accepted, the master must be in possession of all the transmit tokens before sending an acceptance message. This way, the master ensures that the new member sees only complete messages. The master may notify its client application about the new member, and it is up to the application to send client state to the new member. MTP is a NACK-based protocol. When a group member detects a lost message, it sends a NACK to the message producer, who multicasts the requested message to the entire group. However, if the producer has discarded the requested message, it informs the source of the NACK. It is up to the application to recover or decide to withdraw from the group. MTP s flow control mechanism is based on a fixed-size transmission window. Every group member agrees on the current window size upon joining a group. This window specifies the maximum number of data packets a member can send to the group within a specified period of time, and is shared between new and retransmitted packets. Thus, the more retransmissions, the fewer new packets are injected into the group. Refer [**?**] for further details.

2.5.3 Scalable Reliable Multicast

The Wb distributed whiteboard tool uses the Scalable Reliable Multicast (SRM) transport protocol to reliably distribute data among all participating sites. Participants multicast NACKs to request retransmission of lost data, which can be answered by any member that has the information. To avoid generating multiple copies of retransmitted data, retransmissions are multicast to the group. To further reduce the multiple copy

problem, a site waits a random period of time before sending a NACK or retransmitting data, and suppresses its own transmission in case it hears it from another member of the group. A new site joins an ongoing whiteboard conference by announcing its presence with a join message. Current members update their view with the new site. The new site's view of the group is gradually updated through periodic session messages generated by current members. Session messages are also used to measure round-trip time (RTT) between members. RTT information is used to set suppression timers. Refer [**?**] for further details.

2.5.4 Log-Based Receiver-Reliable Multicast

The Log-Based Receiver-Reliable Multicast (LBRM) protocol uses a logging server to log all packets transmitted by the source and to retransmit lost packets. The source keeps data that has already been sent until it gets an ACK from the logging server. LBRM uses NACKs between receivers and the logging server and ACKs between the logging server and the source. Logging servers may be organized in a hierarchy. Logging servers decide whether they should multicast or unicast a lost packet based on the number of NACKs they get. Refer [**?**] for further details.

2.5.5 Reliable Adaptive Multicast Protocol

RAMP is layered on top of a network-layer multicast protocol, such as IP multicast, and provides NACK-based reliable delivery. A RAMP sender starts by sending a connect message to an IP multicast address. The sender can start multicasting the data after it receives at least one accept. Data messages contain monotonically increasing sequence numbers, and once a receiver detects a gap in the message sequence it requests a retransmission by unicasting a NACK back to the sender. The application selects whether retransmitted messages are multicast to the group or unicast to the requester. A RAMP sender may operate in two modes, burst or idle, and may switch between the two modes during the life of a connection depending on the amount of data to be transmitted and the number of receivers. In burst mode, the sender marks the start and end of a burst by setting the ACK flag in the connect message and sending an idle message, respectively. By setting the ACK flag, the sender expects reliable receivers to send an ACK back. When the sender does not hear from a known reliable receiver, it retries and eventually ejects the receiver from its membership set. Note that since RAMP assumes a reliable communication medium, it provides a more aggressive recovery mode. In idle mode, the sender multicasts an idle message immediately after it receives the first accept message and when it has no data to send. When a receiver does not receive an idle or data message

it assumes data loss and requests retransmission. RAMP senders manage group membership information, which provides circuit-switched networks with the required information for circuit setup, but does not scale as well as IGMP-style group membership protocols. RAMP uses a simple rate-based flow control in which the sender adjusts its transmit rate based on the feedback received by the slowest receiver in the group. Refer [**?**] for further details.

2.5.6 Multicast Dissemination Protocol

The Multicast Dissemination Protocol (MDP) provides a reliable multicast framework for file distribution. MDP evolved from IMM, image multicaster, a protocol used in disseminating satellite image files over the MBone. The MDP sender fragments the file into a sequence of MDP maximum data units (MDUs) which are multicast to the group using the UDP/IP multicast suite. MDUs are sent at the transmission rate set by the sender application, which can also control the interval between file transmissions. MDP receivers assemble the received data units into the original file, which can be archived or processed using image viewers or text processing tools. MDP's recovery mechanism works in rounds: after transmitting a file, the source asks receivers for retransmission requests. A receiver that has detected sequence number gaps schedules a retransmission request which gets multicast if no other receiver has requested the same retransmission. MDP also provides an operation mode in which receivers may request repairs at any time during the transmission of a file. The MDP source can also request ACKs from receivers, which respond with information about their current state. MDP allows sites to join and leave the multicast group. New sites announce their presence by multicasting a join packet to the group. Currently, MDP does not have any mechanism to prevent latecomers from requesting retransmission of old packets. Refer [**?**] for further details.

2.5.7 Tree-Based Multicast Transport Protocol

The goal of the Tree-Based Multicast Transport Protocol (TMTP) is to provide reliable multicast communication support for one-to-many bulk data dissemination applications. For error and flow control, TMTP organizes group members into a control tree. Each subtree, or domain, is represented by its root, or the domain manager. Domains typically correspond to subnets. At group creation time, the maximum tree degree K (the maximum number of children at each level) is defined. The control tree grows and shrinks as members join and leave the group. Joining the control tree is based on an expanding ring search to locate the nearest domain manager whose degree is less than K. A TMTP sender multicasts packets to the corresponding multicast group. The root of each

subtree (including the data source) only waits for ACKs from its children to reclaim buffer space and advance its transmission window. An intermediate node does not have to wait for ACKs from all its children to unicast the ACK to its parent. ACK timeouts trigger retransmission of the corresponding packet using limited-scope multicast. In addition to this sender-initiated error control approach, TMTP also relies on NACKs. TMTP receivers use limited-scope NACK multicast subject to suppression. TMTP uses a window-based flow control scheme and a predefined maximum transmission rate set at group creation time. TMPT supports dynamic group membership, where sites can join and leave a multicast group anytime during the life of a session. A session directory service provides primitives for creating, deleting, joining, and leaving a multicast group. However, knowledge of group membership is not assumed. Refer [**?**] for further details.

2.5.8 Reliable Multicast Transport Protocol

The Reliable Multicast Transport Protocol (RMTP) supports reliable one-to-many bulk data delivery. Like TMTP, RMTP addresses feedback implosion by organizing group members into a control tree, which governs feedback propagation and processing. Intermediate nodes in the control tree of designated receivers (DRs) are responsible for buffering data received from the source, processing ACKs from their children, and retransmitting lost packets. Receivers periodically unicast ACKs to their DRs informing them what packets have been correctly received. Depending on how many receivers lost the same packet, the DR decides whether it should multicast or unicast the retransmission. Like TMTP, RMTP uses window-based flow control and defines a maximum transmission rate set at group establishment. RMTP allows dynamic group membership: receivers may join and leave a multicast group anytime during a session. However, knowledge of group membership is not required. RMTP not only allows receivers to join at any time, but also guarantees that they receive all of the data reliably. This feature comes at the price of having the DRs buffer data already sent throughout the session. RMTP's congestion control mechanism is based on shutting down the transmission window to one packet if the number of ACKs with retransmission requests exceeds a predefined threshold. Refer [**?**, **?**] for further details.

2.6 Classification of Reliable Transfer over Asymmetric Networks (RTAN)

The protocol proposed by this work, the Reliable Transfer over Asymmetric Networks (RTAN) protocol, is a stop-and-wait protocol. The protocol operates by sending N pack-

ets to all the receivers and doesn't send any new packets until all the N packets are successfully received by all the receivers by the way of retransmissions if any packets are lost. This reliable multicast protocol follows a sender-initiated approach. The server maintains an index for all the packets sent in this cycle and also keeps track of all the receivers. Receivers indicate packet loss by the way of both ACKs and NACKs. They maintain an index for all the packets expected to be received and set or reset the index bits according to the receipt or drop of the expected packets.

Chapter 3

Design of RTAN

(Reliable Transfer over Asymmetric Networks)

3.1 The Problem

The requirement is to transfer files to remote centres over Distance Education Program (DEP) satellite network. The network consists of a half duplex unreliable multicast 512 kbps channel and a full duplex 16 kbps reliable channel. For sending files, TCP like reliability is a must. But the full duplex 16 kbps reliable channel is too small to reliably send files to remote centres in a reasonable amount of time. Hence, a different strategy is needed to counter the problem.

3.2 Solution Overview

One solution to the problem outlined in the previous section is to send data over the 512 kbps multicast channel and exchange information about the lost data on the 16 kbps reliable channel in order to remulticast the lost data again. The above idea is used to develop a protocol for data transfer over the DEP satellite network. Client-server paradigm has been used to implement the protocol. The basic designs of the server and the client are as follows.

- **The Server:** The server consists of three basic threads - the KeepAlive thread, the ConnectionAcceptor thread and the ControlProcessor thread.

  - The KeepAlive thread is used to keep the link alive and keeps multicasting dummy keep alive packets to the group every 2 seconds. The necessity of KeepAlive thread would be explained in later chapters.

  - The ConnectionAcceptor thread keeps listening for clients and opens a TCP connection with the them as they connect to the client.
  - The ControlProcessor thread actually implements the file transfer protocol. It sends various messages to the client as per the protocol and receives acknowledgments and indices from the client.

- **The Client:** The client consists of two basic threads - the Data thread and the Control thread.
  - The Data thread receives multicast data from the server and stores them in buffer.
  - The Control thread exchanges control information with the server Control thread and maintains the protocol.

The protocol's basic structure will work for any asymmetric network topology which consists of a low bandwidth full duplex reliable channel and a high bandwidth half duplex unreliable channel. But this basic protocol may not be very efficient under conditions like high error rate of the network or the presence of a very high number of clients. Hence, variations to the basic protocol have also been proposed. The following sections give an outline of the basic protocol and the proposed variations to the basic protocol.

3.3 Version 1.0 The Basic Protocol

When the server is invoked, the server main thread invokes KeepAlive and ConnectionAcceptor threads. The ConnectionAcceptor thread starts listening for clients. Clients connect on the 16 kbps reliable channel by opening a TCP connection to the server. ConnectionAcceptor thread keeps listening for client connections throughout the life of the program and creates a list of clients as they get connected. All messages to and from the client are exchanged on the 16 kbps channel. Only the data to be sent is multicast on the 512 kbps channel. Every other communication occurs on the 16 kbps reliable channel.

When a file has to be transferred, a ControlProcessor thread is invoked. The ControlProcessor thread takes the client list and sends each client the information about the file to be sent. This information includes the file name, file size and the file identifier. The server sends the file in parts. The whole file is divided into packets of a predetermined fixed size. The complete data transfer consists of many transfer cycles. Each transfer cycle, in turn, consists of a data multicast and several recovery cycles. The major phases of the protocol are described in the following.

1. In each transfer cycle, the server reads a predetermined number of packets of predetermined size in a buffer and multicasts them on the network. The number of packets read from the file and the size of packets remain fixed throughout the transfer of the file. Each packet sent is given a unique sequence number.

2. After multicasting the packets, the server enters the recovery cycle. It asks all the clients for an index indicatiing the receipt of all the multicast packets. If the client had received the packet then the corresponding bit for that packet must be set in the received index otherwise it is reset. All those packets are resent which have not been received by at least one client. Then the server again asks for index from all clients. The above process repeats until all the clients have received all the packets multicast in this cycle. At this stage the server exits the recovery cycle and one transfer cycle is complete.

3. Now, the server checks if the whole file has been transferred or not. If more data has to be sent, the server initiates next transfer cycle or goes back to step 1. This process goes on until the whole file has been transferred.

The data packets are multicast over 512 kbps multicast channel and all the control information such as indices is sent on the 16 kbps control channel using a TCP connection. Refer to figure 4.1 for having an overview of the protocol sequence. Figure 3.1 provides a view of the channel usage by the protocol over the DEP satellite network. The next chapter provides a detailed understanding of the implementation of the basic protocol.

3.4 Version 1.1 Multiple Multicast Per Cycle

The basic protocol works well if the error rate of the satellite link low. But if the error rate is very high such as if the network is dropping more than half of the packets being sent then the efficiency can drop to unacceptable levels. The time taken to transmit the file may be far too large. One modification which can improve performance is to multicast all the packets being sent in a transfer cycle multiple times. This means that if the server has to send N packets in a multicast, it will send all N packets again after multicasting them once. The server can vary the number of times it is multicasting each set of packets depending on the current error rate of the network.

3.5 Version 1.2 Ignore Unruly Client

One of the main drawbacks of the basic algorithm is that it is at most as efficient as the least efficient client. Although, the multiple multicast solution works fine if all the

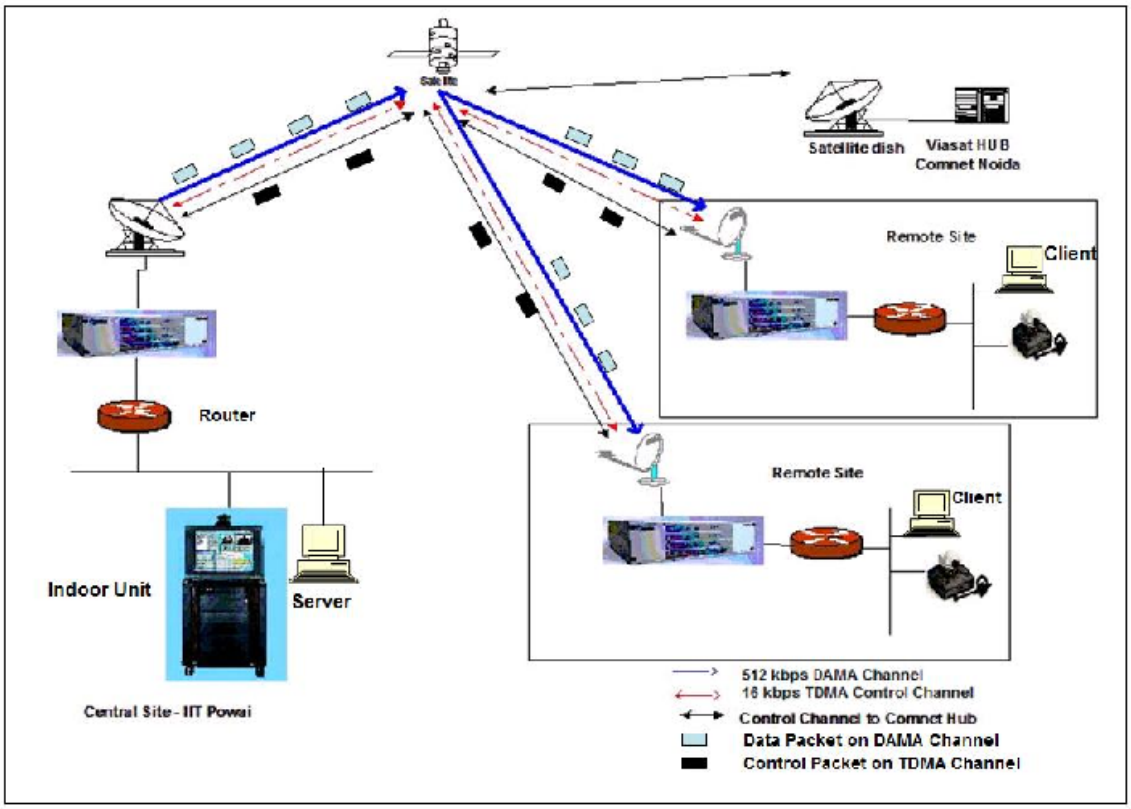

Figure 3.1: RTAN Channel Usage on DEP Network

clients are dropping at a constant high rate, it doesn't help in the case of few unruly clients. If one of the clients is dropping a large number of packets as compared to all other clients, the file transfer would proceed at a snails pace. Since, the file is transferred in parts, the next part is not sent until the previous part is successfully received by all the clients. Hence, one faulting client can make a huge difference in the overall efficiency of the protocol. One solution to this problem is to keep track of the packet drop rates of all the clients and ignore the client if it constantly keeps dropping packets higher than a set threshold.

3.6 Version 1.3 Regional Division of Clients

One of the main features of the basic protocol is to maintain a reliable (TCP) connection with the client for control information exchange while the file transfer proceeds. The server waits for indices from all the clients. If the number of clients become greater than a threshold, the efficiency of the protocol may drop again as the time spent waiting for responses from all the clients may be too large. In such cases, it may be wiser to serve the clients in multiple rounds. The idea is to divide the clients into regions and then pick up a fixed number of clients from each region for the first round. The file is then transferred to all the clients picked for this round. The interesting thing is that while the file is being transferred to the clients selected for a particular round, all other clients which haven't been selected also receive the packets. This is because the transmission is on a multicast channel. Hence, the unselected clients also get some portion of the file while the selected clients get served the whole file. When the time comes at which the unselected clients are picked, they do not require the whole file but only the portion that they dropped during the transfer of the file to the previously selected clients. Hence, throughput of the protocol can be improved.

Chapter 4

Details of Basic Implementation

The protocol has been implemented using a client-server architecture. The server, as explained in an earlier chapter, consists of three main threads - the KeepAlive thread, the ConnectionAcceptor thread and the ControlProcessor thread. The KeepAlive thread keeps sending dummy packets so that link doesn't go down while the transfer is taking place. The ConnectionAcceptor thread keeps waiting for connections from clients. The ControlProcessor thread actually implements the transfer protocol. The client, on the other hand, consists of two main threads - the DataProcessor thread and the ClientControlProcessor thread. The ClientControlProcessor thread opens a TCP connection with the server ControlProcessor thread and interacts for all control operations like sending acknowledgments. The DataProcessor thread receives all the multicast packets sent on the 512 kbps multicast channel. The following sections explain the basic algorithm as well as provide a visual understanding through the associated state transition diagrams, protocol graph and class diagrams.

4.1 Algorithm design

The complete transfer of a file consists of many transfer cycles. Each transfer cycle, in turn, consists of a data multicast and several recovery cycles. The data multicast and the recovery multicast is done on the 512 kbps multicast channel. All the control information is exchanged on the 16 kbps full duplex reliable channel. The following subsections explain the design of the various threads which together constitute the server and the design of the various threads which together constitute the client. Refer to figure 4.1 for the protocol sequence overview.

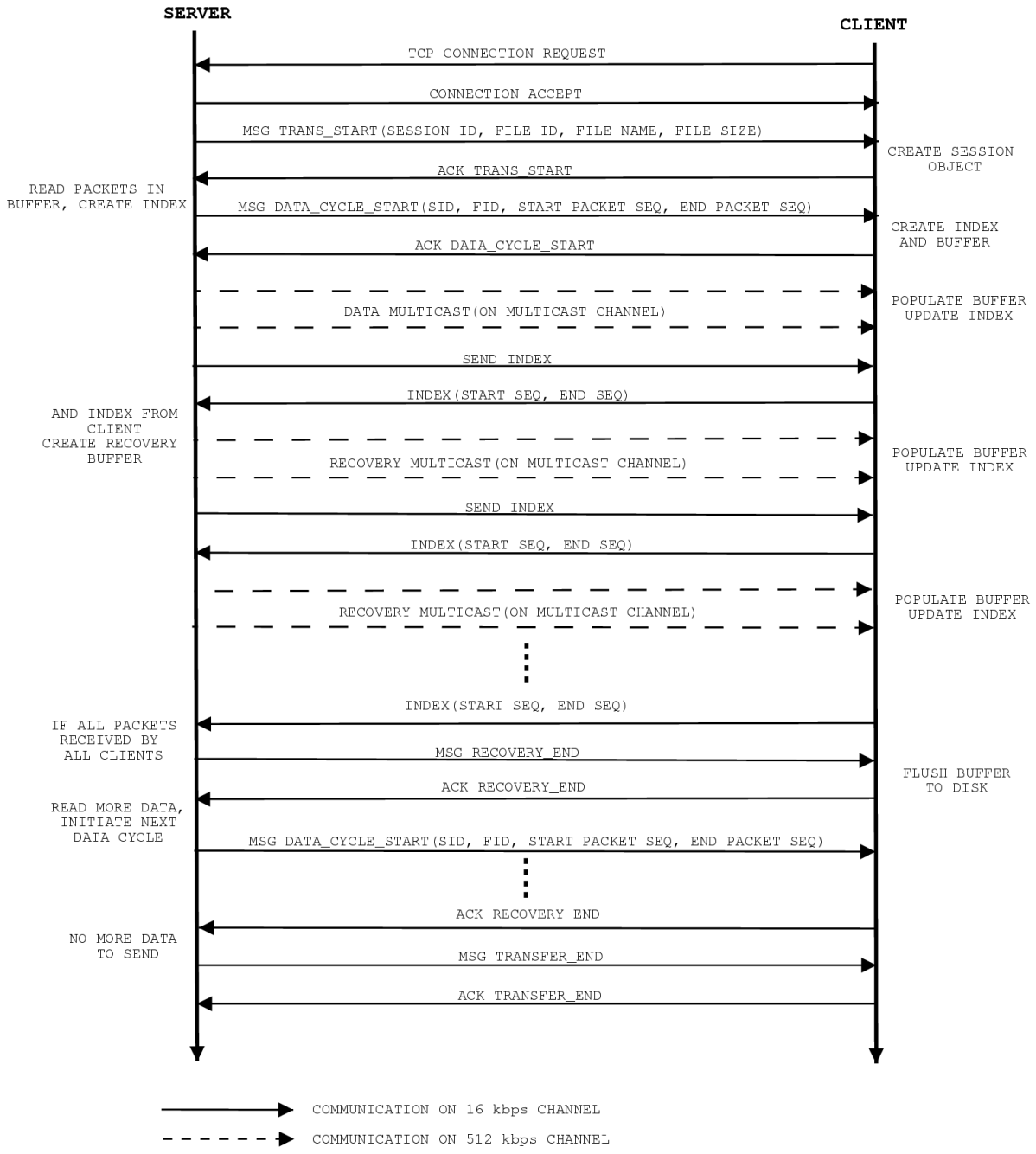

Figure 4.1: The Protocol Sequence

4.1.1 Server Implementation

Refer to figure 4.2 for understanding server state transitions.

4.1.1.1 The Server Main thread

1. Read configuration and create a configuration object for later reference in the program. The main configuration items are the Multicast group, multicast port and TCP port used.
2. Start ConnectionAcceptor thread. This thread waits for clients to connect. This thread runs throughout the life of the program. For each connection request do the following. The corresponding state in the state diagram is the LISTEN state. Go to step 10 on exit request.
   (a) Create a connection object for the client connecting.
   (b) Add the above connection object to a global connection list.
3. Wait for transfer request. The corresponding state in the state diagram is the CLIENT CONNECTED state. Go to step 10 on exit request.
4. Get the file to be transferred. Obtain information such as size of file to be transferred, file name etc. Create a session object for the transfer of this file. If no files are left to be transferred, go to step 3.
5. Start the KeepAlive thread.
6. Invoke the ServerControlProcessor thread which actually performs the transfer.
7. Wait for the ServerControlProcessor thread to finish.
8. Stop the KeepAlive thread.
9. Go back to step 4.
10. Exit. The corresponding state in the state diagram is the CLOSED state.

4.1.1.2 The ServerControlProcessor thread

1. Mark all the clients that are connected to the server as "in session".

   This is because the file is sent to all the clients that are connected at the time of file request. Any clients connecting to the server after a transfer has started are not taken into consideration. The INITIATE TRANSFER state corresponds to this step.

2. Create a transfer start message containing information about the file to be transferred. This information includes the file name and file size. Apart from that a *Sessionid* for the current transfer and a file identifier for the current file being transferred in the current session is also sent. The node FILE INFO SENT denotes the corresponding state in the Server state diagram.

   This is because more than one file can be transferred as a batch in one session. *Sessionids* are required for handling multiple sessions simultaneously in future versions of the system. The transfer start message has to be sent to all the clients in session. The client sends and acknowledgment when it receives the message. In lieu of sending messages to the clients one by one in a loop, separate threads are spawned to deliver the message to all the clients in session concurrently. In fact, separate threads are spawned for all control messages that are to be sent to the clients during the course of transfer. This reduces the processing time required for the entire operation.

3. After acknowledgment from all current in session clients has been received, read a segment of the file to be sent. Check if end of file has been reached. If yes then it signals the end of the transfer. So, go to step 15 else go to step 4. The node READ FILE denotes the corresponding state in the Server state diagram.

4. Fixed sized packets are read from the file and a Transfer Cycle buffer is created. After creating the buffer, create an index for the packets to be sent. Each index entry is initially set to FALSE indicating that no packet has been successfully transferred to ALL the clients in session. The nodes CREATE BUFFER, BUFFER CREATED and INDEX CREATED denote the corresponding states in the Server state diagram.

   The file is sent in parts and in terms of packets as against in terms of bytes. The size of packets is determined by the maximum transmission unit (MTU) of the network which in our case happens to be 1500 bytes. Leaving the headers 1440 bytes can be put as application data in the multicast packet created to transfer each file packet. The number of packets read are predetermined in the basic algorithm and has been reached through live experiments on the satellite network. This number constitutes the "window" of packets that are to be successfully sent to the client in this TRANSFER CYCLE. The future versions will change this predetermined value automatically according to the error rate of the network. . The packets have unique sequence numbers. The index entries correspond to the sequence numbers of the packets.

5. Create DATA_START message which indicates the start sequence number and end sequence number of the packets to be transferred in this transfer cycle. Spawn threads for all the clients in session and send the message. The node TRANSFER CYCLE INDEX INFO SENT denotes the corresponding state in the Server state diagram.

6. After the acknowledgment has been received from all the clients in session, its time to initiate multicast. A thread is spawned which reads the transfer cycle buffer created in step 4, creates multicast packets and multicast them on the 512 kbps multicast channel. The node INITIATE MULTICAST denotes the corresponding state in the Server state diagram.

7. Now, this transfer cycle enters the recovery mode. A complete transfer of a file consists of many transfer cycles. Each transfer cycle, in turn, consists of a data multicast and several recovery cycles. Data multicast denotes the initial multicast of the packets read from the file. Before each data multicast, a DATA_START message is sent indicating the packets that are going to be sent in this transfer cycle. Some of the packets of this initial data multicast are expected to be dropped by the network. Hence, these must be recovered by the server. This recovery is done in recovery mode or recovery cycle. The node RECOVERY MODE denotes the corresponding state in the Server state diagram.

8. Set all the entries in the server index bitmap to TRUE. The node ALL INDEX ENTRIES SET TO TRUE denotes the corresponding state in the Server state diagram.

9. Send message SEND_INDEX to all the clients in session. The node MSG SEND INDEX SENT denotes the corresponding state in the Server state diagram.

10. Each client sends its index for the current transfer cycle to the server. The node INDEX RECEIVED denotes the corresponding state in the Server state diagram.

    The index is a bit index. Each bit corresponds to one packet. If the bit for a packet is set then it indicates that the packet has been received by the client. Otherwise, it is reset.

11. Perform a boolean AND of the client indices with the server index. Create a recovery list of packets that need to be retransmitted.The nodes RECOVERY INDEX CREATED and INITIATE RECOVERY MULTICAST denote the corresponding states in the Server state diagram.

After each client has sent its index, all the packets that have not been received by at least one client must be retransmitted. A boolean AND of all the client indices with the server index yields the desired packet sequence numbers. All packets with their corresponding bit indices reset must be remulticast. A recovery list is created listing such packets.

12. Multicast all the packets in the recovery list. Go back to step 8. Repeat the steps from step 8 to step 11 until recovery list is empty.

    An empty recovery list implies that all clients in session have successfully received all the packets being sent in this transfer cycle. Hence, ANDing indices from all clients with the server index does not yield any packet with its corresponding bit in the index set to FALSE. Hence, this recovery cycle is successful and all the packets have been sent to all the clients.

13. Send RECOVERY_END message to all clients. The node MSG REC_END denotes the corresponding state in the Server state diagram.

    Since, all the packets have been received successfully by all the clients in session, the transfer cycle has come to an end. Hence, RECOVERY_END message is sent to all the clients. On receiving this message, the clients write their buffers to the disk as they have successfully received a contiguous segment of a file.

14. Go to step 3. The steps from 3 to 13 are repeated until the whole file has been sent successfully.

15. Since, all of the file has been sent successfully to all the clients in session, this transfer is complete. The node TRANSFER END denotes the corresponding state in the Server state diagram.

16. Send message TRANSFER_END to all clients. The node MSG TRANSFER END SENT denotes the corresponding state in the Server state diagram.

17. Exit after acknowledgment from all the clients in session has been received.

4.1.1.3 The KeepAlive thread

1. Create a dummy multicast data packet with sequence number 0 and flag byte set to the value 4. Setting the value of flag byte to 4 sets the third bit of the flag byte and resets rest of the bytes. No data bytes are put in the packet.

   The client checks the flag byte and drops the packet if it happens to be a keep alive packet.

2. Send the packet on the same multicast group and port as being used for data transfer.

3. Sleep for 2 seconds.

4. Go to step 1.

4.1.2 Client Implementation

Figure 4.3 provides an overview of the client state transitions.

4.1.2.1 The Client Main thread

1. Read configuration and create a configuration object for later reference in the program.

2. Create a session object for receiving files in this session. This is the global object which contains the buffer and index for the file being received and is shared by the ClientControlProcessor and the DataProcessor.

3. Start the DataProcessor.

4. Start the ClientControlProcessor. The ClientControlProcessor object makes connection request to the server when it is created and establishes a TCP connection. If it fails to establish the connection, it kills the client program. The nodes CONNECT TO SERVER and CONNECTION REQUEST SENT denote the corresponding states in the client state diagram. Go to step 5 if connection is refused by the server.

5. Exit. The node CLOSED denotes the corresponding state in the client state diagram.

4.1.2.2 The ClientControlProcessor thread

1. Wait for messages from the server ControlProcessor. The node WAIT FOR SERVER MESSAGES denotes the corresponding state in the client state diagram.

2. If the message is TRANSFER_START then it receives all the transfer information like the session id, file id, file name, file size. Update session object with the received information. Send an acknowledgment for the message. The node FILE INFO RECEIVED denotes the corresponding state in the client state diagram. Go back to step 1.

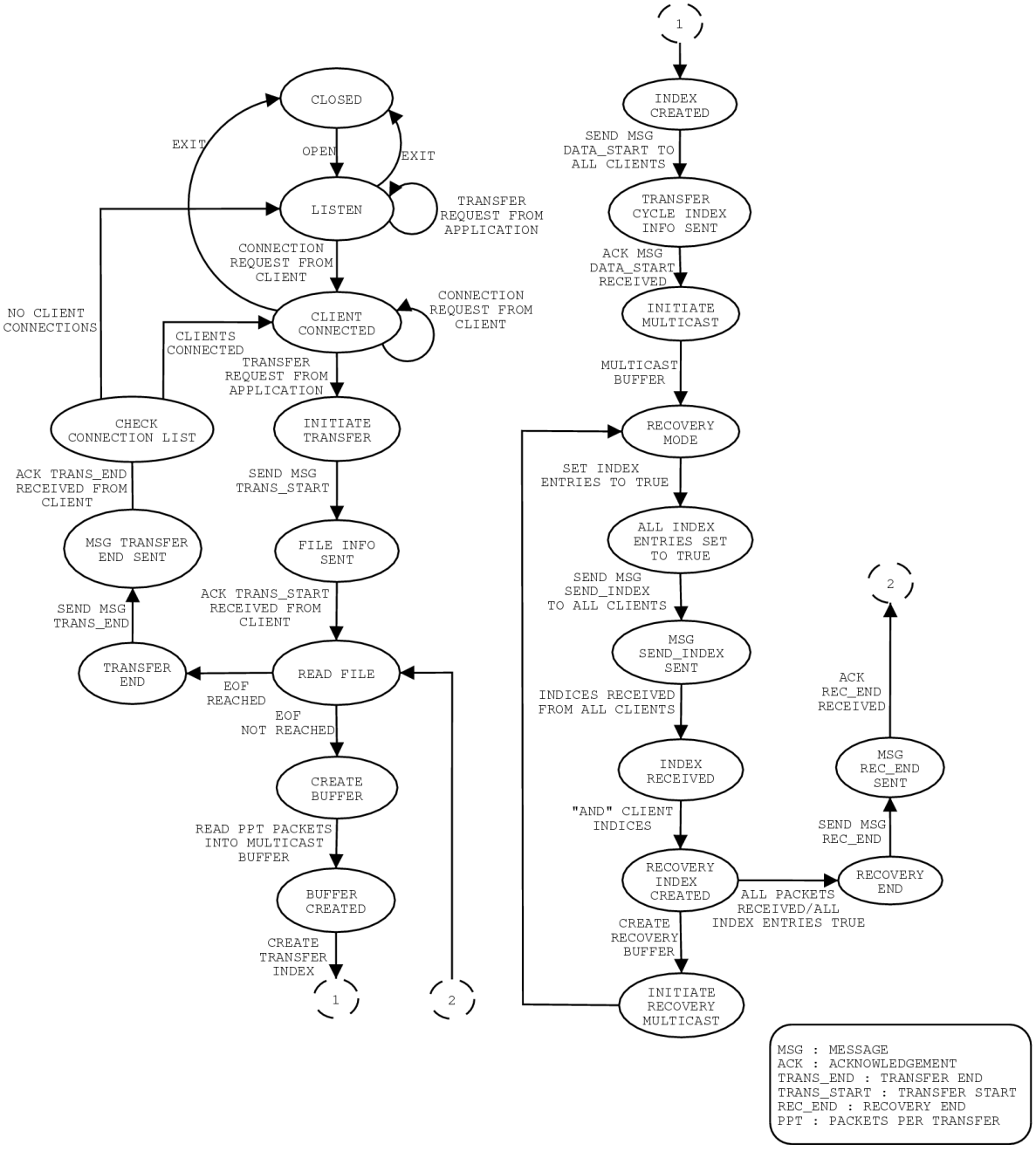

Figure 4.2: The Server State Transition Diagram

3. If the message is DATA_START then create a bit index of the size of number of packets being sent in the new transfer cycle. Reset the whole index or the set each bit value to FALSE. Also, create a buffer to store the received packets. Send an acknowledgement for the message.The nodes TRANSFER CYCLE INFO RECEIVED and INDEX AND BUFFER CREATED denote the corresponding states in the client state diagram. Go back to step 1.

4. If the message is SEND_INDEX then write the index to the TCP socket or send the index to the server ControlProcessor. The node SEND INDEX denotes the corresponding state in the client state diagram. Go back to step 1.

5. If the message is RECOVERY_END then all the packets have been received by the client. Flush the buffer to the disk. Discard both the buffer and the index for the current transfer cycle. The nodes TRANSFER CYCLE END and TRANSFER SEGMENT WRITTEN denote the corresponding states in the client state diagram. Go back to step 1.

6. If the message is TRANSFER_END then the whole data or file has been received by the client. The node TRANSFER OVER denotes the corresponding state in the client state diagram. Go back to step 1.

4.1.2.3 The DataProcessor thread

1. Listen for the multicast packet.

2. If the packet received is a dummy keep alive packet then discard it. The FLAG value in the received packet indicates whether the received packet is a data packet or a keep alive packet. Go to step 1.

3. If the packet received is data packet then if its corresponding index is set then discard the packet. If the corresponding entry is reset then it hasn't been received earlier and hence must be stored in the buffer. Store such packets in buffer. Go to step 1.

4.2 The Multicast Data Packet

The multicast data packet consists of two parts - the header and the payload. The contains information about the data in the multicast packet and the payload contains the actual data that has to be transferred. The size of the multicast packet can be changed

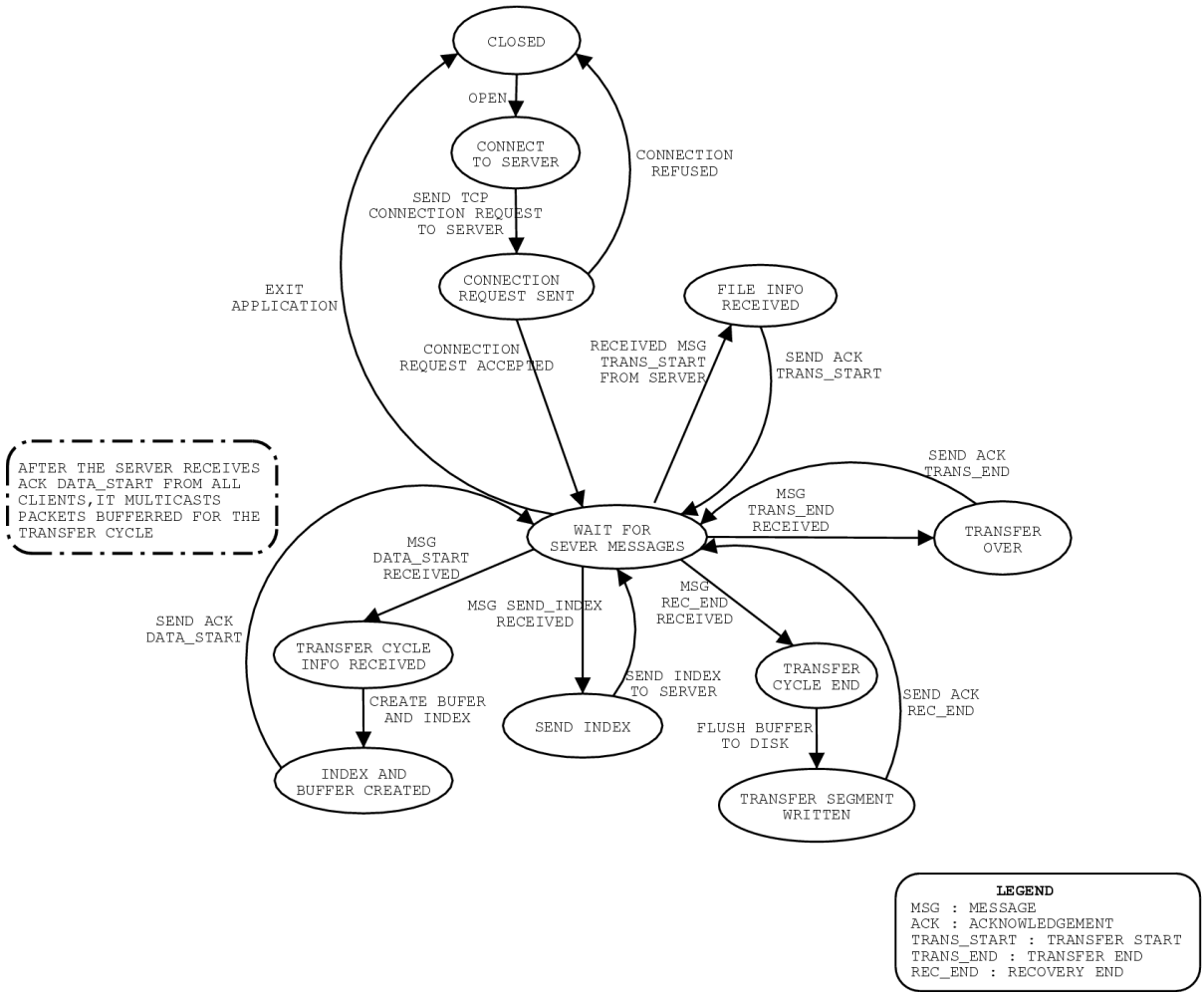

Figure 4.3: The Client State Transition Diagram

by changing the configuration file. The maximum transmission unit (MTU) of the DEP satellite network is 1500 bytes. Barring the various headers, 1440 bytes of application data can be put in each MTU. Hence, the maximum size of multicast data packet can be 1440 bytes.

The header size is 17 bytes. It consists of the following fields:

- Session Id (4 bytes): Session identifier distinguishes between sessions. Currently, only one session can be handled. Future versions of RTAN would be able to manage multiple sessions simultaneously. This implies that RTAN would be able to transfer two different files to two different set of clients utilizing the 512 kbps multicast channel to the optimum.
- File Id (4 bytes): Identifies the file being transmitted. Each file is assigned a unique identifier by RTAN before the transmission begins.
- Sequence Number (4 bytes): The sequence number of the packet being transmitted.
- Payload Size (4 bytes): The number of data bytes contained in this packet.
- Flags (1 byte): Indicates whether the received packet is a data packet or a dummy keep alive packet. One bit is being used for indication of packet type. Rest of the bits are reserved for future use.

4.3 Class Diagrams

The class diagrams for RTAN server and client are shown in figures 4.4 and 4.5 respectively.

4.3.1 The Server Class Diagram

The following must be noted about the server class diagram as shown in figure 4.4:

1. MFTPServer is the main class. This class invokes various threads and creates various objects which together comprise the Server. This class intializes the classes:
   (a) TransferBatchReader which reads the names of all the files to be transferred in a batch,
   (b) Config which reads the configuration,
   (c) ConnectionList which has a global list for storing all the incoming connections,
   (d) ServerControlProcessor which implements the control part of the Server,

(e) KeepAliveThread which keeps the link alive by sending dummy packets periodically,

(f) ConnectionAcceptorThread which listens for incoming client connections and

(g) MFTPSession which maintains information about the session like file name, file id, buffer, index etc.

2. The class ConnectionAcceptorThread has a reference to the global ConnectionList object. As a new client connects, the ConnectionAcceptorThread object creates a Connection class object for it and stores it in the ConnectionList.

3. The class ServerControlProcessor actually implements the server side RTAN protocol. It has an object of class BufferAndIndexHandler for managing the server buffer and index. It also creates ControlCycleImplementor objects for implementing various control cycles and MulticastCycleImplementorThread object for performing data multicast.

4. The class BufferAndIndexHandler uses the object of class FileDataReader to read data bytes from the file being transferred and stores them in the buffer.

5. The class FTPPacket provides method to create a multicast packet with the RTAN header attached to the buffer data bytes when provided with appropriate values. FTPPacket is used by the class MulticastCycleImplementorThread.

6. The class ControlCycleImplementorThread implements various cycles constituting the protocol and interacts with the clients for exchanging control information.

7. The classes FileIndex and FileDataBuffer implement index and buffer respectively. The classes MessageDataStart and MessageMFTPStart provide message objects of appropriate type.

8. The class RecoveryList provides methods to create a list of all the packets dropped by the clients after consulting the server index.

4.3.2 The Client Class Diagram

The following must be noted about the client class diagram as shown in figure 4.5:

1. The class MFTPClient is the main class. It instantiates various classes which together constitute the client.

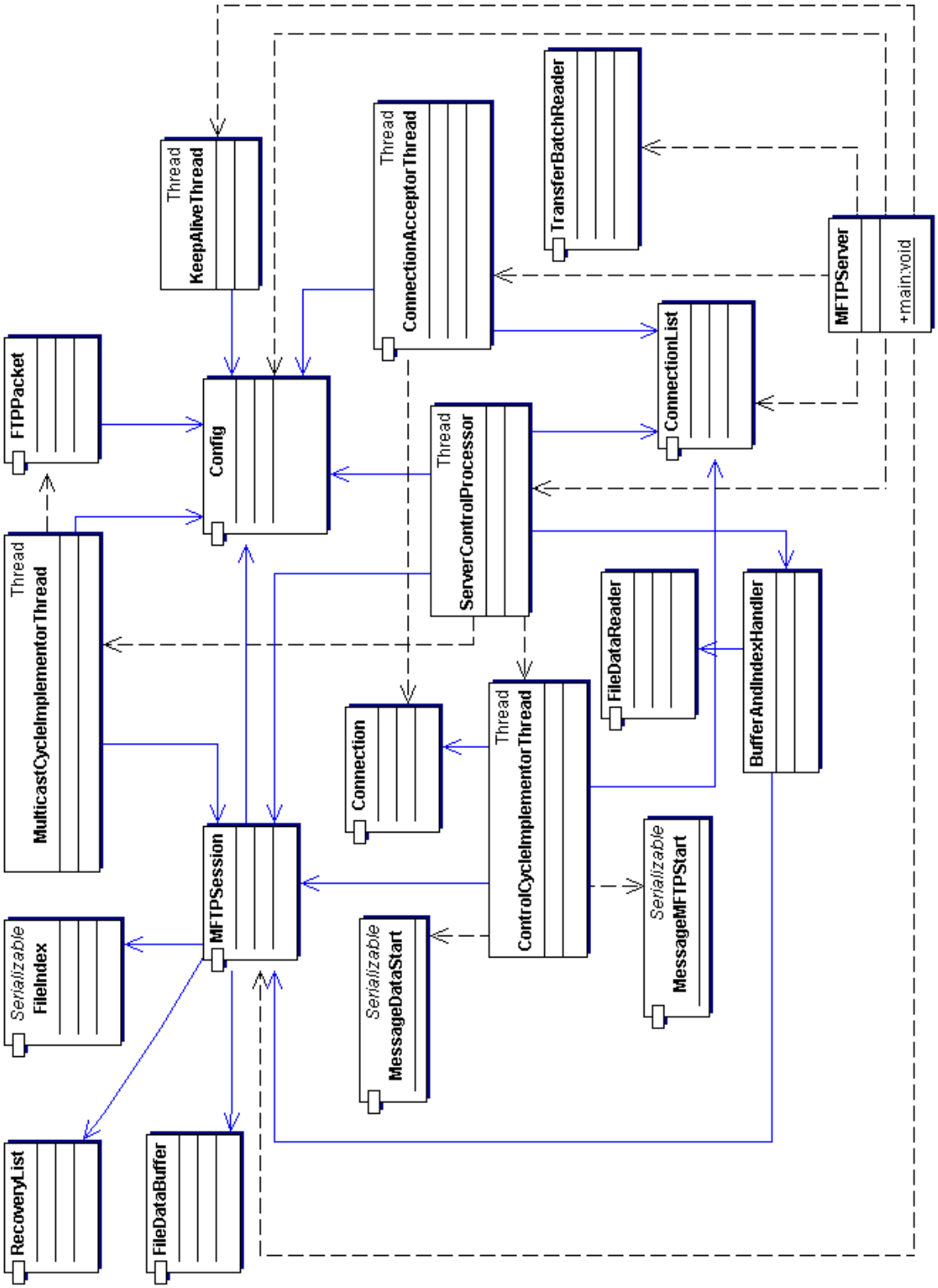

Figure 4.4: Server Class Diagram

(a) Config which reads the configuration,

(b) ClientControlInfoProcessor which implements the control part of the Client,

(c) MFTPSession which maintains information about the session like file name, file id, buffer, index etc.

(d) ClientDataProcessor which reads the packets multicast by the server.

2. The class FTPPacket provides method to get the various values out of the received multicast packet such as the sequence number of the packet, the file data bytes in the packet etc. FTPPacket is used by the class ClientDataProcessor to populate the buffer, change index values etc.

3. ClientControlInfoProcessor has an object of class MessageHandler which has methods to deal with all the messages sent by the server. These methods perform the required control information exchange with the server.

4. The classes FileIndex and FileDataBuffer implement index and buffer respectively. The classes MessageDataStart and MessageMFTPStart provide message objects of appropriate type.

4.4 LAN Simulation

Before being deployed on the DEP satellite network, the protocol implementation was thoroughly tested on a 100 Mbps LAN. Experiments were set up by installing the server on a host machine and multiple clients on separate host machines. Files of varying sizes were successfully transferred over LAN. In one set of experiments, the server data rate was kept at around 6 MBps. This resulted in observable drop of about 15% largely because of the receiver buffer overflow at the client side. The protocol implementation was able to recover all the dropped packets. The received file was found to be uncorrupted and of the same size as sent. A throughput of around 4.6 MBps was achieved. In a second set of experiments, the server was throttled according to the actual data transmission rate of the multicast channel of the satellite network and delays incorporated to account for the time required to exchange control information on the 16 kbps control channel on the satellite network. These simulations provided a fair idea about the time required to transfer the file on the actual satellite network. Now, the protocol implementation was ready to be deployed on the actual network, the DEP satellite network.

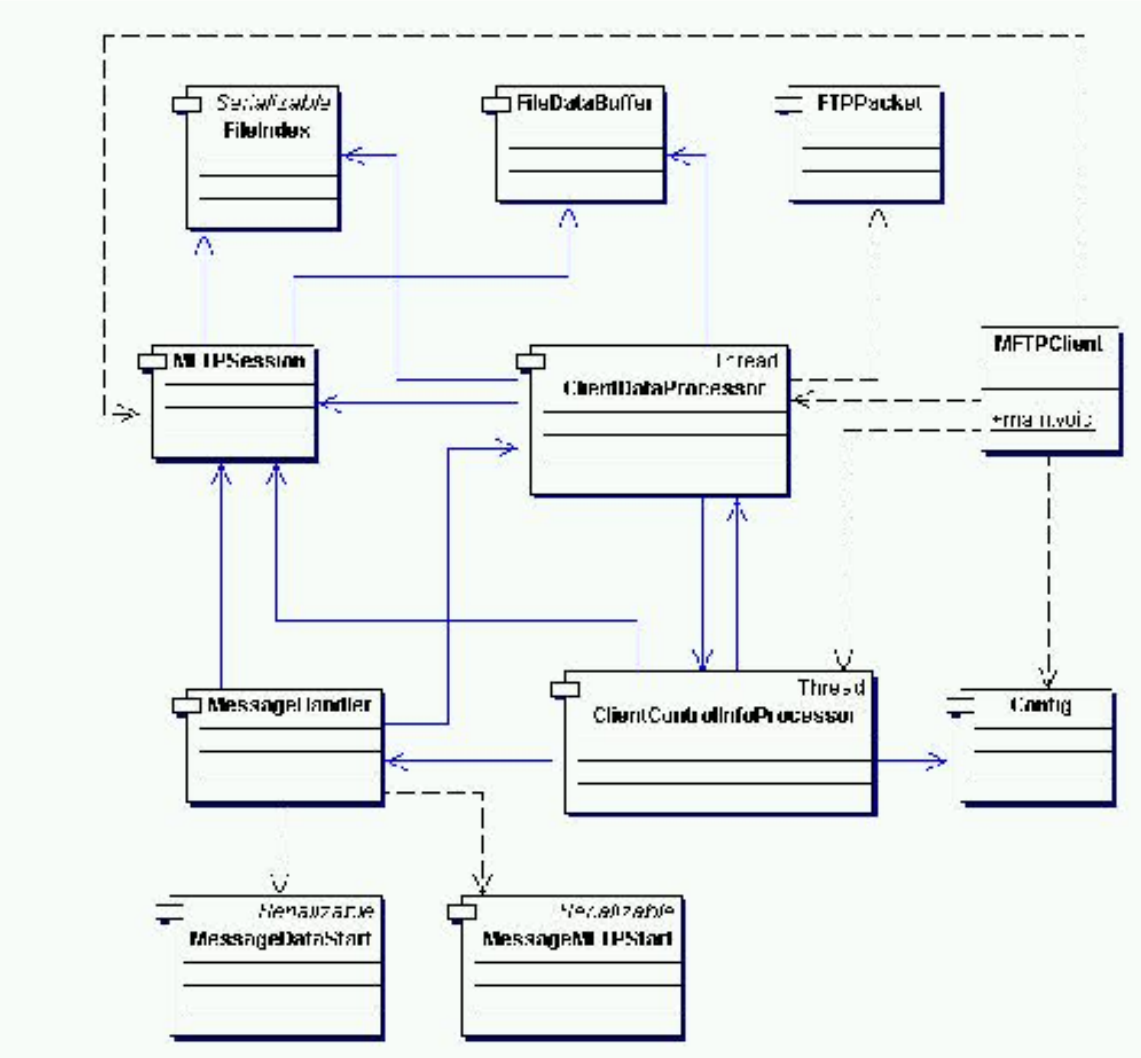

Figure 4.5: Client Class Diagram

Chapter 5

What you foresee is not what you get

Once the protocol was successfully implemented on LAN, it was time to test it on the Distance Education Program (DEP) satellite network. The server was deployed at IIT Bombay. The client was deployed at one of the remote centres, specifically at National Centre for Software Technology (NCST), Juhu, Mumbai. Since, the protocol implementation was working fine on LAN, it was expected to work at the live setup on the DEP satellite network also. But the implementation failed due to various unforeseen problems related to the DEP satellite network. This led to a few changes in the protocol. The following sections describe the various problems which led to the failure of the protocol when it was ported on the DEP satellite network. The observations, guesses, causes and the solution to problems are also described.

5.1 The Router Configuration problem

- **Naive Assumption:** Since, the protocol is working correctly on LAN, it should work correctly on the DEP satellite network also.
- **Observation:** Although, the messages were getting through on the control channel, the client running at the remote centre seemed to be losing all the multicast packets sent.
- **The Guess:** The problem might possibly be in inappropriate router configuration or the multicast group being used was not allowed on the satellite LAN.
- **The Cause:** The cause was the routers present in the DEP satellite network which are absent in LAN. The DEP network consists of routers connecting various centres.

Every host machine that needs to send data to a multicast group or receive multicast packets must be added to the multicast route at the router of the corresponding centre. The machine IP address and the multicast port on which the multicast data will be sent or received must be added to the configuration of the router.

Since, the router had not been configured for the host on which the server and the clients were being run, the multicast packets were getting dropped by the router. One important point is to check the multicast group being used for multicasting. The multicast groups are assigned by the network providers. On a proprietary network, a multicast group chosen at random will not work though the same multicast group will work fine for a LAN setup.

- **The Solution:** The router configurations at IIT Bombay and NCST were modified to add the machines hosting the server and the client respectively to the multicast route. We have used the same multicast group as used for DEP lecture transmissions. The port number at which the multicast traffic is sent was chosen randomly and the routers at both ends were configured properly.

5.2 The Time To Live problem

- **Naive Assumption:** Since, the protocol is working correctly on LAN and the routers at centres hosting server as well as client have been configured for the multicast group properly, it should work correctly on the DEP satellite network also.

- **Observation:** Although, the messages were getting through on the control channel, the client running at the remote centre seemed to be losing all the multicast packets sent.

- **The Guess:** Either the packets were being dropped at some router in between the source and destination or were not being forwarded by the satellite modem at the sender end.

- **The Cause:** The cause was the Time To Live(TTL) value of the IP packets and the presence of routers in the network. When an IP packet arrives at a router, they decrement the TTL field of the packet. If the TTL value becomes zero after decrementing, routers drop that packet.

  The TTL field of the IP packets was being set to 1 by the operating system on the host machine running the server. When the mutlicast packet generated by the

server reached the router, the router decremented its value to 0 and dropped it. Thats why the client was unable to receive any multicast packets as all the packets were getting dropped at the server router itself. As there are two routers in the server to client path, the TTL value must be greater than 2.

- **The Solution:** The TTL value was randomly set to a value greater than equal to 3, specifically 5. Since, there are 2 routers in the path from the server to the client, one router at the IIT Bombay end and the other at NCST end, the TTL value must at least be 3. It must be noted that setting TTL to 1 by the source is illogical but was happening nevertheless.

5.3 The Router Table Update problem

- **Naive Assumption:** Since, the protocol is working correctly on LAN, the routers have been configured for the multicast group properly and the Time To Live value of the multicast packets have been set properly, it should work correctly on the DEP satellite network also.

- **Observation:** Although, the messages were getting through on the control channel, no multicast packets were getting across for the first couple of seconds. After that a limited number of packets got across to the client but the rest got dropped. The number of packets getting across more or less remained fixed.

- **The Guess:** The router was taking time for multicast route discovery and the router buffer was getting overwritten by too high data rate.

- **The Cause:** The cause was the table update time required by the router for the multicast route on which the packets were being sent. The router takes a few seconds to update its routing tables for obtaining the route for the multicast group to which the packets are being sent. For that period, any packets arriving at the router for that multicast group are dropped by the router.

- **The Solution:** For the DEP satellite network, the architects define this period to be around 5 seconds. The experimentally found out value comes out be 2 to 3 seconds. The only solution for this problem is to send a packet for the desired multicast group and then wait for the next couple of seconds before sending any more data.

5.4 The Link Data Rate Synchronization problem

- **Naive Assumption:** Since, the protocol is working correctly on LAN, the routers have been configured for the multicast group properly, the Time To Live value of the multicast packets have been set properly and the initial waiting time has been introduced to account for the router table update, it should work correctly on the DEP satellite network also.

- **Observation:** In every multicast cycle in which a fixed number of packets were sent, a limited number of packets were getting across to the client but the rest got dropped. The number of packets getting across more or less remained fixed. Also, all multicast packets got dropped for the next 20 seconds. After this, some packets again got across. The above cycle repeated continuously until there remained no data to multicast.

- **The Guess:** The router dropped all packets after a fixed time had expired and was deleting entry for that multicast from its routing tables.

- **The Cause:** The cause was the difference in the data rate of the local LAN hosting the server and data rate of the satellite link. The LAN data rate is 10 Mbps which is far higher than the satellite link data rate which is 512kbps.

  Since, the multicast packets were being generated at a far higher rate than what the router could transmit, the packets were getting dropped at the router itself. So, multicast data must be generated in sync with the data rate of the link on which the data has to be sent. Though this may seem trivial, this point may remain unnoticed owing to attention being diverted to solving other unforeseen problems first. Also, there is no reliable method to generate data at the source at a rate exactly matching the link data rate. The data rate at the source may be approximated by putting the program to sleep for a few milliseconds. But this method leads to unreliability as the context switch time is unpredictable. Also, it varies from host machine to host machine. Hence, exact sleep time has to be found out experimentally.

- **The Solution:** Appropriate sleep was introduced in between the generation of consecutive multicast packets so as to keep the data generation rate of the server not more than the satellite link capacity. The actual data rates have been reported in the chapter on field experiments.

5.5 The Link Up-Down problem

- **Naive Assumption:** Since, the protocol is working correctly on LAN, the routers have been configured for the multicast group properly, the Time To Live value of the multicast packets have been set properly, the initial waiting time has been introduced to account for the router table update and the multicast packet generation rate of the server has been synchronized with the link speed, it should work correctly on the DEP satellite network also.

- **Observation:** After every multicast cycle in which a fixed number of packets were multicast, all multicast packets got dropped for the next 20 seconds. After this, one multicast cycle was successful again. The above cycle repeated continuously until there remained no data to multicast. There were a few packet drops in each successful multicast cycle also. On debugging the router it was observed that the link went down for 20 seconds after each successful multicast.

- **The Guess:** The router debug trace showed that the link was being brought down after a fixed interval and remained at that state for the next 20 seconds. This explained the observed packet drop period of 20 seconds while running the program. The guess was that the routers were indulging into some kind of dynamic route discovery or route updates very frequently. Hence, the dynamic routing had to be disabled at the router and static routing used.

- **The Cause:** The above problem was due to the design of the DEP satellite network. The router drops the link for the multicast group if no packet is received on the port for that group for a time interval of 2 seconds. The router removes the entry for that multicast group from its routing tables. The link is brought up again only after 20 seconds. All multicast packets at that port are dropped for the period for which the link remains down.

  As outlined in the protocol described in previous chapters, after each multicast cycle in which a fixed number of packets are sent by the server, the server waits for an index from the client. This control operation of querying the client for the index and receiving the index from the client takes approximately 3 to 4 seconds. For this time, the server did not send any multicast packets. Hence, the router removed the entry for the multicast group and brought the link down.

- **The Solution:** The solution to the above problem is to keep sending at least one dummy keep-alive packet at the concerned multicast port at the router. The server

was modified by adding a KeepAlive thread which kept sending the required dummy packets every 2 seconds.

Chapter 6

Field Experiments

Once various problems in deploying RTAN were taken care of and data transfer had become successful over the DEP satellite network, a number of field experiments were performed to get a fair view of the performance of the protocol and the various parameters effecting the performance. This chapter explains the various field experiments done and results obtained. Before the experiments are described, the initial sections explain a few observations about the DEP network and the protocol implementation which effect the experiments performed.

6.1 Data Rates

It is necessary to synchronize the data rate of the sender with the data rate of the network in order to get optimum performance. The multicast channel of the DEP satellite network, which bears the brunt of data transfer, has a data rate of 512 Kilo bits per second or 64 Kilo bytes per second. The host application, on the other hand, can generate data a very high rate. This rate depends on the processing time required for generating data packets which can be quite less. Hence, the application data rate can be quite high as compared to the satellite link capacity. Hence, the host application data rate must be synchronized with that of the link. Unfortunately, exact synchronization is not possible. This is because there is no sure way of stopping the host application for a specified period of time in between sending consecutive packets. As observed, apart from the processing time required for generating the multicast packet a sleep of few milliseconds must be put in between two packets. For a 512 kbps link having maximum transmission unit of 1500 bytes, consecutive packets must be generated every 23 milliseconds to match the link capacity. These 23 milliseconds must account for both the processing required and the extra sleep time that may be required. The processing time is hard to guess and varies from host machine to host machine depending on machine's configuration (clock rate etc).

Also, its not possible to generate a sleep of a few exact milliseconds. For example, the cumulative sleep time generated by inserting a sleep of 1 millisecond and 10 milliseconds is the same as observed for the host machine used for our experiments. Similarly for the ranges 11 to 20, 21 to 30 and there after. So, the granularity of the sleep time was found to be 10 milliseconds. Hence, exact data rate matching is not possible. We experimented with the data rates of 24, 30, 37.5, 49.5 and 75 kilo bytes per second.

6.2 Error rate of the satellite network

The satellite network has been observed to have very low error rate. At a data transmission rate less than that of the link capacity, very less packets get dropped, less than 0.5% during good channel conditions. But a drop of 2 to 3 percent is observed during early morning hours. As long as the data rate is lesser than the satellite link transmission rate, the error rate adheres to the above conclusions irrespective of the number of packets being sent per transfer cycle. But at a data rate of more than 64 kilobytes per second substantial packet drops occur owing to the buffer overflow at the sender router. At an experimental data rate of 75 kilobytes per second, 10% to 12% packets get dropped. This drop rate is found to be independent of the number of packets being sent per transfer. Even at this comparatively higher drop rate, least transfer times are achieved when data rate is kept at 75 kilobytes per second. 75 KBps is the only data rate that we were able to achieve above the link data rate of 64 KBps. Since, the error rate of the network has been found to be very low, the experiments focus on the transfer time variation with respect to varying packets sent per transfer cycle, file sizes and host data rates.

6.3 Data packet size

The Maximum Transmission Unit (MTU) of the DEP satellite network is 1500 bytes. The maximum application data that can be put into one MTU is 1440 bytes excluding the various headers. The multicast packet has a 17 byte header of its own. Hence, the maximum number of data bytes to be transferred can be 1423 per MTU. We have used 1400 bytes of data bytes per MTU in all the experiments.

6.4 Experiment 1: Transfer Time vs File Size

The experiment consisted of sending files of various sizes ranging from small files of size 4 MB to large files of size 122 MB and noting the time taken for the transfer. The intent was to measure the effect of control information exchange that occurs during each

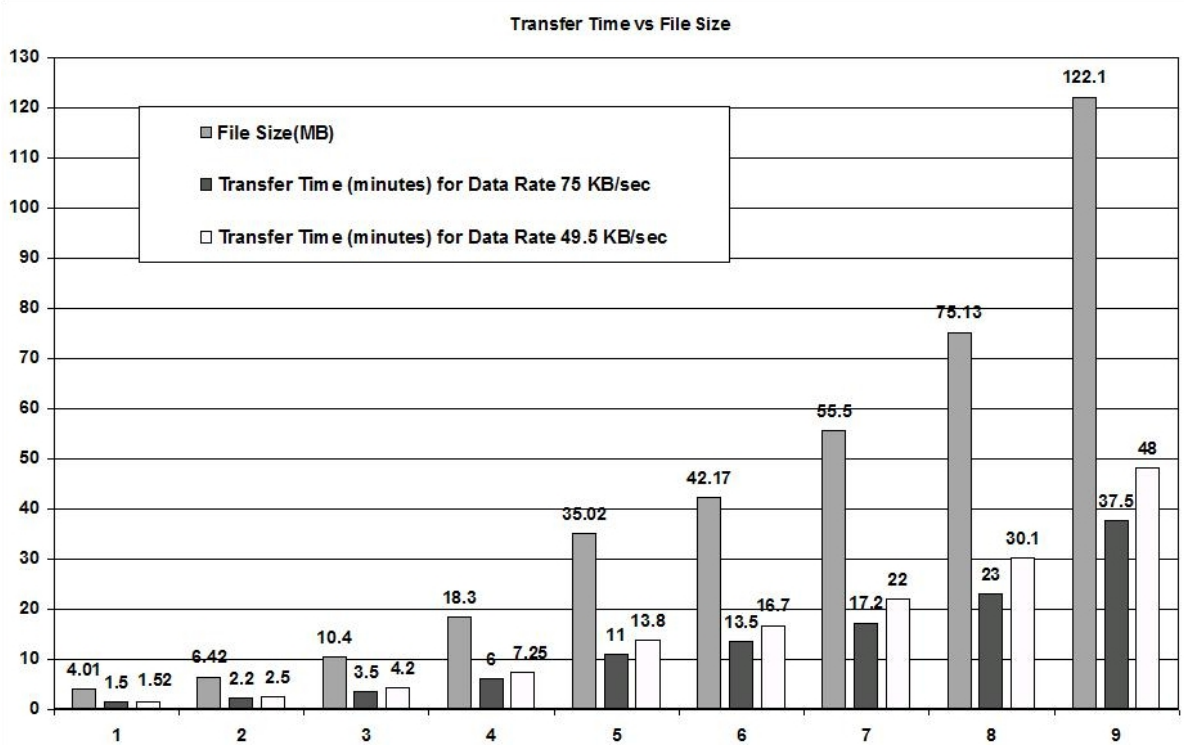

Figure 6.1: Effect of File Size on Transfer Time

transfer cycle on the total transfer time. For a larger file, more transfer cycles are needed to transfer the file. The value of packets per transfer cycle was kept at 10,000 for the experiment.

As can be seen from figure 6.1, the transfer time to file size ratio for smaller files is less as compared to larger files. For small files less than 18 MB, the time consumed in exchanging control information has more weightage in total transfer time as opposed to larger files. Hence, the transfer time to file size ratio is larger for small files. The transfer times for all the files are shown for two different data rates in the figure.

6.5 Experiment 2: Transfer Time vs Packets Per Transfer

The intent of the experiment was to find out the effect of time spent in exchanging control information per transfer cycle on the total transfer time of the data. Varying the number of packets sent per transfer cycle should give an appropriate idea as smaller number of packets per transfer cycle would lead to higher total transfer cycles and vice versa. Hence, a fair idea about the above can be obtained from the experiment. The data rate for the

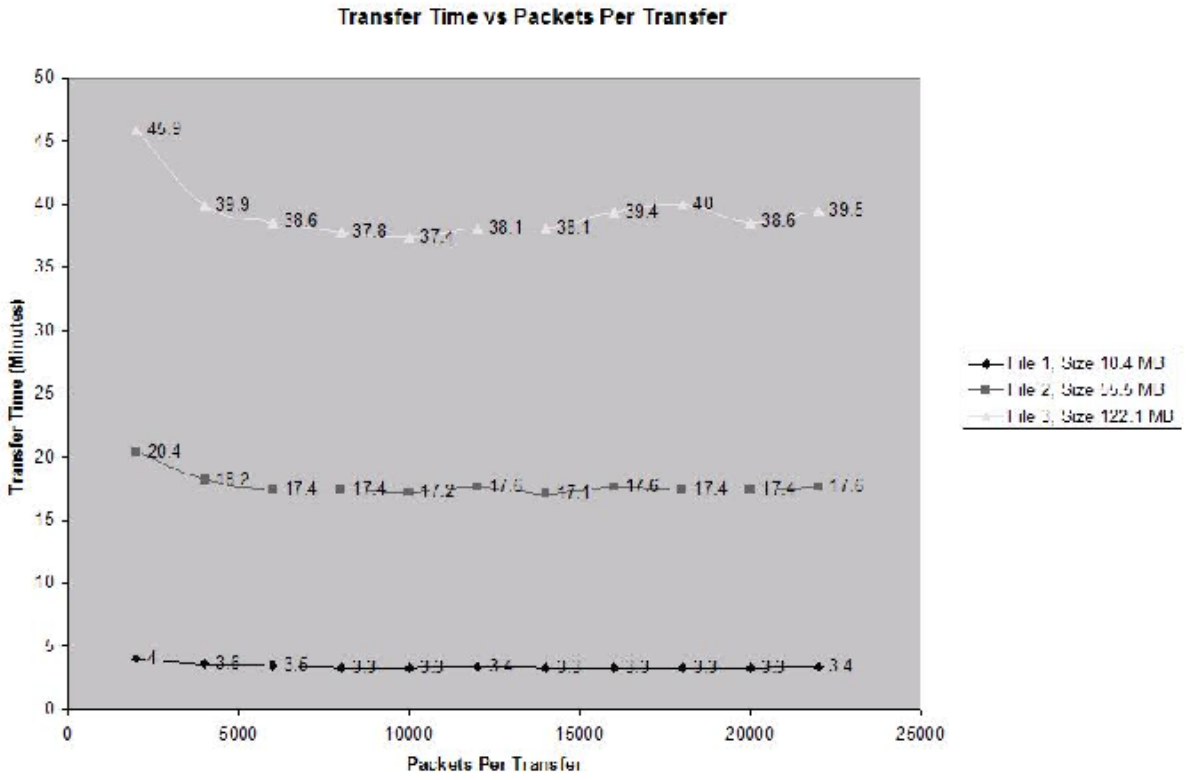

Figure 6.2: Effect of Packets Per Transfer on Transfer Time

experiment had been set to 75 kilobytes per second. The experiment was performed for three different file sizes. The initial value of packets per transfer was set to 2000 and was changed in increments of 2000.

As can be seen from the figure 6.2, for all file sizes, the transfer time remains nearly constant with in a minute for nearly all packets per transfer values. For larger files, a substantial difference in transfer times can be observed for a values less than 4000 packets and greater than that. The lesser values result in larger transfer times as compared to values greater than 4000. An optimum transfer time seems to be achieved at a value of around 10000 packets per transfer. Hence, it can be concluded that the main component of data transfer time is governed by the data multicast time on the multicast channel. The control information exchange does not contribute much to the total transfer time.

6.6 Experiment 3: Transfer Time vs Data Rate

A file of size 42 MB was transferred with a constant value of 10000 packets per transfer and transfer times measured for varying data rates.

As can be seen from the figure 6.3, the lowest transfer time is achieved when sending

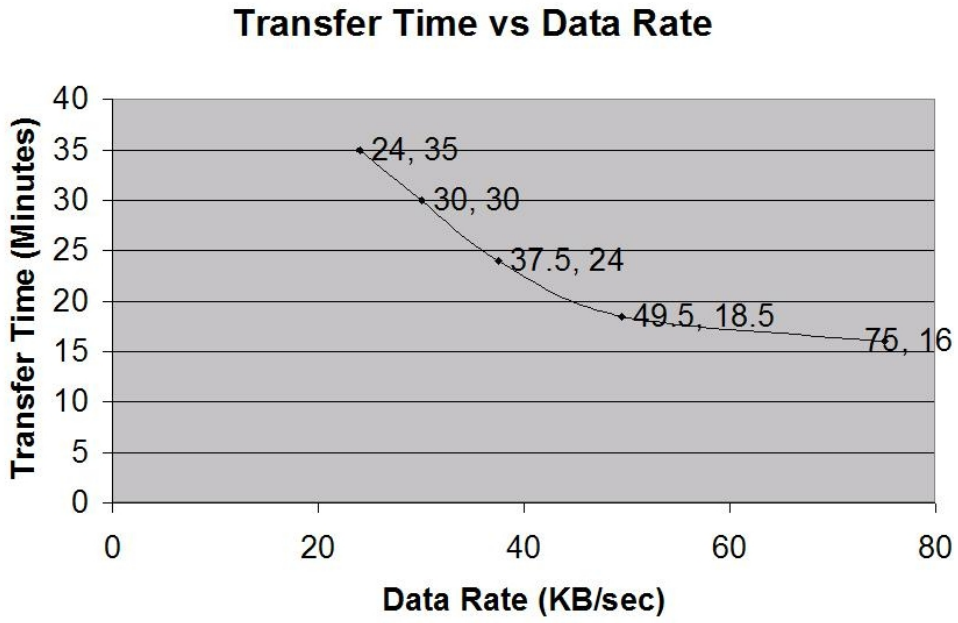

Figure 6.3: Effect of Data Rate on Transfer Time

at a rate of 75 kilobytes per second (which is slightly greater than the link capacity of 64 kilobytes per second). Further, there is not much difference in transfer time at data rates of 75 and 49.5 (lesser than link capacity and the closest). There is steady increase in transfer times, as can be expected, as the data rate is further reduced.

6.7 Experiment 4: Transfer Time vs Packets Per Transfer at Different Data Rates

The intent was to find out the effect on the total transfer time of data on varying packets per transfer cycle value when the data rate is changed. A file of 75 MB was transferred at two different data rates of 37.5 kilobytes per second and 75 kilobytes per second.

As can be seen from the figure 6.4, the main contributing factor to transfer time is the data rate and the value of packets transferred per transfer cycle contributes only minorly to the overall transfer time.

6.8 Conclusions from the Experiments

The following conclusions can be drawn from the performed experiments.

- There is a greater effect of time spent in exchanging control information on the 16

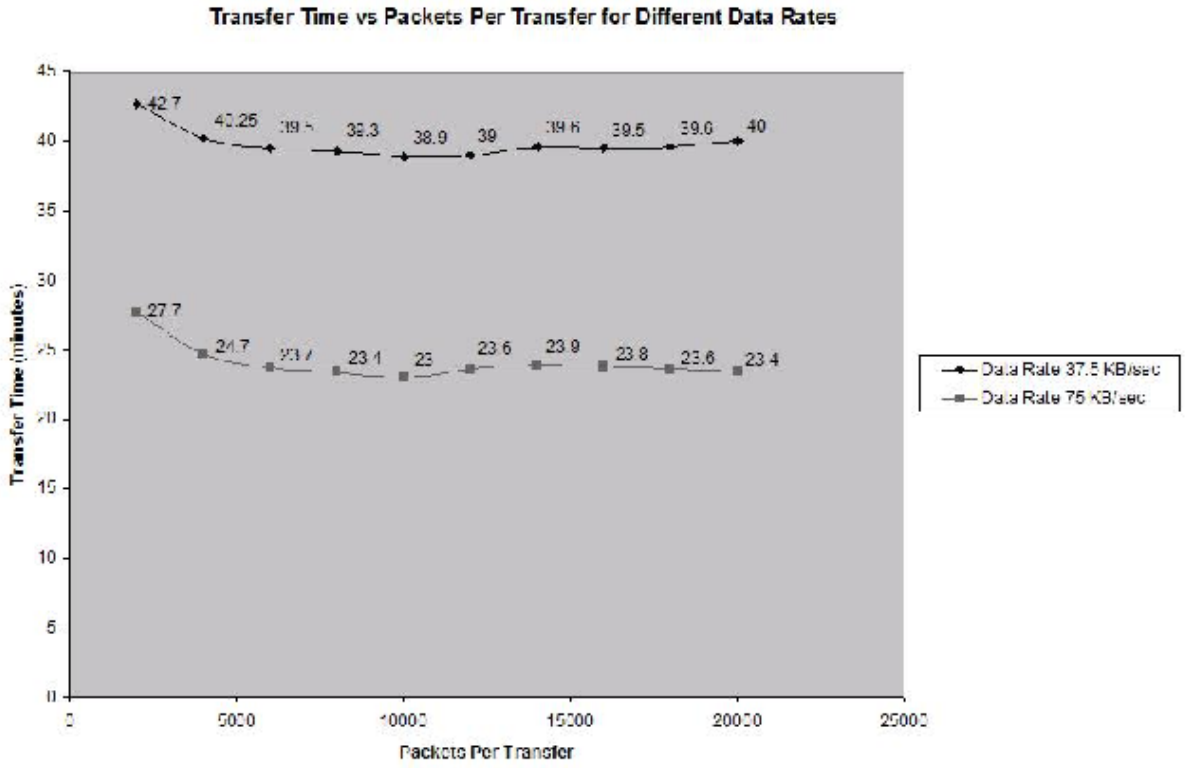

Figure 6.4: Effect on Transfer Time on varying Packets Per Transfer at different Data Rates

kbps link on the overall transfer time in the case of small files less than 18 MB as compared to larger files. For larger files, this effect slowly stabilizes.

- The data rate is the main contributing factor in determining the overall transfer time. Any value of *packets per transfer* greater than 6000 doesn't make a significant difference to the overall transfer time for any file size.
- Data rates closest to the multicast link capacity result in maximum throughput.

Chapter 7

Conclusions and Future Work

7.1 Work Accomplished

A protocol for reliable transfer of data over asymmetric multicast networks was proposed through this work. The proposed protocol is general enough to be applicable to a variety of asymmetric networks having a low bandwidth full duplex unicast channel and a high bandwidth half duplex multicast channel. The protocol builds over the services provided by the transport layer in the TCP/IP protocol stack. The central idea is to use high bandwidth multicast channel for data transfer and use low bandwidth unicast channel for control information exchange in order to perform loss recovery. Specifically, the protocol was implemented in developing a file transfer utility for the Distance Education Program (DEP), IIT Bombay. The requirement was to transfer files ranging from lecture videos of the order of one gigabyte to small tutorials of the order of few kilobytes to various Remote Centres(RCs) scattered across the country. DEP is run on a VSAT satellite network topology. The file transfer utility was successfully developed and deployed over the satellite network to transfer the required files.

The basic protocol divides the data to sent into packets of fixed size m and sends them in groups of fixed size N. After each group is sent, the sender requests a bit index from each client which indicates the receipt or drop of each of the N packets in the group. The sender multicasts all those packets which have not been received by at least one client. Various optimizations to the basic protocol have also been proposed which are discussed in the following section.

Field experiments on the DEP satellite have shown the link to be very stable and having very low error rate. Hence, no error model could be learnt from the satellite network. Thus, the basic protocol works quite well for the DEP satellite network.

7.2 Optimizations

In certain scenarios, the basic protocol may not perform efficiently thus reducing the overall throughput. In order to maintain acceptable levels of throughput, various optimizations have been proposed. These optimizations are being implemented in various versions of the basic protocol.

Satellite links are prone to weather conditions and other external parameters. At very high error rates, most of the packets multicast may get dropped. Then it may be a good idea to multicast all the packets being sent per cycle more than once consecutively. Thus, the clients may be expected to have received a larger fraction of packets successfully as compared to one multicast per cycle scenario. Hence, the recovery cycle can be small.

One of the major drawback of the basic protocol is that it is only as fast as the most faulting client. When the link is exhibiting good throughput in general but some clients are experiencing high packet drop rates then it may be good idea to ignore those clients. Hence, all the clients with good reception rate are not slowed down unnecessarily because of the faulting clients.

Scalability is also a major concern when designing multicast protocols. The basic protocol queries all clients connected in session for an index indicating the receipt or drop of packets on the low bandwidth control channel. The efficiency may drop drastically if large number of clients of the order of hundreds need to be served both because of greater computation time required to process all client indices as well as the inability of the low bandwidth control channel to open connections for all the clients. One idea is to serve clients in iterations. When a set of clients are being served, the other clients also keep listening to the multicast. Hence, by the time file transfer is over for the first set of clients, other clients have also received most of the file. So, when the next set of clients are selected, they are expected to require far less retransmissions than they would have had the file been transferred from scratch. The set of clients is chosen based on a regional division of clients and selecting a selecting a specified number of clients from each region for each iteration.

7.3 Future Work

The current protocol design suffices for a limited DEP network topology with a limited number of remote centres. The protocol design might be needed to be changed in case of addition of large number of remote centres to the network. Changes in the network topology are also in order. So, one direction of work is to adopt the protocol to the extended DEP network topology. Another direction is to extract a reliable transport

layer protocol for asymmetric networks which can provide TCP like primitives for reliable transport.