

**PERFORMANCE ANALYSIS OF MDP IN  
SIMULATED SPACE CHANNEL  
ENVIRONMENT**

**by**

**Anirban Chakraborti and Stephen Horan  
NMSU-ECE-02-009**

# **PERFORMANCE ANALYSIS OF MDP IN SIMULATED SPACE CHANNEL ENVIRONMENT**

---

Anirban Chakraborti and Stephen Horan  
Manuel Lujan Space Tele-Engineering Program  
New Mexico State University  
Las Cruces, NM

Prepared for

NASA Goddard Space Flight Center  
Greenbelt, MD

under Grant NAG5-9323

December 19, 2002



Klipsch School of Electrical and Computer Engineering  
New Mexico State University  
Box 30001, MSC 3-O  
Las Cruces, NM 88003-8001

PERFORMANCE ANALYSIS OF MDP IN SIMULATED  
SPACE CHANNEL ENVIRONMENT

BY

ANIRBAN CHAKRABORTI

A thesis submitted to the Graduate School  
in partial fulfillment of the requirements  
for the degree  
Master of Science in Electrical Engineering

New Mexico State University

Las Cruces, New Mexico

December 2002

“Performance Analysis of MDP in Simulated Space Environment,”  
a dissertation prepared by Anirban Chakraborti in partial fulfillment of the  
requirements for the degree, Master of Science in Electrical Engineering, has  
been approved and accepted by the following:

---

Linda Lacey  
Dean of the Graduate School

---

Stephen Horan  
Chair of the Examining Committee

---

Date

Committee in charge:

Dr. Stephen Horan, Chair

Dr. Eric Johnson

Dr. Richard Oliver

## ACKNOWLEDGEMENTS

He who is one, who is above all color distinctions,  
who dispenses the inherent needs of men of all colors,  
who comprehends all things from their beginning to the end,  
let Him unite us to one another with wisdom, which is the wisdom of goodness.

---- Rabindranath Tagore  
Nobel Laureate, 1931

I take this occasion to gratefully acknowledge the help and support I have received from my advisor, Dr. Stephen Horan, throughout my research. Without his inspiration and advice I would not have been able to accomplish as much or be the person that I am today. I have benefited much from his guidance in both personal and professional spheres. I am also indebted to my committee members Dr. Eric Johnson and Dr. Richard Oliver for their advice, suggestions and comments, which have shaped up my research work.

I am equally thankful to my family members and friends whose encouragement has kept me going strong all through the years of graduate study.

Certainly, I would especially like to thank NASA for their support under the Grant NAG5-9323 and experts of NASA Goddard Space Flight Center, and NASA Glenn Research Center and JPL who are interested in our study.

## VITA

April 2, 1979	Born at Asansol, West Bengal, India
1996	Graduated from DAV Jawahar Vidya Mandir, Ranchi, Jharkand, India
2000	Bachelor of Engineering in Electronics and Telecommunication, Utkal University, India
2000-2002	Research Assistant, Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, New Mexico

## PROFESSIONAL AND HONORARY SOCIETIES

Institute of Electrical and Electronic Engineers

## PUBLICATIONS

Anirban Chakraborti, "MDP in Telemetry Data Transfers," *23<sup>rd</sup> International Telemetry Conference, October 2002*

Stephen Horan, Anirban Chakraborti, Sandeep Muddasani and Srinivasulu Narina. "Testing MDP in Simulated Space channel Environment," *IEEE Transactions in Computer Networking* (Submitted).

ABSTRACT

PERFORMANCE ANALYSIS OF MDP IN SIMULATED  
SPACE CHANNEL ENVIRONMENT

BY  
ANIRBAN CHAKRABORTI

Master of Science in Electrical Engineering

New Mexico State University  
Las Cruces, New Mexico, 2002

Dr. Stephen Horan, Chair

The objective of this thesis is to study the performance of a Unicast Datagram Protocol (UDP)/Internet Protocol (IP) based data transfer protocol in a simulated space environment. Previous studies have shown that Transmission Control Protocol (TCP) suffers from limitations in the space network environment and has low transmission efficiency as well. One possible way to increase efficiency is to consider UDP based architectures and protocols for usage in space communications. We have chosen to analyze Multicast Dissemination Protocol (MDP), developed by the Naval Research

Laboratory in 1995 and is available in the public domain for further modification and development.

MDP had been originally designed for reliable multicast and file dissemination on the top of generic UDP/IP platform. MDP is adaptable to a wide range of network environments and uses negative acknowledgement mechanism to ensure end-to-end file transfer.

The key challenges in space communications always have been high Bit Error Rates (BER), long propagation delays, intermittent connectivity and asymmetric channels. Hence we have directed our attention to test the suitability of using MDP in space environment and uncover features that might affect its performance.

The testing was carried out on our in-house Space Channel Simulator, which has been earlier developed to facilitate space link simulation. It has variable settings for channel error rate, channel delay and connectivity options. The configuration uses Point to Point Protocol (PPP) to model point to point satellite links and corresponds to low capacity systems as found in small satellite systems.

We found that MDP performs better than TCP based protocols like File Transfer Protocol (FTP) in the face of high error environments and propagation delays. It is even capable of transmitting files in completely asymmetric or simplex links. MDP is able to recover and restore lost links without session restarts and thus can handle intermittent link outages. The protocol allows for

proactive use of Reed Solomon forward error correction, which reduces retransmissions and leads to higher throughput. The studies showed that MDP has desirable features, which makes it attractive for bulk data transfer in asymmetric and space Internet work applications

TABLE OF CONTENTS

LIST OF FIGURES ..... xii

LIST OF TABLES..... xiii

LIST OF ACRONYMS..... xiv

1. INTRODUCTION..... 1

1.1 Background..... 1

1.2 Problem Outline and Motivation ..... 4

1.3 Topic Development ..... 5

2. MULTICAST DESSIMINATION PROTOCOL..... 6

2.1 MDP Background ..... 6

2.2 MDP Protocol Operation ..... 7

2.2.1 MDP Protocol Session ..... 7

2.2.2 Server Transmission ..... 9

2.2.3 Client Reception and Synchronization ..... 10

2.2.4 Client NACK Process ..... 10

2.2.5 Server NACK Aggregation and Repair..... 11

3. SPACE CHANNEL SIMULATOR TEST-BED AND PROTOCOL CONFIGURATIONS ..... 13

3.1 Space Channel Simulator Test-Bed ..... 13

3.1.1 Channel Simulator..... 13

3.1.2 User Interface..... 15

3.1.3 Error Generation Methodology ..... 18

3.1.4	Space Channel Simulator Configuration for the Experiment .....	19
3.1.5	Workload Parameters .....	20
3.2	Protocol Configuration.....	22
3.2.1	Protocol Stack .....	22
3.2.2	PPP Configuration.....	25
3.2.2.1	On the receiver side .....	25
3.2.2.2	On the transmitter side.....	26
3.2.3	MDP Configuration .....	28
3.2.3.1	Configuration on the server side .....	28
3.2.3.2	Configuration on the client side.....	28
3.2.3.3	Command line parameter and option descriptions.....	29
3.2.3.4	Parameter ranges (for our configuration) .....	31
4.	PRELIMINARY TESTING AND ANALYSIS ON MDP .....	34
4.1	Optimum Transmission Rate.....	34
4.1.1	Methodology.....	34
4.2	Throughput.....	35
4.2.1	Methodology.....	36
4.3	Propagation Delays.....	38
4.3.1	Methodology.....	39
4.4.	Simplex Channels .....	40
4.4.1	Methodology.....	41
5.	ADVANCED FEATURES OF MDP .....	42

5.1	Behavior on Interrupted Links .....	42
5.1.1	Problem Outline .....	42
5.1.2	Hypothesis .....	43
5.1.3	Methodology for simulating Intermittent Links .....	43
5.1.4	Test Results .....	44
5.1.5	Discussion on Intermittent Links.....	47
5.1.6	Methodology for simulating Scheduled Links cuts.....	48
5.1.7	Test Results .....	48
5.1.8	Discussion.....	49
5.1.9	Verification of Hypothesis.....	50
5.2	File Update Performance .....	51
5.2.1	Problem Outline .....	51
5.2.2	Hypothesis .....	51
5.2.3	Methodology.....	51
5.2.4	Test Results .....	52
5.2.5	Discussion.....	53
5.2.6	Verification of Hypothesis.....	54
5.3	Effect of Parity Blocks on Performance.....	54
5.3.1	Problem Outline .....	54
5.3.2	Hypothesis .....	55
5.3.3	Methodology.....	55

5.3.4	Statistical Analysis .....	56
5.3.5	Test Results and Analysis .....	58
5.3.6	Verification of Hypothesis.....	66
6.	CONCLUSIONS.....	67
APPENDICES		
A.	TRANSFER DURATION OBTAINED FOR 1MB FILES AT BER = .00001 .....	71
B.	TRANSFER DURATION OBTAINED FOR 10MB FILES AT BER = .00001 .....	72
C.	TRANSFER DURATION OBTAINED FOR 1MB FILES AT BER = .0001 .....	73
D.	TRANSFER DURATION OBTAINED FOR 10MB FILES AT BER = .0001 .....	74
	REFERENCES .....	75

## LIST OF FIGURES

1.1 TCP timing diagram .....	3
1.2 MDP/UDP timing .....	3
3.1 A typical satellite link model .....	13
3.2 Space channel link simulator .....	16
3.3 User interface for the error generation module .....	17
3.4 User interface for the 2X1 module .....	17
3.5 Outline showing test factors and different levels of each factor .....	20
3.6 Protocol stack .....	22
3.7 MDP protocol stack .....	24
3.8 Receiver configuration .....	25
3.9 Transmitter configuration .....	26
4.1 The average value of file transfer time using MDP at different BER ...	37
4.2 The average value of file transfer time using FTP at different BER ....	37
5.1 Effect of file updates on file integrity .....	52
5.2 Effect of parity blocks on 1MB files (BER =. 00001) .....	60
5.3 Effect of parity blocks on 10MB files (BER =. 00001) .....	62
5.4 Effect of parity blocks on 1MB files (BER =. 0001) .....	64
5.5 Effect of parity blocks on 10MB files (BER =. 00001) .....	65

## LIST OF TABLES

3.1 Typical data files for designated link BER.....	21
3.2 Typical one-way link delays .....	22
4.1 File transfer duration obtained at different set transmission rates for 1MB files .....	35
4.2 The average values of file transfer times (using MDP) at different BER .....	36
4.3 Effect of 5 sec propagation delay on transfer duration.....	39
5.1 Effect of intermittent link cuts on transfer duration (zero error) .....	44
5.2 Effect of intermittent link cuts on transfer duration (BER = .00001) ....	45
5.3 Effect of intermittent link cuts on transfer duration (BER = .0001) .....	46
5.4 Resynchronization time for scheduled link cuts .....	49
5.5 Average time taken by 1MB files at different number of deliberately inserted parity packets (BER = .00001) .....	59
5.6 Average time taken by 10MB files at different number of deliberately inserted parity packets (BER = .00001) .....	62
5.7 Average time taken by 1MB files at different number of deliberately inserted parity packets (BER = .0001) .....	63
5.8 Average time taken by 10MB files at different number of deliberately inserted parity packets (BER = .0001) .....	65

## LIST OF ACRONYMS

BER	Bit Error Rate
FTP	File Transfer Protocol
H	Hypothesis
FEC	Forward Error Correction
N	the number of measurements in a data set
IP	Internet Protocol
MDP	Multicast Dissemination Protocol
ACK	Positive Acknowledgement
NACK	Negative Acknowledgement
NMSU	New Mexico State University
$q$	Critical value for a standardized distribution
$s$	Standard deviation in a set of measurements
PPP	Point-to-Point Protocol
SGLS	Space to Ground Link Simulator
$t$	Critical value for a t-distribution
$t_i$	Time measurement for an experiment run
TCP	Transmission Control Protocol
$z$	Critical value for a normal distribution
$\alpha$	Significance level
$\mu$	Mean value for a set of measurements
$\nu$	Number of degrees of freedom
UDP	User Datagram Protocol

## **1. INTRODUCTION**

### **1.1 Background**

The use of Internet-type protocols for space communication is no longer considered a “new” topic of investigation. Research has been primarily on the subject of Transmission Control Protocol (TCP) in space conducted at the National Aeronautics and Space Administration (NASA), Department of Defense (DOD), contractor and university facilities for a few years now. Much of the research has focused on the performance of TCP in space from the observations of simulations and experimental test bed results [1], [2], [3]. The purpose of these studies was to identify a better TCP variant for use in long-delay networks, such as those found in the satellite environment and to investigate the effect of latency on aggregated network utilization.

One next logical evolution in the process is to consider the satellite to be another node on the Internet. Research on this subject has been conducted at government, contractor, and university facilities for several years now; see, for example, [4], [5], and [6], and references therein. The intent of this configuration is not only to use the satellite as part of the infrastructure backbone but to also have the ability to terminate the link within the satellite itself to support space operations. As part of our satellite communications research program, we have been investigating the performance of both

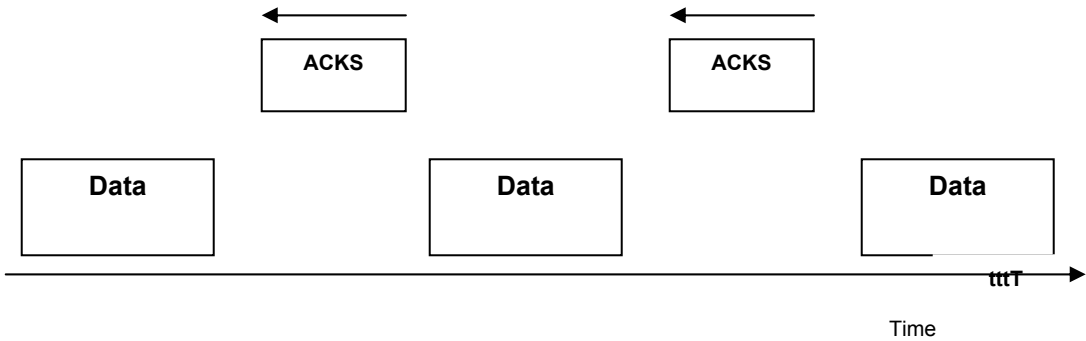
connection-oriented and connectionless protocols for satellite data transfer to a ground control point.

The space channel environment poses a number of challenges to providing reliable, end-to-end data communications with a user-specified level of service. Losses due to transmission errors, large point-to-point propagation delays, constrained bandwidth, asymmetric link data rates, and intermittent link connectivity all conspire to severely limit the performance of any ground-network-based protocol.

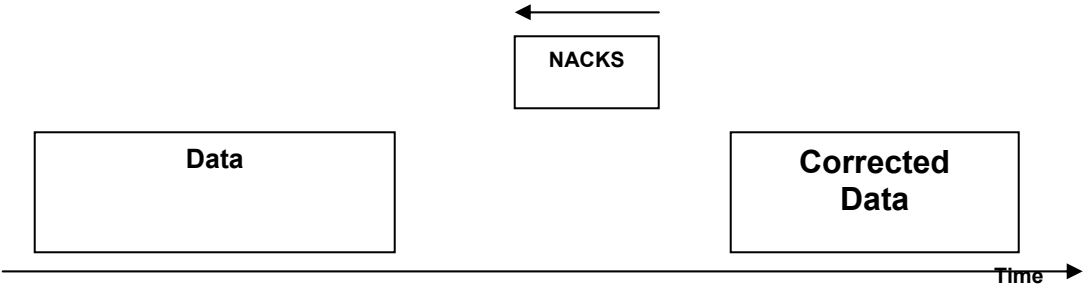
TCP is known to have limitations in network environments with long operational delays and high error rates [7]. TCP does provide reliable transfer of data but it needs positive acknowledgement from the receiver for transfer of each data packet, which, in turn, reduces the throughput. In particular, TCP-based protocols have a tendency to confuse channel errors with link congestion and thereby causing a reduction in transmission efficiency when the protocol invokes its rate back-off mechanism. TCP also typically uses some form of slow-start or ramp-up mechanism at the beginning of the transmission as it probes for the link congestion state. Since the typical satellite communications channel with one node originating or terminating at the satellite is an uncongested link with few competing flows, these mechanisms reduce link transmission efficiency as well.

While TCP-based protocols can have the advantage of end-to-end connection management, the inefficiency in link utilization to support the management may not be appropriate in satellite channels where the pass time, and hence the connection life, may be short. One way to improve

efficiency is to consider Unconnected Datagram Protocol (UDP) based transmission architectures. The UDP protocols will typically use a best-effort means to deliver the data but the delivery is not guaranteed. In UDP the application layer needs to provide for a feedback mechanism for error detection and correction for reliable data transfer. UDP will then have the expected advantages of minimal overhead and maximum data flow when compared with connection-oriented protocols. Conversely, the UDP-based protocols will not be as reliable as connection-oriented protocols without additional overhead.



**Figure 1.1** TCP timing diagram



**Figure 1.2** MDP/UDP timing diagram

Fig 1.1 and 1.2 compares and contrasts the TCP based data transfer processes with an UDP based. We find that in TCP an explicit acknowledgement is required for every data block transmitted and then only the next data block is sent. But in the case of UDP based data transfer continues on a best effort basis. Unlike TCP based protocols the error detection mechanism and feedback is built in MDP rather than UDP. When MDP detects a corrupted or missing data blocks it sends negative acknowledgement for the missing block only and then the server process duly retransmits the data block.

## **1.2 Problem Outline and Motivation**

The basic motivation of this thesis is evaluate an UDP based protocol, Multicast Dissemination Protocol [8] for use in small satellite environment. Previous studies [9] had compared the performance of MDP and FTP in simulated space environment and found out that MDP and FTP behaves the same in uninterrupted channel links and low error environments. In high error environments it was discovered that MDP performs better because of its negative acknowledge scheme.

In this study we are primarily interested in studying features of MDP such as restoring lost links and forward error correcting ability of parity packets in high error environment which are not available in FTP. On the broad side our aim is to be able to answer the following questions on MDP.

1. What are the basic features of MDP and how does it help its

performance under normal operating conditions of our Space Channel Simulator.

2. Is there an advantage to a MDP in this link environment when a link outage problem occurs and can the protocol recover.
3. Can MDP be used as a data transport protocol for real time applications where file sizes are dynamic.
4. Does proactive policy on inserting parity blocks in MDP coding blocks affect the throughput and in what way.

### **1.3 Topic Development**

The rest of the thesis is organized as follows. In Chapter 2, we discuss about the multicast dissemination protocol and its workings. In Chapter 3 we describe the Space Channel Simulator and outline the investigative procedures, system boundaries and work load parameters. In Chapter 4 we describe the testing done on some basic features of MDP and its analysis. In Chapter 5 we propose three hypotheses and test their validity. Chapter 6 summaries the thesis and provides the conclusions and future work for our study.

## **2. MULTICAST DESSIMINATION PROTOCOL**

### **2.1 MDP Background**

We have concentrated on the study of Multicast Dissemination Protocol, an UDP based application layer protocol that has mechanisms integrated in the protocol for reliability and transaction management. Though MDP had been designed for reliable multicast file and data delivery [10], we have evaluated the protocol in a unicast mode in a simulated space environment.

MDP was developed in 1995 by Brian Adamson of New Link Global Engineering Corporation and Joe Macker of Naval Research Laboratory with an objective of producing reliable communications for multiple users while also promoting efficient utilization. Reliability means ability to ensure time bounded delivery without compromising the files integrity. It uses ordered delivery, which is the packets are delivered to the receiver's application layer in the sequence they were transmitted. The client uses negative acknowledgments to report missing sequence number to the server. It makes no design assumptions about the network structure and works in both commercial Internet and space environments.

MDP is capable of handling link outages and high channel errors. This is done via the use of selective negative acknowledgments, the use of parity for frame repairs, and link management options. The positive acknowledgement scheme followed by TCP based protocols does not work

efficiently in high error environments. This is because the positive acknowledgements also can be corrupted in the return channel leading to redundant transmissions for packets, which had already been successfully received at the receiver. Thus the MDP type of protocol design is also expected to be useful in the space channel environment even if a single user, point-to-point link is being used. The MDP protocol does not use a windowing mechanism to control data transmission and avoid congestion. Rather, a TRANSMISSION\_RATE parameter that can be adjusted in real-time is used to control data flow. Since the typical satellite command or telemetry channel is not congested by other users, the rate control parameter can be set to a value that works best (gives the lowest data transfer duration) with the transmission hardware.

## **2.2 MDP Protocol Operation**

In this section we have provided a brief overview of how MDP operates. The description relates to a multicast data transmission scenario but is valid for unicast operations too. One of the primary reasons for studying a multicast protocol is to adapt the control techniques designed for multiple receivers and transmissions for an unicast mode [11].

### **2.2.1 MDP Protocol Session**

We begin by describing the makeup of an MDP reliable multicast session. To start with, the session participants exchange User Datagram Protocol (UDP) packets over the generic Internet Protocol (IP) network on a

common, predetermined address and port number. We call this a MDP protocol session.

An MDP sender primarily generates messages of type MDP\_DATA and MDP\_PARITY to carry data content and related parity-based repair information [12] for the bulk data (or file) objects being transferred. MDP\_PARITY information is usually sent only on explicit repair demand, but the protocol implementation supports “proactive” MDP\_PARITY, by which parity packets are sent with the initial data transmission itself. Another sender message of the type MDP\_INFO is also defined and used to carry context (file features such as file size) information for a given transport object (the file to be transmitted). A sender also generates messages of the type MDP\_CMD to perform certain protocol operations such as end-of-transmission indication, object flushing, round trip estimation, optional positive acknowledgement requests and session squelch commands. Sequencing information is used to detect transmission losses and is embedded in all transmitted packets from the sender.

An MDP receiver generates messages of the type MDP\_NACK or optionally MDP\_ACK in response to the transmission periods or commands from the sender. The MDP\_NACK messages are generated in response to transmission losses and inform the sender about the detected missing or corrupted sequence. MDP\_ACK messages are not required by MDP for default protocol operation but can be used for object receipt book keeping and active congestion feedback.

### **2.2.2 Server Transmission**

The protocol activity within a session is initiated by the transmission of data by the source node. Session data are compromised of serialized segments of objects enqueued for transmission by the source application. For example a file can be defined as a single object that is further subdivided into number of segments (depending on the default segment size or the size explicitly set by the user). The TRANSMISSION\_RATE parameter (configurable on individual participants) governs the peak rate at which data are transmitted by a session the participant in units of bits per second. The multicast transmission rate by the source, including the data, repairs, and commands, is upper bounded by this parameter. The SEGMENT\_SIZE parameter describes the maximum datafield size the source uses for packet transmission.

The BLOCK\_SIZE and MAX\_PARITY parameters used during a session affect how the server calculates, maintains, and transmits parity based repair messages. The MDP protocol uses shortened, Reed-Solomon encoding to construct repair segments based on a block of data segments. The BLOCK\_SIZE parameter corresponds to the number of MDP\_DATA messages per Reed Solomon encoding block while the MAX\_PARITY parameter corresponds to the number of repair (MDP\_PARITY) segments the source calculates and maintains per block.

### **2.2.3 Client Reception and Synchronization**

Upon reception of MDP\_INFO or MDP\_DATA messages from a new session source, an MDP client will "synchronize" with a source by beginning to maintain state on the source using the object segmentation and encoding parameters and current transmission sequencing information embedded in the received messages. For this reason, certain information is embedded in all MDP\_INFO, MDP\_DATA, and MDP\_PARITY messages transmitted by the source [13].

### **2.2.4 Client NACK Process**

After the synchronization of the client with the server, the client begins tracking the sequence of transmitted objects using the OBJECT\_ID and offset fields contained in the data and command sent by the server. If the client detects missing data from the server at the end of an encoding block, end of an object transmission, or upon receipt of an MDP\_CMD\_FLUSH command, it initiates a process to request repairs from the server. The end of block or end of object boundaries are detected either explicitly by presence of the MDP\_FLAG\_BLOCK\_END flag in a received message or implicitly by the receipt of a message for an object or encoding block beyond the last incompletely received block.

To initiate the repair request process and to facilitate the suppression of redundant NACK transmission, clients use a random holdoff timer to delay repair requests. Thus, if another NACK for the same (or more) repair information (or the repair information itself) arrives before the timeout ends, the client suppresses its transmission of an MDP\_NACK message. After transmission or suppression of the NACK occurs, an additional timeout period is used before reinitiating the same repair request attempt (this additional timeout allows the source time to provide a group repair response to the previous request).

#### **2.2.5 Server NACK Aggregation and Repair**

Upon receipt of an MDP\_NACK from a client, the server parses and tracks the repair request data and begins a holdoff timeout period before responding to pending repair requests. This allows the source to receive and aggregate multiple repair requests from other clients. Note that during this repair response holdoff time, the server continues to transmit data for new or other pending objects. The exception to the holdoff timeout is that retransmission of MDP\_INFO messages occurs immediately upon receipt of the MDP\_NACK message. If required, that repeat retransmission of duplicate MDP\_INFO is restricted to a maximum of once per timeout period.

The MDP protocol attempts to maximize its use of parity segments, which it has calculated for repair transmissions. For example, if during an initial repair cycle for an object, receivers request a portion of the available parity

segments, the server uses parity segments from any remaining unused portion for subsequent repair cycles in the same encoding block. There is advantage to this approach as the client parity decoding process fills missing data segments (erasures) with any combination of the same number of, but different, parity segments. Thus, when reliability requires multiple repair cycles (more than one repair attempt) to complete an encoded object block, receivers are freed from the difficulty of tracking and requesting explicit sequences of parity segments. Given a sufficient number of parity segments calculated by a source and nominal packet loss, sources generally never need to send the same segment twice, thus maximizing the use of the parity information and minimizing the maintenance of redundant data requests amongst receivers.

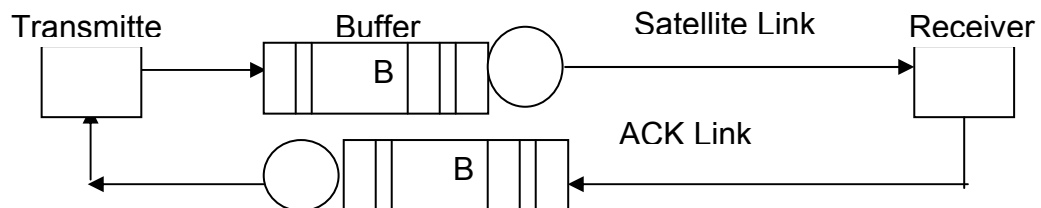
The usage of negative acknowledgement mechanisms rather than positive acknowledgement helps MDP to work in the constraints of the space environment, as we shall see in the succeeding chapters. The next chapter describes our Space Channel Simulator and the MDP protocol configurations used in our testing.

### 3. SPACE CHANNEL SIMULATOR TEST-BED AND PROTOCOL CONFIGURATIONS

This chapter describes NMSU Space Channel Link Simulator test-bed facility, the procedures and the protocol entities we have used to conduct the experiments.

#### 3.1 Space Channel Simulator Test-Bed

The Space Channel Link Simulator [16] is used to perform the error generation and link delay used to test the protocol suite performance. In the following subsections, we introduce the simulator and discuss the experiment methodologies we have used to conduct the tests.



**Figure 3.1** A typical satellite link model

##### 3.1.1 Channel Simulator

A typical satellite link model is given in Figure 3.1. Basically it consists of the transmitter, receiver, link buffer and satellite link. The Space Channel Simulator has been developed at NMSU to model space channel characteristics experienced in transmitting data. The simulator is described

fully in [14], [15], and [16]. Basically, the simulator configuration allows the user to configure the simulated channel to

- Allow for simultaneous bi-directional data flow (forward and return channels),
- Allow user-selectable error rates and statistical descriptions of the channel,
- Allow different data rates on the forward and return links as would be found in satellite links, e.g. 2400 baud forward, 115,200 baud return
- Provide for a simulated delay up to 5 seconds on each link.
- User selectable link cut off time in both directions to simulate the intermittent connectivity

The Space Channel Simulator utilizes the LabVIEW programming language to control data throughput through the simulator, mix the baseband data stream with the user-selected error vector, and provide for the user-selectable link delay value. The hardware configuration is illustrated in Figure 3.2. The LabVIEW software is run as an application on each of the Space Channel Simulator computers. Typically, the LabVIEW modules are the only applications software running on the computers. This configuration was developed to model point-to-point satellite links in its current configuration. The bandwidth-delay product for the system under a 115,200 bps symmetric link with no imposed channel delay is 671 bytes [14]. As a comparison, a T-1 line crossing the United States has an estimated bandwidth delay product of

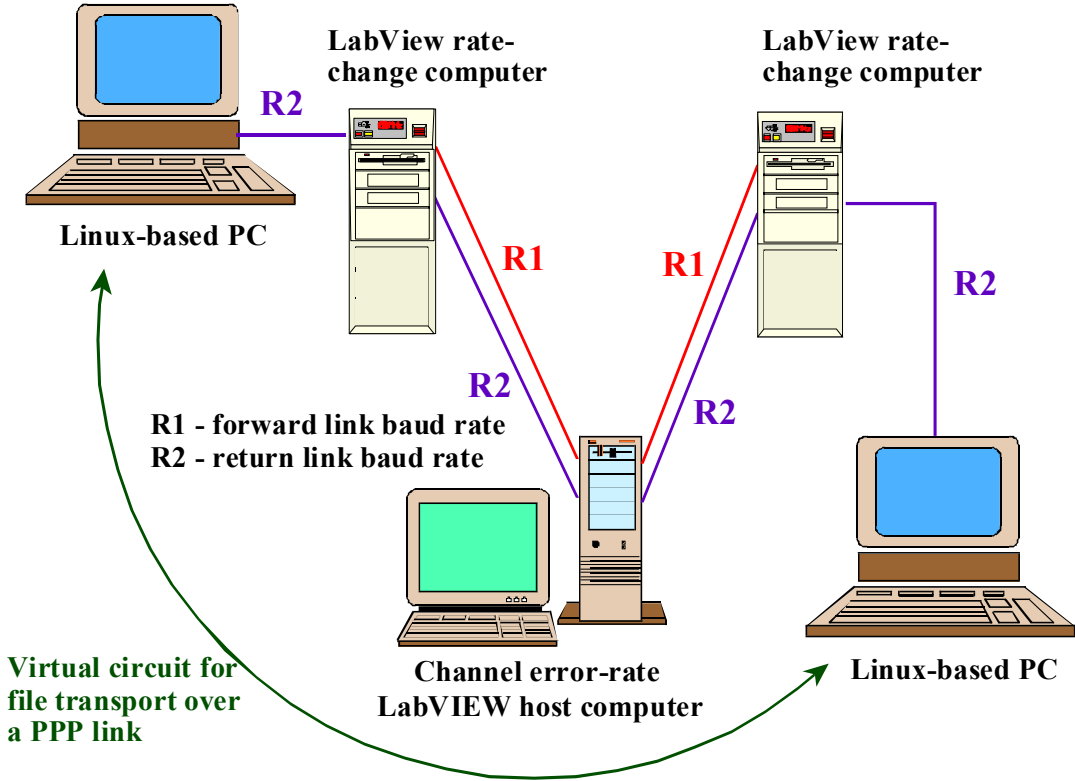
11,580 bytes. Therefore, this simulator corresponds to a relatively low BDP system.

The three PCs in the simulator are Dell 600-MHz computers with 128 Mbytes of memory running Windows 98 second edition. The first Linux-based PC is a Dell 266 MHz computer. This is our logical ground station computer. The second Linux-based PC is a Gateway 166 MHz. This is our logical satellite computer. Both Linux computers are running Red Hat Linux version 7.1. The simulator is connected to the Linux computers using serial cables connected to the COM serial ports on each computer. The data connections are configured without hardware or software handshaking to allow for a simulation that would be similar to interfacing with a satellite radio system. In all cases, the links between the simulator and the Linux computer were set to 115,200 bps (R2 in Figure 3.2). The simulations run with symmetric links had the forward link (R1 in Figure 3.2) also set to 115,200 bps. The simplex link tests were conducted with only the forward channel set to the designated link speed and the return channel closed.

### **3.1.2 User Interface**

The user interface for the error generation module is given in Figure 3.3 while the user interface for the two rate-split and delay components is illustrated in Figure 3.4. The input for the user-selectable parameters is done using the LabView Text Tool on the panel. The specification of the external data computer (External Port Number), and link simulator interface ports

(Channel Forward Port Number and Channel Return Port Number) can be selected by incrementing the selection slide on the user interface using the Operating Tool. The error generation VI gives the user a Windows file manager interface to select the forward and return error files. The software enable/disable is done using the button switch on the VI panel. This can be clicked by using the mouse to halt processing. The Lab VIEW execution is initiated by clicking the left-pointing arrow (⇐) on the command bar using the mouse.



**Figure 3.2** Space channel link simulator

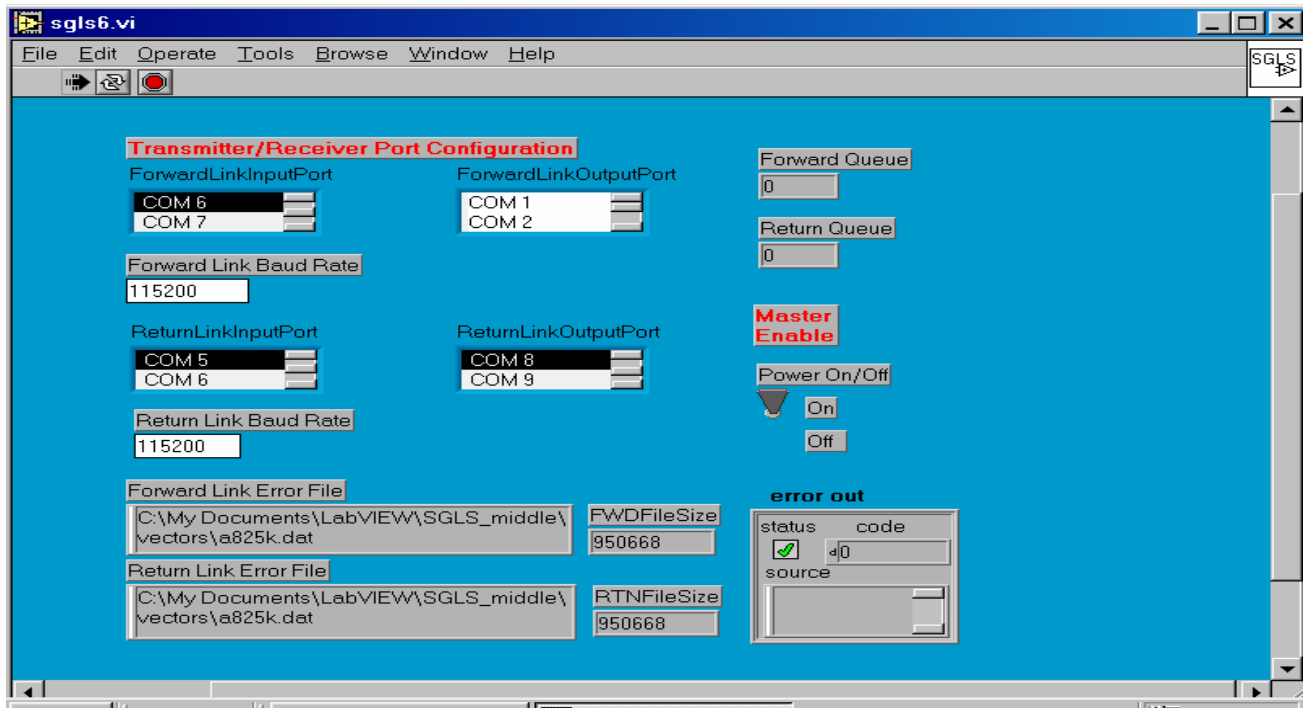


Figure 3.3 User interface for the error generation module

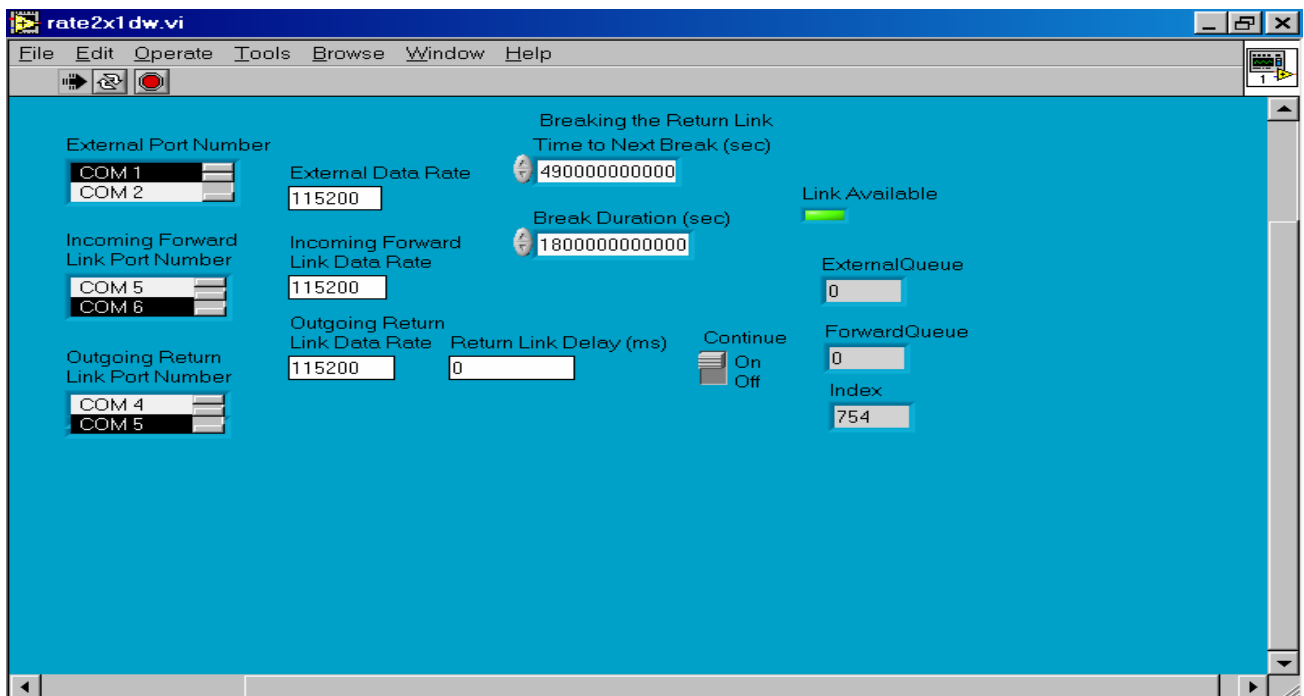


Figure 3.4 User interface for the 2x1 module. The 1x2 module is similar

### 3.1.3 Error Generation Methodology

To properly model a channel, the user needs a proper statistical description of the channel error generation mechanism [16]. A typical channel error statistical description is Additive White Gaussian Noise (AWGN) where the errors are described by a Gaussian random process parameterized by the link Energy-per-bit-to-Noise-density ratio ( $E_b/N_0$ ) [17]. Here, we use a computer program [18] to develop a library of error vectors for use in the simulator so that the error process does not need to be computed in real-time and thereby slowing down the simulation process. In the program, the user specifies an  $E_b/N_0$  value, the number of bit errors to be generated, and the type of statistics to be used and the program would produce a vector meeting this specification. The error vector will be all 0s except for 1s at the locations where the bit errors are to occur. The 1s are distributed over the vector according to the statistics specified by the user. The program was designed to develop vectors for AWGN, radio frequency interference, and mixed noise-and-interference environments. Other statistical distributions can be generated by modifying the program to generate the desired statistical model. For all of the testing shown here, the AWGN statistical model was used.

When performing protocol testing, the data is organized into channel transfer frames at the source computer. Each frame will have a maximum size of  $N$  bits that will be a function of the framing protocol used and how the user configures the protocol. For example, if the default Point-to-Point Protocol

(PPP) frame size is used,  $N$  will be 1500 bytes. The asynchronous serial ports do not bring an entire frame into the simulator; rather a sub-block of  $M < N$  bits will be brought in each time slice. As the data enters the simulator, a sub-block of the error vector having the same length as the input data, is chosen from the next  $M$  bits of the error vector. If  $r_i$  is a single bit within the input data sub-block and  $e_i$  is the corresponding element of the error vector, then an output sub-block, representing the data at the output of the channel, is formed by the set of  $M$  bits  $t_i$  where [19]

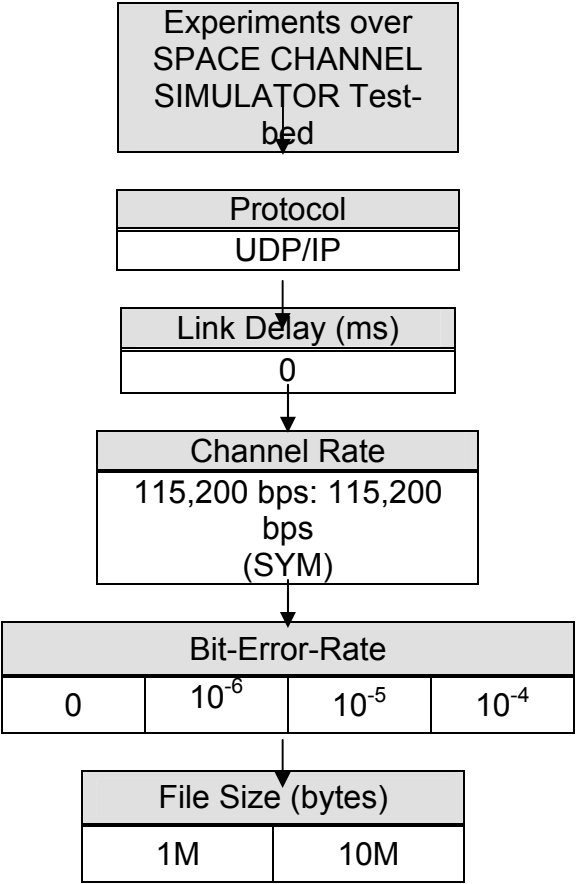
$$\mathbf{t}_i = \mathbf{r}_i \oplus \mathbf{e}_i \quad (1)$$

As each new sub-block is brought in, the position along the error vector is moved to match the input data sub-block size. When the end of the error vector is reached, the index is wrapped around to the beginning of the vector. Typical error vector sizes are on the order of  $10^6$  bits in length. The error generation module then consists of a while-loop structure where the input ports are scanned for available data. If data is found, Equation (1) is applied to the input block and then it is immediately written to the output port. The while-loop keeps executing as long as the user has the VI running.

### 3.1.4 Space Channel Simulator Configuration for the Experiment

Figure 3.5 outlines test factors and different levels of each factor for experiments over Space Channel Simulator testbed. The test conditions

include link delay, channel rate, Bit-Error-Rate (BER) and transmission file size, which represent satellite orbit status, channel operating mode, space channel noise and user transmission load individually.



**Figure 3.5** Outline showing test factors and different levels of each factor

**3.1.5 Workload Parameters**

The test data set will be typically be a data file composed of random data. Other files, such as voice or video data can also be used in specific experiments to investigate how the channel errors affect the data structures in

addition to the effects of the channel conditions on the protocol itself. The random data files can be taken from a standard set of random data files of length 1MB and 10 MB. In practice, these individual file sizes are not exactly the length given in the name. Rather, a length that is a prime number close to the desired length is used. This will keep interactions between the file length and the channel error vector length to a minimum. In the standard set, the following file sizes are available:

- ❖ 1MB files are 1,000,001 bytes long.
- ❖ 10MB files are 10,000,007 bytes long

Typical data files for various white gaussian noise Eb/No designs are given in Table 3.1 Typical link delay values for specific link types are given in Table 3.2

**Table 3.1** Typical data files for designated link BER

File	BER	Number of errors in the file
Infinite.dat	0	0
a1050.dat	$10^{-6}$	10
a950.dat	$10^{-5}$	100
a825k.dat	$10^{-4}$	1000

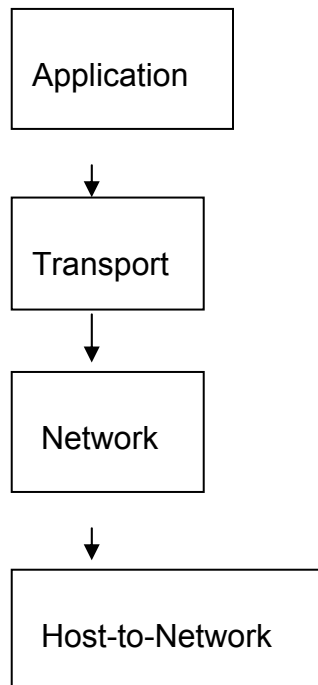
**Table 3.2** Typical one-way link delays

Delay	Link Length	Link Type
2.5ms	750km	LEO
120ms	35,700km	GEO
1.28ms	384,000km	lunar
5s		Triana

## 3.2 Protocol Configuration

### 3.2.1 Protocol Stack

The protocol stack is the combination of different protocols at various layers. It is normally considered to be a four-layer system as shown in Figure 3.6.



**Figure 3.6** The layers of protocol stack

In Figure 3.6 each layer has a different responsibility

1) The application layer handles the details of the particular application. MDP is an application layer protocol

2) The transport layer provides a flow of data between two hosts, for the application layer above. In the transport layer there are two vastly different transport protocols namely

- TCP (Transmission Control Protocol).
- UDP (User Datagram Protocol).

TCP provides a reliable flow of data between two hosts. It is concerned with things such as dividing the data passed to it from the application in to appropriately sized chunks for the network layer below, acknowledging received packets, setting timeouts to make certain the other end acknowledges packets that are sent and so on. Because this reliable flow of data is provided by the transport layer, the application layer can ignore all these details.

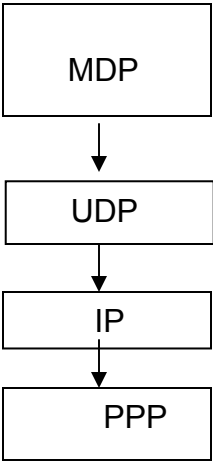
UDP, on the other hand, provides a much simpler service to the application layer. It just sends packets of data called datagrams from one host to the other, but there is no guarantee that the datagrams reach the other end. Any desired reliability must be added by the application layer.

3) The network layer handles the movement of packets around the network. Routing of packets takes place here. IP (Internet Protocol), ICMP (Internet

Control Message Protocol), and IGMP (Internet Group Management Protocol) provide the network layer.

4) The UDP/IP protocol stack does not say much about host to network layer except it is a medium or protocol by which the host connects to the network for sending the IP packets over it. Commonly this is the Ethernet layer or the PPP layer.

Corresponding to the layers described in Figure 3.6 the protocols used in this report are divided as shown in Figure 3.7.



**Figure 3.7** Different levels of MDP

For our testing we need to configure the PPP (Point to Point Protocol) and the MDP (Multicast Dissemination Protocol) at the sender and the receiver. The host operating system at the session participants takes care of the transport and the network layer. The next section explains how to configure PPP.

### 3.2.2 PPP Configuration

The communication between transmitter and receiver will be through a serial port. The communication port used on both transmitter and receiver is `/dev/ttyS0` (COM1 in DOS/Windows). In Linux PPP is implemented by the PPP daemon `pppd`. Its configuration is done through files in the `/etc/ppp` directory. In the next sections we illustrate how to configure the PPP daemon for both the transmitter and receiver.

#### 3.2.2.1 On the Receiver side

An options file `/etc/ppp/options.ttyS0` is created which contains PPP options specific to connections through `/dev/ttyS0`. The entries in `/etc/ppp/ttyS0` file are given in the figure 3.8

```
local  
noauth  
10.0.0.2:10.0.0.3  
115200  
mru 1500  
mtu 1500
```

**Figure 3.8** Receiver configuration

The meaning of each line is as follows:

- *local* indicates that modem lines are not used
- *noauth* indicates that it would not need any authorization for communication with transmitter.
- *10.0.0.2:10.0.0.3* indicates that *10.0.0.2* is the remote IP address and *10.0.0.3* is the local IP address.
- *mr**u* indicates the maximum receiving unit. By changing this parameter we adjust the PPP frame size. By default this is 1500 bytes.
- *mtu* indicates the maximum transmitting unit. The default is 1500.

### 3.2.2.2 On the Transmitter side

On the transmitter side the entries in */etc/ppp/ttyS0* file are similar to that of receiver side except the remote and local IP addresses will be swapped. The configuration file contains the entries illustrated in Figure 3.9

```
local  
noauth  
10.0.0.3:10.0.0.2  
115200  
mru 1500  
mtu 1500
```

**Figure 3.9** Transmitter configuration

The PPP daemon can be installed in two ways

- `getty`-like installation
- Installing manually

In `getty`-like installation we can have the `pppd` (PPP daemon) start when we boot the system. A typical way of achieving this is to edit the `/etc/inittab` file. This file contains information for initializing the system.

We add the following to this file:

*# Start pppd for the serial laplink.*

```
pd:2345:respawn:/usr/sbin/pppd /dev/ttyS0 nodetach
```

This is interpreted as follows: for run levels 2, 3, 4 and 5 start

`/usr/sbin/pppd /dev/ttyS0 nodetach` and if it dies (at the end of a connection) respawn (start a new one). The `"nodetach"` option makes that `pppd` stays connected to the terminal that started it, rather than forking and exiting. This option is necessary because the `"init"` process would respawn a new one immediately otherwise.

We can start the PPP connection manually by typing

```
/usr/sbin/pppd /dev/ttyS1 nodetach
```

at the command line.

We can kill `pppd` by typing

```
killall pppd
```

at the command line.

### 3.2.3 MDP Configuration

The exact configuration of MDP along with the workload parameters varied depending upon the intended operational characteristics we desired to analyze. However here we shall examine the most general configuration and explain the involved parameters.

MDP takes an UNIX command line argument both on the server and the client side. The commands to configure the server and the client are given next.

#### 3.2.3.1 Configuration on the Server side

The following command is entered at the command line interface of the server for its configuration.

```
mdp [-A <address/port>] [-F <interface address>][-r <txRate> (bit/sec)][-L  
<logfile>] [-d <debuglevel>] [-b <blocksize>] [-p <numparity>][-n<autoparity>] [-  
l <debuglogfile>] [-g <initialGrtt>]<tx file/dir names>
```

#### 3.2.3.2 Configuration on the Client side

The following command is entered at the command line interface of the client for its configuration.

```
mdp [-A <address/port>] [-F <interface address>][-r <txRate> (bit/sec)][-  
L <logfile>] [-d <debuglevel>][-l <debuglogfile>] -D <archive/cache directory>  
[-a]
```

### 3.2.3.3 Command line Parameter and Option descriptions

In this section we describe the command line parameters used for the configuration of the client and the server.

- a      Configure client to permanently store (archive) files instead of just using the "-D" archive Directory as a temporary cache for received files.
  
- u      Configure client to unicast protocol messages back to the server. This may be useful for some asymmetric network topologies.
  
- A <address/port>      Destination IP address/port of transfer session.
  
- F<interface Address>      Specify the IP address of the network interface to use for multicast traffic
  
- r <txRate>      Maximum transmission rate in bits per second.
  
- b <block size>      Number of data segments per MDP coding block.

- p<numparity>                    set number of available parity segments per MDP coding block
- n<autoparity>                    set number of parity packets automatically sent by the server per MDP block
- d <debug Level>                Set debug level. Determines level of debugging detail displayed or logged.
- Debug levels currently include:
- 0 : Misc errors only.
  - 2: Major events (e.g. file tx/rx start/stop) plus client statistic reports.
  - 4: Detailed file reception progress
  - 6: General protocol operation.
  - 8: More detailed protocol operation.
  - 10: High level NACK construction/handling
  - 12: Detailed NACK construction/handling
- l <debugLogFile>                Specify debug message log file.
- L <log File>                    Log transmit/receive completion events to specified log file.

<tx directory/file names> Files/Directories to transmit (These should be last on the command-line)

#### 3.2.3.4 Parameter Ranges (For our Configuration)

In this section we describe the ranges of command line parameters used for the configuration of the client and the server.

- Address parameter: The target address for the client is 10.0.0.1 and 10.0.0.2 for the server.
- Interface Parameter: It is 10.0.0.2 for the client and 10.0.0.1 for the server.
- Port Number: MDP can use any of the unreserved ports for its process ranging from 1200 to 65536. The port number should be same at the server and the client for both sender and receiver process to communicate successfully.
- Transfer rate: The default transfer rate is 64 Kbps. The rate is usually set depending upon the channel capacity. We had it set at 88,000 bps for a channel capacity of 115200 bps. On the Ethernet it can be set up as high as 4Mbps on a 10 Mbps maximum link speed.

- Block Size: MDP splits data in MDP coding blocks. By default each coding block has 64 segments, which are of 1024 bytes each. There can be a maximum of 256 segments per block and can be set accordingly.
- Numparity: It is the number of parity packets that can be deliberately sent in a MDP coding block. The default number is 32. It has a maximum of 128 so that  $(\text{Block Size} + \text{Num Parity}) \leq 256$ .
- Autoparity: It is number of parity packets automatically sent per MDP coding block. Since insertion of parity packets implies less number of data segments can be accommodated, its default is set to 0.
- Debug level and Debug file: The higher debug level implies higher detail of the protocol is captured on the debug file, which helps in understanding the inner workings of the protocol.
- Log file: On the client side, the log file records the name of the file received, size of the file and the time stamps at which the MDP\_INFO and last data vector was received for the transmitted file. On the server side, the log file records the same information

except for time stamps, which record the time at which MDP\_INFO and last data vector, was sent for the transmitted file.

## **4. PRELIMINARY TESTING AND ANALYSIS ON MDP**

The objective of this section is to study the basic operational characteristics of Multicast Dissemination Protocol under simulated space environment. Towards this end we have designed experiments to understand the working of the protocol under conditions of high error rate, simplex channels and propagation delays. We will present the methodology of each of the tests to enable future replication. At this point our main concern is to analyze the response of the protocol to the simulated settings and thus the number of test runs varied accordingly. The present results are merely reflective of the protocols behavior and makes no claims of being statistically significant.

### **4.1 Optimum Transmission Rate**

The transmission rate at which data transfer takes the least time or the throughput is highest under zero delay and delay error conditions are known as optimum transmission rate. It is independent of the file size. The optimum transmission rate is required to calculate the transmission lifetime for a particular file size. The transmission rate depends on the hardware settings.

#### **4.1.1 Methodology**

Set the transmission rate parameter of MDP at different values ranging from 64000 bps up to maximum channel speed of 115200 in zero error and zero delay conditions to measure the respective file transfer durations.

The TRANSMISSION\_RATE parameter of MDP can be set according to the channel conditions; the maximum is limited only by the channel capacity and the hardware. Its default is set at 64kbps, but we have set it up to 4Mbps on the Ethernet. On the Space Channel Simulator the maximum channel speed can be set to 115200 bps. However any rate set above 88000 bps leads to longer throughput time than at 88000 bps itself. Therefore a functional maximum link rate of 88000 bps is used for testing. This behavior is assumed to be due to buffer capacity of the link interfaces.

**TABLE 4.1** File transfer duration obtained at different set transmission rates for 1MB files

Set Transmission Rate (bps)	Achieved Transfer Duration (sec)
85,000	96
88,000	95
90,000	97
1,00,000	98
1,15,200	103

**4.2 Throughput**

Throughput is an indicator of the rate of data flow in the channel under simulated conditions. Consequently higher the throughput, the lower is the time required for end-to-end file transmission. Throughput depends on number of factors including the error conditions, propagation delays and block size of

the underlying data link layer. In our case the data link layer was PPP and we had set the block size at the default value of 1500 bytes.

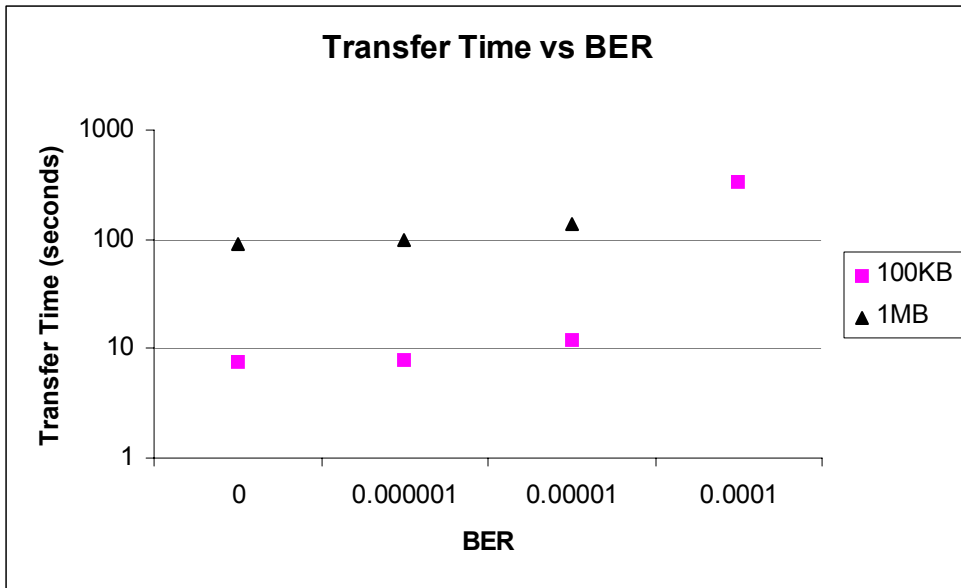
**4.2.1 Methodology**

The purpose of the this test is to find the throughput times for 1MB and 10MB files at set simulator conditions as described below.

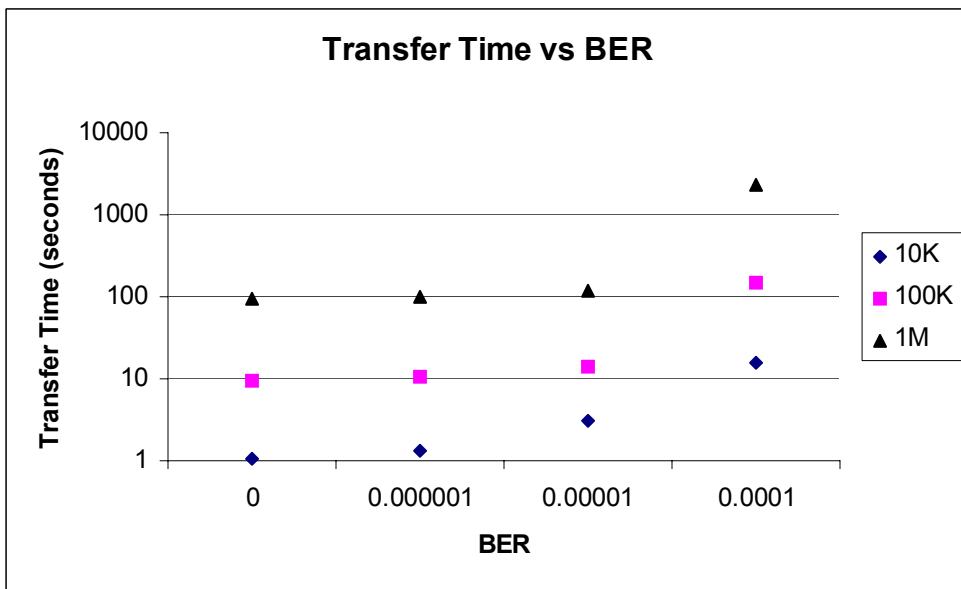
1. The transmission rate is set at 88,000 bps
2. The error vectors are selected to achieve BER of 0,  $10^{-6}$  and  $10^{-5}$  and  $10^{-4}$  respective throughput times are recorded for each of 1MB and 10MB files
3. The bit directional delays are set to zero.
4. The frame size at the PPP link level is set to default value 1500 bytes.
5. Each of the data file category (1MB, 10MB) were test run 16 times (see section 5.3.4) to compute the average throughput time.

**Table 4.2** The average values of file transfer times at different BER's using MDP [9]

File Size In Bytes	$\mu$ at BER:0	$\mu$ at BER: .000001	$\mu$ at BER: .00001	$\mu$ at BER: .0001
1M	95.3125	99	107.438	321.56
10M	953.375	1026.2	1118.813	3220.5



**Figure 4.1** The average value of File Transfer time using MDP at different BER



**Figure 4.2** The average value of File Transfer time using FTP at different BER

In a zero error environment the transfer duration of MDP is slightly higher than TCP based protocols (figures 4.1 and 4.2). But in case of high bit error rates like  $10^{-4}$  MDP is able to complete the data transfer whereas TCP based protocols like File Transfer Protocol (FTP) ceases transmission. Therefore FTP is unable to transfer large files at high bit error rates. The fact that MDP is capable of maintaining data transmission even at relatively high error rates makes it suitable for space applications.

### **4.3 Propagation Delays**

The objective of this experiment is to determine the effect of propagation delay on protocol performance. Space channels like commercial Internet suffer from latencies with only the magnitude being several times higher for the former. The latency on the commercial internet can be anywhere between  $100\mu$  secs to  $5000\mu$  secs [20]. We have taken the worst-case scenario of channel latency or delay of 5 sec (the longest delay obtainable on our simulator) into consideration and evaluated the ability of UDP based protocol to complete end-to-end data transfer. We did not simulate any channel errors for we wanted to ascertain the effect of propagation delay alone on the file transfer.

**4.3.1 Methodology**

The purpose of the following tests is to determine the effect of propagation delays on file transfers.

- 1. The transmission rate is set at 88,000 bps.
- 2. The error vector is selected for 0 BER
- 3. The forward and return channel propagation delay is set at 3000 ms
- 4. Five test runs were conducted under each data file category and average result was computed.

**Table 4.3.** Effect of 5 sec propagation delay on transfer duration

File Size	Bit Error Rate	Propagation Delay (sec)	Transfer time (sec)
1MB	0	5	136
10MB	0	5	1387

The results show that MDP is able to complete the data transfer though the throughput time increases considerably. Tests conducted on FTP ceased data transfer and was not able to complete end-to-end transfer [9]. This indicates that TCP based protocols in their present mode is not suitable for high latency space channels.

TCP works on the positive acknowledge mechanism in which it waits for receiving an acknowledgement for the last block of data sent before it continues transmission. If the acknowledgement is not received before the

timeout interval, it retransmits the last block again and again until it is acknowledged. Under high propagation delay of 5 seconds TCP has to wait for a long time for the acknowledgement for each block of data and thus there are redundant re-transmissions but the file is never completely transferred. While in the case of MDP, the negative acknowledgement is required only for the missing packets and not for every packet so the server does not have to wait to continue with the transmission.

#### **4.4 Simplex Channels**

TCP based protocols like FTP rely on positive acknowledgement mechanisms for data transmission. A block of data needs to be acknowledged before the next block is sent. Thus it requires the presence of a return channel for successful data transfer. On the other hand certain UDP based protocols like MDP work on negative acknowledgement mechanisms, which does not require every successfully transmitted block to be acknowledged. Tests were conducted to ascertain the effect of simplex channel, i.e. only the forward channel is available and there is no return channel, on the success of data transfers. However, a return channel does need to be available sometime otherwise the transmission has no closure.

#### **4.4.1 Methodology**

The purpose of the following tests is to determine the behavior of MDP in the presence of simplex channels.

1. The transmission rate is set to 88,000 bps
2. Zero error and zero delay conditions are set.
3. The return link is cut for the duration for least as long as the transmission lifetime of the file to transmitted.
4. The file is then transmitted.
5. The result was ascertained by conducting 10 end-to end data transfers

Unlike TCP based protocols, which rely on positive acknowledgements MDP can work in simplex channels. Because of the need of positive acknowledgements TCP based protocols cannot complete data transfers. While for MDP, in absence of any feedback from the client, the server continues on transmitting in a best effort service fashion until all the files and directory have been transmitted from its end. When the return link is again available the client may send aggregated NACKS for the transmitted files and server goes into repair session and completes the data transfer.

## **5 ADVANCED FEATURES OF MDP**

### **5.1 Behavior on Interrupted Links**

#### **5.1.1 Problem Outline**

In this chapter we investigate the behavior of MDP in the event of interrupted links and its role in the recovery and resuming file transmission after dropouts.

We have categorized interrupted links in two parts. Intermittent link cuts and scheduled link cuts. We will mainly discuss the effect of these link cuts on the individual files but towards the end of the chapter we shall also discuss its effect on directory management.

Intermittent link cuts are one of the major challenges in space communications. Data transfers may be suddenly interrupted due to link outages. Such link cuts, which occur without any forewarning, are mostly due to atmospheric attenuation and inherent spin of the satellite and may last from few seconds to several minutes. The ability to handle intermittent cuts and restore them is an important characteristic of any space communication protocol.

Scheduled links cuts occur during the time satellite is not seen over the ground station and hence no data transfer can take place. We have categorized scheduled link cuts as link cuts that are much larger duration than the transmission lifetime of the files. The importance of scheduled link cuts lies in the fact that if this happens amidst of the file transfer and after the link is

restored, the file may need be retransmitted completely or may be picked up from the point where the disruption occurs.

#### 5.1.2 Hypothesis

Given that the duration of the link cut is less than the lifetime of the file, transfer recovery is possible in all cases. In the case of link cut duration exceeding the lifetime, transfer recovery is not possible and it requires a link restart and resynchronization to complete the transfer

#### 5.1.3 Methodology for Simulating Intermittent Links

Lifetime is a function of file size, set transmission rate, bit error rate and possible latencies of the channel. The average lifetimes of 1MB and 10MB files have been given in Section 4.2 and 4.3. In all cases we have simulated link outages on these files under zero delay conditions.

The Procedure for simulating intermittent link cuts as follows:

1. The transmission rate is set to 88,000 bps.
2. The link cut interval is set at some percentage of the transmission lifetime of the file to be transmitted and the duration is set so as the sum of link cut interval and transfer duration does not exceed the transmission lifetime.
3. The file is transferred.
4. Each of the files below was test five times under similar conditions and the average of the transfer duration was taken.

The percentage in the following tables denotes the total percent of the file transferred when the link cut occurred. Total transmission time is the total time taken for the end-to-end file transfer and includes gap duration time.

#### 5.1.4 Test Results

The following test results were obtained.

**Table 5.1** Effects of intermittent link cuts on transfer duration (Zero Error)

File Size	%	Gap Duration (sec)	Total transmission time (sec)
1MB	25	15	110
1MB	50	15	110
1MB	75	15	110
1MB	25	30	126
1MB	50	25	120
10MB	50	120	1076
10MB	75	120	1076
10MB	25	120	1076
10MB	50	90	1046
10MB	25	90	1045

**Table 5.2** Effect of intermittent links on transfer duration (BER=. 00001)

File Size	%	Gap Duration (sec)	Total transmission time (sec)
1MB	25	50	159
1MB	50	50	157
1MB	75	20	131
1MB	25	30	139
1MB	50	25	124
10MB	40	100	1213
10MB	60	180	1290
10MB	70	200	1311
10MB	50	90	1200
10MB	25	90	1200

**TABLE 5.3** Effect of intermittent links on transfer duration (BER=. 0001)

File Size	%	Gap Duration (sec)	Total transmission time (sec)
1MB	50	50	374
1MB	50	50	373
1MB	20	20	346
1MB	25	30	354
1MB	50	25	350
10MB	50	180	3415
10MB	80	250	3481
10MB	70	100	3333
10MB	50	90	3323
10MB	25	100	3332

### 5.1.5 Discussions on Intermittent Links

At this point we can draw a few conclusions on the protocol performance in face of link cuts.

- If the transmission lifetime of a file under given set of channel conditions is  $t$  sec and link is cut when  $\%p$  of the transmission is complete or  $z$  sec of transmission have taken place and link cut duration was  $l$  sec where  $(l+z) < t$  sec then total time taken for complete transmission would be about  $(t+l)$  sec and is not dependent on the  $\%p$ .
- In case of intermittent links no resynchronization of client or server is required nor any restart takes place. File transmission resumes after the link outage is cleared from the point where it was left at the time of the outage. The apparent penalty of link drop out is on the transfer duration which increases by an amount equal to link cut duration.

For example consider a 1MB file with a transmission lifetime of 95 sec that is disrupted when already 50% of the file has been transferred and the link is restored after 15 sec (disruption interval), then the file transfer resumes again and the end-to-end transfer is completed in a total time of 120 seconds including the disruption interval. The total time will always be the sum of the

disruption interval and transmission lifetime as long as the disruption interval is less than the transmission lifetime and is independent of the percentage of file already transferred when the disruption occurred.

In case of directories is too, if there is a link cut and link is restored within the lifetime of the server transmission, then the time taken for total transfer is about the usual time + link cut duration. This was expected because individual files behave in that way.

#### **5.1.6 Methodology for Simulating Scheduled Link Cuts**

The following tests were conducted to ascertain the behavior of MDP in the face of scheduled link cuts.

1. The transmission parameter is set at 88,000 bps
2. The link cut interval is set at some percentage of the transmission lifetime of the file to be transmitted and the duration is set so as the sum of interval and duration exceeds the transmission lifetime and is equal to duration of scheduled link cuts.
3. The file is transferred.
4. Each of the files were test run 5 times and average resynchronization duration range was computed.

#### **5.1.7 Test Results**

*Resynchronization time = Total transfer Duration – Lifetime of the file.*

Tests were conducted on both 1MB and 10MB files and following data set was obtained. The percentage in the following tables denotes the total percent of

the file transferred when the link cut occurred. Both gap duration and reorganization times are in seconds.

**Table 5.4** Resynchronization time for scheduled link cuts

%	Gap Duration (sec)	Resynchronization time (sec)
25	900	120-159
25	1800	105-158
25	5400	150
75	1800	130
50	5400	150-165
75	5400	160
50	1800	110-158

**5.1.8 Discussion**

In the case of intermittent link cuts, we see that the total time taken for the file transfer was the sum of lifetime and gap duration without any requirement for resynchronization. In cases where the link cut duration is greater than the lifetime of the file we find it takes some time to resynchronize and data transfer commences again. During the resynchronization interval only MDP control packets are exchanged and no data transfer takes place. The time taken for resynchronization is independent of the transmission lifetime of the file and percentage of the file transmitted when the link cut occurred, this

happens to be in a fairly constant range for link cut durations of 15 mins to 1.5 hours. The exact value however could not be ascertained though it was always between 105 sec to 165 seconds in all of the tests. Similar tests conducted by Computer Science Corporation [21] indicate that buffer size on the client or server affects the resynchronization time. More investigation is needed to ascertain this.

Hence the decision to retransmit the file fully or resume from the point of interruption depends on the transmission lifetime of the file and resynchronization time. For example, if a 1MB file with a transmission lifetime of 90 sec is disrupted, it makes sense to retransmit the file again when the link is restored rather than waiting for resynchronization and start the transmission again from the point it was left because the transmission lifetime is lower than resynchronization time. For of 10 MB files with transmission life time of 955 sec resynchronization and transmission makes more sense than complete retransmission except the case where resynchronization time plus link availability time exceeds the transmission lifetime.

In the case of directory transmission, if the link is cut and the link cut duration exceeds the server transmission lifetime (sum of the lifetimes of all the files in the directory), then also the client is able to recover all the files in the directory. This is because the MDP server process keeps the track of the total number of files in the directory and the number of files that has been successfully transmitted.

### **5.1.9 Verification of Hypothesis**

We can assert that the first statement in the hypothesis is true. The file transfer durations observed complies with the proposed link interruption algorithm. However we are unable to devise such an algorithm for the second case. Hence we can say the second case is also true, for resynchronization is required to restart file transfer.

## **5.2 File Update Performance**

### **5.2.1 Problem Outline**

In this section we investigate the effect of updating of updating a file on MDP during the course of it's transfer. This characteristic was analyzed to determine whether MDP can be used in real time broadcasts. In real time applications, like broadcasts, the files are updated on the fly and hence the file size increases in the process. Therefore any protocol intended for real time use must be able to handle dynamic file sizes.

### **5.2.2 Hypothesis**

File cannot be successfully updated during its transfer without restarting the client and server processes.

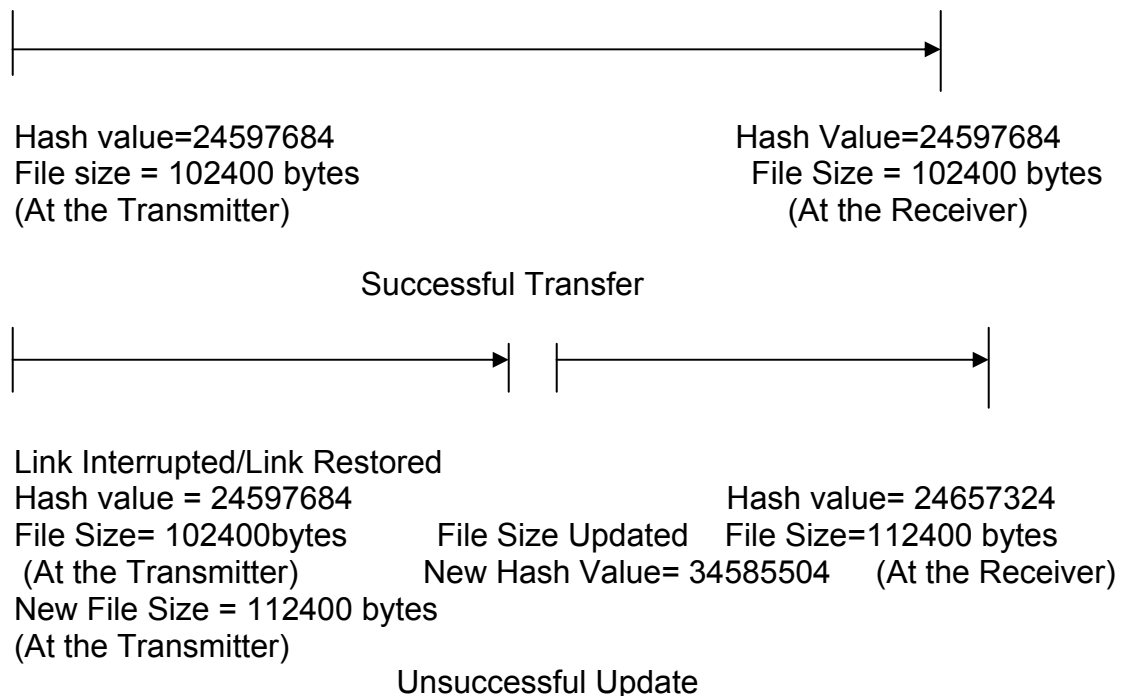
The following tests were conducted to ascertain the behavior of MDP when the file size is updated in the middle of the file transfer

### **5.2.3 Methodology**

1. The transmission rate is set at 88,000 bps

2. Zero error and zero delay conditions are set
3. A file whose transmission lifetime is previously known is chosen
4. The link cuts are fixed at any time interval from the start of transmission but less than the lifetime and the duration are kept sufficient to allow updating of the file at the server.
5. File transfer is allowed to take place
6. At the link cut duration the size of the file is increased and decreased.
7. After the link is restored the transfer is allowed to be completed.
8. The hash value of the received file at the client is compared against the hash value of the updated file at the server.

#### 5.2.4 Test Results



**Figure 5.1** Effect of file updates on file integrity

### **5.2.5 Discussion**

The original size of the file (a readable data file on the server) under test was 102400 bytes and had a corresponding hash value of 24597684. Data transfer was allowed to commence but was interrupted before end-to-end transfer of the file was completed. In the period of interruption the file was updated from an original size of 102400 bytes to 112400 bytes and a new hash value of 34585504 was computed. After the link was restored the file transfer resumed and was completed after some time (depending on transmission life time and link interruption duration). Then the size of the file transferred was noted on the client and was found to be 102400 bytes in spite of the size update. Hash value of the file, which was transferred was again computed and was found to be 24657234 which is different from the hash values of the original file and updated file. Also the file could not be opened for reading. It was assumed that it was due to lack of format and structure in the file.

At the start of transmission of a file MDP sends a packet known as MDP\_FILE\_INFO concerning the particulars of the to be transmitted file including it's size. Hence the file may not be updated until its transmission is

complete or it is rendered useless due to format errors. If the size of the file is increased during transmission, up to original file size is transmitted and if its decreased during the transmission MDP assumes the file to be corrupted and discards the transmitted file on the receiver.

### **5.2.6 Verification of Hypothesis**

We can now assert that the hypothesis made is correct because the hash values of the updated file and the file received do not match. Also the received file size remains same as the file size at the start of the data transfer. Thus we can conclude that MDP is unsuitable for real time data transfers where the file size is dynamic.

## **5.3 Effect of Parity Blocks on Performance**

### **5.3.1 Problem Outline**

The objective of this section is to analyze the effect of deliberately inserted in the MDP coding block on file transfer durations in high error environments.

In its default configuration, MDP sends no parity packets with the data and follows a reactive policy [22] that is, parity packets are inserted by the protocol during the error correcting sessions and not in the original transmission. Transmissions in high error environments generate a large number of retransmission requests. Hence sometimes it pays to follow a proactive policy, deliberately inserting parity packets in each MDP coding block and this should result in a number of NACKS are generated are few

along with fewer retransmissions. It is particularly useful where the channel is simplex though it adds to the overhead and leads to higher transfer duration. In few cases it actually decreases the transfer duration than it would be without the parity packets [23].

### **5.3.2 Hypothesis**

There is an optimum number of parity blocks that can be sent without increasing the transfer duration and that the number is a function of file size and channel BER. The optimal number is the number of parity blocks deliberately inserted in each MDP coding block gives the lowest throughput time. The optimal number can be also be zero indicating that the protocol is left best to follow a reactive policy under that circumstance.

### **5.3.3 Methodology**

1. The transmission parameter is set at 88,000 bps
2. The error vector is selected as desired depending on the test
3. Initially data transfer is done with the auto parity parameter “n” set to 0 and the throughput times are noted.
4. In the subsequent cases the parity packets are deliberately inserted in the MDP coding block by setting the auto parity parameter to 10,20...60 until the throughput time is lower than with the one recorded at step 3. The numbers of parity packets are increased till the throughput time exceeds that of step 3.
5. In each test 16 random files of 10MB and 1MB were test run.

#### 5.3.4 Statistical Analysis

Based on the analysis done by [14] we chose the number of times each file to be sent as 16, which is sufficiently large to statistically detect the significant mean difference of 1 second at a 95% confidence level. Most satellite transfers can be thought of a single-attempt trial. The network would have no memory of previous results or chance to optimize based on previous data streams. We consider 16 replicate observations will be representative of these single shot attempts at data transfers.

When comparing the file transfer time, the initial measurement is the average transmission time,  $\mu$ . The average transmission time  $\mu$  for each file, namely 1M and 10M at  $0, 10^{-6}, 10^{-5}$  and  $10^{-4}$  BER is calculated using the formula

$$\mu = \frac{1}{N} \sum_{i=1}^N t_i \quad (1)$$

Where  $t_i$  is the transfer time for a single run of a file through Space Channel Simulator in a set of N files to be transmitted.

The standard deviation in the time, s is computed using the following equation:

$$s = \frac{1}{N-1} \sum_{i=1}^N (t_i - \mu)^2 \quad (2)$$

Our experimental goal is to determine if there is any statistically meaningful difference between the throughput times obtained by inserting parity packets under the similar error conditions. For this we need the average time and standard deviation for the statistical comparison and the desired confidence level for the comparison is chosen as 95%.

The experimental analysis then becomes a testing of the hypothesis that the mean transfer time,  $\mu_A$ , of Configuration A (Transfer duration obtained by inserting a particular number of parity packets at a given error rate) is statistically same (or statistically different) as the mean transfer time,  $\mu_B$ , of configuration B (Transfer duration obtained by inserting a particular number of parity packets at the same error rate) at a confidence level of 95%.

The hypothesis testing problem can be written as (6):

- $H_0$ : The mean of the first configuration,  $\mu_A$  is same as the mean of the second configuration,  $\mu_B$
- $H_1$ : The mean of the first configuration,  $\mu_A$  is different from the mean of the second configuration,  $\mu_B$

To test the difference in the means of two configurations, we use the basic confidence level method of discriminating between two means.

$$\mu_A - \mu_B = \bar{x}_A - \bar{x}_B \pm z_{\alpha/2} \text{SE} (\bar{X}_A - \bar{X}_B) \quad (3)$$

Here we are looking at the equation for the difference in the means assuming that a large enough set of measurements is available that the normal

distribution can be used. If a small number ( $n < 30$ ) of measurements is available, the critical value for the t-distribution is used instead of the critical value for the normal distribution, z. In this test procedure, if the 95% confidence level includes 0, then the hypothesis  $H_0$  is accepted and if the 95% confidence level does not include zero the hypothesis  $H_1$  is accepted. For a 95% confidence level,  $\alpha = 0.05$  and  $\alpha/2 = 0.025$  and the number of degrees of freedom,  $\nu$ , is  $\nu = N_A + N_B - 2$  where  $N_A$  and  $N_B$  are the number of data points in each configuration. In equation (3), the standard error is computed using

$$SE (\bar{X}_A - \bar{X}_B) = s_{pool} \sqrt{\frac{1}{N_A} + \frac{1}{N_B}} \quad (4)$$

Here the protocol variance is computed using

$$s_{pool}^2 = \frac{(N_A - 1)s_A^2 + (N_B - 1)s_B^2}{N_A + N_B - 2} \quad (5)$$

The standard deviation for each of the two groups is computed using Equation (2).

### 5.3.5 Test Results and Analysis

These results were obtained from the test run of 16 random files of 1MB and 10MB under bit error rates of  $10^{-5}$  and  $10^{-4}$  with different number of parity packets deliberately inserted in the MDP coding block at the start of data

transmission. We noted the time taken for end-to-end file transfer in each case and computed the average data transfer time and corresponding standard deviation. The standard deviation was especially calculated in order to statistically distinguish the computed averages and determine the optimal number of parity packets required to achieve the least transfer time. We have also plotted the graph between the number of parity packets inserted and the average time taken for data transfer. Though the points on the any particular graph may seem scattered and different from each other, they might not be statistically distinguishable.

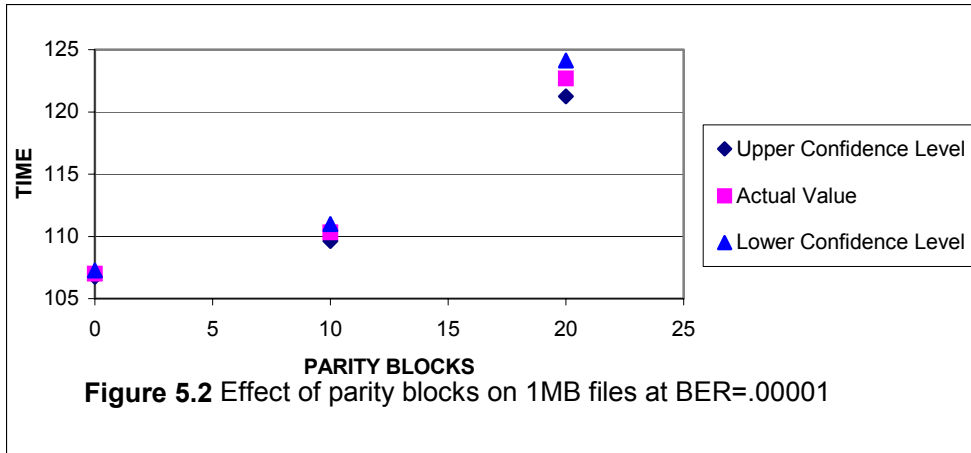
**1> BER = 0.00001**

**i. 1MB files**

File Transfer time for all 1MB files (random set) under similar conditions are given in Appendix - A

**TABLE 5.5** Average time taken by 1MB file at different number of deliberately inserted parity packets.

File Size	Number of parity Packets	Average	Std Dev	Upper Confidence Level	Lower Confidence Level
1MB	0	107.43	2.104	107.25	106.75
1MB	10	110.31	1.4	110.99	109.62
1MB	20	122.68	2.937	124.12	121.24



To determine whether two points are statistically the same or different we used the statistical analysis described in the previous section. We shall illustrate the computation for a pair of points in this section and determine if they are statistically distinguishable from each other. The computations in the following sections have been done in the same way.

$$\bar{X}_A = 107.43 \quad \bar{X}_B = 110.31, \quad N_A = 16 \quad N_B = 16, \quad S_A = 2.104 \quad S_B = 1.4$$

For a 95% confidence level,  $\alpha = 0.05$  and  $\alpha/2 = 0.025$

$$z_{\alpha/2} = 1.960 \text{ (t- distribution)}$$

$v = N_A + N_B - 2$ , that is equal to 30.

$$\text{Protocol Variance } s_{pool}^2 = \frac{(N_A - 1)s_A^2 + (N_B - 1)s_B^2}{N_A + N_B - 2} = 3.193$$

$$SE(\bar{X}_A - \bar{X}_B) = s_{pool} \sqrt{\frac{1}{N_A} + \frac{1}{N_B}} = .6293$$

$$\mu_A - \mu_B = \bar{x}_A - \bar{x}_B \pm z_{\alpha/2} \text{SE} (\bar{X}_A - \bar{X}_B) = (4.113, 1.647)$$

If  $\mu_A - \mu_B$  includes zero in the range computed or that the confidence interval obtained includes zero, the pair of points are not statistically similar. If it does not include zero then they are said statistically different from each other. In the above case the confidence interval does not include zero and hence they are statistically distinguishable.

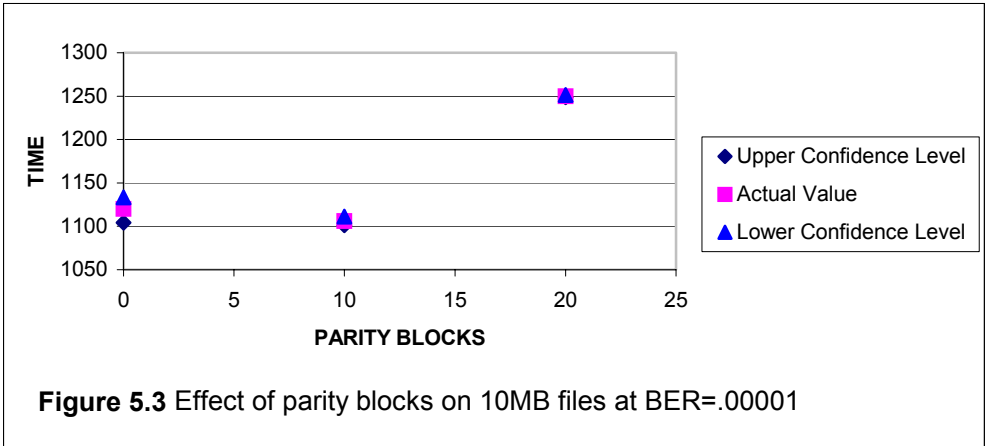
Thus this case (table 5.5) the transfer time obtained for 1MB file with no parity packet inserted is 107.43 sec with a standard deviation of 2.104 sec which is statistically significant from the average transfer time obtained with the deliberate insertion of 10 parity packets. So by hypothesis  $H_1$  the optimal number of parity packets for 1MB file size under error rate of  $10^{-5}$  is same as the default configuration with no parity packets inserted. Figure 5.3 confirms the observation that the least data transfer time was obtained with no parity packets deliberately inserted.

## ii. 10MB Files

File Transfer time for all 10MB files (random set) under similar conditions are given in Appendix - B

**Table 5.6** Average time taken by 10MB file at different number of deliberately inserted parity packets.

File Size	Number of parity Packets	Average	Std Dev	Upper Confidence Level	Lower Confidence Level
10MB	0	1120	2.10	1133.36	1104.26
10MB	10	1106.12	10.21	1111.12	1101.11
10MB	20	1250	2.87	1251.4	1248.59



**Figure 5.3** Effect of parity blocks on 10MB files at BER=.00001

In this case (Table 5.6) the transfer time obtained for 10MB file with no parity packet inserted is which is not statistically different from the average transfer time obtained with the deliberate insertion of 10 parity packets. But the mean transfer duration obtained from inserting 20 parity packets is statistically different from the means obtained by inserting null and 10 parity packets. Hence here the optimal number of parity packets inserted can either be null or 10 to achieve the best throughput time. Looking at figure 5.4 we observe that the time achieved at 0 and 10 parity packets are very similar while data

transfer time increases significantly at 20 parity packets. The statistical analysis confirms our observation.

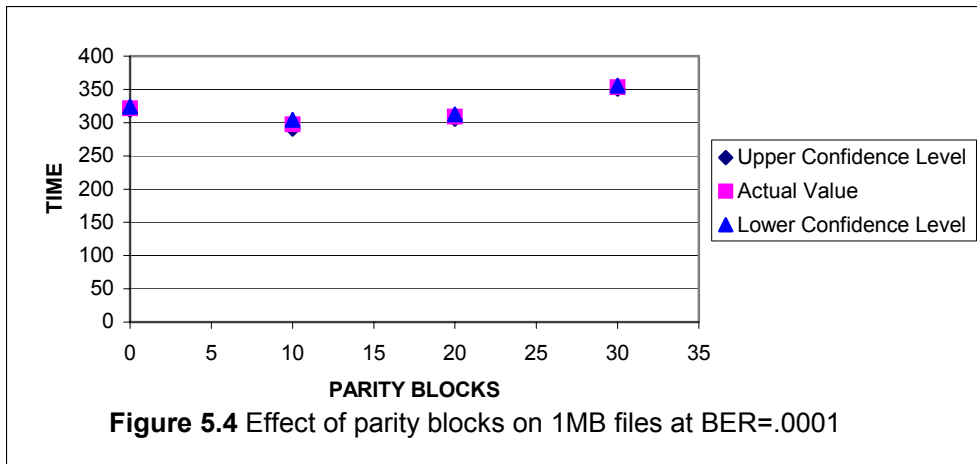
**2> BER = .0001**

**I. 1MB Files**

File Transfer time for all 1MB files (random set) under similar conditions are given in Appendix - C

**TABLE 5.7** Average time taken by 1MB file at different number of deliberately inserted parity packets.

File Size	Number of parity Packets	Average	Std Dev	Upper Confidence Level	Lower Confidence Level
1MB	0	321.56	4.71	323.86	319.25
1MB	10	297.43	13.78	304.18	290.67
1MB	20	308.87	6.83	312.21	305.52
1MB	30	325.37	5.22	355.93	350.81



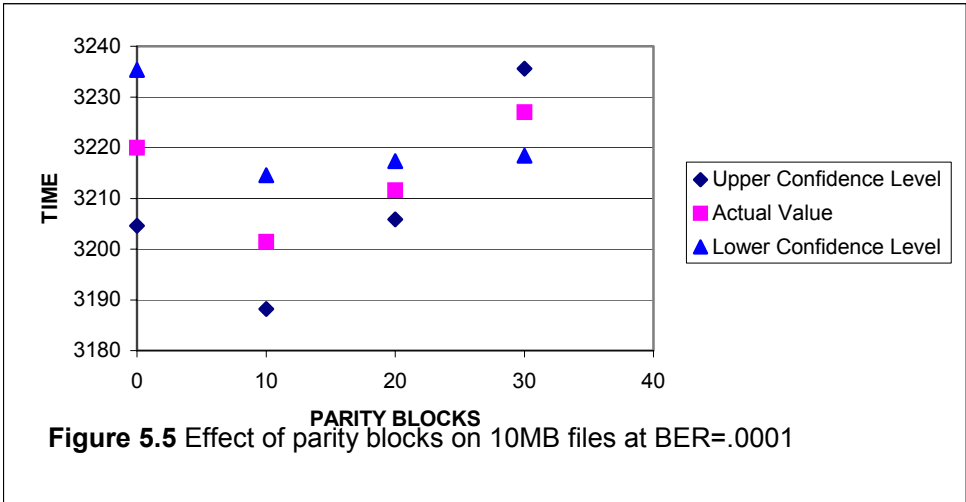
In this case (table 5.7) the transfer time obtained for 1MB file with no parity packet inserted is statistically significant from the average transfer time obtained with the deliberate insertion of 10 parity packets. But the mean transfer duration obtained from inserting 20 parity packets is not statistically significant from the means obtained by inserting zero and 10 parity packets. Hence here the optimal number of parity packets inserted can either be 10 or 20 to achieve the best throughput time. Although from figure 5.5 it appears that the least time is achieved at 10 parity packets, the statistical analysis shows us that the results are statistically indistinguishable. In other words inserting 10 or 20 parity packets shall give a lower data transfer time than that achieved at 0 parity packets

## II. 10MB Files

File Transfer time for all 10MB files (random set) under similar conditions are given in Appendix – D

**TABLE 5.8** Average time taken by 10MB file at different number of deliberately inserted parity packets.

File Size	Number of parity Packets	Average	Std Dev	Upper Confidence Level	Lower Confidence Level
10MB	0	3220.5	31.35	3235.36	3204.63
10MB	10	3201.43	27.01	3294.6	3188.19
10MB	20	3211.62	11.19	3217.34	3205.89
10MB	30	3227	17.55	3235.59	3218.4



In this case (table 5.8) the transfer time obtained for 10MB file with no parity packet inserted is which is not statistically different from the average transfer time obtained with the deliberate insertion of 10 parity packets. Neither

the mean transfer duration obtained from inserting 10 parity packets is statistically significant from the means obtained by inserting 20 and 30 parity packets. With none of the obtained throughput times can be statistically distinguished from each other we can assert that the optimal number of parity packets in this case is null .The protocol is left to best following a reactive policy to ensure the best throughput time. Though from figure 5.6 it appears that time durations achieved at 0,10,20 and 30 parity packets are different from each other and the lowest obtained time is at 10 parity packets but they are statistically indistinguishable due to high standard deviations achieved on them.

#### **5.3.6 Verification Of Hypothesis**

On the basis of the statistical analysis computed in section 5.3.5 we can assert that there exists an optimal number of parity blocks depending on the file size and channel error rate. Thus the hypothesis is shown to be true.

## 6. CONCLUSIONS

In this thesis we have examined the suitability of using the Multicast Dissemination Protocol in an environment that would be expected to be found in a small satellite communications environment. These environments have limited data rates, high channel error rates, and gaps in the transmission.

We did some preliminary testing on MDP to evaluate the basic features of MDP like throughput, optimum transmission rate, effect of propagation delays and protocol operation in simplex channels. We discovered that the optimum transfer rate on which file transfer duration depends is independent of the file size and is dependant on the hardware settings like buffer capacity of the network card or serial port. We saw that in the zero error environment MDP performed the same as FTP in terms of throughput. But MDP takes an edge in high error conditions (BER  $10^{-5}$  and above) and is able to successfully transmit large files of the size of 10MB or more while FTP ceases transmission or takes much more time to complete the transfer compared to MDP. We also found that MDP, being UDP based, successfully handles high propagation delays but at the cost of higher file transfer duration than that achieved in the absence of any propagation delay in the link. MDP is able to function in case of simplex channels or where there is no return channel available on a best effort basis. FTP being positive acknowledgement based fails to work without a return channel being present.

We investigated the MDP protocol operation on interrupted links, both intermittent and scheduled. While doing so we introduced a new variable named TRANSMISSION\_LIFETIME, which is dependent on the file size and the channel error conditions. We proposed an algorithm to predict the performance of MDP in face of intermitted links and subsequent tests proved it to be true. We also discovered that restart and resynchronization might be required in case of scheduled link cuts. Next we tested MDP as a protocol for real-time data and file dissemination and found that it may not be used with applications where the file size is dynamic. Lastly we tried to decide the optimal number of deliberated inserted parity packets or blocks in MDP coding block to influence throughput times. We found that the optimal number is dependant on the file size and the channel error rate and in some cases it is better to follow a reactive policy on parity packets than a proactive one. In the other words under some combinations of file sizes and channel errors the protocol works best when left in the default mode. But the advantage we discovered in using parity packets though at the expense of increasing the transfer duration is that it reduced the number of retransmissions. This might be useful in file transfer scenarios where the channel error rate is high and the return channel is unreliable

Overall MDP handling negative acknowledgement scheme at the application layer and using connectionless UDP at the transport gives it advantages in link management and reliable transfers in high error

environments. The studies shows that MDP has desirable features, which makes it attractive for bulk data transfer in asymmetric and space Internet work applications. Proactive FEC based repairing improves protocol performance in very high bit error rate scenarios. The ability of the server to hold state information is very useful in handling link cuts and initiate repair sessions long after initial transmission is over. It is capable of operating independent of the network structure which makes it attractive for real time data transfer in space like environments like the wireless systems because of high error rates link disruptions and frequent packet drops.

## **APPENDICES**

**A. Transfer Durations Obtained for 1MB files at BER=. 00001**

File Name	Number of Parity Packets		
	0	10	20
1MB-1	108	112	124
1MB-2	107	110	122
1MB-3	107	110	124
1MB-4	108	110	124
1MB-5	108	111	124
1MB-6	107	110	124
1MB-7	107	110	124
1MB-8	108	110	124
1MB-9	108	110	120
1MB-10	108	110	122
1MB-11	107	108	124
1MB-12	107	180	117
1MB-13	107	41	128
1MB-14	108	114	122
1MB-15	107	108	116
1MB-16	107	109	124

**B. Transfer Durations Obtained for 10MB files at BER=. 00001**

File Name	Number of Parity Packets		
	0	10	20
10MB-1	1230	1103	1251
10MB-2	1115	1105	1250
10MB-3	1112	1104	1251
10MB-4	1109	1103	1251
10MB-5	1110	1105	1253
10MB-6	1111	1103	1251
10MB-7	1110	1102	1250
10MB-8	1109	1104	1249
10MB-9	1114	1104	1250
10MB-10	1111	1107	1250
10MB-11	1112	1104	1251
10MB-12	1111	1106	1250
10MB-13	1114	1110	1250
10MB-14	1109	1139	1253
10MB-15	1112	1087	1250
10MB-16	1112	1112	1240

**C. Transfer Durations Obtained for 1MB files at BER=. 0001**

File Name	Number of Parity Packets			
	0	10	20	30
1MB-1	318	328	317	335
1MB-2	326	327	322	331
1MB-3	320	287	301	330
1MB-4	328	300	308	318
1MB-5	319	285	305	325
1MB-6	322	297	305	320
1MB-7	320	289	310	325
1MB-8	317	300	303	325
1MB-9	316	312	322	330
1MB-10	322	287	303	326
1MB-11	318	287	306	330
1MB-12	317	295	305	318
1MB-13	321	290	317	325
1MB-14	330	288	303	319
1MB-15	320	287	308	320
1MB-16	331	300	307	329

**D. Transfer Durations Obtained for 10MB files at BER=. 0001**

File Name	Number of Parity Packets			
	0	10	20	30
10MB-1	3198	3189	3227	3219
10MB-2	3119	3181	3210	3208
10MB-3	3228	3226	3219	3234
10MB-4	3233	3180	3224	3241
10MB-5	3235	3206	3191	3251
10MB-6	3191	3193	3211	3238
10MB-7	3222	3225	3202	3219
10MB-8	3224	3209	3197	3209
10MB-9	3239	3224	3223	3185
10MB-10	3224	3234	3198	3237
10MB-11	3251	3217	3212	3228
10MB-12	3215	3156	3221	3221
10MB-13	3217	3137	3202	3217
10MB-14	3252	3216	3219	3231
10MB-15	3220	3214	3223	3252
10MB-16	3239	3216	3207	3242

## REFERENCES

- [1] M. Allman, C. Heyes, H. Kruse, S. Osterman, "TCP Performance over Satellite Links," Proc.5<sup>th</sup> international Conf on Telecom Systems, 1997.
- [2] Y. Zhang, E. Yan, S. K. Dao, "A measurement of TCP over Long Delay Network," Proc.6<sup>th</sup> International Conf on Telecom Systems, 1998.
- [3] M. Mathis, J. Mahdavi, S. Floyd, "TCP Selective Acknowledgement Options," IETF, Read for Comments RFC 2018, October 1996.
- [4] Consultative Committee for Space Data Systems, "Space Communications Protocols Specification(SCPS)-Rationale, Requirements, and Application Notes," CCSDS 710.0-G-0.3, April 1997.
- [5] S. Horan, "Design of a Space Channel Simulator Using Virtual Instrumentation Software," IEEE Instrumentation and Measurement Technology Conference Budapest, Hungary, May 21-23, 2001.
- [6] S. Horan and R. Wang, "Effects of Network Configurations On Channel Throughput in Space," Proc.14<sup>th</sup> AIAA/USU Conference on Small Satellites, August 2000.
- [7] S. Horan and R. Wang, "Effects of Protocol Options on Data Throughput over a Simulated Environment," NMSU-ECE-00-007, August 2000.
- [8] J. Macker and R. Adamson, "The Multicast Dissemination Protocol (MDP)," Naval Research Laboratory, October 1999
- [9] S. Horan and S. Muddassani, "Comparison of FTP vs MDP in Simulated Space Channels." Tech Report, New Mexico State University, August 2002.
- [10] A. Mankin, A. Romanow, S. Bradner, V. Paxson, "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols", RFC 2357, June 1998.
- [11] S. Floyd, V. Jacobson, S. Macne, C. Lin, L. Zhang, "A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing," Proc ACM SIGCOMM, August 1995.
- [12] J. Macker, "Reliable Multicast Transport and Integrated Erasure-based Forward Error Correction," Proc IEEE MILCOM 97, Oct 1997

- [13] B. Adamson, "The Multicast Dissemination Protocol Toolkit", Technical White paper, July 1996.
- [14] S. Horan and R. Wang, "Internet-Type Protocol Testing in a Simulated Small Satellite Environment," IEEE Conference on Aerospace and Electronic Systems, March 2001.
- [15] R. Wang, "The Behavior of TCP and Its Extensions in Space," Phd Dissertation, New Mexico State University, August 2001.
- [16] S. Horan and R.H. Wang, "Design of a Channel Error Simulator Using Virtual Instrument Techniques for the Initial Testing of TCP/IP and SCPS protocols," IEEE Transactions on Instrumentation and Measurement, October 2002.
- [17] B. Sklar, Digital Communications, Prentice Hall, Upper Saddle River, NJ, 2001.
- [18] J.C. Moser and W.P. Osborne, "Error Pattern Generation for Coded BPSK," NMSU-ECE-95-007, August 1995.
- [19] Sklar, *ibid*, p. 276.
- [20] Latency Information at [www.mids.org/weather](http://www.mids.org/weather).
- [21] E. Crisculo, "MDP: Reliable File Transfer for Space Mission," Computer Science Corporation, May 21, 2002.
- [22] D. Gossink, J. Macker, "Reliable Multicast and Integrated Parity Retransmissions with Channel Estimation," IEEE GLOBECOM 1998.
- [23] A. Tanenbaum, Computer Networks, Prentice Hall PTR, Upper Saddle River, NJ, 1989.