MASTER'S THESIS

Error Control for Satellite and Hybrid Communication Networks

by D.E. Friedman
Advisor: A. Ephremides

CSHCN M.S. 95-1
(ISR M.S. 95-10)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

Abstract

Title of Thesis: Error Control for Satellite and Hybrid Communication Networks

Name of degree candidate: Daniel E. Friedman

Degree and year: Master of Science, 1995

Thesis directed by: Professor Anthony Ephremides
Department of Electrical Engineering

Both forward-error-correction (FEC) and automatic-repeat-request (ARQ) error control schemes are used for assuring the accuracy of information transferred through imperfect channels. In satellite systems, in which propagation times are typically large, ARQ error control can result in poor throughput to the destination. Also, an ARQ protocol for satellite multicast communication must be carefully crafted to assure good throughput to all destinations regardless of which stations require retransmissions.

Supplementing a satellite link with a parallel terrestrial link may allow mitigating some problems of using ARQ in satellite communication systems. ARQ acknowledgments, and possibly retransmissions as well, can be sent terrestrially in such a hybrid network, and so avoid the large satellite propagation delay. The satellite transmission capability of a receiving station which communicates

with the transmitter exclusively by terrestrial means can be eliminated and the system cost correspondingly reduced. Further, multicasting with a hybrid network may allow retransmissions to be conducted without interrupting the flow of new information to all destinations, so throughput need not drastically suffer if retransmissions are required.

The degree to which throughput can be improved by adopting a hybrid network is not clear. A hybrid network's effect on the fidelity of information delivered to the destination(s) is also not clear. An experiment is presented for investigating such error control issues of hybrid networking.

Error Control for Satellite and Hybrid Communication Networks

by

Daniel E. Friedman

Thesis submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1995

Advisory Committee:

Professor Anthony Ephremides, Chairman/Advisor
Professor John Baras
Professor Evaggelos Geraniotis

Acknowledgments

I wish to express my thanks to some of the many individuals who made this work possible. Foremost among these is my advisor, Professor Anthony Ephremides, who has provided me excellent opportunities for learning and patiently given me invaluable guidance.

I am also grateful to Professors John Baras and Evaggelos Geraniotis for kindly consenting to join the defense committee and review this thesis.

Without the help of the Computing Support Staff of the Institute for Systems Research, in particular Haisong Cai, Amarendranath Vadlamudi and Irving Hsu, it would have been impossible to conduct the experiment which is described in this thesis. Their many efforts, including reconfiguring Sun workstations, making sense of incomplete software documentation, finding disk space to store hundreds of megabytes of data and answering numerous questions are greatly appreciated.

Yannis Konstantopoulos, now of MCI Communications Corporation (Reston, Virginia), assisted with the UNIX networking aspects of the software used for

the experimentation.

The NASA Advanced Communication Technology Satellite (ACTS) Experiments Office (Lewis Research Center, Cleveland, Ohio) granted the proposal for experiments with ACTS, and was kind enough to loan us an ACTS terminal. I would like to particularly thank our experiment liaison to NASA, Barry Fairbanks of Analex Corporation, and Mike Jarrell and Kevin McPherson of the ACTS System Engineering Office, who were especially helpful. Barry helped arrange for the loan of the satellite terminal and continually campaigned on our behalf for more satellite time, while Mike and Kevin helped keep our terminal in working order.

Marjorie Weibul of the National Telecommunication and Information Administration (NTIA) and Dr. Stanley Bush, Fan Huang, Gil Weiss, Neill Cameron, Wenjin Yang and David Bailey of the University of Colorado at Boulder made it possible to use a second satellite terminal in this work, and responded with good nature and sincere effort to our many requests for checking and reconfiguring equipment to suit experimentation.

This work was supported by the Graduate School of the University of Maryland at College Park through a graduate fellowship, and by the Center for Satellite and Hybrid Communication Networks and the Institute for Systems Research through a research assistantship.

Finally, I wish to thank my family and friends for their support and encouragement.

Table of Contents

Section	Page
List of Tables	**vii**
List of Figures	**viii**
1 Introduction	**1**
1.1 Error Control	1
1.2 Error Control for Satellite Communication	3
1.3 Hybrid Networks	5
1.4 An Experiment in Error Control for Satellite and Hybrid Networks	9
1.5 Overview of the Thesis	11
2 Experiment Description	**12**
2.1 Motivation	12
2.2 Experiment Hardware	13
2.2.1 Advanced Communications Technology Satellite (ACTS)	15
2.2.2 ACTS T1 Very-Small Aperture Terminal (VSAT)	15
2.2.3 Frame Relay Access Switch (FRACS)	16

2.2.4 Apparatus in College Park, Maryland 17
2.2.5 Apparatus in Boulder, Colorado 17
2.3 Experiment Design 18
2.3.1 Artificial Noise 19
2.3.2 FEC Tests 20
2.3.3 Point-to Point ARQ Tests 24
2.3.4 Multicast ARQ Tests 28
2.4 Software Overview and Implementation Details 34
2.4.1 FEC Software 34
2.4.2 ARQ Software 35
2.4.3 Additional Details 38

3 Experiment Results and Discussion 40

3.1 Difficulties Experienced 40
3.2 FEC Results 41
3.3 ARQ 41

4 Conclusion 46

A Software Listings 49

A.1 Introduction 49
A.2 Software for FEC Tests 49
A.2.1 Encoders and Decoders 49
A.2.2 FEC Transmitter 50
A.2.3 FEC Receiver 52
A.3 Software for ARQ Tests 54
A.3.1 ARQ Transmitter 54

A.3.2 ARQ Receiver 65
A.3.3 Delayer 72

List of Tables

Number		Page
2.1	FEC transmission blocks.	24
2.2	Combinations of networks and artificial noise bit error rates tested with point-to-point GBN and SR ARQ protocols.	27
2.3	Combinations of networks and artificial noise bit error rates tested with point-to-multipoint GBN and SR ARQ protocols.	33
3.1	Transfer times (in seconds) for FEC tests.	42
3.2	Residual bit error rates for FEC tests.	42

List of Figures

Number Page

1.1 A hybrid network. 6

2.1 Experiment infrastructure. 14

2.2 FEC testing concept. 21

2.3 Point-to-Point ARQ testing concept. 25

2.4 Hybrid network for point-to-two point ARQ tests. 29

Error Control for Satellite and Hybrid
Communication Networks

Daniel E. Friedman

October 29, 1996

This comment page is not part of the dissertation.

Typeset by LaTeX using the `dissertation` class by Pablo A. Straub, University of Maryland.

Chapter 1

Introduction

1.1 Error Control

A communication system is used for transferring information. The communication channel is typically imperfect and may distort information, causing the received data to be an erroneous version of the information which was to have been transmitted. The errors in the received information are undesirable and so error-control techniques must be adopted to transfer information accurately.

The two broad types of error-control techniques are forward-error-correction (FEC) and automatic-repeat-request (ARQ). In FEC, k information symbols are mapped to a larger number of symbols, n, which are sent through the channel. The mapping—the FEC code—is devised so that there is some redundancy in the symbols and so, upon receipt, the redundancy can be exploited to detect and correct errors which may have developed in transport. Thus error control is achieved by sending more symbols than would have been sent if no error control were needed. In general, codes with lower code rates (n/k) or complex codes can protect against more errors. Hence, higher fidelity transmission can be achieved

at the expense of sending more symbols in the channel and/or complexity.

In ARQ-based error control, the information is segmented into packets, to which are attached check symbols calculated from the packet in a manner similar to the mapping in FEC. These symbols are used [generally] only to detect errors, and so n is usually only slightly greater than k. If an error is detected in a received packet, the receiver sends to the transmitter a request for the packet to be retransmitted: a negative acknowledgment. If no error is detected, a positive acknowledgment of the received packet is sent instead. The code used for error detection is usually a cyclic redundancy check (CRC), and such codes provide excellent error detection capabilities while admitting simple implementations.

In some cases, FEC may be preferred for error control, while ARQ may be better in others. If the channel is very noisy, information sent through it will be severely corrupted. If ARQ is used in this case, many retransmissions will be required. A retransmission sent through the same channel as the original transmission will also likely be corrupted. Hence, if channel conditions are poor, information delivery may be slow or non-existent if ARQ error control is employed. A FEC code which can correct many errors would be a better error control choice in this case. Such a code would would have many check symbols although such overhead may be tolerable if high-fidelity communication is required.

If the channel is fairly noise-free, ARQ may be a better choice. Usually, less overhead is required for to provide excellent error detection capability than to achieve a comparable degree of error correction. Hence, less overhead is required for high-fidelity communication if ARQ is employed than if FEC is used. With less overhead, information can be delivered more quickly by ARQ than FEC if

the channel symbol rate is fixed and errors are unlikely. Of course, if no channel is available for feedback from the receiver to the transmitter, ARQ is not a feasible error control option, and FEC must be employed.

It is clear, then, that selecting an error-control scheme entails considering the interrelationships of several factors which include required fidelity of information delivered, amount of acceptable overhead, error characteristics of the channel, implementation cost and the availability of a feedback channel.

1.2 Error Control for Satellite Communication

Communication via satellite has historically been conducted predominantly with satellites in geostationary orbits (at an altitude of 22,300 miles/35,800 kilometers [1]). A characteristic of communication with such satellites is a large propagation delay, more than two-tenths of a second, for transmission via the satellite from one earth station to another. If ARQ is used for error control, the ARQ protocol must be able to provide efficient communication despite this great delay. In the stop-and-wait (SW) ARQ protocol, for example, the transmitter waits for an acknowledgment after sending each packet. With this protocol, new packets can be sent at intervals no smaller than about four-tenths of a second (the propagation times of a packet and an acknowledgment) and new information is transferred to the receiver during less than half the time the protocol operates. Thus the inefficiency of SW ARQ is magnified in satellite operation. This protocol is relatively simple to implement, however.

A more efficient (and more complex) ARQ protocol is the go-back-N (GBN) protocol. With GBN, the transmitter can send new packets continuously if no

retransmissions are required (and the positive acknowledgments from the receiver arrive at the transmitter). If an error in a packet is detected by the receiver, the packet is rejected, a negative acknowledgment is sent to the transmitter, and no other packets are accepted until the one requested is received. While the negative acknowledgment travels to from the receiver to the transmitter, new packets arrive at the receiver but are rejected since none of them is the packet the receiver will now accept. When the transmitter receives the negative acknowledgment, it "goes back" to send the requested packet and continues transmission with packets immediately following the one retransmitted. The number of packets which are sent again is approximately N, so each retransmission request from the receiver causes the transmitter to send approximately N packets another time. The number N, the "window size," is a parameter of the protocol operation and is related to the propagation times for a packet and an acknowledgment. If these propagation times are great, N may be large as well. Consequently, many packets must be sent again for a retransmission request generated by GBN operation in a satellite network. However, if the channel is fairly noise-free, then such retransmissions will be rare and the transmitter will be able to send new information packets to the receiver continuously. Thus, GBN is a more efficient ARQ protocol than SW especially under good channel conditions.

Yet, each GBN retransmission requires retransmitting N packets. It would be more efficient if the negative acknowledgment were to precisely specify the packets the receiver needed to have sent again, and if only these specified packets would have to be retransmitted. This improvement is incorporated in the selective-repeat (SR) ARQ protocol. With SR ARQ, if an invalid packet arrives at the receiver, valid packets received subsequently can be accepted, and

need not be discarded as in GBN. This ability to accept packets out-of-order, combined with the feature of retransmitting only specific packets, makes SR the most efficient of the three ARQ protocols discussed here. A SR ARQ implementation is correspondingly the most complex, and requires buffer space to allow for re-sequencing of packets into their correct order. With the great propagation delays of satellite communication, many packets may arrive at the receiver between the time a retransmission is requested and the time the retransmission arrives, so the buffering requirements may be substantial. This discussion shows that if ARQ is used for error control in satellite communication, a sophisticated protocol is required for reasonable throughput to be achieved. Such a protocol will be complex and require significant amounts of memory.

However, while both the complexity and memory requirements add costs to the system, the cost for the required feedback satellite channel may be far greater. In order to use the satellite for feedback, the ground terminal of the receiving station must be able to transmit on the satellite channel. Such satellite transmission capability can compose a substantial portion of the terminal's cost. This large cost to implement ARQ for satellite communication, combined with the great propagation delay which magnifies ARQ inefficiencies, reduce the attraction of ARQ for satellite communication error control. Accordingly, FEC methods are more commonly used for error control in satellite communication.

1.3 Hybrid Networks

If the propagation delay and the cost of a feedback channel could both be reduced, perhaps ARQ would be a more attractive error control scheme for satellite

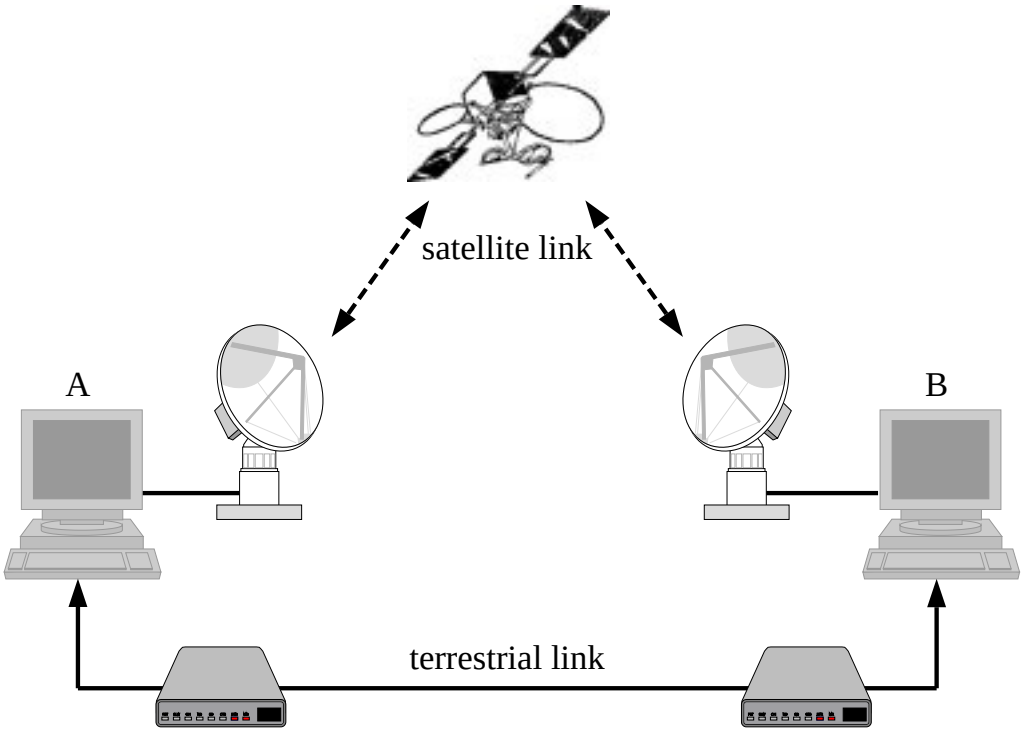

Figure 1.1: A hybrid network.

communication. This motivates supplementing the satellite communication link with a parallel terrestrial link, such as a modem connection through the public telephone network, as depicted in Figure 1.1. Such a network of parallel satellite and terrestrial links is termed in this work a *hybrid network*.

Supplementing a satellite link with a parallel terrestrial link offers additional possibilities for error control, particularly ARQ. A retransmission request can be sent terrestrially and so avoid most of the propagation delay experienced with satellite transport. A terrestrially-carried retransmission request will arrive at the transmitter sooner than one sent via satellite, so the transmitter can begin retransmission sooner. Thus the receiver need not wait as long for its desired retransmission to arrive. Thus suggests it may be possible to achieve a higher net rate of information transfer with a hybrid network. It is possible the memory

requirements for ARQ can be relaxed as well in a hybrid network. Further improvement might be attainable by sending retransmitted packets terrestrially as well. The degree of throughput improvement will be discussed below in 1.4.

An additional benefit of a hybrid network is one of cost reduction. Suppose A and B are stations in a hybrid network, and A sends to B a body of information through the satellite using ARQ error control. Station B transmits only positive and negative acknowledgments, which can be carried terrestrially instead of via satellite. If B transmits only on the terrestrial link, it does not use its capability to transmit on the satellite link. Hence the satellite transmission capability of station B can be eliminated. As satellite transmission capability composes a substantial portion of a satellite terminal's cost, a significant cost reduction can be achieved by adopting a hybrid network architecture.

A hybrid network concept may also be used to supplement a terrestrial link. For example, many people access the Internet and other computer networks through modems and the public telephone system. Commonly, a user will request much more information from the network than he will send to/through it. The speed of the user's modem connection may be the principal constraint upon the rate at which requested information can be downloaded. Even with the fastest modems available, downloading a file, image, etc. from the network may require several minutes. Now, if a relatively high-rate satellite link were to be used for the link from the network to the user, download times could be greatly reduced. To achieve such improvement, the user would require a receive-only satellite terminal, which can be purchased for less than $1000. So, the user can obtain at reasonable cost a means for retrieving information from a network at speeds much greater than attainable through modems. Hughes

Network Systems has developed the hybrid network concept for this purpose into its recently-announced *DirecPC* product.[1] This is an example of the great commercial potential of hybrid networks, driven by the availability of low-cost receive-only satellite terminals.

The difference in rates of the satellite and terrestrial links of a hybrid network poses a restriction on the sort of communication which can be supported by this type of network. Commonly, there is much information to send in one direction of a communication, but much less to send in the opposite direction. Examples of such *bandwidth-asymmetric communication* are file transfer and multimedia database lookup. In such communication, one station requests a large amount of information from another, and the requested information is then sent to the first station. A hybrid network can be used for this type of communication: the request can be sent terrestrially and the bulk information can be returned via satellite. So, while the hybrid network is well-suited for bandwidth-asymmetric communication and perhaps not appropriate if the bandwidth requirements are symmetric, much communication falls into the former category, making the hybrid network a good architecture for many communication needs.

While the discussion to this point has been about point point-to-point (unicast) communication, a satellite is an excellent facility for point-to-multipoint (multicast) communication. Accordingly, the hybrid architecture ought be considered for multicasting as well as for unicasting. The general problem with using ARQ error control for multicast communication is that the need of one receiver for a retransmission may cause cause the throughput for all receivers to be reduced. More precisely, a retransmitted packet sent over the multicast

[1] *DirecPC* is a trademark of Hughes Network Systems.

channel interrupts the flow of new packets to all stations and does not benefit those receivers which do not require the retransmission. However, if there is a terrestrial link between each receiver and the transmitter, retransmitted packets can be sent terrestrially only to the stations requiring them, while the transmitter can continue sending new packets over the satellite multicast link. With such a multicast hybrid network, if only some stations require retransmissions, the throughput to all stations need not suffer so drastically as in the case of using satellite-only network. A challenge here is to devise the ARQ protocol so that such benefits can be achieved.

1.4 An Experiment in Error Control for Satellite and Hybrid Networks

It has been suggested above that throughput to receiver(s) in both unicast and multicast satellite networks could possibly be improved by adopting a hybrid network. The degree of this improvement will depend on the ARQ protocol as well as on the size of a packet, the transmission rates through the satellite and terrestrial links and the propagation delays of these links. For example, while the terrestrial link has smaller propagation delay than the satellite link, the former has a lower rate than the latter, so the terrestrial transmission time of, say, an ARQ acknowledgment, may offset the gain of lesser propagation delay. Hence, it is possible a hybrid network may not provide any throughput improvement for some combinations of network and protocol parameters. Conversely, given the propagation times and transmission rates of both links, there is a challenge to find the optimal combination of window size, ARQ timer period and ARQ

protocol.

A terrestrial link typically has different error characteristics than a satellite link, so an ARQ protocol may behave differently in a hybrid network than in a satellite-only network. Thus the hybrid network presents both additional possibilities and problems for error control. The precise differences of satellite and hybrid networks, including effects on throughput and error control, as well as best protocols to use with a hybrid network, are not immediately seen.

An experiment, discussed in the next two chapters, was conceived and developed by the Center for Satellite and Hybrid Communication Networks (CSHCN) at the University of Maryland to investigate such error control issues of hybrid networking. This experiment was one of a group of advanced networking experiments proposed by the CSHCN in 1992. The experiments were originally motivated by the availability of low-cost receive-only satellite terminals and the commercial potential of hybrid networks, but their scope was later expanded to include other advanced networking concepts. Three experiments were ultimately devised:

1. An examination of dynamic allocation of satellite bandwidth in response to variations in amount of traffic to be sent through the satellite.
2. Investigation of error-control schemes for use in a hybrid network, with particular attention given to multicasting.
3. Remote multimedia database access via a hybrid network and performance comparison of networking protocols in local area network (LAN) interconnection.

1.5 Overview of the Thesis

The second of the three experiments is discussed in the next two chapters of this thesis. The experiment was developed between May 1993 and August 1995, but was not completed in time for all results to be included in this thesis. Chapter 2 describes the design and implementation of experiment, and Chapter 3 presents the available results and discusses the experiment further. Finally, Chapter 4 presents conclusions and ideas for further work. Listings of experiment software are provided in a following appendix.

Chapter 2

Experiment Description

2.1 Motivation

The terrestrial link of a hybrid network can be used as a path for feedback from the receiver to the transmitter. This presents additional possibilities for error control, particularly ARQ, as well as additional problems, due to differing characteristics of satellite and terrestrial links. Also, since communication with a terrestrial link does not incur the great propagation delay of satellite transport, it has been speculated that throughput in a satellite network could perhaps be improved with a hybrid network. The effects on error control are not clear, however, as described in the previous chapter. Hence an experiment was proposed to investigate ARQ schemes in hybrid networks. Since a satellite is an excellent medium for point-to-multipoint communication, both point-to-point (unicast) and point-to-multipoint (multicast) communication was considered.

To gauge the improvement offered by a hybrid architecture, an investigation and comparison of error control techniques in a hybrid network should include examining error control techniques used in a satellite-only network. In particular,

FEC is commonly used for satellite communication, so FEC error control schemes should be included in the comparison. Also, the relative throughput and fidelity of ARQ and FEC schemes depend on many factors, including the incidence and nature (bursty, random, etc.) of channel errors, propagation times and channel rates. For such reasons, the experiment also considered FEC error control.

Accordingly, the experiment may be outlined as follows:

1. Testing of FEC schemes in a satellite-only network.
2. Testing of ARQ schemes for point-to-point communication in satellite-only and hybrid networks.
3. Testing of ARQ schemes for point-to-multipoint communication in satellite-only and hybrid networks.

For all tests, the measures of chief interest were the time required to transfer a prescribed amount of information and the bit error rate (BER) of that information after receipt (and decoding, if applicable). The experiment was designed to allow comparison of these results not only within each of the three parts, but also between parts 1 and 2 and parts 2 and 3. Such comparisons are discussed in Chapter 3.

The experiment hardware will be presented next, followed by the experiment design.

2.2 Experiment Hardware

The experiment was conducted with a satellite and two earth stations, one at the University of Maryland at College Park (UMCP) and the other in Boulder, Colorado. This infrastructure is diagrammed in Figure 2.1. Communication

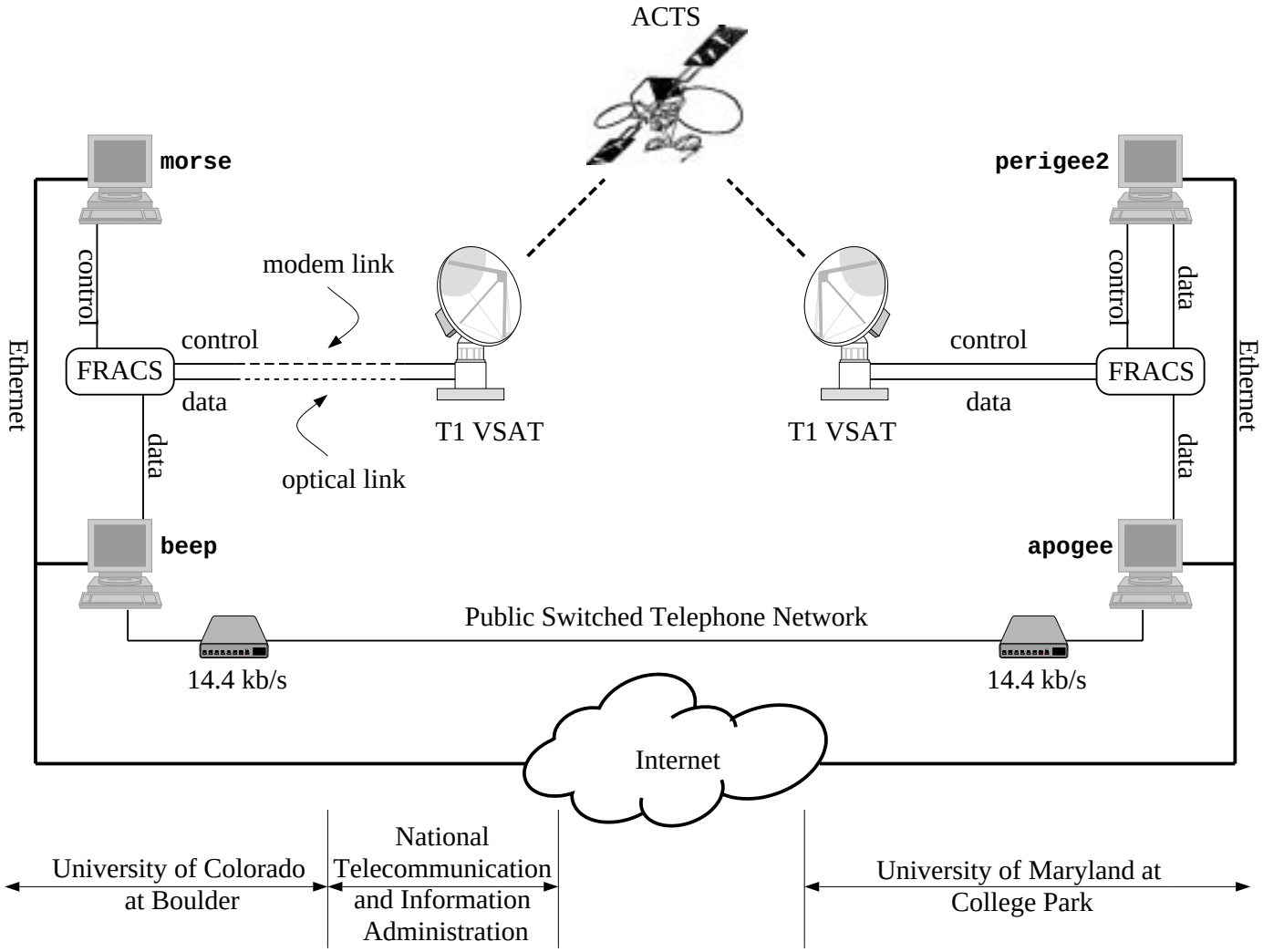

Figure 2.1: Experiment infrastructure.

between these stations was conducted with Sun computers running both commercial and specially-developed software (described in 2.3).

2.2.1 Advanced Communications Technology Satellite (ACTS)

The satellite used in this experiment was NASA's Advanced Communications Technology Satellite (ACTS), which was launched in September 1993. The technological innovations incorporated in this satellite include operation at K_a-band (30 GHz uplink/20 GHz downlink), hopping spot beams, on-board processing, and signal regeneration through demodulation/remodulation. Each of the features just listed allows the use of terminals with very small antennas. In ACTS's baseband processor (BBP) mode, which is the mode used for this experiment, FEC can be applied as needed to combat fading due to precipitation. The satellite also has a microwave switch matrix (MSM) mode, which can support very high rate satellite-switched TDMA operation.

ACTS is controlled from a master control station located at the NASA Lewis Research Center in Cleveland, Ohio [2].

2.2.2 ACTS T1 Very-Small Aperture Terminal (VSAT)

The most common way of using ACTS's BBP mode is with a T1 very-small aperture terminal (VSAT). The name "T1 VSAT" is somewhat inaccurate, for this terminal actually supports a maximum of 28 64 kb/s channels. This terminal supports both voice calls, carried at a rate of 64 kb/s, and data calls, at multiples of 64 kb/s.

Physically, the T1 VSAT comprises an outdoor dish antenna and associated electronics, a rack of indoor electronics and connecting cabling. The antenna diameter is usually either 1.2 or 2.4 meters.

If suitably configured, the T1 VSAT can accept commands from an external computer for making and breaking calls through ACTS [3, 4].

2.2.3 Frame Relay Access Switch (FRACS)

An example of such an external processor is the Frame Relay Access Switch (FRACS) used in the CSHCN ACTS experiments. The FRACS is a Motorola 68040-based unit custom-developed for the CSHCN by COMSAT Laboratories (Clarksburg, Maryland). In addition to controlling the VSAT, the FRACS served as a frame relay interface to the ACTS system. While frame relay was not integral to this experiment, this protocol was examined in a separate CSHCN experiment (Experiment 3). Further, the FRACS can be used for dynamically allocating bandwidth to suit traffic requirements, a capability utilized in yet another CSHCN experiment (Experiment 1) [5].

In this experiment, the FRACS was used solely as a traffic interface to the VSAT, and for making/breaking calls. Traffic and control information were carried between the VSAT and the FRACS over T1 and 9600 b/s RS-232 connections, respectively. The FRACS provides commands for reporting packets sent and received via its T1 interface and via ACTS, for changing the allocated satellite bandwidth, and for configuring interfaces, frame relay virtual circuits, and the bandwidth allocation algorithm. For this experiment, the bandwidth allocation algorithm was disabled and all ACTS circuits were established manually.

2.2.4 Apparatus in College Park, Maryland

As mentioned above, one of the earth stations used in this experiment was located in College Park, Maryland. A T1 VSAT, kindly loaned by NASA for CSHCN experimentation on ACTS, was installed at the University of Maryland in July 1994. This VSAT had an antenna 1.2 meters in diameter.

A FRACS was used as an interface unit between the VSAT and two Sun workstations which ran experiment software. Each of these workstations, named `apogee` and `perigee2`, was connected to the FRACS through a Sun High-speed Serial Interface (HSI). This interface afforded a traffic data connection, of rate up to 1.536 Mb/s (T1), between each workstation and the FRACS [6]. SunLink Frame Relay software was used on the workstations for this connection. Additionally, the FRACS was controlled and monitored through a 4800 b/s RS-232 serial connection to `perigee2`. (This connection was made to the FRACS's "console" port.)

The two computers were connected to an Ethernet LAN, and so to the Internet. `Apogee` was also linked to a peer computer in Boulder via a dialed-up 14.4 kb/s modem connection through the public switched telephone network (PSTN).

The VSAT indoor equipment, FRACS and workstations were located in the Systems Integration Laboratory (SIL) of the Institute for Systems Research at the University of Maryland.

2.2.5 Apparatus in Boulder, Colorado

The hardware installation in Boulder, Colorado was similar to that in College Park, Maryland. Again, two computers, named `beep` and `morse`, were used.

Of these two, only `beep` was equipped with a Sun High-speed Serial Interface, and so was the only computer in Boulder which could transmit data through a FRACS. Hence, `beep` was the only computer in Boulder which operated experiment software. As with `apogee` and `perigee2` in College Park, `beep` operated SunLink Frame Relay software during this experiment. The second computer in Boulder, `morse`, was used exclusively for controlling the Colorado FRACS (primarily making and breaking calls).

As in College Park, both of the computers in Boulder were connected to the Internet through an Ethernet LAN. A 14.4 kb/s modem was attached to `beep` for communication with `apogee` through the PSTN.

A significant difference between the hardware arrangements in Boulder and College Park was that the VSAT in Boulder was not located with the experiment computers. Rather, the Boulder VSAT was installed at the National Telecommunication and Information Administration (NTIA). (This VSAT had a 2.4 meter diameter antenna.) The balance of the Boulder apparatus was located in the Telecommunications Laboratory of the University of Colorado. A two-way infrared beam through air—an "optical" link—was used for conveying traffic data between the VSAT and the experiment computers, while a 4800 b/s modem link carried control information between the VSAT and the FRACS. Equipment at the University of Colorado was controlled from College Park via the Internet.

2.3 Experiment Design

Everything between the transmitting software and the receiving software—including the FRACSes, VSATs, and ACTS—was regarded as the "satellite

channel" in this work. This channel was used with a rate of 128 kb/s consistently through the experiment. This rate was chosen to avoid overloading the 14.4 kb/s modem connection used in the hybrid network ARQ tests, and to avoid overwhelming the experiment software. Note that the rate in the physical ACTS channel was greater than 128 kb/s due to overhead added by frame relay software, the FRACS and other sources (as will be discussed below).

2.3.1 Artificial Noise

It had been originally intended to conduct tests using a satellite link and so letting the transferred data suffer the noise phenomena of this link. However, it became clear in developing the experiment that the errors accrued by the traffic during satellite transport could not be directly observed. The reason for this is that every frame sent through the FRACS has a cyclic redundancy check (CRC) protecting it [5]. Thus, if a frame develops an error in satellite transport, the frame will fail the CRC check in the destination FRACS and so will be discarded.

Furthermore, it was learned from other ACTS experimenters that the ACTS satellite link exhibits very low BER when not compromised by severe weather. Experience with the T1 VSAT at UMCP corroborates this finding. Thus, not only does the FRACS prevent observing naturally-produced errors, such errors were infrequent.

Hence, in order to study error control schemes, it was necessary to inject artificially-produced noise, in the form of bit inversions, into received data.

Upon resolving to use such artificial noise effects, a model to characterize the noise effects was required. A reasonable choice was a more severe version of the noise experienced with the ACTS channel, when such noise appears. Now it had

been originally expected that the errors experienced with the ACTS satellite link would be primarily burst errors, particularly due to disturbances of the ACTS K_a-band channel by water in the atmosphere. The results of another ACTS experimenter [7] helped in this case, too: these results indicate that a binary symmetric channel (BSC), not a burst channel, best characterizes the development of errors in the ACTS channel. Such a model was used in the experiment software for corrupting received data.

These noise effects were generated with bit error rates of 10^{-5}, 3.16×10^{-5}, 10^{-4}, 3.16×10^{-4}, 10^{-3}, and 3.16×10^{-3}. A control case of zero artificial noise was included as well. To make meaningful statistical inferences from the experiment data, it was decided to send at least ten million information bits from one station to another when operating the noise effects at BER=10^{-5} (the minimum selected non-zero BER setting). For purposes of consistency and easy comparison, at least ten million information bits were sent in all scenarios of the experiment.

2.3.2 FEC Tests

The FEC testing conducted in this experiment is conceptually depicted in Figure 2.2.

For this testing, a short plain text message was encoded, for each of the FEC coding options described below. Several codewords were concatenated to form a block of encoded information (stored as a file) about 125 bytes long. This length was chosen because such a length is common for packets in ARQ communication via satellite, and was adopted for ARQ testing in this experiment. To allow straightforward comparisons, information was communicated in blocks about 125 bytes long throughout the experiment.

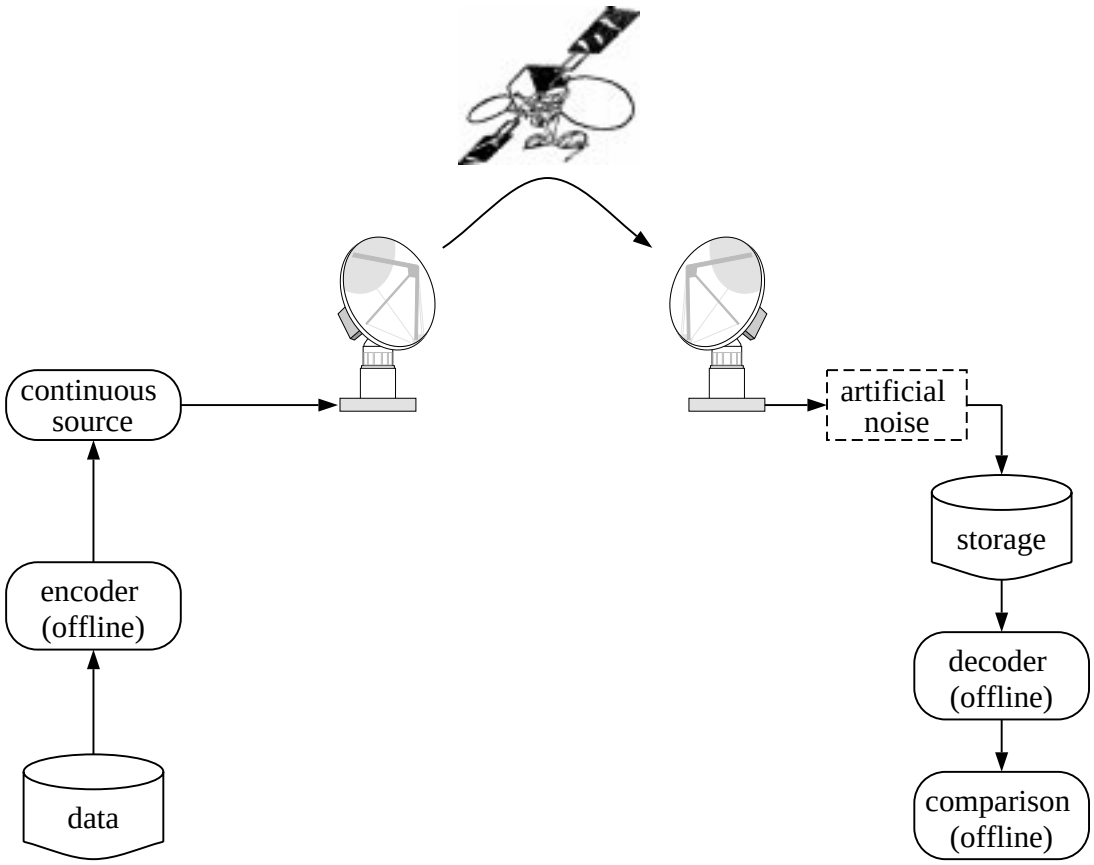

Figure 2.2: FEC testing concept.

This block of encoded information was then transmitted via satellite at 128 kb/s from `apogee` to `perigee2`. The block was sent repeatedly so that at least ten million information bits were transfered. By sending a brief message repeatedly instead of sending a single huge message, far less computer storage space was required at the transmitting station. Thus the actual information transfered comprised thousands of copies of the encoded short text message. The time to transfer the many blocks—the period starting with receipt by the receiver of the first block and ending with receipt of the last block—was recorded.

Upon arrival at the receiving station, received blocks were corrupted with

the artificial noise described earlier, and then stored. Later, the blocks were decoded, and the decoded information—multiple copies of the short message—was compared bit-by-bit to the original message. Erroneous bits in the received information were counted to obtain a residual BER.

It might appear that the FEC measurements obtained in this configuration would nothing more than simulated results of FEC schemes over a BSC, results which are well-known (see for example [8]). It might further appear the actual use of the satellite is superfluous. However, since an aim of the experiment was to compare error control techniques for satellite and hybrid communication, it was necessary to conduct the FEC tests in exactly the same fashion as the ARQ tests, which indeed require the use of a satellite and the hardware shown in Figure 2.1.

Codes

The codes used in the FEC portion of this experiment were linear block codes. While convolutional codes are often used in satellite communication [9], it was feared software implementations of such codes would be complex and so would run slowly. In fact, even for the selected block codes, both the software encoder and decoder were found to operate too slowly to support real-time operation coordinate with the desired ACTS channel rate of 128 kb/s. Accordingly, all encoding and decoding operations were conducted offline. Such offline operation was deemed acceptable because specialized hardware would ordinarily be used for conducting coding operations in real-time in an actual system.

Two of the codes selected were the BCH (15, 7) and Golay (23, 12) codes, since these codes were relatively simple to implement and have code rates of

about one-half. The code rate was significant because it had originally been hoped to compare the performance of these codes with that of the rate one-half code used by ACTS (a convolutional code with constraint length 5) to combat rain fading [2]. This goal was abandoned upon realizing the performance of ACTS's coding could not be measured since it was impossible to examine received data for errors before ACTS's coding corrected them.

A third code, the BCH (15, 11) code, was later added to provide a broader set of FEC choices. For comparison purposes, plain uncoded text was also selected as an FEC option. (The Reed-Solomon (127, 123) and (127, 121) codes constructed over GF(2^7) were originally included as FEC options since they protect against burst errors, but were abandoned when the actual non-bursty noise model was adopted as described earlier.) Thus the following four FEC options were considered in this experiment:

1. BCH (15, 7)
2. Golay (23, 12)
3. BCH (15, 11)
4. Plain text

Each of these options was tested with the seven settings for the noise effects BER mentioned above (in 2.3.1), yielding 28 FEC scenarios.

As mentioned earlier, each block was sent many times so that at least ten million information bits were transfered. The number of blocks sent for each code is shown in Table 2.1.

It might appear that the described FEC tests would represent nothing more than the simulated results of the FEC schemes over a BSC, which are well-

Code	Block size (bits)	Information bits per block	Blocks sent	Information bits sent
BCH (15, 7)	960	448	22,322	10,000,256
Golay (23, 12)	920	480	20,834	10,000,320
BCH (15, 11)	960	704	14,205	10,000,320
Plain text	1000	1000	10,000	10,000,000

Table 2.1: FEC transmission blocks.

known [8]. It might further appear that the actual use of the satellite is superfluous. However, the purpose of this work was examine and compare error control techniques for satellite and hybrid communication, and so it was necessary to conduct the FEC portion of the experiment in exactly the same fashion as the ARQ portions, which indeed required the use of the actual satellite and hybrid architecture of Figure 2.1.

2.3.3 Point-to Point ARQ Tests

The point-to-point (unicast) ARQ testing conducted in this experiment is conceptually depicted in Figure 2.3. Paralleling the FEC tests, in each ARQ test a set of data packets was sent over the satellite at 128 kb/s. Each packet comprised a 125-byte plain text message, a 2-byte ARQ sequence number and a 2-byte CRC (CRC-CCITT). This packet was sent ten thousand times so that ten million information bits were transfered.

As each packet was received, it was corrupted with artificial noise. The resulting packet was then checked for errors using the CRC. Valid packets were stored and invalid packets were discarded. The receiver then generated a corresponding acknowledgment message according to the ARQ protocol in use, and sent this reply to the transmitter. Data stored by the receiver was later compared bit-by-bit with the original message to obtain a residual BER.

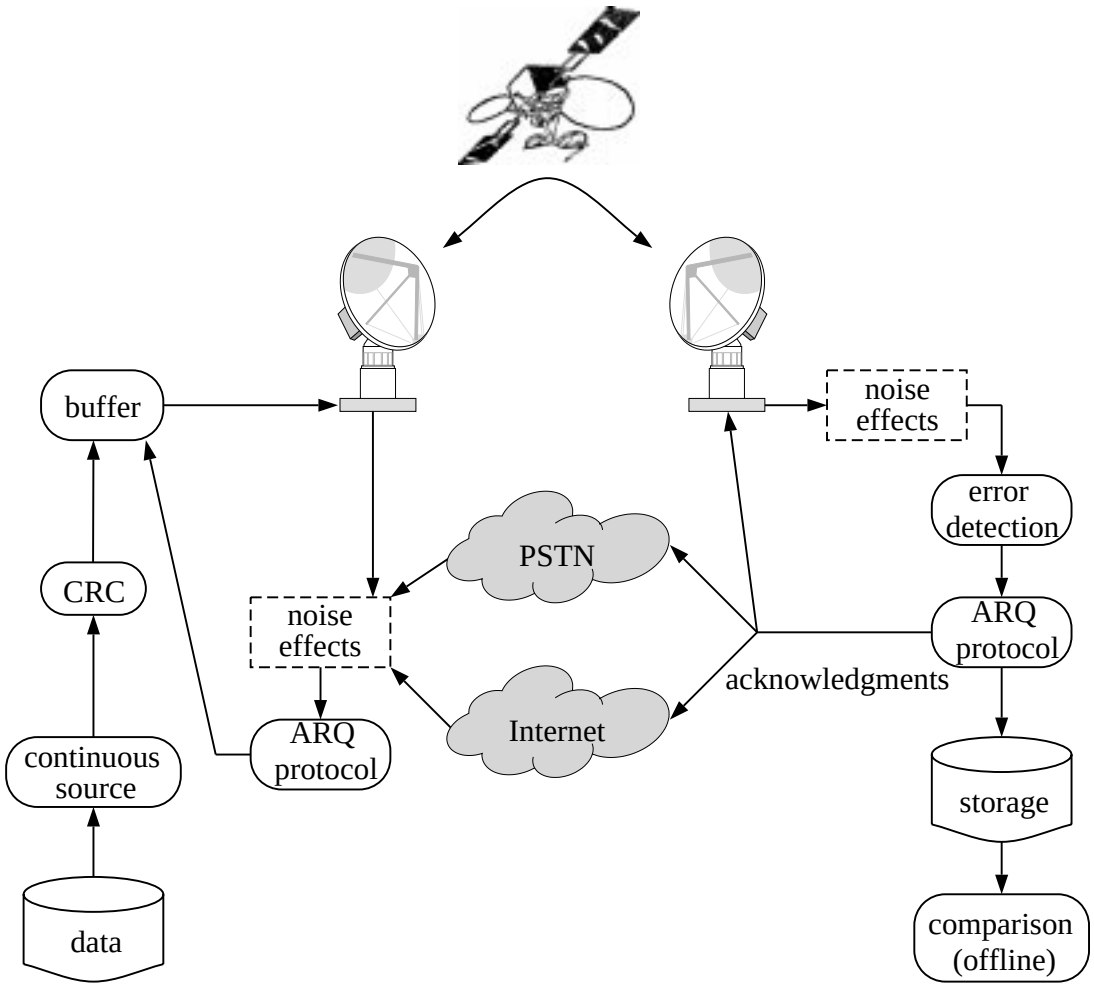

Figure 2.3: Point-to-Point ARQ testing concept.

Acknowledgments were carried over ACTS or terrestrially, via the Internet or via the public switched telephone network. For cases in which acknowledgments were carried terrestrially, retransmitted packets were sent either over ACTS or over the same terrestrial link (in the opposite direction). Retransmitted packets were always sent via satellite if the acknowledgments were so transported.

The time to send the ten thousand information packets was recorded in each test. This transfer time was defined as the period starting with receipt by the receiver of the first valid packet and ending with receipt of the last valid packet.

The message and acknowledgment packets sent, received and received in error on each link were counted and also recorded.

Both go-back-N (GBN) and selective-repeat (SR) ARQ protocols were tested in this experiment. The GBN and SR protocols tested followed the logic of the REJ protocol and SREJ protocol with multi-selective reject option, respectively, specified in [10, 11]. The parameters of the protocols, such as window size, were modified for satellite experimentation, and some parts not integral to error control, such as call setup and termination, were not included in the software implementation of the protocols since they were accomplished by other means.

The ARQ timer period was set to 0.788 s for satellite-network operation and 0.674 s for hybrid operation. Each of these times accounts for the time to transmit a packet on the satellite link, the propagation time through this link, the transmission time for an acknowledgment on the feedback link, the propagation time of the acknowledgment, and time for software processing on the ground. (For such purposes, the Internet was conservatively assumed to have the same bandwidth and propagation time as the 14.4 kb/s modem connection.) The period was less for hybrid operation than for satellite operation because of the lesser propagation delay through the terrestrial link than the satellite link. It might be expected the timer period for hybrid operation should be even less than just specified because the terrestrial propagation delay is substantially less than that of the satellite. However, the terrestrial link bandwidth was less than one-eighth that of the satellite link, which partly offset the effect of the former's smaller propagation delay in the calculation of the ARQ timer period.

The window sizes (N) corresponding to the aforementioned timer periods were 76 for operation in a satellite network, and 65 for the hybrid network.

Network type	Artificial noise BER						
	0	10^{-5}	3.16×10^{-5}	10^{-4}	3.16×10^{-4}	10^{-3}	3.16×10^{-3}
Satellite	√	√	√	√	√	√	√
Hybrid-1:							
Internet	√	√		√		√	
Modem	√		√		√		√
Hybrid-2:							
Internet	√	√		√		√	
Modem	√		√		√		√

Hybrid-1: Acknowledgments sent terrestrially.
Hybrid-2: Acknowledgments and retransmitted packets sent terrestrially.

Table 2.2: Combinations of networks and artificial noise bit error rates tested with point-to-point GBN and SR ARQ protocols.

It would have been possible to conduct the satellite network unicast ARQ tests using `apogee` and `perigee`. However, the terrestrial propagation delay would be unrealistically small if these two machines would have been used for the hybrid network tests. Hence `beep` and `apogee` were used not only for the hybrid network tests, but, for purposes of consistency and better comparison, for the satellite network tests as well. Because all received traffic data and measurements had to be stored and processed at UMCP, `beep` transmitted to `apogee` during all unicast ARQ tests.

Due to time constraints, it was not feasible to test every possible combination of network architecture, type of terrestrial link and artificial noise BER. Accordingly, only the combinations indicated in Table 2.2 were tested. All the indicated combinations were tested with both GBN and SR ARQ, yielding a total of 46 unicast ARQ scenarios.

2.3.4 Multicast ARQ Tests

A group of point-to-two point ARQ tests was included in this experiment as the start of an inquiry into point-to-multipoint ARQ schemes for satellite and hybrid networks. In a purely-satellite multicast ARQ system, each destination station must have a relatively expensive two-way (receive and transmit) terminal in order to receive information and send acknowledgments. Also, if a single receiving station requires a packet retransmission, then the transmitting station must send the packet over the satellite link. This retransmission interrupts the stream of new packets for all destination stations. Only one receiving station benefits from the retransmission; the others are forced to wait during this time. This delay might be circumvented, and the information delivery rate to each receiver possibly increased, by sending the ARQ acknowledgments and retransmitted packets terrestrially. Again, a receive-only satellite terminal suffices for the destination stations, and a significant cost savings may be achieved in addition to the aforementioned throughput improvement. This portion of the experiment investigated these concepts.

The multicast ARQ testing was similar to the unicast testing except two destinations were used (Figure 2.4). `Apogee` transmitted simultaneously to `perigee2` (a loopback transmission from the perspective of ACTS) and to `beep`. Two network configurations were employed: either all data packets and acknowledgments were sent over the satellite links and nothing was sent over terrestrial links, or data packets were originally sent via satellite while both acknowledgments and retransmitted packets were sent terrestrially.

Both the Internet and the PSTN were used for the terrestrial link between `apogee` and `beep`. Terrestrial communication between `apogee` and `perigee2`

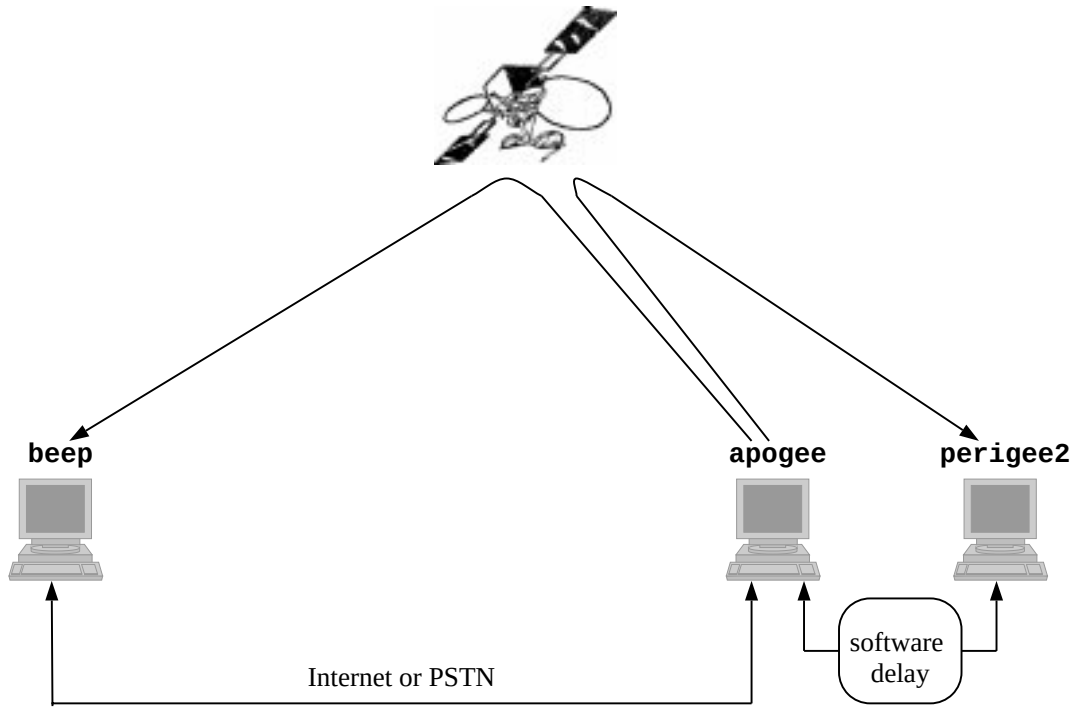

Figure 2.4: Hybrid network for point-to-two point ARQ tests.

was conducted with Ethernet. This Ethernet communication was delayed with software to simulate the propagation delay which would have been realistically experienced had these two stations not been located so near each other. For each test, this delay was matched to the bona fide terrestrial link used between `apogee` and `beep` so that terrestrial transmissions between `apogee` and `beep` and between `apogee` and `perigee2` experienced similar delays. The software delay was not fixed, but varied stochastically with each packet to reflect the non-constant propagation times experienced with Internet and with the telephone system. For simulating Internet, the delay value was uniformly distributed on [0.090, 0.150) seconds. (This interval was half-open and not closed only because the random number generator produced values in [0,1).) For simulating the modem connection, the delay was uniformly distributed on {0.160, 0.170, 0.180}

seconds. Both of these models were obtained experimentally, using the UNIX `ping` command between `apogee` and `beep` through Internet and through a modem connection. While ordinarily it may be possible for packets to arrive out of sequence, the delay software preserved the correct order.

For the ARQ multicasting tests, the GBN and SR ARQ protocols used in the unicast testing were modified slightly to accommodate two receiving stations. While the ACTS system supports multipoint transmission, this feature was not used because it was not supported by the FRACS. Instead, each packet to be sent via ACTS to the two destinations had to be sent twice, once on the `apogee-perigee2` satellite connection and once on the `apogee-beep` satellite connection. However, a single transmitting process was used to send packets over the satellite, process acknowledgments and conduct retransmissions as necessary. Thus the system implemented was a point-to-two point ARQ system, not two simultaneously-operating point-to-point systems. As with the unicast ARQ tests, the transfer times to each receiver and the number of packets and acknowledgments sent, received and received in error on each link were recorded.

Multicast ARQ Protocols for Hybrid Networks

The multicast ARQ protocols tested in this experiment were developed from their point-to-point counterparts, and merit some discussion. One important concept is the ARQ transmitter's "window." As for the point-to-point case, the window was defined as the range of sequence numbers of packets which may be sent but remain unacknowledged until expiration of the ARQ timer. Hence, if the minimum of the sequence numbers of the packets awaited by receiver A and those of the packets awaited by B was x, and the window size was N, packet

number $x + N - 1$ could be transmitted, but $x + N$ could not be sent until x was positively acknowledged by both stations. For SR operation, a receiver would discard a valid packet if the packet had already been received or if the packet's sequence number less N exceeded the minimum of the sequence numbers of packets not yet delivered to the receiver. (Such operation is a direct extension of the point-to-point operation specified in [11].)

The problem in a satellite multicast ARQ system of a retransmission for a single receiving station forcing all stations to suffer a throughput reduction was described earlier and should be examined more closely. Consider, then, a hybrid network employing multicast SR ARQ. Assume that each destination can receive simultaneously on both its satellite and terrestrial links. If one station requires a retransmission, the retransmitted packet can be sent terrestrially and the flow of new packets need not be interrupted for the station which does not require the retransmission. In fact, the receivers can, to a limited degree, accept packets out of order because SR ARQ is used. Hence the flow of new packets to the station requiring the retransmission also need not be interrupted. Thus, if channel conditions permit, the hybrid network allows the transmitter to send fresh packets continuously to all receivers even while one may require a retransmission, and the throughput need not suffer significantly. This approach was taken in implementing the multicast SR ARQ protocol for this experiment.

Now consider the situation if GBN ARQ is used. If receiver A requires a retransmission while receiver B does not, the latter can continue to receive new packets over the satellite link. Suppose k new packets are successfully delivered via satellite to B during the time between A sends its retransmission request and A sends the positive acknowledgment for the retransmission. During

this time, receiver A will reject all packets other than the one it requires—the one requiring retransmission—so A will discard these k packets. Thus, after sending the positive acknowledgment for the retransmission, A will lag B by k packets. Next the transmitter will have to send via satellite the k packets for A, an action which does not benefit B, since B already has these k packets. Thus $2k$ packets will be sent via satellite to deliver k packets to each of the two stations. This is no more efficient than operating two point-to-point ARQ systems simultaneously, and does not take advantage of the satellite's multicast capabilities. The situation in practice likely will be worse than described since both receivers may require retransmissions of different packets.

A more efficient approach, taken in this experiment, is to suspend sending fresh packets on the satellite link until the retransmission event for receiver A is concluded. In this case, no new packets need be transmitted twice, although receiver B is still forced to wait. Hence it appears GBN ARQ is not an efficient protocol for ARQ multicasting in satellite nor hybrid networks. This is not highly surprising, since GBN ARQ is less efficient than SR for point-to-point communication [12].

If all the receivers of a large network require retransmission of the same packet, it is more efficient to retransmit the packet via the high-bandwidth multicast satellite channel than via the separate low-bandwidth terrestrial channels. Such operation was adopted for the experiment's configuration of two receiving stations. That is, a packet was retransmitted via satellite if both receivers requested the packet be sent again.

The results from the unicast ARQ tests could be expected to render unnecessary performing multicast ARQ tests with some combinations of network

Network type	Artificial noise BER (Receiver 1, Receiver 2)		
	$(10^{-5}, 10^{-5})$	$(0, 10^{-5})$	$(10^{-3}, 10^{-5})$
Satellite	√	√	√
Hybrid-1: Internet Modem	(No tests with hybrid-1 network)		
Hybrid-2: Internet	√	√	√
Modem	√	√	√

Hybrid-1: Acknowledgments sent terrestrially.
Hybrid-2: Acknowledgments and retransmitted packets sent terrestrially.

Table 2.3: Combinations of networks and artificial noise bit error rates tested with point-to-multipoint GBN and SR ARQ protocols.

architectures, type of terrestrial link and artificial noise BER. In particular, it was expected the point-to-point tests would show sending both acknowledgments and retransmissions terrestrially yields superior throughput performance than using the terrestrial link for acknowledgments alone. Accordingly, the combinations of networks and artificial noise bit error rates indicated in Table 2.3 were selected for testing with GBN and SR multicast ARQ protocols, a total of 18 multicast ARQ scenarios. As shown, the noise bit error rates at the receivers were set independently. This allowed simulating cases of reception conditions at one receiver being much poorer than at the other (as when one satellite station suffers from deep rain fading while the other enjoys a clear sky). In all cases, all acknowledgments were corrupted by artificial noise effects (with BER set to 10^{-5}) upon receipt by the transmitter, and ten thousand packets (ten million information bits) were transmitted.

2.4 Software Overview and Implementation Details

General descriptions of the software programs used in the experiment, as well as some implementation details, are provided in this section. More details regarding the software may be found in Appendix A.

The software for this experiment was written in the C language and utilized User Datagram Protocol/Internet Protocol (UDP/IP). UDP was chosen over Transmission Control Protocol (TCP) because the latter has an ARQ scheme of its own, which would interfere with testing the error control protocols of interest in the experiment.

2.4.1 FEC Software

Nine programs were used in FEC experimentation. Six of these were encoders and decoders, an encoder-decoder pair being required for each of the three FEC codes tested. The other three programs were the FEC transmitter, the FEC receiver and the program for comparing bit-by-bit the received information with the original information.

To prepare for an FEC test, an encoder was used to process a portion of a 125-byte plain text file, yielding an encoded block with the properties given in Table 2.1. The resulting block was stored as a file.

The task of the FEC transmitter software was to read a file specified by the user and sent it multiple times via satellite to the receiver. This file constituted the packet which was sent repeatedly by the transmitter program. With this arrangement, selecting the FEC code to test corresponded to selecting the

appropriate source file for the transmitter.

The transmitter used UDP/IP to send packets to a particular [logical] port of the receiver. The receiver listened for packets on this port and recorded them in a file as they arrived. Later, the received packets were processed with an appropriate decoder and the decoded information was compared bit-by-bit with the original text.

For testing with un-encoded text, no encoder nor decoder was required, and the 125-byte message file was used directly as the packet sent via satellite.

2.4.2 ARQ Software

Five programs were used in ARQ experimentation: a point-to-point transmitter, a multicast transmitter, two receivers, and a delayer. Originally, the point-to-point and multicast ARQ transmitter programs were written separately; the two were later combined into a single body of C code. The combination facilitated debugging and adding new features, for oftentimes during program development a change was deemed necessary for both the point-to-point and the multicast transmitters. The production of two programs from a single body of code was made possible by using conditional compilation directives.

A single body of program code was used for the receivers as well. The receiver programs used for multicasting differed only in port numbers, which were also selected with conditional compilation directives. One of the two receivers was used for the point-to-point tests while both were used for multicast tests.

Three logical links, to which corresponded three port numbers, were defined between the transmitter and each receiver:

1. A link for the transmitter to send new packets to the receiver, via satellite.
2. A link for the transmitter to retransmit packets, via terrestrial link.
3. A link for the receiver to send acknowledgments to the transmitter, via satellite or via terrestrial link.

In tests which did not use a terrestrial link for retransmissions, the transmitter sent retransmitted packets over the satellite link.

In a fashion similar to the FEC transmitter, the ARQ transmitter used a specified file as the information portion of the packet which was sent repeatedly. A sequence number and CRC were added to each packet before transmission, as described earlier.

The heart of the transmitter program is a loop comprising four parts:

1. Send a packet requiring retransmission (if any).
2. Send a new packet (if any).
3. Process acknowledgments (if any).
4. Check for expiration of ARQ timers.

Each step was conducted for all destinations before proceeding to the next step, instead of repeating the four steps for each destination. This was necessary so that the transmitter would treat the receivers as a group and not individually.

For example, if a packet is successfully sent to all destinations, positive acknowledgments from the receivers will arrive at the transmitter nearly simultaneously if propagation delays are the same between the transmitter and all destinations. The transmitter should regard all the acknowledgments as having arrived together. If all earlier packets have already been acknowledged, the transmitter should discard the newly-acknowledged packet and slide forward one position the ARQ window. The transmitter should *not* possibly regard the acknowledgments as having arrived at different times because of delay in sequentially processing an acknowledgment from each destination. Similarly, time-stamping of events such as sending a packet was carefully conducted to avoid incorrect operation due to such sequential processing delays.

The order of steps in the transmitter loop was determined by design. Packets requiring retransmission were sent before new packets because the need for a retransmission impedes the progress of the system. That is, until all receiving stations acknowledge a packet, that packet cannot be discarded by the transmitter, and the lower-edge of the ARQ window cannot be advanced beyond that packet. Also, if GBN ARQ is used, the receiver will refuse any new packet while a retransmitted packet is awaited. Hence, it is desirable to attend to retransmissions as soon as possible. Similarly, the receiver checked for packet arrivals on its terrestrial link before checking its satellite link since all packets arriving on the terrestrial link are retransmissions.

Also, acknowledgments were processed before checking for timer expirations because it is wasteful to declare an expiration of the timer for packet number z, and so order an retransmission, while an acknowledgment for z may have arrived at the transmitter but not yet have been processed.

Separate ARQ timers were required for each destination since the receivers operated independently. For example, one receiver may positively acknowledge a packet while another requests retransmission. The transmitter must then send the requested packet again to the second station and somehow regard the fact of a packet having been sent more recently to the second destination than to the first. The solution to this problem is to use separate timers for each destination.

For multicasting, it was necessary to delay terrestrial communication between `apogee` and `perigee2`. This delay was accomplished by a separate program, which queued each packet and released it after the packet had been queued for a specified time. The amount of delay varied stochastically, as described in 2.3.4.

2.4.3 Additional Details

For each data packet sent by experiment software via the satellite, UDP/IP added 28 bytes of overhead [13]. SunLink Frame Relay software was used to transport the UDP/IP datagrams between FRACSes and between each FRACS and workstations connected to same; this software added two bytes of overhead for addressing purposes [14]. The High-level Data Link Control (HDLC) protocol in the Sun HSI added to each frame a one-byte opening flag, a two-byte CRC and a one-byte closing flag, constituting an additional four bytes of overhead (ignoring bit stuffing) [6]. For frames sent via satellite, the FRACS added five bytes of overhead for its own purposes (such as frame sequence restoration) [5]. Thus, for every experiment packet sent over the satellite, 28+2+4+5=39 bytes of overhead were sent through ACTS. For an experiment packet 129 bytes long (as in the ARQ tests) sent at 128 kb/s, this corresponds to a rate of 169.3 kb/s supported by ACTS. Accordingly, at least three 64 kb/s ACTS channels (a

total bandwidth of 192 kb/s) were required to conduct this experiment. Due to particulars of FRACS operation, a margin of additional bandwidth was deemed necessary and so the entire experiment was conducted using four ACTS channels (a total bandwidth of 256 kb/s) for each satellite link required.

The Point-to-Point Protocol (PPP) was used for sending packets over the modem link. PPP added an overhead of eight bytes to every packet produced by experiment software [13, 15]. The modems were set to operate with V.42bis and MNP5 compression algorithms enabled.

Chapter 3

Experiment Results and Discussion

3.1 Difficulties Experienced

The previous chapter presented an experiment developed to examine a hybrid network's implications with respect to error control. Unfortunately, a number of difficulties hampered conducting the experiment. These difficulties included:

1. The software for ARQ experimentation had been originally written improperly (by another party) and so had to be nearly completely rewritten.

2. The optical link in Colorado was often disrupted by precipitation, and suffered several failures.

3. The T1 VSAT in Maryland suffered several problems, including:

    - Incorrect configuration information (resulting in high-BER operation, or no operation at all).
    - Two Intermediate Power Amplifier failures.
    - Two High-Power Frequency Doubler failures.
    - Partial failure of VSAT control computer.

Because of such difficulties, the results from only the FEC tests are available for inclusion in this thesis.

3.2 FEC Results

The number of blocks sent for each code tested was specified in Table 2.1. The times required to transfer these blocks in the FEC tests are given in Table 3.1. These results show the time to transfer the ten million information bits depends in each test depends on the code rate and is independent of the degree of channel noise, as would be expected.

In each test, the received data was decoded and compared to the original message and the received information bits in error were counted to yield a residual BER. The residual BERs so obtained are given in Table 3.2. The nonzero residual BER from the test of the BCH (15, 11) code with noise BER of 3.16×10^{-5} may appear anomalous, but may be attributed to the statistical nature of the results.

3.3 ARQ

The point-to-point and point-to-multipoint tests were not completed in time for their results to be included in this thesis. Some comments about the results expected, and the comparisons which would have been made, can nonetheless be advanced here.

Regardless of the noise BER, the residual BER would be expected to be zero for all ARQ tests, with possibly some rare exceptions. This can be claimed because the CRC is an excellent error-detecting code. In fact, the CRC is more

Noise Effects Bit Error Rate	Code			
	Plain Text	BCH (15, 11)	Golay (23, 12)	BCH (15, 7)
3.16×10^{-3}	78.539	107.263	150.659	168.476
10^{-3}	78.596	107.257	150.658	168.540
3.16×10^{-4}	78.607	107.195	150.668	168.460
10^{-4}	78.615	107.216	150.634	167.818
3.16×10^{-5}	78.621	107.219	150.609	168.534
10^{-5}	78.613	107.170	150.651	167.802
0	78.515	107.194	150.644	168.573

Table 3.1: Transfer times (in seconds) for FEC tests.

Noise Effects Bit Error Rate	Code			
	Plain Text	BCH (15, 11)	Golay (23, 12)	BCH (15, 7)
3.16×10^{-3}	2.042×10^{-3}	9.990×10^{-5}	1.180×10^{-5}	1.300×10^{-6}
10^{-3}	8.163×10^{-4}	1.160×10^{-5}	1.100×10^{-6}	0
3.16×10^{-4}	2.877×10^{-4}	1.100×10^{-6}	0	0
10^{-4}	9.970×10^{-5}	0	0	0
3.16×10^{-5}	3.210×10^{-5}	2.000×10^{-7}	0	0
10^{-5}	1.010×10^{-5}	0	0	0
0	0	0	0	0

Table 3.2: Residual bit error rates for FEC tests.

powerful at detecting errors, and so prompting a retransmission, than are any of the tested FEC codes at correcting errors. Accordingly, the residual error rates for point-to-point ARQ tests with a particular noise effects BER would, in general, be expected to be less than or equal to the residual BERs in FEC tests at the same noise level.

The transfer times in ARQ tests would be expected to increase with the noise effects BER, since ARQ operation with poorer channel conditions results in more retransmissions. Accordingly, to achieve similar residual BERs, it would be expected that, if the amount of noise is below a certain level, an ARQ test would have smaller transfer time than obtained with a particular FEC code. Above that noise level, the opposite would be expected to be true. The extent to which such comparisons can be made is limited by the CRC being able to detect a larger fraction of error patterns (and signal the need for a retransmission) than the error-correcting codes can correct. That is, as stated in the previous paragraph, the residual BERs would be expected to be less for ARQ tests than for the FEC tests.

The GBN ARQ protocol does not permit the receiver to accept packets out of order. Accordingly, if the receiver is waiting for a retransmission, new packets are rejected until retransmitted packets arrive. These packets may be accepted if SR ARQ is employed. Further, far more packets need be sent again with GBN than SR if a retransmission is requested. Accordingly, transfer times in GBN tests would be expected to be greater than for SR tests.

It is possible retransmitted packets can be delivered more quickly to the receiver via a low-delay, yet low-bandwidth, terrestrial link than via a high-bandwidth, yet high-delay, satellite link. However, the exact parameters of the

situation (the packet size, the window size, the rates and propagation delays of each link) may be such that the retransmitted packets can be delivered more quickly via satellite than terrestrially. Hence it is not immediately clear if tests with the hybrid network and GBN ARQ would be expected to yield lesser transfer times than would corresponding tests with the satellite-only network. It may well be that sending both acknowledgments and retransmitted packets terrestrially instead of by satellite *reduces* the throughput because each retransmission requires sending N packets through the low-rate terrestrial link. Of course, determining if—and to what extent—the throughput can be improved is one of the purposes of the experiment.

In SR ARQ tests, however, the availability of a separate path for retransmitted packets, combined with the protocol's [limited] capability to accept packets out of order, would suggest smaller transfer times for hybrid network operation than for satellite-only operation. If the transmitter is able to retransmit a packet terrestrially while continuing without interruption to send new packets on the satellite link, then, if retransmissions are infrequent, the throughput can be nearly as great as in a zero-noise case. Even if the channel noise is severe enough to cause more frequent retransmissions, or if the transmitter must briefly interrupt sending new packets in order to retransmit, some throughput improvement may be expected. Again, this depends greatly on the operating parameters.

The transfer times in the multicast ARQ hybrid network tests would be expected to be less than those in the corresponding satellite network tests. Multicast ARQ tests would be expected to have longer transfer times than would point-to-point ARQ tests, since in the former multiple receivers may require retransmissions and the information rate to all receivers might suffer, even in a

hybrid network.

A surprising finding gleaned in developing and testing the experiment is a great delay when using the modem link. In particular, the round-trip time for a UNIX `ping` message between `apogee` and `beep` was found to average about 350 ms. To eliminate most of the propagation delay between College Park and Boulder, an identical test was performed with `apogee` and `perigee2`, using two phone lines in the Systems Integration Laboratory. This second test yielded a round-trip time of 285 ms. Subtracting the time to transmit the `ping` message leaves 192 ms (96 ms in each direction) of delay comprising propagation time through the local phone system and the modems. The modems' share of the delay is speculated to be due to compression/decompression of the data carried and the modems' trellis-coded modulation scheme (particularly the Viterbi decoding). As the delay experienced with modems is a significant fraction of the single-hop propagation delay through a geostationary satellite, it is possible transfer times obtained in a pure-satellite architecture might not be significantly reduced by adopting a hybrid network with a modem-based terrestrial connection. (This finding does not, however, diminish the significant cost savings, mentioned in Chapter 1, which may be achievable by adopting a hybrid network instead of a pure-satellite network.)

Chapter 4

Conclusion

Error control is required for assuring the accuracy of information transfered through an imperfect channel. FEC and ARQ are the two broad categories of error control schemes. The great propagation delay of a satellite channel presents challenges for ARQ schemes provide good throughput to a destination station. The challenges are compounded for multicasting, where the throughput to all stations may suffer if a retransmission is required, even if only for one station.

A hybrid network has been suggested for mitigating such problems. ARQ acknowledgments, and perhaps retransmissions as well, can be sent terrestrially, and greater throughput possibly achieved. A hybrid network may help tremendously for ARQ multicasting by allowing retransmissions to be conducted without drastically reducing throughput.

The precise effects a hybrid network may have on throughput and fidelity are not clear. An experiment to investigate such error control effects was presented. This experiment examined FEC in a point-to-point satellite network, ARQ in point-to-point satellite and hybrid networks, and ARQ multicasting in satellite and hybrid networks. It was seen that an ARQ protocol and its parameters must

be carefully tailored to suit point-to-point satellite communication. Similarly, a satellite multicast ARQ protocol must incorporate features of unicast protocols as well as other measures in order to operate well. While all results were not available for inclusion in this thesis, it is strongly believed that the experiment, when completed, will show a throughput advantage of a hybrid network, especially for ARQ multicasting.

A problem suffered throughout the experiment was the need to send overhead from UDP/IP, frame relay, and the FRACS. Such overhead, as calculated in 2.4, amounts to more than 20% of the bits transmitted via the satellite, and did not improve error control. Such overhead was not regarded in calculating residual BERs and transfer times, yet a penalty of additional satellite bandwidth was suffered for carrying it. It is likely smaller transfer times could have been achieved if there had been less overhead.

The great delays experienced with modem links were unexpected, and suggest other terrestrial links may be better for hybrid networking.

Throughout this work, the satellite was assumed to be in a geostationary orbit. Several systems employing low- and medium-earth orbit (LEO and MEO) satellites, which provide less propagation delay than geostationary satellites, have been proposed in recent years. It may be speculated that, because propagation delays are smaller, a hybrid architecture with these newer satellites will not provide benefits as great as for higher-altitude satellites. However, considering the features and applications foreseen for LEO and MEO satellite systems may suggest other advantages. Hybrid networking with LEO and MEO satellites is therefore a worthy topic for future examination.

Pure FEC and pure ARQ were the error control schemes considered in this

work. Hybrid ARQ uses FEC in conjunction with ARQ to provide high-fidelity, high-throughput communication with fewer retransmissions than with pure ARQ and with less overhead than required with pure FEC. This form of error control was not examined in this work. Combining hybrid ARQ schemes and hybrid networks offers additional possibilities for error control, and remains a topic for inquiry.

Two standard point-to-point ARQ protocols were modified to suit multicasting in this experiment. While these multicast ARQ protocols were perhaps not optimal for communication over a pure satellite network, they offered a standard for gauging the benefits offered by operation in a hybrid architecture. Accordingly, this thesis motivates future inquiry and development in the field of multicast ARQ protocols for satellite and hybrid networks. The author hopes to develop these techniques and others for exploiting the hybrid architecture.

Appendix A

Software Listings

A.1 Introduction

An overview of the experiment software was presented in Section 2.4. This appendix presents the software in a mix of actual C code and C-like pseudo-code. Concentration is given to elements unique to the experiment; details such as declaration of variables, file operations and networking with UNIX sockets are largely omitted.

A.2 Software for FEC Tests

A.2.1 Encoders and Decoders

For each of the three codes tested, an encoder and a decoder were required. Most of the encoders and decoders were software implmentations of circuits described in [9] (to which the page numbers below refer).

All codes were encoded using a software implementation of the cyclic shift circuit described on pages 95-96. The decoder for the BCH (15, 7) code was

a type-II one-step majority logic decoder descrived on pages 188-190. For the Golay (23, 12) code, Kasami's error-trapping technique was used for decoding (pages 135-138; also [16]). Finally, table-lookup was used for decoding the BCH (15, 11) code.

A.2.2 FEC Transmitter

The operation of the FEC transmitter is determined by the combination of a command-line argument and information in a configuration file. The command-line argument is an Internet host name/address which is the destination to which packets should be sent. The name/address used in the experiment corresponded to the local interface to the satellite link to the destination.

```
/*
 FEC Transmitter
 */

main(argc, argv)
    int             argc;
    char           *argv[];
{
    seed_random_number_generator();

    /* read from configuration file:
        output rate (bit/s),
                        (variable: Output_rate)
        Number of packets to send
                        (variable: Source_size)
        Name of file to send repeatedly
    */
    get_parameters();

    /* Read the data to repeatedly send */
    read_source_file();
    N = length_of_source_file();

    /* here we calculate the bit/sec rate */
    /* The time between sending successive packets is
        given by:
    (bits/packet) / (bits/second in the channel) */
    txinterval = (N * 8 / Output_rate);

    /* Open a "connection-oriented" UDP connection to
        the destination; returns the socket descriptor the
        program will use for sending data */
    txsock = init_connection(argv[1]);

    run_tx();
}

run_tx()
{
    struct timeval lastsend, timenow;
    double timediff;
```

```
    /* must put some initial time into lastsend */
    gettimeofday(&lastsend, NULL);

    pktssent = 0;
    do {
        /* send a packet if it's time to */
        gettimeofday(&timenow, NULL);
        if ((it is time to send a packet) {
            if (Tx(databuf, N) >= 0) {
                pktssent++;
                lastsend = timenow;
            }
        }
    } while (pktssent < Source_size);
}
```

A.2.3 FEC Receiver

A configuration file, but no command-line argument, is used to control operation of the FEC receiver.

```
/*
    FEC Receiver
*/

main()
{

   seed_random_number_generator();

    /*
    Read from configuration file:
        Number of packets to expect to receive
                        (variable: Source_size)
        Enable/disable noise (variable: NOISE)
    */
    get_parameters();

    run_rx();
```

```
}


struct timeb    first_arrival, latest_arrival;
run_rx()
{

    int length, i, n, NumFrames;
    unsigned char buf[FRAMESZ], noisebuf[FRAMESZ],
        filenamedata[12], filenameinfo[12];
    unsigned char codeword[BCHOUT],
        decode[PKTZ_FRAME];
    struct timeval  time0;
    int             udp_cleanup();

    /* Open a "connection-oriented" UDP connection from
        the transmitter; returns the socket descriptor
        the program will use for reading data */
    rxsock = init_connection();

    counter = 0;

    /* the number 2500 a few lines below is an
    arbitrarily-chosen limit for the size of a packet
    */
    do {
        bytes_received = recv(rxsock, buf, 2500, 0)  ;
        if (bytes_received>0) {
            ftime(&latest_arrival);
            if (!counter) {
                /* First packet has just arrived */
                first_arrival = latest_arrival;
            }
            counter++;
            if (NOISE==ENABLED) {
                /* corrupt the received data (buf) with
                i.i.d. noise, and store result in bufC;
                (BER is set in a header file) */
                corrupt(buf, bufC);
                record_to_disk(bufC);
            }
            else
```

```
                record_to_disk(buf);
            }
        } while (bytes_received != 0 && counter < Source_size);
        /* Record the elapsed time between receipt of first and
           last packets; number of packets received */
        record_measurements();
}
```

A.3 Software for ARQ Tests

A.3.1 ARQ Transmitter

As described in Section 2.4, a single body of code was used to generate the ARQ point-to-point and multicast transmitters. Conditional compilation was used to select which type of transmitter was produced at compile-time.

As wih the FEC transmitter, the operation of the ARQ transmitter was controlled command-line arguments and information in a configuration file. In the point-to-point transmitter, either one or two command line arguments were required. The first argument was an Internet name/address for the local interface to the satellite link to the destination. The second argument, if any, was the name/address to which retransmissions should be sent. When Internet was used as the terrestrial link, the second name was simply the name of the destination computer (`apogee`, for example). If the program had been compiled to support two destinations, two addresses were required for each destination: one address for new packets, and one address for retransmitted packets.

The key data structure in the ARQ transmitter is a special array, used as queue, from which packets are sent at regular intervals. The array is accessed in a circular fashion using the C % (modulo) operator. It was found that % operates

fairly slowly so auxilliary variables were introduced to allow the modulo operator to be used sparingly.

For both the transmitter and the receiver, the CRC was encoded and decoded with a combined cyclic shift/table-lookup algorithm.

```
/*
    ARQ Unicast/Multicast Transmitter
*/

    /*
    NUM_DEST is the number of destinations; the
    program supports values of 1 or 2.  The value for
    NUM_DEST is set by a -DNUM_DEST=1 or -DNUM_DEST=2
    option for the "cc" compiler
    */
#if (!defined (NUM_DEST) || (NUM_DEST>2))
    /* doesn't yet support more than two destinations;
    alert user with a compiler error */
    undeclaredvar = ;
#endif

typedef short           SEQNUMTYPE;
/* On Sun machines, sizeof(short) is 4 bytes */
#define SEQNUMSZ        4

/* there should NOT be one of these TERR_RETRAN's
for each destination, but one for the entire program */
short           TERR_RETRAN;
    /* FALSE means retransmissions are sent via
    satellite */

SEQNUMTYPE      SOURCE_SIZE;

/* these flags can all be shorts */
short           NOISE, SELECTIVE;
    /* SELECTIVE==FALSE means use GBN */
double          TIMEOUT;
SEQNUMTYPE      WIN_SIZE;
double          RATE;
double          TxOUTRATE_1, TxOUTRATE_2;
```

```
/* rate (bit/s) on satellite link(s),
    terrestrial link(s) */
/* and here (next declaration) are the corresponding
    intervals (seconds) between sending packets
double          txinterval1, txinterval2 ;
char            SOURCE_FILENAME[50];
char            init_guf[32], init_guf1[32], init_guf2[
32], init_guf3[32];

unsigned char         DATABUF[DATASZ];

/* Data Socket descriptors: satellite link(s) */
int             sfwd1[NUM_DEST];
/* Data Socket descriptors: links for terrestrial
    retransmissions */
int             sretran[NUM_DEST];

/* ack. Socket descriptor */
int             sack[NUM_DEST], acksock[NUM_DEST];
/* port numbers */
int             txport[NUM_DEST],
                retxport[NUM_DEST],
                ackport[NUM_DEST];
int             AckFlag[NUM_DEST];

/* indicates retran. in progress */
int             retranflag[NUM_DEST];

SEQNUMTYPE      startofretran[NUM_DEST];
SEQNUMTYPE      endofretran[NUM_DEST];

int             N;
int             rframelen;

/* next two variables used to copy the command
line arguments */
char            freshpktaddr[NUM_DEST][51];
char            stalepktaddr[NUM_DEST][51];
struct hostent *host_entry;
struct in_addr *ptr, inetaddr;

/* ack. packets which fail CRC check */
```

```
int             corruptedpkts[NUM_DEST];

/*
The key data structure in the program is a
buffer/queue, which keeps track of which packets
have been sent to the receiver, which have been
acknowledged, for which ones a retransmission
has been requested, and other such things.

The queue is used circularly by accessing its elements
with expressions of form
    head % queue_size
and the like.   This way, we can use (except when
accessing the queue) the array indices "head" and
"tail" as if the queue were infinitely large.

The tail is the highest sequence number of all packets
transmitted.  This corresponds to the send state
variable V(S) in ISO/IEC 7776. The head is the sequence
number X such that all packets 0 through X-1 have been
acknowledged.  Hence the transmitter can discard the
first "head" packets.  Thus head is the minimum
sequence number of all packets not yet acknowledged.
At all times, we require head-tail <= window size.

Instead of making confusing additional variables, we
will call head and tail SNmin and SNmax respectively,
for indeed head and tail, as defined above, represent
these values.
*/

#define TRUE    1
#define FALSE   0
typedef short BOOLEAN;
SEQNUMTYPE      TXARRSZ;

struct TXARRAY {
    SEQNUMTYPE      seqno;  /* sequence number */
    unsigned char   packet[FRAME_SZ];
        /* packet = sequence number, data, CRC */
    BOOLEAN         valid;
    BOOLEAN         doretran[NUM_DEST];
```

```
        /* TRUE if receiver asked for a
        retransmission, returns to FALSE after
        retransmitting */
    BOOLEAN         acked[NUM_DEST];
    BOOLEAN         sent[NUM_DEST];
    struct timeval  timesent[NUM_DEST];
        /* this is used for checking for timer
        expiration*/
} ;
struct TXARRAY *txarr;      /* [TXARRSZ]; */

/* for ALL destinations */
SEQNUMTYPE      SNmax=-1 , SNmin=0 ;
/* need these for multicasting: */
SEQNUMTYPE      SNmineach[NUM_DEST];


void main(argc, argv)
int             argc;
char           *argv[];
{
    seed_random_number_generator();

    /* open connections */
    txport[0] = CLI_PORT;
    retxport[0] = CLI_PORT_RETRAN;
    ackport[0] = ACK_PORT;
#if (NUM_DEST==2)
    txport[1] = CLI_PORT2;
    retxport[1] = CLI_PORT_RETRAN2;
    ackport[1] = ACK_PORT2;
#endif

    for (m = 0; m<2*NUM_DEST; m+=2) {
        init_connections(m/2, argv[m+1], argv[m+2],
            txport[m/2], retxport[m/2], ackport[m/2]);
    }

    /* Read from configuration file (names of
        associated variables given in parantheses) */:
        Output rate for satellite link;
        Output rate for terrestrial link;
```

```
        Number of packets to send (SOURCE_SIZE);
        Size of data structure (TXARRSZ);
        Window size (WIN_SIZE);
        Noise type: i.i.d or none (NOISE);
        Noise BER (BER);
        ARQ protocol (SELECTIVE);
        ARQ timer period (TIMEOUT);
        Name of file to send repeatedly     */
    get_parameters();

    /* make the array by dynamically allocating memory */
    make_txarray(TXARRSZ);

    /* Read the data to repeatedly send */
    read_source_file();
    N = length_of_source_file();


    /* initialization for all the destinations */
    for (m=0; m<NUM_DEST; m++) {
        SNmineach[m] = 0;
        searchbot[m] = SNmin;
        retranflag[m] = 0;
        corruptedpkts[m] = 0;
    }
    run_arq();

    /* Measurements to record: number of packets sent,
    received, and received in error on each link;
    elapsed time between arrival of first and last
    valid packets */
    record_measurements();
}

/* Once arriving in this function, the program remains
here (or in functions called by it) until the end
of the program */
run_arq()
{
    struct timeval timenow,
        /* time of last transmission on primary link
            for ALL destinations */
```

```
    lastsend1,
    /* time of last transmission on secondary link
        for EACH destination */
    lastsend2[NUM_DEST];

/* Here we calculate the bit/sec rates. The time
between sending successive packets is given by:
    (bits/frame) / (bits/second sent in channel)
Now, TxOUTRATE_{1,2} = output rate in bits/sec. We
must add the time required for the sequence number
and CRC. We will then obtain
txinterval{1,2} = time in seconds between
sending packets */

txinterval1 = ((N +SEQNUMSZ+CRC_SIZE ) * 8 ) /
    TxOUTRATE_1 ;
txinterval2 = ((N +SEQNUMSZ+CRC_SIZE ) * 8 ) /
    TxOUTRATE_2 ;

lastsend1 = timenow;
for (m=0; m<NUM_DEST; m++) {
    lastsend2[m] = timenow;
}

retrandestcount=0;
do {
    for (m=0, retrandestcount=0; m<NUM_DEST; m++) {
        if ( retranflag[m] )
            retrandestcount++;
    }

    /* Sending of packets */

    /* In two steps, we see if it is time to send a
    packet; that is, if enough time has passed since
    sending the last packet.  We use one step for
    packets to be sent over the primary link, and
    one for the secondary link */

    /* We want to get rid of any packets requiring
    retransmission as soon as possible, so we'll
    consider first the retransmission link */
```

```
        if (TERR_RETRAN && retrandestcount) {
            gettimeofday(&timenow, NULL);
            for (m=0; m<NUM_DEST; m++) {
                /* in case it is not yet time, or no
                packet to send */
                nexttosend[m] = -1;
                if (!retranflag[m]) {
                    continue;
                }
                if (it's time to send a packet
                    terrestrially) {

                nexttosend[m] = findpkttosend(m, 1);
                /* first arg. of findpkttosend() is
                the destination number, second tells
                if the search should be only for
                packets to be retransmitted
                terrestrially */

                }
            }

            /* Below, we don't want to timestamp it
            *now*, rather use the earlier time of "timenow",
            since otherwise the acheived transmission
            rate is strictly less than the
            desired transmission rate, significantly */

#if (NUM_DEST>1)
    /* See if perhaps it would be more efficient to
    send the packet over the satellite link instead of
    the terrestrial link */
    /* Here we make use of knowing that NUM_DEST>1,
    then NUM_DEST is actually 2 */
/* (NUM_DEST==2) */
            if (nexttosend[0] == nexttosend[1] &&
                nexttosend[0] != -1) {
                /* Send same frame to all destinations
                over the satellite link */
                for (m=0; m<NUM_DEST; m++) {
                    qq = nexttosend[m] % TXARRSZ;
```

```
                    Tx(txarr[qq].packet,
                    SEQNUMSZ+DATASZ+CRC_SIZE, m);
                    txarr[qq].doretran[m] = FALSE;
                    txarr[qq].timesent[m] = timenow;
                    if (nexttosend[m]>SNmax) {
                        SNmax = nexttosend[m];
                    }
                }
                lastsend1 = timenow;
            }
            else {
                for (m=0; m<NUM_DEST; m++) {
                    if (nexttosend[m] == -1) {
                        continue;
                    }
                    qq = nexttosend[m] % TXARRSZ;
                    ReTx(txarr[qq].packet,
                        SEQNUMSZ+DATASZ+CRC_SIZE, m);
                    lastsend2[m] = timenow;
                    txarr[qq].doretran[m] = FALSE;
                    txarr[qq].timesent[m] = timenow;
                }
            }
#else /* ==> NUM_DEST ==1 */
            if (nexttosend[0] != -1) {
                qq = nexttosend[0] % TXARRSZ;
                ReTx(txarr[qq].packet,
                    SEQNUMSZ+DATASZ+CRC_SIZE, 0);
                lastsend2[0] = timenow;
                txarr[qq].doretran[0] = FALSE;
                txarr[qq].timesent[0] = timenow;
            }
#endif  /* NUM_DEST>1 */
        }

        /* Now search for a packet to send via
        satellite;
        If we are engaged in a retransmissions for GBN,
        we don't want to send some packets which are
        not going to be accepted at the stations
        requiring those retransmissions.  Also, we want
        to be sure we send only retransmitted packets
```

```
--and on the correct links--for GBN
retransmissions */
if (! (!SELECTIVE && retrandestcount &&
    TERR_RETRAN)) {
    for (m=0; m<NUM_DEST; m++) {
        /* in case it is not yet time,
        or no packet to send */
        nexttosend[m] = -1;
        if (it's time to send a packet on the
            satellite link)
        nexttosend[m] = findpkttosend(m, 0);

        /* we will not send the packet now, but
        collect them all and send them together
        a few lines below... */
    }

    /* ...and here is where we send them */
    for (m=0; m<NUM_DEST; m++) {
        if (nexttosend[m] == -1)
            continue;
        qq = nexttosend[m] % TXARRSZ;
        Tx(txarr[qq].packet,
            SEQNUMSZ+DATASZ+CRC_SIZE, m);
        txarr[qq].sent[m] = TRUE;
        txarr[qq].doretran[m] = FALSE;
        txarr[qq].timesent[m] = timenow;
        if (nexttosend[m]>SNmax) {
            SNmax = nexttosend[m];
        }
        /* we actually have to do this only
        once, but must be sure to do it
        only if we actually send something
        on the primary link */
        lastsend1 = timenow;
    }
}

/* process acknowledgments */
/* Structure of an acknowledgment:
- List of packets requiring retransmission
    (SR only, and only if necessary);
```

```
- Request number
- CRC on all above */
for (m = 0; m<NUM_DEST; m++) {
    GetAck(m);
}
/* consolidate the information from the
acknowledgments and release packets all
packets 0, 1,..., X such that these X+1
packets have been ACKed by all
destinations */
i = SNmin;
jj = i % TXARRSZ;
while (i<=SNmax) {
    ackcount=0;
    for (m=0; m<NUM_DEST; m++) {
        if (txarr[jj].acked[m] == TRUE)
            ackcount++;
    }
    if (ackcount == NUM_DEST) {
        /* packet number i may be discarded,
        everyone's ack'ed it */
        for (m=0; m<NUM_DEST; m++) {
            txarr[jj].doretran[m] = FALSE;
            txarr[jj].sent[m] = FALSE;
        }
        txarr[jj].valid = FALSE;
        SNmin++;
        i++;
        jj++;
        if (jj == TXARRSZ)
            jj = 0;
    }
    else {
        break;
    }
}


/* check for time out */
gettimeofday(&timenow, NULL);

/* must do this for all the destinations */
```

```
        for (m = 0; m<NUM_DEST; m++) {
            if (SNmax == -1)
                break;

            jj = SNmineach[m] % TXARRSZ;
            if (txarr[jj].sent[m] == TRUE &&
                    txarr[jj].doretran[m] == FALSE) {
                if (more than TIMEOUT seconds have
                passed since sending the earliest
                un-ack'ed packet to destination m) {
                    /* prepare data structure for
                    timeout-based retransmission */
                    i = SNmineach[m] ;
                    jj = i % TXARRSZ;
                    while (i<= SNmax) {
                        txarr[jj].doretran[m] = TRUE;
                        i++;
                        jj++;
                        if (jj == TXARRSZ)
                            jj = 0; /* % is slow */
                    }
                    searchbot[m] = SNmineach[m];
                    retranflag[m] = 1;
                }
            }
        }
    } while (SNmin <= SOURCE_SIZE -1 ) ;
}
```

A.3.2 ARQ Receiver

The ARQ receiver requires a single command-line argument, the address to which acknowledgments should be sent. A configuration file is used for this software as well. A data structure similar to the one used in the transmitter software is used in the receiver. Conditional compilation is used to select which set of port numbers the receiver uses.

```
/*
    ARQ Receiver
*/

/* v26: try to make Selective Repeat work properly. To
do this, we make a buffer/queue, which keeps track of
which packets have arrived in good condition, and
other such things.

The queue is used circularly by accessing its elements
with expressions of form
    head % queue_size
and the like.  This way, we can use (except when
accessing the queue) the array indices "head" and "tail"
as if the queue were infinitely large.

The tail is the highest sequence number of all accepted
packets.  The head is the sequence number X such that
all packets 0 through X-1 have been accepted, and
released (=written to the data output file, in this
context).  Thus head is the minimum sequence number of
all packets not yet received.

Instead of making confusing additional variables, we
will call head and tail RN and RNmax, respectively.
Note that this definition of RN complies with
ISO/IEC 7776.

For go-back-N ARQ, after receiving a valid, in-sequence
packet,
    RNmax = RN-1;
*/

#define TRUE 1
#define FALSE 0
typedef short BOOLEAN;
SEQNUMTYPE RXARRSZ;

struct RXARRAY {
    SEQNUMTYPE      seqno;  /* sequence number */
    unsigned char   data[PKTZ_FRAME];
    BOOLEAN         valid;
```

```
    /* BOOLEAN         acked; */
} ;
struct RXARRAY *rxarr;
SEQNUMTYPE      RN ;
/* minimum seq. num. of all packets not yet received */
SEQNUMTYPE      RNmax ;
/* maximum seq. num. of all received packets */

/* Compose the information field for a compound SR
ACK in this variable; naklist is an array, in which
we actually store sequence numbers of packets not
yet received correctly in chunks of SEQNUMSZ
( = sizeof(SEQNUMTYPE)) bytes, at 0, SEQNUMSZ,
2*SEQNUMSZ, etc. */
#define NAKLISTSZ 60    /* unit is bytes */
unsigned char   naklist[NAKLISTSZ], *nlp;
    /* the pointer is NakListPointer */
int             nakcount = 0;
    /* how many NAKs in the naklist ? */


void main(argc, argv)
int             argc;
char           *argv[];
{

    int i;

    seed_random_number_generator();

    /* Read from configuration file (names of
        associated variables given in parantheses) */:
        Output rate for acknowledgment link;
        Number of packets to expect (SOURCE_SIZE);
        Size of data structure (RXARRSZ);
        Window size (WIN_SIZE);
        Noise type: i.i.d or none (NOISE);
        Noise BER (BER);
        ARQ protocol (SELECTIVE); */
    get_parameters();
```

```
    /* make the array by dynamically allocating memory*/
    make_rxarray(RXARRSZ);

    /* Open connections for acknowledgments to be sent
    to the transmitter and for data to arrive from the
    transmitter */
    init_connections(argv[1]);

    RN = 0;
    RNmax = -1;

    run_rx_arq();
}

run_rx_arq()
{
    int             n;
    SEQNUMTYPE      j;
    int             rval1, rval2;
    unsigned char   buf1[FRAMESZ + CRC_SIZE],
                    buf2[FRAMESZ + CRC_SIZE];
    SEQNUMTYPE      seqno = 0;

    struct timeval  timenow, lastsend;
    int             bytessent;
    short           ack_to_send;
    /* indicates if there is an ack to send */

    counter = 0;
    ack_to_send = 0;

    /* we need an initial time in lastsend; we will
    subtract several seconds from the time so that
    the first ack will not be delayed */
    gettimeofday(&lastsend, NULL);
    lastsend.tv_sec -= 100;

    /* We also need an initial sendinterval, the time
    between sending packets; given by:
    (bits/frame) / (bits/second sent in channel) */
    sendinterval =  ((SEQNUMSZ+CRC_SIZE) * 8) /
    OUTPUT_RATE ;
```

```
do {
    /* two basic parts to the loop: checking for
    and processing new packets, which is done in
    every pas of the "do" loop, and sending
    acknowledgments, if it is time to do so */

    rval2=recv(sfwd2, buf2, FRAMESZ + CRC_SIZE, 0)

    if (rval2 <= 0) {
        rval1 = recv(sfwd1, buf1,
            FRAMESZ + CRC_SIZE, 0);

    if (rval2 > 0 || rval1 > 0) {
        if (rval2 > 0) {
            rxpkts2++;
        }
        else if (rval1 > 0) {
            rxpkts1++;
        }

        /* use this for determining if to send an
            ack now */
        gettimeofday(&timenow, NULL);

        /* Read the SEQNUMSZ-byte sequence number */
        if (rval2 > 0) {
            seqno = * (SEQNUMTYPE *) buf2;
            bcopy(buf2+SEQNUMSZ, tmpframe,
            PKTZ_FRAME);
        }
        else if (rval1 > 0) {
            seqno = * (SEQNUMTYPE *) buf1;
            bcopy(buf1+SEQNUMSZ, tmpframe,
                PKTZ_FRAME);
        }

        if (!NOISE) {
            if (rval2 > 0)
                check = check_crc(buf2);
            else if (rval1 > 0)
                check = check_crc(buf1);
```

```
}
else if (NOISE == 1) {
    if (rval2 > 0)
        check = check_crc(corrupt1(buf2));
    else if (rval1 > 0)
        check = check_crc(corrupt1(buf1));
}

    if (rval1>0)
        corruptedpkts1++;
    else if (rval2>0)
        corruptedpkts2++;
}
else if (!check    /* valid information ? */
        && ( (!SELECTIVE && seqno == RN)
            || (SELECTIVE /* SR ? */
            && seqno >= RN
                /* don't have it already ? */
            && seqno < RN + WIN_SIZE - 1))
                /* within the window ? */
        && rxarr[seqno % RXARRSZ].valid ==
            FALSE ) {
    /* accept the packet */
    counter++ ;

    /*  store in the buffer/array */
    rxarr[seqno % RXARRSZ].seqno = seqno;
    bcopy(tmpframe, rxarr[seqno % RXARRSZ].
        data, DATASZ);
    rxarr[seqno % RXARRSZ].valid = TRUE;

    /* we need RNmax for composing the SR
    acknowledgment */
    if (seqno > RNmax)
        RNmax = seqno;

    /* release the data--adjust RN, too */
    while (rxarr[RN % RXARRSZ].valid==TRUE
            && RN<=RNmax) {
        /* write to output file */
        record_pkt_to_disk(RN%RXARRSZ);
        rxarr[RN % RXARRSZ].valid =
```

```
                FALSE;
                RN++;
            }
            /* now RN, the "head" of the queue, is
            the minimum sequence number of all
            packets not yet received */

            /* Next, compose and send the ack */
            nakcount = 0;

            if (RNmax > RN && SELECTIVE) {
                /* compose compound ACK for SR */
                /* put RN in the sequence number
                field--this is done in SendAck */

                /* put information about other
                needed packets into the information
                field, which we will compose in
                naklist */
                nakcount = 0;
                nlp = naklist;
                j = RN + 1 ;
                while (j <= RNmax+1 && nlp-naklist
                     < NAKLISTSZ) {
                    if (rxarr[j % RXARRSZ].valid ==
                        FALSE ) {
                        bcopy((unsigned char *) &j,
                            nlp, SEQNUMSZ);
                        nakcount++;
                        nlp += SEQNUMSZ;
                    }
                    j++ ;
                }
            }
            ack_to_send = 1;
        }
        else if (!check && !SELECTIVE && seqno>RN) {
            nakcount = 0;
            ack_to_send = 1;
        }
    }   /* if (rval2 > 0 || rval1 > 0) */
```

```
        gettimeofday(&timenow, NULL);
        if (ack_to_send && it's time to send an ack) {
            bytessent = SendAck(sack);
            ack_to_send=0;
            /* compute the new sendinterval */
            sendinterval =  (bytessent * 8)/OUTPUT_RATE ;
            lastsend = timenow;
        }
    } while (rval1 != 0 && counter < SOURCE_SIZE);
    cleanup();
}               /* run_rx_arq() */

cleanup()
{
    /* release any data still in the data structure */
    while (rxarr[RN % RXARRSZ].valid  == TRUE &&
            RN<=RNmax ) {
        /* write to output file */
        record_pkt_to_disk(RN%RXARRSZ);
            rxarr[RN % RXARRSZ].valid = FALSE;
            RN++;
    }

    /* Measurements to record:
        Time elapsed between arrivals of first and last
            valid packets;
        Number of packets sent, received, and received
            in error on each link;
    */
    record_measurements();
}
```

A.3.3 Delayer

The delayer is implemented as a queue. Each queued packet is released after having been queued for a specific amount of time. This amount of time may be fixed or set stochastically to model propagation through a modem connection or through Internet, as described in Section 2.3.4 on page 29.

```
/*
    Delayer: Program to simulate propagation
    delays found in channels such as satellite
    channels, Internet, and others.
*/

#define QUEUE_SIZE 5000 /* more than adequate */

unsigned long loopcount;

int             input_port, output_port;
int             use_tcp;
double          delay_value; /* time in seconds */
char            output_address[51];
int             in_socket, out_socket, sock_in;
int             dummy=0;

double drand48();
unsigned long pktssent, pktsrecvd;
double sum_delays;
double actual_delay, max_delay, min_delay;

#if (defined INTERNET) && (defined MODEM)
    /can't define them both, only one: */
    undeclared_var = ;
#endif

main()
{
    /* Read operating parameters:
        input_port,
        output_port,
        output_address,
        delay model.
    */

    get_parameters(argc, argv[1]);

#if ((defined INTERNET) || (defined MODEM))
    seed_random_number_generator();
#endif
```

```
    /* Open UDP connections from a source and to a
    destination */
    open_connections();

    give_and_take();

    cleanup();
}

int give_and_take()
{
    struct {
        /* this is system time in
            seconds.milliseconds */
        double arrtime;
#if ((defined INTERNET) || (defined MODEM))
        float delay;
#endif
        unsigned char frame[500];
        short size;
    } qa[QUEUE_SIZE] /* queue-array */;
    int tail=-1, head=-1;

    loopcount=0;

    while ( 1 ) {
        rval = recv(sock_in, buf, sizeof(buf), 0) ;

        if (rval >= 0) {
            ftime(&tb_temp);
            ++tail;
            if (tail == QUEUE_SIZE)
                tail=0;
            if (tail == head) {
                printf("queue overflow,\
program terminated\n");
                exit(3);
            }
            if (head== -1)
                head=tail;

            qa[tail].arrtime = tb_temp.time +
```

```
            (double)tb_temp.millitm/1000.0;
                memcpy(qa[tail].frame, buf, rval);
                qa[tail].size = rval;
#if (defined INTERNET)
                /* Uniform on [0.090, 0.150] seconds */
                qa[tail].delay = 0.090 +
                    (drand48() * 0.060);
#elif (defined MODEM)
                /* Uniform on {0.160, 0.170,
                    0.180} seconds */
                qa[tail].delay = 0.160 +
                    ((lrand48() %3) * 0.010);
#endif
        }

        /* now see if it's time to send out a packet */
        if (head != -1) {
            ftime(&tb_temp);
            if ( (actual_delay = (double) tb_temp.time +
                (double)tb_temp.millitm/1000.0
                - qa[head].arrtime)
#if ((defined INTERNET) || (defined MODEM))
                    >= qa[head].delay
#else
                    >= delay_value
#endif
            ){
                Tx(qa[head].frame, qa[head].size,
                out_socket);
                pktssent++;
                if (head==tail)
                    head= -1;
                else {
                    head++;
                    if (head == QUEUE_SIZE)
                        head=0;
                }
            }
        }
        loopcount++;
    }
}
```

Bibliography

[1] Wilbur L. Pritchard and Joseph A. Sciulli. *Satellite Communication Systems Engineering.* Prentice-Hall, 1986.

[2] National Aeronautics and Space Administration, Lewis Research Center, Cleveland, OH. *System Handbook: Advanced Communications Technology Satellite (NASA TM-101490)*, 1993.

[3] Harris Corporation. *ACTS T1 VSAT Operation and Maintenance Manual*, August 1993. Prepared under NASA contract NAS3-25860.

[4] Mary L. Rivett, Zubin H. Sethna, and Jeffrey D. Speigler. *Advanced Communications Technology Satellite (ACTS) Call Manager (CM) Dialing and Call Control Specification (NASA Doc. CM-USR-002), Revision 4.0.* NYMA, Inc., April 1995. Prepared under NASA contract NAS3-27186.

[5] COMSAT Laboratories, Clarksburg, MD. *COMSAT Frame Relay Access Switch (FRACS) Reference Manual [Software Release 2.2]*, June 1994.

[6] Sun Microsystems, Inc., Mountain View, CA. *High Speed Interface/SBus (HSI/S) Installation and Administration Guide*, 1991.

[7] David P. Kennedy. Personal communication, May 1994. INTELSAT, Washington, DC.

[8] John G. Proakis. *Digital Communications.* McGraw-Hill, New York, 2nd edition, 1989.

[9] Shu Lin and Daniel J. Costello, Jr. *Error Control Coding: Fundamentals and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1983.

[10] International Standards Organization. *International Standard ISO 4335, "Information processing systems – Data communication – High-level data link control procedures – Consolidation of elements of procedures".*

[11] International Standards Organization. *Proposed Draft International Standard ISO 7776/DAM 2, "Information processing systems – Data communication – High-level data link control procedures – Description of the X.25 LAPB-compatible DTE data link procedures – DAM 2: Multi-selective Reject Option".*

[12] Dimitri Bertsekas and Robert Gallager. *Data Networks.* Prentice-Hall, 2nd edition, 1992.

[13] W. Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, 1994.

[14] Sun Microsystems, Inc., Mountain View, CA. *1.0 SunLink Frame Relay Installation and Administration Guide*, 1992.

[15] D. Perkins. *"The Point-to-Point Protocol for the Transmission of Multi-Protocol Datagrams Over Point-to-Point Links," RFC 1171.* Carnegie-Mellon University, July 1990.

[16] Arnold M. Michelson and Allen H. Levesque. *Error-Control Techniques for Digital Communication.* John Wiley & Sons, New York, 1985.

[17] Fred Halsall. *Data Communications, Computer Networks, and Open Systems.* Addison-Wesley, 3rd edition, 1992.

[18] W. Richard Stevens. *UNIX Network Programming.* Prentice-Hall, 1990.

[19] W. W. Wu. *Elements of Digital Satellite Communication*, volume 2. Computer Science Press, Rockville, MD, 1985.