August 1993

LIDS-TH-2188

Research Supported By:

National Science Foundation
Grant NCR 8802991

Defense Advanced Research Projects Agency
Grant MDA972-92-J-1038

ATT Bell Laboratories

End-To-End Reliable Communication in Data Networks

Jane Marie Simmons

August 1993

LIDS-TH-2188

Sponsor Acknowledgments

National Science Foundation
Grant NCR 8802991

Defense Advanced Research Projects Agency
Grant MDA972-92-J-1038

ATT Bell Laboratories

End-To-End Reliable Communication in Data Networks

Jane Marie Simmons

This report is based on the unaltered thesis of Jane Marie Simmons submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy at the Massachusetts Institute of Technology in August 1993.

This research was conducted at the M.I.T. Laboratory for Information and Decision Systems with research support gratefully acknowledged by the above mentioned sponsors.

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

END-TO-END RELIABLE COMMUNICATION

IN DATA NETWORKS

by

JANE MARIE SIMMONS

B.S.E. EECS, Princeton University, 1985
M.S. EECS, Massachusetts Institute of Technology, 1990

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the
Degree of

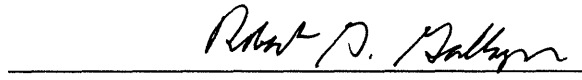
DOCTOR OF PHILOSOPHY

at the

Massachusetts Institute of Technology

August 1993

Signature of Author 
Department of Electrical Engineering and
Computer Science, August, 1993

Certified by 
Robert G. Gallager
Thesis Supervisor

Accepted by _____
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Studies

END-TO-END RELIABLE COMMUNICATION

IN DATA NETWORKS

by

JANE MARIE SIMMONS

Submitted to the Department of
Electrical Engineering and Computer Science in August, 1993
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

ABSTRACT

Reliable communication in data networks is defined as the ability to reproduce at the receiver exactly what has been transmitted by the source, with very high probability. This thesis establishes guidelines for designing network protocols that ensure reliable end-to-end communication. The first section of the thesis focuses on error detection protocols; the second section examines data retransmission schemes.

The ability to detect errors in a network, such as bit errors on the data line and lost data due to buffer overflow, is obviously an important issue. Nevertheless, it appears there has been little prior research done on the fundamental issues involved with designing an error protection scheme. A five step methodology is presented that provides insight into, first, the order in which errors should be considered when designing an error detection scheme, second, which types of error detection mechanisms are most effective, and third, which layer should be responsible for detecting a given type of error. The issues of overall effectiveness, efficiency, and robustness of an error detection scheme are addressed.

Once an error has been detected, the system must be able to either correct the error or retransmit the portion of data in error. Only retransmission options are explored in this work. A delay criterion for evaluating the performance of general retransmission schemes is presented. This is used to examine the tradeoffs involved with retransmission schemes that are based on a polling mechanism and schemes that employ a timer mechanism. The interplay between the requirements, resources, and error characteristics of a network and the design of a retransmission scheme is discussed. Much of the analysis is devoted to

poll-based schemes since they have been proposed for several high speed networks currently under development.

To provide perspective in examining these issues, the error detection and recovery schemes of Asynchronous Transfer Mode (ATM) systems and Transport Control Protocol/Internet Protocol (TCP/IP) systems are analyzed.

Keywords: Error Detection, Retransmission, Network Protocols, ATM, TCP/IP

Thesis Supervisor: Robert G. Gallager
Title: Fujitsu Professor of Electrical Engineering and Computer Science

ACKNOWLEDGEMENTS

I admire Bob Gallager for his deep understanding of so many different fields. I appreciate his sense of humor and his willingness to adapt. I thank him for being my friend.

Dimitri Bertsekas very generously allowed me to use his computer for the past few years. I thank Steve Finn for always being very enthusiastic about my research.

Dan Grossman of Codex Corp. first got me involved with the ATM retransmission scheme, which led to a major portion of this thesis. He also kept me up-to-date with the latest ATM standards.

Finally, I thank my friends for all the laughter.

This work was partially funded by the NSF, DARPA, and ATT Bell Laboratories.

TABLE OF CONTENTS

1. Introduction 7
    1.1 Overview 7
    1.2 Layering Concepts 8
    1.3 ATM 10
    1.4 TCP/IP 11
    1.5 ATM vs. TCP/IP 12
    1.6 Networks of the Future 12
    1.7 Summary of Major Contributions 13
2. Cyclic Redundancy Checks and Checksums 14
    2.1 Cyclic Redundancy Check 14
    2.2 Checksum 22
    2.3 Implicit Error Checking 24
3. Error Detection Schemes 27
    3.1 Overview 27
    3.2 Description of Layers 29
    3.3 Degree of Difficulty in Detecting An Error 36
    3.4 Guidelines for Designing Error Detection Schemes 37
    3.5 Summary 53
4. Error Detection in ATM 55
    4.1 ATM Overview 55
    4.2 General Properties of ATM Networks 56
    4.3 Error Characteristics of ATM Networks 57
    4.4 Design of ATM Error Detection Scheme 59
    4.5 Summary of ATM Error Detection Scheme 70
    4.6 Implementation 71
    4.7 Alternative Schemes 71
    4.8 SONET 76
    4.9 Conclusions 80
5. Error Detection in TCP/IP 82
    5.1 TCP/IP Overview 82
    5.2 TCP/IP Description 84
    5.3 Assumptions 88
    5.4 Analysis of TCP/IP Error Detection Scheme 89
    5.5 Conclusions 103
6. Retransmission Schemes 106
    6.1 Introduction 106
    6.2 Poll-based Retransmission Schemes 110
    6.3 Timer-based Retransmission Schemes 133
    6.4 Poll-based Schemes vs. Timer-based Schemes 137
    6.5 Combination of Poll-based and Timer-based Schemes 141
    6.6 Selective Repeat vs. Go Back N 145
7. Retransmission in ATM 150
    7.1 Introduction 150
    7.2 Description of Proposed Retransmission Scheme 151
    7.3 Proof of Proper Operation 155
    7.4 Potential Problems 171
    7.5 Poll-based Scheme in ATM Environment 175

8. Retransmission in TCP/IP 181
8.1 Introduction 181
8.2 Description of TCP Retransmission Scheme 181
8.3 Retransmission Policy Analysis 185
8.4 Estimation of Round Trip Delay 195
8.5 Poll-based Scheme in TCP/IP 199
8.6 Conclusions 202
9. Conclusions 203
9.1 Major Contributions 203
9.2 Stabilization Properties of Retransmission Schemes 204
References 207
Biography 210

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

All real-world data network systems are subject to errors. Thus, it is necessary to develop network protocols that detect and recover from errors. This thesis focuses on establishing guidelines for designing protocols that ensure reliable end-to-end communication, where reliable communication is defined as the ability to duplicate at the receiver exactly what has been transmitted by the source, with very high probability. Obviously the many data networks currently in use include mechanisms for this purpose. However, there does not appear to be an established systematic approach to dealing with errors. Many systems that are currently being designed have taken a rather ad hoc approach to providing reliable communication. The major contribution of our research is to identify the fundamental issues involved with providing error protection and to specify guidelines for designing protocols that effectively and efficiently handle errors.

There are two major facets of providing reliable data delivery: error detection and error recovery. Error detection usually takes the form of including check fields along with the data, such as parity checks or a field indicating the amount of data sent. Undetected errors are inevitably going to occur in any system regardless of the number of error detection fields included. We determine which error detection fields are appropriate to add, given the characteristics of the network, in order to reduce the rate of undetected errors to a level acceptable to users. In chapter two, we describe common techniques for detecting errors. In chapter three, a step-by-step algorithm for designing effective and efficient error detection schemes is presented. These guidelines are applied to actual systems in chapters four and five.

Throughout our research we will be concerned with several different levels of errors. The lowest level errors are due to physical properties of the network. For example, this includes bit errors on the data line, burst errors due to equipment malfunction, and lost data due to buffer overflow. These low-level errors may propagate up through the various layers of the network. For instance, an undetected bit error in an address field may result in data being routed to the wrong destination; if the stray data is not detected, extra data may be delivered to the user at this wrong destination. Thus, the choice of error detection fields at one layer impacts the probability of error at higher layers. This is explored in detail in chapter three.

Once an error has been detected, the system must be able to either correct the error or retransmit the portion of data in error. Our discussion focuses on the retransmission option. We present a delay criterion for evaluating the performance of retransmission schemes in general. This criterion is used to explore the tradeoffs involved with retransmission schemes that are based on a polling mechanism and schemes that employ a timer mechanism. Based on this analysis, we specify how the requirements, resources, and error characteristics of a network should influence the design of a retransmission scheme. These guidelines are applied to actual systems in chapters seven and eight.

To provide perspective in examining the issues of reliable communication, we analyze two very different systems: Asynchronous Transfer Mode (ATM) systems, and Transport Control Protocol/Internet Protocol (TCP/IP) systems. An overview of these systems is given below in Sections 1.3 and 1.4; the details of the protocols will be described in later chapters. The next section discusses layering issues in data networks to provide insight into the functionality of ATM and TCP/IP.

1.2 LAYERING CONCEPTS

Most networking systems are designed with a layered architecture. Each layer has specific functions that it must perform. A layer receives input from the layers immediately above or below it, processes the input, and passes the result to a layer below or above it. The adjoining layers are unaware of the specific processing details; they need only be concerned with the interface. This modularity allows designers to work on one layer independently of all other layers, allows new versions of a layer to be incorporated without affecting other layers, and results in a number of interchangeable layers.

The lowest layer in a networking system is the communication link connecting the network nodes. The layers above this are distributed i.e., modules of the same layer exist at both ends of a link. Modules which operate at the same layer are referred to as peer modules. We can consider two types of communication in which a module participates. First, there is communication directly with the modules above or below it. Second, and more important, there is indirect communication with peer modules at other network nodes. It is the combination of these two types of communication that ultimately provides a meaningful exchange of information among nodes in a network.

Peer modules can communicate on a node-by-node basis, or on an end-to-end basis. The lower layers of a network generally operate on a node-by-node basis, and are involved with the transfer of data at each node along the data path. The higher layers generally operate end-to-end; only the source and destination process the data at these layers.

Different organizations have established different layering standards. Below, we give an overview of the seven layers of the Open Systems Interconnection (OSI) architecture, which was proposed by the International Standards Organization. First, we review the conventions for describing data entities, as established in the OSI model[MeP82]. A Service Data Unit (SDU) is the data unit that a layer receives from the next higher layer. A Protocol Data Unit (PDU) is the data unit exchanged between peer modules. It is necessary to distinguish between these different data units in order to maintain modularity of design. Consider any given layer, which we refer to as layer N. It may interface with a variety of layer N+1 entities (where we consider layer N+1 to be the layer above layer N). The layer N SDU is the data passed down from a layer N+1 entity so that layer N can perform the functions requested by layer N+1. Layer N must determine the correct Protocol Control Information (PCI) to append to the SDU; the PCI is the information exchanged between the layer N peer modules in order to co-ordinate their operation. The PCI will depend on the services requested by layer N+1, and the peer modules with which layer N is interacting. Thus, different PCI could be added to an SDU depending on the circumstances. The combination of the layer N SDU and the layer N PCI is the layer N PDU.

The top three layers of the OSI model operate end-to-end. The highest layer is the application layer. Below that is the presentation layer, which performs tasks such as data compression and encryption. The next layer down is the session layer, which deals with setting up connections. These three layers will not be discussed further.

The transport layer, which also operates end-to-end, is below the session layer. The SDU at the transport layer is often referred to as a message. The functions of the transport layer are: divide messages into smaller entities at the source and reassemble them at the destination; multiplex sessions that have the same source and destination; and provide end-to-end reliable data delivery. It is the functions of the transport layer with which we are most concerned in this research.

Below the transport layer is the network layer, which operates network-wide and provides routing and flow control services for network layer PDUs. PDUs at the network layer are usually called packets for connection oriented sessions, and datagrams for connectionless sessions. In connection oriented sessions, a data path is mapped out between source and destination, and all packets from the session follow that path. In connectionless sessions, there is no pre-established path; datagrams are routed independently of each other.

The next layer down is the data link control (DLC) layer, which provides reliable node-by-node delivery of packets. The DLC PDU is usually called a frame. Much of the frame overhead is for the purpose of error detection and recovery. The lowest layer of the OSI model is the physical layer. Its function is to transmit bits over a physical communication channel.

The ATM and TCP/IP protocols do not conform to specific OSI layers. Rather, they are more an amalgamation of OSI layers.

1.3 ATM

ATM is a protocol being designed for use on Broadband Integrated Services Digital Network (B-ISDN) systems. The goal of B-ISDN is to provide transmission services for voice, data, and video at data rates of 150 Mb/sec and higher. ATM provides a common format for transmitting these different traffic types. We deal specifically with reliability issues for the connection oriented, variable bit rate service class of ATM.

ATM is comprised of the ATM layer protocol and the ATM Adaptation Layer (AAL) protocol, which lies above the ATM layer. There is not a direct correspondence between these two layers and the OSI layers. The ATM layer operates node-by-node and performs some of the functions associated with the physical and network layers. The AAL operates end-to-end and performs functions associated with the transport layer in providing multiplexing and end-to-end reliable data delivery. It should be noted that the unit of data

the AAL receives from higher layers is called a frame, thus prompting some researchers to consider both the ATM and AAL layers as part of the physical layer.

In general, when we refer to ATM we will be referring to ATM systems and not the ATM layer protocol. ATM has not yet been finalized as a standard by the International Telegraph and Telephone Consultative Committee (CCITT). In working on this thesis we have attempted to keep up with the current status of the proposed standard. Also, we kept our analysis as general as possible so that changes in the proposed standard will not significantly affect our results.

ATM is to be run over fiber optic lines, a very reliable medium; thus, we expect a low bit error rate on ATM data links. The chief source of low level errors is expected to be data lost due to congestion. The error detection scheme originally proposed by the CCITT is fully described in chapter four. However, we will show that by applying the error detection design techniques outlined in chapter three, we can come up with a scheme that is both more effective and efficient than the CCITT proposal. A scheme similar to ours has been proposed by another researcher also, although without any analysis[Lyo91]. The CCITT is currently considering adopting such a scheme as part of the ATM standard[CCI92a].

The retransmission scheme proposed by the CCITT is based on a polling mechanism, rather than on a traditional timing mechanism. In chapter seven, we show that the characteristics of ATM systems actually favor the use of a timing mechanism.

1.4 TCP/IP

The second representative system we will examine is the Internet and its related protocols, IP and TCP. The goal of the Internet is to provide data communication services between thousands of diverse computer networks distributed worldwide. The speed, resources, and capabilities of these interconnected networks are very different. Also, the transmission speed and error characteristics of the lines connecting these networks span a wide range.

The various networks are connected by computers that serve as gateway nodes. IP performs some of the functions of a network layer and provides a connectionless unreliable datagram delivery service between the gateway nodes. Datagrams may be lost, duplicated, misordered or damaged while in transit. TCP is a connection oriented transport protocol

that operates above the IP layer on an end-to-end basis. It provides connection management, reliable data transport, and flow control services.

The TCP/IP protocol has been in use for about a decade. The error detection mechanisms of TCP/IP are described in chapter five. We show that the TCP/IP error detection scheme closely corresponds to the design guidelines presented in chapter three. TCP uses a timer-based retransmission scheme. In chapter eight, we analyze why such a scheme performs poorly in the TCP/IP environment.

1.5 ATM vs. TCP/IP

We chose to analyze ATM and TCP/IP due to the very different nature of these protocols. The ATM environment is expected to be relatively uniform, i.e., high speed and reliable. The networks over which TCP/IP is run, however, are very diverse in terms of speed and error rates. ATM is purposely being designed to carry a wide range of traffic types. TCP/IP was designed for a homogeneous traffic environment. The ATM packet delivery service that we analyze is connection oriented; traffic at the IP layer is connectionless. Despite these differences, we show that the guidelines we develop for the design of error protection protocols are applicable to both systems.

1.6 NETWORKS OF THE FUTURE

The major goal of this thesis is to provide guidelines for the design of error protection protocols for general network systems. These guidelines present a systematic approach to designing systems of the future. Many of the networks that have been proposed for the future, including ATM, are referred to as high speed networks, since the data rates are expected to be very high. The fact that the data speed is high does not in itself imply that a system is very different from a system where the data speed is low. For example, if the data rate, packet size, and buffer size are all scaled by the same factor (and flow control is equally effective), then transmission and queueing delays do not change.

From the point of view of designing an error protection scheme, a more significant difference in these proposed networks of the future is that the high speeds will be accompanied by the greater reliability of fiber optics. Another significant difference is that many of the proposed networks are expected to carry a wide range of traffic types that have very different requirements in terms of data rate, burstiness, and sensitivity to delay and error. Thus, the protocols that are developed for these networks should be suitable for a

range of traffic types. Where appropriate, we discuss how these characteristics of future networks affect the design process of error protection schemes.

1.7 SUMMARY OF MAJOR CONTRIBUTIONS

Our research makes several original contributions in the areas of error protection protocols for data networks. The ability to detect errors in a network is obviously an important issue. Nevertheless, it appears there has been little prior research done on the fundamental issues involved with designing an error protection scheme. By addressing the fundamental issues, we develop a logical and coherent approach to the problem. The methodology presented in chapter three provides insight into, first, the order in which errors should be considered when designing an error detection scheme, second, which types of error detection mechanisms are most effective, and third, which layer should be responsible for detecting a given type of error.

We have also made several contributions in the area of retransmission schemes. First, an in-depth study of poll-based schemes is provided. Despite the fact that such schemes have been in existence for over a decade, and have been proposed for networks currently being designed, there has been little published on how such schemes perform. We analyze the advantages and disadvantages of poll-based schemes, compare them with timer-based schemes, and examine in which network environments a poll-based scheme and a timer-based scheme are best suited. We formally prove the correctness of the proposed ATM poll-based retransmission protocol, under certain conditions. As of the writing of this thesis, such a proof had not been provided by the CCITT. Also, in analyzing the validity of the ATM retransmission scheme, we further define the protocol.

CHAPTER 2

CYCLIC REDUNDANCY CHECKS AND CHECKSUMS

Cyclic redundancy checks (CRCs) and checksums are both forms of redundancy checks that are commonly used to detect errors in data. Since they play an important role in the error protection schemes we will examine in chapters three through five, we review their error detection properties below. Section 2.1 describes CRCs and Section 2.2 analyzes checksums. In Section 2.3, we describe the technique of using a CRC or checksum to implicitly check on other error detection fields. In some situations, this technique provides an additional amount of error protection without any additional bits of overhead.

2.1 CYCLIC REDUNDANCY CHECK

A CRC code is a type of parity check code. Below, we describe how a CRC code is generated and analyze its error detection properties.

Let D be the number of data bits we wish to check, and L be the number of check bits we wish to add. The L check bits are referred to as the CRC. Denoting the data bits by $s_{D-1}, s_{D-2}, \ldots, s_0$, where s_{D-1} is the first bit of data, we can represent the data by the polynomial:

$$s(X) = s_{D-1}X^{D-1} + s_{D-2}X^{D-2} + \ldots + s_0, \text{ where X is an indeterminate.}$$

Similarly, we can represent the CRC by the polynomial:

$$c(X) = c_{L-1}X^{L-1} + c_{L-2}X^{L-2} + \ldots + c_0.$$

Let g(X) be a polynomial of degree L, referred to as the generator polynomial. Then, for each possible data string s(X), we generate the CRC as follows:

$$c(X) = \text{Remainder}[s(X)X^L/g(X)] \quad \text{(using binary arithmetic).}$$

We can represent the combination of data bits and check bits by:

$$q(X) = s(X)X^L + c(X) \qquad (2.1)$$

We can think of q(X) as being a codeword in the code generated by g(X). It can be shown (see [BeG92]) that any codeword q(X) is divisible by g(X), and that any polynomial divisible by g(X) must be a valid codeword. If g(X) factors X^{L+D} -1, then the code generated by g(X) is a cyclic code i.e., any cyclic shift of a codeword is another codeword.

It will also be helpful sometimes to represent the j^{th} bit of the CRC as follows:

$$c_j = \sum_{i=0}^{D-1} s_i \, c_j^{(i)} \quad 0 \le j < L \qquad (2.2)$$

where $c^{(i)}$ is the CRC that results from the data string with a single 1 in position i.[BeG92] From this representation we see that a CRC is indeed a parity check.

Assume that the codeword q(X) representing the data and CRC is sent over a noisy channel. Let e(X) be the polynomial representing the errors that occur in the transmission of the codeword. The coefficient e_i is 1 if a bit error occurs in position i of the transmitted sequence; otherwise e_i is 0. The received sequence can thus be represented by r(X) = q(X) + e(X), where the addition is bitwise modulo 2. The decoding rule at the receiver is: accept r(X) as error-free if and only if r(X) is divisible by g(X) (i.e., iff r(X) is a valid codeword). Since q(X) is divisible by g(X), this acceptance rule is equivalent to requiring e(X) to be divisible by g(X). Thus, if the error sequence is not all zeros, then the CRC fails to detect an error if and only if the errored bits are located such that e(X) is divisible by g(X) (i.e., iff e(X) is a valid codeword).

2.1.1 Extended Hamming Codes

The error detecting properties of the CRC depend on the choice of the generator polynomial. If g(X) is chosen to be the product of the polynomial (X+1) and a primitive polynomial of degree L-1, where

$$2^{L-1} - L \ge D+1 \qquad (2.3)$$

then the resulting code is called an Extended Hamming code. (For a discussion of primitive polynomials see [Gal68].) The codewords of the Extended Hamming code generated by g(X) are the even weight codewords of the Hamming code generated by $\frac{g(X)}{(X+1)}$. (The weight is defined as the number of 1's in the codeword.) Extended Hamming codes are better than ordinary Hamming codes for error detection purposes; their error detection properties are discussed below.

All codewords in an Extended Hamming code have even weight since (X+1) is a factor of the generator polynomial and hence a factor of all codewords. It also can be shown (see [BeG92]) that the codewords never have weight two. Since a CRC only fails to detect an error if e(X) is a valid codeword, we conclude that an Extended Hamming code CRC can detect all odd numbers of bit errors and all double bit errors. Furthermore, it can be shown that such a CRC is capable of detecting all error bursts within a span of L bits.[BeG92] Finally, the probability of the CRC failing to detect errors in a completely random string is:

$$2^{-L} \qquad (2.4)$$

From the above discussion, we see that at least four bit errors must occur, in the combination of the data and the CRC bits, in order for the CRC to fail to detect an error. This is equivalent to stating that the minimum distance between codewords is four. The distance between codewords refers to the number of bit positions where the codewords differ. (In ordinary Hamming codes, the minimum distance is only three.) This does not imply that all combinations of four or more errors result in an undetected error. Only those error patterns that correspond to a valid codeword cause undetected errors.

Consider the Extended Hamming code produced by the degree L polynomial g(X) where L is chosen such that equation (2.X) holds with equality. Thus, $L+D = 2^{L-1} - 1$ and we refer to the code as a full-length code. For a full-length Extended Hamming code, it is known precisely how many error patterns will not be detected by the CRC. Let A_i be the number of error patterns of weight i that will not be detected by the Hamming code of length $2^{L-1} - 1$ generated by $\frac{g(X)}{(X+1)}$. Equivalently, A_i is the number of weight i codewords in the Hamming code. Letting $T = 2^{L-1} - 1$ (where T is referred to as the block length), A_i can be shown to satisfy the following relation[VaV89]:

$$A_0 = 1, \quad A_1 = 0$$
$$(i+1)A_{i+1} + A_i + (T-i+1)A_{i-1} = \binom{T}{i} \qquad 1 \le i \le T, \qquad (2.5)$$

Let B_i equal the number of error patterns of weight i that will not be detected by the full-length Extended Hamming code CRC. Then:

$$B_i = A_i \quad i \text{ even}$$
$$B_i = 0 \quad i \text{ odd} \qquad (2.6)$$

If bit errors occur randomly with probability P_R, then the overall probability of a full-length Extended Hamming code CRC failing to detect random bit errors is:

$$P_E = \sum_{i=1}^{T} B_i \, P_R{}^i \, (1-P_R)^{T-i} \qquad (2.7)$$

If P_R is small, P_E will be dominated by the term due to 4 bit errors. From equations (2.5) and (2.6) we see that:

$$B_4 = \frac{1}{4}\left(\binom{T}{3} - \frac{1}{3}\binom{T}{2}\right) \qquad (2.8)$$

Thus,

$$P_E \approx B_4 P_R^4 = \frac{1}{4}\left(\binom{T}{3} - \frac{1}{3}\binom{T}{2}\right) P_R^4 \qquad (2.9)$$

By examining the Hamming code of block length T we can show why B_4 (and A_4) equals $\frac{1}{4}\left(\binom{T}{3} - \frac{1}{3}\binom{T}{2}\right)$. First, we derive A_3. The Hamming code is a perfect code: any sequence of length T is either a codeword or at distance 1 from a codeword. We know that there are no weight-1 or weight-2 codewords in a Hamming code. Take any two bit positions i and j and consider the sequence that has a 1 in these two positions and zeros elsewhere. This sequence is not a codeword since it has weight 2. Thus, it must be at distance 1 from a codeword. Since there are no weight-1 codewords, it must be at distance 1 from a weight-3 codeword. There can only be one such weight-3 codeword. If there were more than one, then there would be a weight-3 codeword with 1's in positions i, j, k and a weight-3 codeword with 1's in positions i, j, k', where $k \neq k'$. But these two weight-3 codewords would be at distance 2 from each other which is impossible for a Hamming code.

Thus, for each choice of i and j there is exactly one corresponding weight-3 codeword with a 1 in positions i, j, and some k. There are $\binom{T}{2}$ ways to choose 2 positions. However, this includes choosing (i,j) and (i,k) and (j,k), all of which have the same corresponding weight-3 codeword (otherwise there would be codewords at distance 2 from each other). Thus, there must be exactly $\frac{1}{3}\binom{T}{2}$ weight-3 codewords in the Hamming code.

We can use a similar argument to derive A_4. Take any three bit positions, a, b, c, and consider the sequence with a 1 in these three positions, and a 0 elsewhere. This sequence is either a codeword itself or at distance 1 from a weight-4 codeword. (Recall there are no weight-2 codewords in a Hamming code.) We know there are $\binom{T}{3}$ possible (a, b, c) combinations, and A_3 of them correspond to weight-3 codewords. The remaining sequences must be at distance 1 from a weight-4 codeword. Using the same argument as

above, there is at most one weight-4 codeword at distance 1 from the sequence. (Let d be the location of the fourth 1). Thus, $A_4 = B_4 = \frac{1}{4}\left(\binom{T}{3} - A_3\right)$. There is a factor of $\frac{1}{4}$ since the combinations of (a,b,c) and (a,b,d) and (b,c,d) and (a,c,d) are at distance 1 from the same weight-4 codeword (otherwise there would be codewords at distance 2 from each other).

2.1.2 Shortened Codes

If equation (2.3) holds with strict inequality then we have a shortened code, i.e., $L+D < 2^{L-1} - 1$. The codewords in a shortened code are related to codewords in the full-length code. Let T= L+D. For any codeword $c_0\ c_1\ ...\ c_{T-1}$ in the shortened code, there is a codeword $c_0\ c_1\ ...\ c_{T-1}\ 0\ 0\ ...\ 0$ in the full-length code, and vice-versa. Since we are truncating 0's in order to generate the shortened codewords, it must be true that the minimum weight of the shortened code is at least as great as the minimum weight of the full-length code. It is possible that the minimum weight of the shortened code is greater than the minimum weight of the full-length code. Consider a shortened Extended Hamming code of length T. If there are no codewords in the full-length code that have exactly four 1's in the first T positions and all 0's in the last $2^{L-1} - 1 - T$ positions, then there will be no weight-4 codewords in the shortened code. Thus, the minimum weight of the shortened code will be at least 5.

In a shortened Extended Hamming code, the exact number of error patterns that will not be detected by the CRC is not known in general. Assume P_R is very small so that the probability of error due to random bit errors is dominated by the term due to 4 bit errors. We use the following argument to upper bound the number of four-bit error patterns that will not be detected by the CRC. The error sequence e(X) must be a codeword in order for the CRC to fail to detect the error. Choose any three distinct bit positions i, j, and k, and let $e_i = e_j = e_k = 1$. Given that errors have occurred in these three positions, there is at most one position where the fourth bit error can occur to yield an e(X) that is a valid codeword. Let's assume this were not true. Let e(X) represent errors in positions i, j, k and z, and let e'(X) represent errors in positions i, j, k, and z', where $z \neq z'$. Since e(X) and e'(X) must themselves be codewords, this implies that the minimum distance between codewords is no more than two. This contradicts the fact that the minimum distance between codewords in an Extended Hamming code is four. We conclude that given bit errors in any three locations, there is at most one location where the fourth bit error can

occur to possibly cause the CRC to fail to detect the error. Thus, letting T represent the total number of data and CRC bits, the number of possible four bit patterns that can cause an undetected error can be upper bounded by:

$$\binom{T}{3}\frac{1}{4} \qquad (2.10)$$

The factor of $\frac{1}{4}$ is necessary since there are 4 ways of choosing 3 positions out of (i, j, k, z). Equation (2.10) is only an upper bound; a shortened Hamming code is not a perfect code, thus we can not use the same technique as in the previous section to derive the exact number of error patterns. Note that this bound is valid for the case of a full-length code also.

This bound might be very loose. As we discussed at the beginning of this section, it is possible that if the code is shortened enough, it will contain zero weight-4 codewords. Note that in the literature, the upper bound typically used is $\binom{T}{4}$; thus, our bound is tighter by a factor of T.

2.1.3 Error Correction

Since the minimum distance between codewords in an Extended Hamming code is four, the CRC is capable of correcting all single bit errors. Using a CRC to correct errors is described in [Gal68]. Here, we outline the process. Based on the generator polynomial g(X), the receiver forms what is called a parity-check matrix H. The matrix H has the property that for any sequence y of length T, yH=0 iff y is a valid codeword. Let q be the transmitted sequence and r be the received sequence. The error sequence e can be expressed as e = q + r, where the addition is bitwise modulo-2. We know q is a valid codeword; thus qH = 0. The receiver calculates what is called the syndrome S, where S = rH = (r + q) H = eH. If S equals 0, the receiver assumes no error has occurred. If S does not equal 0, then the receiver uses a table to determine the minimum weight sequence e' that produces such a syndrome and corrects the received sequence to r + e'. For an Extended Hamming code, this allows us to correct all single bit errors i.e., all error sequences of weight-1 appear in the syndrome table.

Using an Extended Hamming code CRC for correction rather than just detection increases the probability an error will not be detected. In either mode, if an error sequence happens to be a valid codeword, the error will go undetected. In the correction mode, however, it is

possible the CRC will 'correct' the received sequence to the wrong value, since three or more bit errors may produce a syndrome that corresponds to a single bit error.

Assume that the weight-4 sequence with 1's in positions a, b, c, and d corresponds to a valid codeword. If an error sequence has 1's in exactly three of these four positions, the CRC will make a 'false' correction. Let B_4 be the number of weight-4 sequences in the CRC code. Then 4 B_4 is the number of weight-3 error sequences that lead to false corrections. Assuming P_R is small, we approximate the probability of undetected error due to random bit errors by: $4\ B_4\ P_R^3$. (Compare this to $B_4\ P_R^4$ if the CRC is only used for detection.) For a full-length code, using equation (2.8), this corresponds to:

$$\left(\binom{T}{3} - \frac{1}{3}\binom{T}{2}\right) P_R^3 \qquad (2.11)$$

For a shortened code, we use the upper bound of equation (2.10), which leads to the upper bound:

$$\binom{T}{3} P_R^3 \qquad (2.12)$$

If we wish to be able to correct single bit errors, but still want to be able to detect up to three bit errors with certainty, then we can double the length of the CRC and use a BCH code, as described in Section 2.1.6 below.

The probability that a burst error will not be detected by an L bit CRC in the correction mode is the probability that a random sequence is a codeword or is at distance 1 from a codeword. If D is the number of data bits being checked, then there are 2^D possible codewords (corresponding to each possible data string); there are T sequences that are at distance one away from a given codeword, where T = D + L. After the burst error hits, any given sequence will occur with probability 2^{-T}. Thus, the probability the CRC will not detect the burst error is:

$$\frac{2^D + T\ 2^D}{2^T} = (T + 1)\ 2^{-L} \qquad (2.13)$$

For the full-length Extended Hamming code, this probability is .5. If we wish to perform error correction but still want to provide about the same level of protection against burst errors as a CRC of length L used in the detection mode, then we would need to double the length of the CRC to 2L.

2.1.4 Summary of Extended Hamming Code CRC

In practice, we are usually dealing with shortened codes rather than full-length codes. Thus, we will upper bound the probability of four random bit errors occurring and going undetected by $\binom{T}{3} \frac{1}{4} P_R^4$. For small values of P_R, terms due to more than four random bit errors will be insignificant in calculating the probability of undetected random bit errors. Also, if the transmitted sequence is affected by an error burst such that the bits are randomly 0 or 1, an L bit CRC fails to detect the error with probability 2^{-L}. These probabilities of undetected random bit errors and undetected burst errors are used repeatedly throughout our research.

2.1.5 CRC-32

The standard 32 bit CRC used in practice is generated by:

$g(X) = X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+ 1.$

This g(X) is a primitive polynomial of degree 32; it does not have X+1 as a factor. Thus, the resulting code is a ordinary Hamming code rather than an Extended Hamming code and the minimum weight of the full-length code is 3. The full length of this code is $2^{32} - 1$. However, in practice, the length of the data plus CRC is usually much smaller than this. The effect of shortening the CRC-32 code has been studied in [FKL89]. If the length of the shortened code is between 4096 and 121444 bits, then the minimum weight is 4. If the length is less than 4096, then the minimum weight of the code is 5. The probability of undetected error is smaller in the shortened code.

2.1.6 BCH Codes

Another type of effective CRC code is the Extended BCH code. From [Pet61], we know that if we choose an appropriate generator polynomial of length L, where L is roughly twice the minimum specified in equation (2.3) and (X+1) is a factor of the generator polynomial, then the minimum distance of the code is at least 6. Thus, at least 6 bit errors must occur before an error will go undetected.

For a shortened Extended BCH code, we can follow the proof given in Section 2.1.2 to upper bound the number of weight-6 error sequences that are not detected by the CRC. Given bit errors in any 4 positions, there is at most one set of locations where the fifth and sixth bit errors can occur to possibly cause the CRC to fail to detect the error. If this were not true, the minimum distance of the code could be no larger than 4. Thus, letting T be the

total number of data and check bits, the number of possible weight-6 error patterns that can cause an undetected error can be upper bounded by:

$$\binom{T}{4}/\binom{6}{4} \qquad (2.14)$$

Compared to the Extended Hamming code, the Extended BCH code requires twice the number of check bits but decreases the probability of undetected error by roughly a factor of P_R^2 T.

Since the Extended BCH code has a minimum distance of six, it is capable of correcting all single and double bit errors. However, in the correction mode, four bit errors are capable of causing an undetected error.

2.2 CHECKSUM

In this section we look at the error detection properties of checksums. In general, checksums are less effective than CRCs in detecting errors, but they are easier to implement in software.

Assume the length of the checksum is L bits. There are different methods of calculating the checksum; we will analyze the method used in TCP and IP. This method makes use of the one's complement sum, which is defined as the L bit sum using an end around carry. Thus, the arithmetic is performed modulo 2^L-1. For example, if L equals 3, the one's complement sum of the binary numbers (100) and (101) is (010) (i.e., 4+5 = 2 mod7). Note that the value zero is represented by both 2^L-1 and 0.

The algorithm for calculating the checksum in TCP and IP is: partition the data being checked into L-bit 'words' (each word can be thought of as being the binary representation of some integer); calculate the one's complement sum of all L-bit words; the one's complement of this sum becomes the checksum. (The one's complement inverts each bit.) When the calculation is performed at the sender, 0's are put in the position of the checksum. At the receiver, the one's complement sum is performed over the data and the checksum; if there are no errors, the one's complement sum at the receiver should yield all 1's i.e., 2^L-1.

Again let's consider an example, where L equals 3. Let the data string be (100101). The two 3-bit words are (100) and (101). The one's complement sum of these words is (010). The checksum for this data is then the inverse of (010), which is (101); thus, the

transmitted string is (100101101). If there are no errors in transmission, the receiver performs the one's complement sum of (100), (101) and (101), which is (111).

Below, we analyze the protection a checksum provides against random bit errors and burst errors.

Let

L = Length of checksum

T = Total length of data being checked plus the length of the checksum

Assume data is padded with 0's so that T is an integral multiple of L

N=T/L = Number of L bit 'words'

S = Sum of all L bit words in the transmitted sequence, including the checksum

P_R = Probability of random bit error

2.2.1 Random Bit Errors

The receiver calculates the one's complement sum over the received sequence, including the checksum. It decides that the received sequence is error-free if the one's complement sum equals 2^L-1. Thus, an error will not be detected by the checksum if the error is such that the one's complement sum at the receiver is still 2^L-1.

Consider the case of one random bit error. Let us assume the error is in position i in some L bit word, where $0 \leq i \leq (L-1)$. If the error is such that a 0 is changed to a 1, the sum at the receiver increases by 2^i; if the error changes a 1 to a 0, the sum at the receiver decreases by 2^i. S is the sum of the error-free sequence. Thus, $S \pm 2^i$ is the sum of the errored sequence. In order for the error to go undetected at the receiver, we require: $(S \pm 2^i) \bmod (2^L-1) = 2^L-1$. Since S mod (2^L-1) equals 2^L-1, this implies that 2^i must be a multiple of 2^L-1, which is impossible. Thus, all single bit errors are detected.

Next, consider two bit errors. Assume the errors are in position i of a word, and in position j of a word, possibly the same word, where $0 \leq i, j \leq (L-1)$. In order for the errors to go undetected we need: $(S \pm 2^i \pm 2^j) \bmod (2^L-1) = 2^L-1$. If i = j, this equation can be satisfied, as long as one bit error is from 0 to 1, and the other is from 1 to 0. If $i \neq j$, we need $(\pm 2^i \pm 2^j)$ to be a multiple of 2^L-1. This is impossible for $0 \leq i, j \leq (L-1)$.

We conclude that the only double bit errors that go undetected are when the errors are in the same position in two different words, and the errors have opposite 'polarities'. There are L

choices for the position and $\binom{N}{2}$ ways to choose two different words. Thus, the probability of undetected error due to two random bit errors is:

$$L\binom{N}{2}\frac{1}{2}P_R^2(1-P_R)^{T-2} \qquad (2.15)$$

The 1/2 term is the probability that the two errors have opposite polarities.

The analysis for three bit errors is similar. For triple bit errors to go undetected, we need: $(S \pm 2^i \pm 2^j \pm 2^k) \bmod (2^L-1) = 2^L-1$ for $0 \le i, j, k \le (L-1)$. This is possible if two of the errors occur in position i of two different words and have the same polarity, and one error of the opposite polarity occurs in position (i+1)mod L, for $0 \le i \le L-1$. There are L choices for the position of the two errors of the same polarity, $\binom{N}{2}$ ways to choose two different words, and N possible words where the third error can occur. Thus, the probability of undetected error due to three random bit errors is: $L\,N\binom{N}{2}\frac{1}{4}P_R^3(1-P_R)^{T-3}$. The 1/4 term is the probability the bit errors have the necessary polarity.

In general, we will assume that terms due to three or more random bit errors are insignificant in calculating the probability of undetected error when a checksum is used.

2.2.2 Burst errors

Now let's look at the effectiveness of a checksum in detecting burst errors. The checksum can detect all bursts of length up to L-1 bits since such bursts cannot change the sum by a multiple of 2^L-1. If a longer burst error hits the data, it is reasonable to assume that the probability the error will go undetected by the checksum is 2^{-L}.

We see from the discussion above that in general a checksum is much worse at detecting errors than a CRC. Checksums can fail due to the occurrence of just two bit errors. There must be a minimum of four bit errors before an Extended Hamming code CRC fails to detect an error. The reason a checksum is used in some applications rather than a CRC is that a checksum is easier to implement in software.

2.3 IMPLICIT ERROR CHECKING

In the sections above, we described transmitting data along with a CRC or a checksum to detect errors in the data. There are likely to be many parameters associated with the data that we could use as further checks that the data has arrived at the correct destination with no errors. One example is the length of the data block; a second example is the address of

the node to which the data is being sent. One option is to explicitly send these parameters along with the data and compare the transmitted values to the values determined by the destination. The drawback to this is it increases the number of overhead bits that need to be transmitted. An alternative to explicitly sending these fields is to use the CRC or checksum to implicitly check on these parameters. Implicit error checking, which is only helpful in certain situations, is described below. We will specifically analyze the implicit error checking properties of a CRC.

We use the term pseudoheader to refer to the set of error detection fields that are implicitly checked by the CRC. At the transmitter, the CRC is calculated as if the correct values for the pseudoheader fields preceded the data block. At the receiver, the data that has been received is used to determine the value for these fields, and again the CRC is calculated as if these fields preceded the data. If data has been received incorrectly, causing a pseudoheader field to have a value at the destination that is different from its value at the source, then the CRC may detect the error. This provides us with some degree of error protection without any bits of overhead.

Consider the example of using the destination address to check that a block of data has arrived at the correct location. The destination address may be quite long in terms of number of bits, so it may not be desirable to explicitly transmit it along with the data. If it is not explicitly included, then it makes sense to use the CRC to implicitly check the address. If no errors occur in the block of data except that it ends up at the wrong receiver, then an implicit check of the destination address may detect the error.

Note that the effectiveness of including a field in a pseudoheader depends on what happens to the block of data at lower layers of the network. Again consider including the destination address in the pseudoheader at some layer, say layer N. Assume layer N-1 (where we assume layer N-1 is below layer N) divides the layer N PDU into two smaller PDUs. Assume one of the layer N-1 PDUs is misdelivered, and is interpreted at the wrong destination as comprising an entire layer N PDU. Having the destination address in the pseudoheader does not help detect the misdirection in this case. To see this, assume an L bit CRC is used to check on the layer N PDU, and assume the misdirected portion of the layer N PDU does not contain the CRC bits. Then L random bits will be interpreted as being the CRC, and the error will go undetected with probability 2^{-L}, regardless of whether or not the destination address is checked implicitly.

There are some error detection fields that never provide any benefit if included in a pseudoheader. For example, if the CRC checks on the whole block of data, it does not provide any benefit to have the CRC also implicitly check the data length. If the length of the data received is incorrect, then we see from equation 2.2 that the CRC calculated from the incorrect data will essentially be random. An L bit CRC will appear to be correct with probability 2^{-L} whether or not the data length is checked implicitly. Thus, the presence of the data length field in the pseudoheader has no effect. In general, it only makes sense to include a field in a pseudoheader if it is a field whose value can be determined by the receiver, and if the purpose of the field is to detect errors that do not cause the CRC to be random.

Including an error check implicitly does not provide as much protection against errors as including it explicitly. Consider the example of transmitting a block of data with an L bit Extended Hamming code CRC, where the destination address is included in a pseudoheader. Assume that the whole block of data is sent to the wrong destination. If the length of the destination address is longer than L bits, then it is possible that if the incorrect address differs from the correct address in at least four bits, the misdirected data will go undetected without there being any true bit errors (except for whatever error caused the data to be misdirected). If the length of the destination address is L bits or shorter, then at least one bit error must occur in transmission for the misdirection to go undetected, since the CRC detects all burst errors of length less than or equal to L. If the destination address is explicitly included and sent along with the data, then at least four actual bit errors must occur before the misdirection could go undetected. Thus, including a check field implicitly may not provide maximum protection, but it has almost no cost.

CHAPTER 3

ERROR DETECTION SCHEMES

3.1 OVERVIEW

Any real-world network system is subject to a variety of errors. The lowest level errors are due to physical properties of the network. For example, this includes bit errors on the data line, burst errors due to equipment malfunction, and lost data due to buffer overflow. Failure to detect an error at one layer of a network may lead to the error propagating up to the next highest layer. Error detection mechanisms may need to be implemented at one or more of the network layers in order to prevent errored data from being accepted as error-free by the destination. Undetected errors are inevitably going to occur in any system regardless of the number of error detection fields included. The purpose of the error detection scheme is to reduce the rate of undetected errors to a level acceptable to users. In this chapter we establish guidelines for designing effective and efficient error detection schemes.

In general, error detection is accomplished by transmitting redundant information along with the data; the receiver checks for inconsistencies between the received data and this redundant information. There are many fields that could potentially be transmitted along with the data to help detect errors. For example, transmitting the destination address along with the data helps detect data that has been sent to the wrong destination; transmitting a length field helps detect lost data. It may be tempting to include a large number of error detection fields along with the data since having a large number of check fields may give the impression that errors will almost certainly be detected. However, as we show in this chapter, many check fields are capable of detecting only certain types of errors. It is important that there be protection against all likely error types. The effectiveness of an error detection scheme is measured by the error scenario that results in the most undetected errors. Adding a lot of overhead to greatly decrease the undetected error rate of one

particular error scenario may not be very helpful if there is another scenario that yields a much higher rate of undetected error. In this chapter, we outline the design steps to follow so that with high probability the resulting error detection scheme provides adequate overall protection.

Another measure of an error detection scheme is the number of bits of overhead required by the scheme. Ideally, the amount of overhead should not be very large. However, when designing a scheme this issue is likely to be of secondary importance compared to the effectiveness of the scheme. One should also consider the complexity involved in implementing an error detection scheme. Again, this is not of prime importance, although it may be a more significant factor when designing protocols for more primitive systems, or for very high speed systems where processing time is critical.

Before designing an error detection scheme, it is important to enumerate the various errors that can be expected in the system and estimate the likelihood of their occurrence. It is important that the error detection scheme be robust. Since it is difficult to predict the precise characteristics of the underlying network, it is desirable to design a scheme that works well under a wide range of conditions. Thus, when estimating the frequency of occurrence of the various errors, reasonable worst case scenarios should be considered.

We want our algorithm for designing an error detection scheme to work for general networks. Our approach will be to look at three layers of a network, which we call layers N, N-1, and N-2, where layer N is the highest layer of the three. These three layers need not correspond to actual OSI layers. Rather, they are a natural way to divide up some of the basic functions common to most networks. We look at the various error detection options at each layer. For example, we will see that for some error types, we have the option to correct the error if we deal with it at one layer, but we don't have the correction option at another layer. Many of the design decisions depend on the underlying error characteristics of the network. Throughout this chapter, we assume we have control over the error detection scheme of all three layers.

In chapter four, we apply the general design approach presented in this chapter to ATM systems; we produce an error detection scheme that is more effective and more efficient than the error detection scheme originally proposed by the CCITT (referred to as AAL Type 3/4), but that is very similar to the new scheme that has been proposed by the CCITT (referred to as AAL Type 5). In chapter five, we apply the error detection design method to

TCP/IP systems. We see that the resulting scheme is similar to the actual scheme used in TCP/IP.

3.2 DESCRIPTION OF LAYERS

As stated above, our approach is to examine error detection at three layers of a network. In this section, we describe the basic functions of each of the three layers. Refer to figures 3.1 and 3.2.

The highest of the three layers is layer N. We call the PDU at layer N a message. A message is comprised of a data portion (the layer N SDU) and a control portion (the layer N PCI). (The PDU/SDU terminology was described in chapter one.) We assume that layer N operates end-to-end. Note that when we discuss ATM, we will see that the Convergence Sublayer corresponds to layer N, and the layer N PDU is called a frame. In TCP/IP, layer N corresponds to the TCP layer, and the layer N PDU is called a segment.

At the source, layer N passes the message down to layer N-1. Layer N-1 is responsible for dividing the message up into smaller units; it may also add control information to each of these units. We call the layer N-1 PDU a packet. The initial fragmentation of the message occurs at the source. However, further division of the packets into smaller packets might be permitted at intermediate nodes along the data path. At the destination, layer N-1 is responsible for merging the packets together to form a message, which is then passed up to layer N. We assume reconstruction of the message only occurs at the destination. In ATM, the Segmentation and Reassembly sublayer is analogous to layer N-1; the PDU is called a segment. In TCP/IP, the IP layer performs the duties we associate with layer N-1; the IP PDU is called a fragment.

We assume layer N-2 is responsible for routing the packets to the correct destination. (For simplicity, we refer to the layer N-2 PDU as a packet also.) It may add further control information to each packet in order to be able to perform the routing. Layer N-2 is involved with the routing at each node along the data path. The ATM layer performs the routing in ATM; the ATM layer PDU is called a cell. In TCP/IP systems, the IP layer performs the routing.

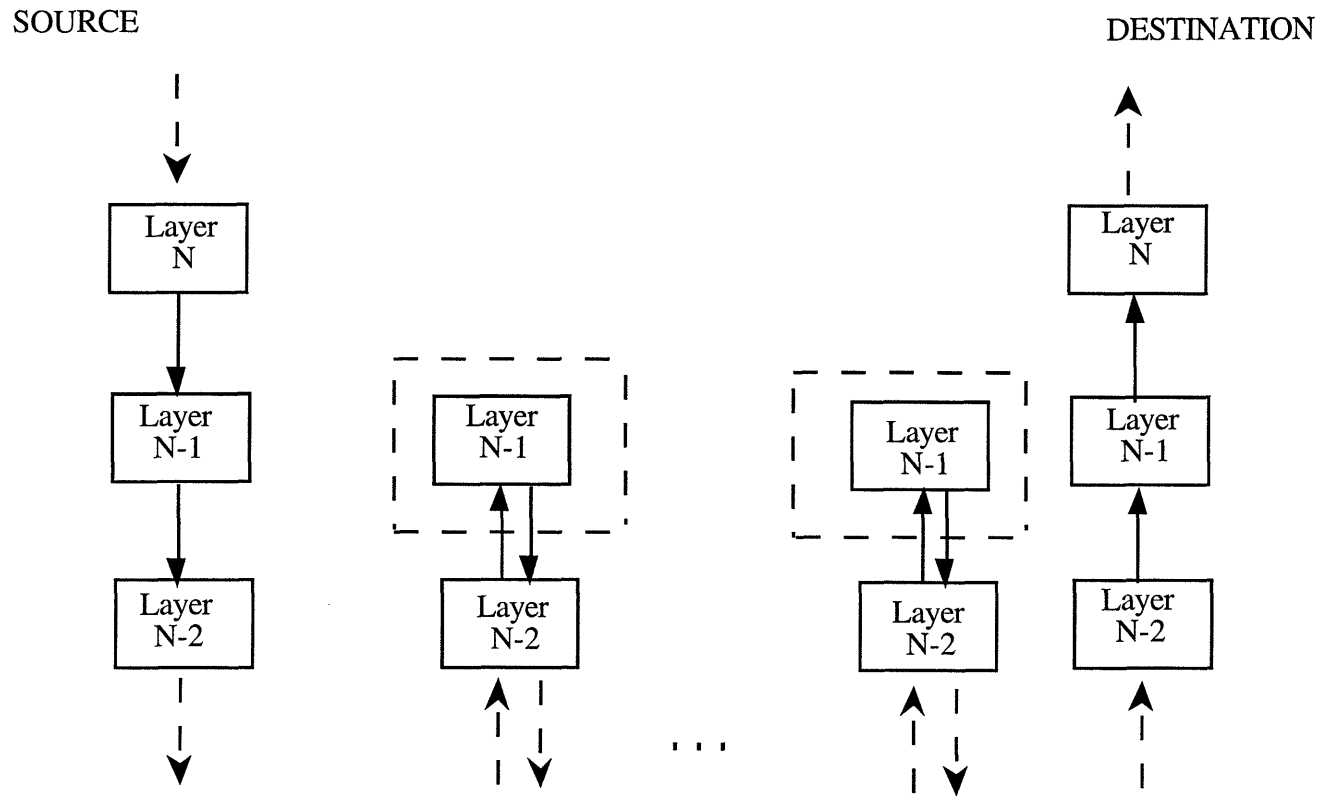


Figure 3.1 The layers of interest along the data path. Layer N-1 is shown in dotted lines at the intermediate nodes since it possibly may fragment the packets further at these nodes, but it does not put the packets together to form a message until the destination.

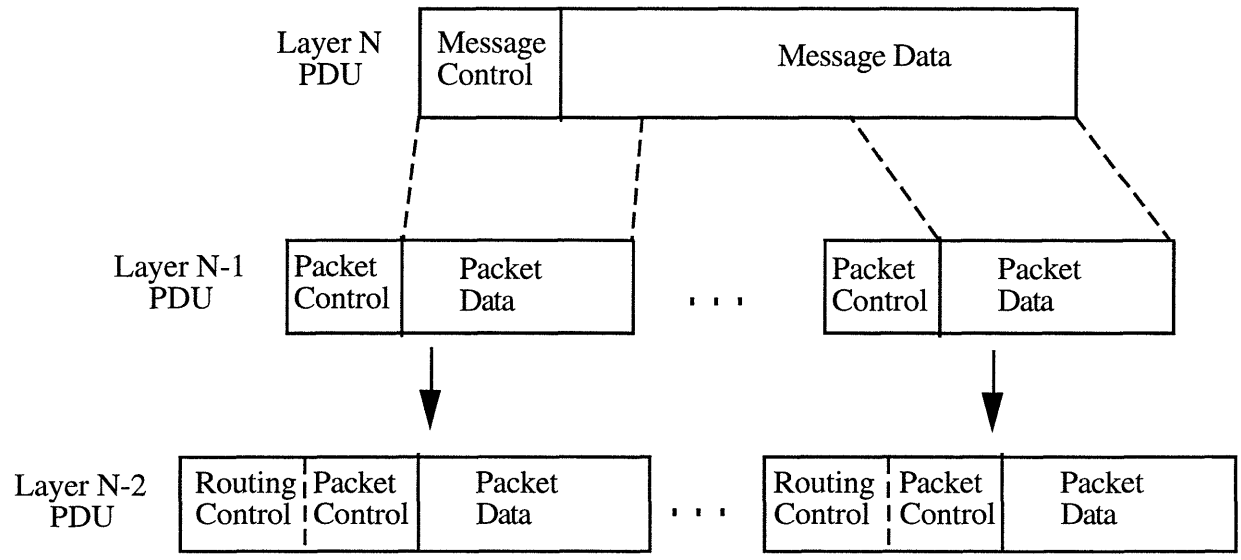


Figure 3.2 The PDUs at each of the three layers.

Error detection may take place at any of the three layers, but we will assume for the reasons below that recovery from errors is performed end-to-end at layer N. The unit of retransmission is a layer N SDU. Different control information may be added when the SDU is retransmitted so that we cannot consider the unit of retransmission to be the message.

Another option would have been to retransmit individual packets at layer N-2 or N-1 rather than the entire layer N SDU. This obviously would be a more efficient use of bandwidth since fewer packets would be retransmitted. However, it is very important to note that in typical systems, the layers that deal with packets do not know what type of data is contained in the packet. For example, if a packet contains video information, it is unlikely that it should be retransmitted if it contains errors. The lower layers of the network, however, would not be aware of this. Normally, the layer that deals with messages, i.e., layer N, is aware of which connections require retransmission of errored data. Thus, we see that end-to-end error recovery at layer N is a reasonable assumption. Also, this is what is implemented in ATM and TCP.

We assume that if errored data is not detected after being processed by layer N at the destination, then the receiver will end up accepting errored data as error free. Our goal is to have the combination of error detection techniques at all three layers result in a rate of undetected errors at layer N that is below some acceptable threshold. In an actual system, there may be layers above N or layers below N-2 that are also capable of detecting errors. However, we will assume that we cannot rely on layers other than N-2, N-1, and N to detect errors.

Below, we examine the various error scenarios that are relevant at each layer. We look at the lowest layer, and then work our way up. Figure 3.3 at the end of the section provides an overview of the errors at the various layers. Table 3.1 lists the specific error scenarios that need to be dealt with when designing an error detection scheme.

3.2.1 Error Detection at Layer N-2

We assume that at some layer below N-2, transmissions are subject to random bit errors and burst errors. Also, data can be lost. It is assumed that we have rough estimates of the probabilities of these various events.

The four types of errors that are pertinent at layer N-2 and their ramifications are as follows:

1) The packet addressing information has been corrupted due to bit errors or burst errors.

Since packets are routed by layer N-2, if an error in the packet address is not detected by layer N-2, the packet may arrive at the wrong destination. We refer to this type of error as misdirected data. It is important to note that layer N-2 does not have the option of letting a layer above it take responsibility for preventing the misdirection (although these higher layers may be able to detect that a packet has been sent to the wrong destination). If layer N-2 detects a corrupt address, it can either attempt to correct the error or it can drop the packet. These options are discussed in Section 3.4.1.

2) The packet data has been corrupted due to bit errors or burst errors.

As we will see, corrupted data is an error that can be dealt with at layer N-2, N-1, or N. In order to detect bit errors in the data some sort of redundancy check on the data needs to be added. If the errors are not detected at layer N-2, then layer N-2 passes a packet that has bit errors in it up to layer N-1 .

3) A link on which layer N-2 would like to route a packet is down.

A link failure may cause packets to be lost. For connection oriented traffic, we assume that if a link goes down, the connection is aborted and re-established along a different path. For connectionless traffic, we assume the packets are re-routed without the source and destination being notified about the link failure.

4) A buffer at an intermediate node is full.

If a buffer at an intermediate node is full, we assume packets are dropped. We assume the node does not send notification to the source or to the destination indicating which packets have been dropped.

3.2.2 Error Detection at Layer N-1

Next, we look at the possible errors at layer N-1. The design decisions at layer N-2 affect the likelihood of these various errors occurring. Recall that the responsibility of layer N-1 is to reconstruct the message at the destination. Undetected errors at layer N-1 result in an errored message being passed up to layer N.

1) Layer N-2 delivers to the destination a packet that does not belong there.

Layer N-1 can add control information to each packet to help detect this error. For instance, it could add an ID field to identify which packets comprise one message. If layer N-1 fails to detect the error, it delivers a message that has extra data in it to layer N.

2) Layer N-2 delivers a duplicate copy of a packet to the destination .

Layer N-1 can add a packet sequence number to help detect a duplicate packet. If layer N-1 fails to detect the error, it delivers a message that has extra data in it to layer N.

3) Layer N-2 fails to deliver a packet to the destination.

Layer N-1 can add a packet sequence number to help detect a lost packet. It could also add an ID field to identify packets belonging to one message, to help prevent the case where lost packets result in packets from multiple messages being merged together. (This results when the "message delimiting" packets are lost.)

4) Layer N-2 delivers a packet out-of-sequence. Note that this is not necessarily an error event; a datagram delivery service is expected to deliver packets out-of-sequence.

Layer N-1 can add a packet sequence number to help detect out-of-sequence packets. If layer N-1 fails to detect the error, it delivers to layer N a message with the correct number of bits, but the bits are scrambled.

5) Layer N-2 delivers a packet that has bit errors in the packet data to the destination .

In order to detect bit errors in the packet data some sort of redundancy check on the data needs to be added. If the bit errors are not detected, then layer N-1 passes a message that has bit errors in it up to layer N .

6) Layer N-2 delivers a packet that has errors in the control information to the destination.

We can distinguish two types of control fields. Some control fields aid the destination in reconstructing the message. For example, there is often a field that indicates which packet is the last packet of a message. If there is an error in this field, then the message may be reconstructed incorrectly. Redundancy checks on these types of fields would help detect such errors. The check fields themselves are the second type of control field. Bit errors in these fields may result in other error scenarios going undetected.

It is important to note that layer N-1 is capable of correcting some of the errors enumerated above. If layer N-1 detects an extra packet (either a misdirected packet or a duplicate packet) it can simply drop it. Or, if packets arrive out-of-sequence and some unique ordering scheme is included in the packet control information, then layer N-1 can reorder the packets correctly. In both of these cases, if layer N-1 does not fix the error, then it passes an errored message up to layer N. Layer N may be able to detect the error, but it likely will not be able to correct it. Thus, dealing with certain errors at layer N-1 rather than at layer N may decrease the number of messages that are dropped.

Another possibility is to use a data redundancy check such as a CRC to correct errors in the data. This is an option at any layer that includes a CRC. However, as shown in Section 3.4.2 below, this is unlikely to be implemented.

3.2.3 Error Detection at Layer N

Next, we look at the possible error scenarios at layer N. Again, the likelihood of these scenarios depends upon the error detection methods used at layers below layer N.

1) Layer N-1 delivers a message to layer N that has the correct number of bits, but contains at least one bit error.

2) Layer N-1 delivers a message to layer N that has too few bits.

3) Layer N-1 delivers a message to layer N that has too many bits.

4) Layer N-1 delivers a message to layer N, where the beginning of the message belongs to one message that was actually sent, and the end of the message belongs to another message that was actually sent.

5) Layer N-1 delivers a message to layer N that is correct except that the message was meant for a different destination.

If any of these errors is not caught by Layer N, then we have an undetected error event. Layer N has a few error detection options:

a) A redundancy check on the whole message would aid in detecting all of the scenarios.

b) A message length check would aid in detecting scenarios 2, 3, and possibly 4.

c) A message ID in the beginning and end of the message would help detect scenario 4.

d) Including the destination address as control information in the message would help detect scenario 5. Or, as discussed in Section 2.3, the destination address could be checked implicitly.

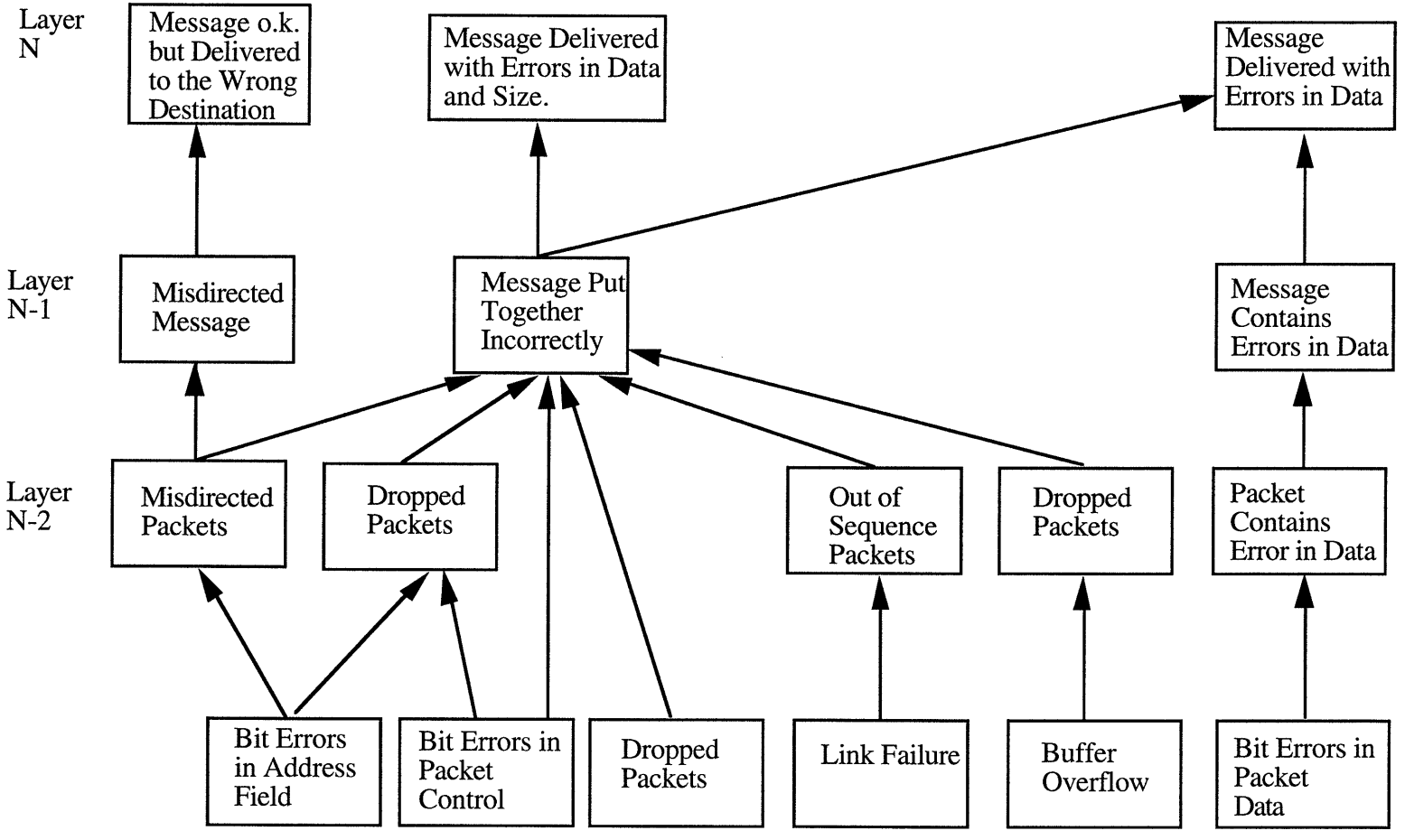


Figure 3.3 Diagram of errors at the three layers of concern. The errors shown at a given layer are the errors passed from that layer to the layer above it.

TABLE 3.1: List of Error Scenarios

1. Stray packets
   - Packets arrive at the wrong destination
   - Packets left over from an old connection
2. Out-of-sequence packets
3. Duplicate packets
4. Lost packets
5. Packets from more than one message are merged into one message
6. Bit errors in the data
7. Bit errors in the control information
   - Error in a flag
   - Error in a length field
   - Error in an ID field

3.3 DEGREE OF DIFFICULTY IN DETECTING AN ERROR

Before continuing our analysis, we define the notion of one error being easier to detect than another. Essentially, the greater the diversity of options that can be used to detect an error, the easier the error is to detect. Another way of looking at this is the more inconsistencies produced by an error, the easier it is to detect the error.

Consider the following example. Assume we have a system where the beginning packet of a message contains a BEGIN flag, and the final packet of a message contains an END flag. Assume that packets are fixed length and are expected to arrive in-sequence. Consider the following three scenarios involving an END packet arriving at the wrong destination.

a) Assume the stray END packet arrives immediately after an END packet that was meant for the destination. Thus, the destination receives an END packet without a corresponding BEGIN packet. In this scenario, the misdirection has occurred in such a way as to produce an invalid message. Thus, the error should be easy to detect.

b) Now assume the END packet arrives immediately after a BEGIN packet. This error is harder to detect than case (a) since a valid message is formed. Assume that the actual message meant for this destination was comprised of more than two packets, so that the message formed by the stray END packet is shorter than the message that was sent. The fact that there is a difference in length provides a means of detecting the error e.g., if the BEGIN packet contains a message length field, it could help detect the error.

c) In the third case, assume the misdirected END packet arrives immediately after a BEGIN packet, and that the actual message meant for this destination was comprised of two packets. The message will appear to be valid, and the message length will appear to be correct. Thus, there are less options for detecting the error than there are in cases (a) and (b). We can say that this scenario is the hardest of the three error scenarios to detect.

3.4 GUIDELINES FOR DESIGNING ERROR DETECTION SCHEMES

In this section we use the above discussion of the various error scenarios to provide guidelines for designing effective and efficient error detection schemes.

3.4.1 Step 1: Reduce Level of Misdirected Data

The first step should be to deal with the error that causes the most serious problems. Of all the errors diagrammed in Figure 3.3, misdirected data that arrives at the wrong destination can be considered to be the most serious error scenario. Misdirected data is a potential security threat whether or not it is detected. Thus, preventing data from being sent to the wrong destination is important (as opposed to just detecting the stray data after it's reached the incorrect destination). Since misdirection results in both lost data at the correct destination and extra data at the incorrect destination, this error scenario can potentially result in multiple undetected error events in addition to causing security problems. If the addressing error is caught and the packet dropped, then there is only lost data to deal with.

Of the 3 layers discussed, only layer N-2 is capable of preventing data from being sent to the wrong destination. In order to prevent misdirected data, a redundancy check on the address field should be included in the layer N-2 packet control information. For example, a CRC could be included. Each intermediate node along the data path would then check the value of the CRC before allowing the packet to be forwarded.

Assume an Extended Hamming code CRC is used. If the length of the address field is A, then we know from equation (2.3) that we should choose the length of the CRC, L, to

satisfy: $2^{L-1} - L \geq A+1$. Let P_R be the probability of random bit errors, and let P_B be the probability that the address field of a packet will be hit by a burst error. For a given L, we can use equations (2.4) and (2.10) to upper bound the probability of misdirected data by: $\binom{L+A}{3} \frac{1}{4} P_R^4 + P_B 2^{-L}$. If this probability of misdirected data is not small enough to satisfy the security concerns of network users, then a longer CRC could be used (if the term due to burst errors is the dominant term) or a more powerful check, such as a BCH CRC code, could be used.

From chapter two, we know that a CRC can also be used to correct single bit errors. Thus, rather than dropping a packet when a single bit error is detected, a node can correct the error. This reduces the number of dropped packets; at least two bit errors rather than one must occur for the packet to be dropped. Thus, using the CRC in the correction mode decreases the probability of dropped packets due to random bit errors in the address from about $(L+A)P_R$ to about $\binom{L+A}{2}P_R^2$. However, as mentioned in chapter two, using a CRC in the correction mode may result in more undetected errors. Using equations (2.12) and (2.13), we see that the probability of misdirected data increases to: $\binom{L+A}{3} P_R^3 + (L+A+1) P_B 2^{-L}$.

Obviously, using the CRC in the correction mode involves a tradeoff. If the increase in misdirected packets is greater than the decrease in lost packets, it does not make sense to use the correction option. Or, if the higher probability of misdirection is unacceptable to network users for security reasons, then the correction option should not be used. Also, it is important to consider whether the decrease in probability of dropped packets is significant. If there are other error scenarios that will result in a probability of dropped packets higher than $(L+A)P_R$, then the decrease will be insignificant, and the correction option should not be used. These design decisions obviously depend on the value of P_R and P_B.

This concludes our general discussion on the prevention of misdirected data. In chapter four, we discuss more elaborate schemes, in the context of ATM systems. Note that a CRC cannot totally eliminate the occurrence of misdirected data. Thus, our discussion below of detecting errors at the various layers also considers the problem of detecting misdirected packets.

3.4.2 Step 2: Add Check Fields to Detect Bit Errors in Data

The problems caused by the remaining errors can be considered to be equally serious. However, the remaining errors can be differentiated on the basis of how easy they are to detect. From the point of view of efficiency, the next logical step is to choose a mechanism to detect the error scenario that is most difficult to detect. The mechanism chosen to detect this error may also help detect other scenarios, but the converse is unlikely to be true.

Of all the errors shown in Figure 3.3, bit errors in the data is the most difficult error scenario to detect. The only inconsistency produced by this error is that the value of one or more data bits is incorrect; it does not affect the reconstruction of the message. All other error scenarios pass through the box in the diagram at layer N-1 entitled "Message Put Together Incorrectly". (Ignore the "Misdirected Message" box for now.) As discussed above, there are many checks that can be added to help detect errors that affect the reconstruction of the message (e.g., packet sequence numbers, packet IDs).

The packet data can be an arbitrary binary sequence. The only method of checking the validity of the data is to add a redundancy check at one of the three layers. (As mentioned previously, it is possible that the layers above layer N or below N-2 could perform error detection also. However, since we cannot be sure what form of error protection these other layers may employ, we assume that we cannot rely on these layers to detect errors.) We must choose a redundancy check that is powerful enough to reduce the level of undetected error due to bit errors in the data below the acceptable threshold, since no other error detection fields will help detect this error. We assume a CRC will be chosen for this purpose. The question remains at which layer or layers should the CRC be added. This is a very critical decision. A CRC at layer N that checks on the integrity of the whole message is capable of detecting most other error scenarios also, whereas a CRC at layer N-1 or N-2 that just checks on the integrity of the packet is not. Thus, this decision likely affects the mechanisms that will be chosen to detect other errors. Of course, it is possible to add a CRC at multiple layers. However, it is most efficient to implement a CRC at just one layer. Thus, given that we have control over the error detection schemes of layers N-2 through layer N, we should choose one of these three layers to perform the redundancy check.

Checking on the data at layer N-2 entails including a CRC with each packet and checking on the validity of the packet data at each node along the data path. If we choose to check on the data at layer N-1, then a CRC is added to each packet, but the validity of the data is only

checked at the destination. Finally, if we choose the layer N option, then a CRC is added to each message, and the data check is only performed at the destination.

The first issue is whether to check on the validity of the data at each node along the data path, or whether to check it on an end-to-end basis only. The advantage of checking the data on a node-by-node basis is that data that picks up an error along a data link can be dropped at the next node, rather than being sent all the way to the destination. Also, it is possible that some errors may be easier to detect if a check is performed node-by-node. For example, recall that an Extended Hamming code CRC can detect with certainty all single, double, and triple bit errors; however, it may fail to detect four or more bit errors. Thus, if picking up a bit error on each link is expected to be a common occurrence, it would be easier to detect the bit errors if the packets were checked at each node. The disadvantage of node-by-node checking is that the CRC has to be calculated at each intermediate node, which may add to the packet delay.

In general, the only reason to perform the node-by-node check would be if it is likely an error will be discovered on a link. We will assume that the data links are reliable enough not to warrant node-by-node checking. We will not consider the case where the links are very error-prone.

The next section compares performing the data check at layer N-1 versus layer N. In both cases, the check is performed on an end-to-end basis; the only difference is whether the CRC is included per-packet (layer N-1) or per-message (layer N). Recall that we assume retransmissions are done on a per-message basis. If retransmissions were done on a per-packet basis, then a per-packet redundancy check would allow the system to determine which packets need to be retransmitted. A per-message check necessitates the retransmission of the entire message even if just one packet in the message is lost or contains errors. (Of course, if errors are very bursty, then the entire message would need to be retransmitted anyway.) Since we assume retransmissions are done on a per-message basis, this issue does not affect whether we include the redundancy check at layer N-1 or layer N.

3.4.2.1 Per-Packet CRC vs. Per-Message CRC

We want to compare the effectiveness and efficiency of a per-packet CRC to that of a per-message CRC. Assume packets and messages are fixed length. Let M be the length of a message, and let K be the length of a packet. Assume M = N K.

From equation (2.3) we know that in general the length of a CRC, L, should satisfy $2^{L-1} - L > D + 1$, where D is the amount of data being checked by the CRC. For the per-packet CRC, the data to be checked has length K-L. Thus, the length of the per-packet CRC should satisfy: $2^{L-1} - L > (K-L) + 1$. We assume the minimum length CRC is used; thus, the per-packet CRC is roughly of length log K. (Logs are taken base 2.) Similarly, the per-message CRC checks data of length M-L. Again we assume the minimum length CRC is used; thus the per-message CRC is roughly of length log M.

3.4.2.1.1 *Overhead*

Let's get rough estimates for the percent overhead involved in these two options. We assume N is much smaller than K.

$$\text{Overhead with the per-packet CRC option} \approx \frac{\log K}{K}$$

$$\text{Overhead with the per-message CRC option} \approx \frac{\log M}{M} \approx \frac{\log K + \log N}{N\ K} \approx \frac{\log K}{NK}$$

Thus, the per-packet CRC requires roughly N times more overhead than the per-message CRC.

3.4.2.1.2 *Random Bit Errors*

Now let's compare the effectiveness of the two methods in detecting random bit errors. Assume the probability of random bit errors is P_R. As shown in chapter two, we can upper bound the probability of four random bit errors causing an undetected error by:

$$\binom{T}{3} \frac{1}{4} P_R^4 \qquad (3.1)$$

where T equals the length of the data being checked plus the length of the CRC. We assume that P_R is small enough that terms due to more than four bit errors are insignificant compared to this.

Since we assume we are using the minimum length CRCs, we are dealing with full-length codes. Thus, we actually know the precise probability of four random bit errors causing an

undetected error. However, the bound in (3.1) is valid, and is fairly tight, for full-length codes; using this bound simplifies our calculations and will not significantly affect the results below.

If a per-packet CRC is used, the probability a message will contain undetected random bit errors is upper bounded by:

$$N \binom{K}{3} \frac{1}{4} P_R^4 \approx N K^3 \frac{1}{24} P_R^4 \qquad (3.2)$$

The factor of N is necessary since the probability of a message containing an undetected error equals the probability that at least one of the packets contains an undetected error.

If a per-message CRC is used, the probability a message will contain undetected random bit errors is upper bounded by:

$$\binom{M}{3} \frac{1}{4} P_R^4 \approx M^3 \frac{1}{24} P_R^4 \approx N^3 K^3 \frac{1}{24} P_R^4 \qquad (3.3)$$

Thus, if we are comparing the effectiveness of a per-packet CRC and a per-message CRC where the length of the CRC is roughly log K and log M, respectively, then undetected bit errors due to random bit errors are roughly N^2 more likely with the per-message CRC.

However, as stated above, a per-message CRC of length log M will result in much less overhead than a per-packet CRC of length log K. Thus, for a fair comparison of the effectiveness of the two methods, we should examine the case where the length of the per-message CRC is increased. If we increase the length of the per-message CRC to roughly 2log M and use an Extended BCH code rather than an Extended Hamming code, then the minimum distance of the code is six. (BCH codes were discussed in Section 2.1.6.) Then, using equation (2.14), the probability of a message containing undetected random bit errors in the data can be upper bounded by:

$$\binom{M}{4} \frac{1}{15} P_R^6 \approx N^4 K^4 \frac{1}{360} P_R^6 \qquad (3.4)$$

If $(N^3 K P_R^2) < 15$, which is likely for reasonable values of these parameters, then this probability is smaller than the probability given by (3.2) for the per-packet CRC option.

Or, rather than using an Extended BCH code, an Extended Hamming code CRC of length greater than log M can be used to provide greater protection. If we increase the length of the per-message CRC while keeping the length of the message data the same, the result is a shortened code. As discussed in Section 2.1.2, if the degree of shortening is very high,

then the minimum distance of the code likely becomes five, so that at least five bit errors are necessary to cause an undetected error. Thus, if the length of the per-message CRC is increased, the probability of undetected errors will likely be significantly smaller than indicated in equation (3.3) (although it likely will not be as small as equation (3.4), where an Extended BCH code is used).

We conclude that if the probability of undetected error using a per-message CRC as given in equation (3.3) is not satisfactory, then the probability can be significantly reduced by increasing the length of the per-message CRC, and using either an Extended BCH code CRC or a shortened Extended Hamming code CRC. If random bit errors are a significant problem, and some design specification prohibits us from using a per-message CRC longer than log M, then the per-packet CRC would be the preferred option. Otherwise, as far as random bit errors are concerned, there is not a significant difference whether a per-packet CRC or a per-message CRC is used.

3.4.2.1.3 *Random Burst Errors*

In this section we compare the effectiveness of the two types of CRCs in detecting random burst errors. Let P_B represent the probability of a random burst error. As stated in the introduction of this chapter, it is important that the error detection scheme be robust. Since it is difficult to predict the precise characteristics of the underlying network, it is desirable to design a scheme that works well under a wide range of conditions. Our analysis should therefore focus on reasonable worst case scenarios. Thus, when looking at random burst errors, we are more concerned with the problem of detecting short burst errors rather than long burst errors. Long burst errors are likely to affect packet control information, and thus cause problems in reconstructing the message. Short burst errors are more likely to affect only the packet data and are thus harder to detect. Below we assume packets are hit randomly by a short burst error with probability P_B.

First, we note that a CRC of length L can detect with certainty burst errors of length less than or equal to L bits. Thus, since the per-message CRC is longer than the per-packet CRC, there are more burst errors that will be caught with certainty by the per-message CRC than the per-packet CRC. For burst errors longer than this but shorter than the length of a packet, we have:

If we use the per-packet CRC option, the probability a message will have an undetected error due to a burst error is:

$$N P_B 2^{-L} \approx N P_B 2^{-(\log K)} = \frac{N P_B}{K} \qquad (3.5)$$

If we use the per-message CRC option, the probability a message will have an undetected error due to a burst error is:

$$N P_B 2^{-L} \approx N P_B 2^{-(\log M)} = \frac{N P_B}{M} = \frac{P_B}{K} \qquad (3.6)$$

The per message CRC is N times more effective in detecting short burst errors. Also, as mentioned in the previous section, it is likely that a per-message CRC of length longer than log M would be used. Assume the length is about 2log M. Then the probability of undetected burst errors is:

$$N P_B 2^{-L} \approx N P_B 2^{-(2 \log M)} = \frac{N P_B}{M^2} = \frac{P_B}{N K^2} \qquad (3.7)$$

Thus, with this doubling of the CRC length, the per-message CRC option is N^2K more effective in detecting burst errors than the per-packet CRC option.

3.4.2.1.4 *Implementation*

We can also compare the two CRC methods in terms of difficulty of implementation. It is probably easier to implement the per-packet CRC. The CRC can be calculated as soon as the packet arrives. If a per-message CRC is used, the CRC can be fully calculated only after the whole message has arrived. If partial calculation of the per-message CRC is done as the packets arrive, then these partial results need to be stored.[DCS91] Packets from different connections are likely to be intermixed, so there will need to be storage for all active connections at the destination.

From the standpoint of implementation, the drawback of the per-packet CRC lies in its inability to detect several error scenarios. It can only check on the integrity of the packet, not the whole message. For example, a per-packet CRC can not detect the case of a lost packet. The limited error detection power of a per-packet CRC forces the need for additional error detection fields. These extra fields will have to be checked at the destination, which adds to the complexity of the scheme.

3.4.2.1.5 *Conclusion*

Above, we compared the performance of a per-packet CRC of length roughly log K, and a per-message CRC of length roughly log M. We showed that, given these minimum length CRCs, the per-packet CRC performs better in terms of detecting random bit errors and the per-message CRC performs better in terms of detecting burst errors. If the length of the

per-message CRC is doubled to 2log M, then the per-message CRC is greatly superior in detecting burst errors, and likely to be superior in detecting random bit errors. Even with this doubling of length, the per-message CRC option requires less overhead than the per-packet CRC option. The per-packet CRC is probably a little easier to implement than a per-message CRC. However, due to the inability of the per-packet CRC to detect certain types of errors, schemes that employ a per-packet CRC are likely to include many other error detection fields. Thus, the implementation of the overall scheme is likely to be equally complex.

We conclude that if random bit errors are a significant problem and we are forced to use a per-message CRC of length log M, then a per-packet CRC option should be used. Otherwise, a per-message CRC is preferable.

3.4.2.2 Correction of Errored Data

As discussed in chapter two, it is possible to use a CRC to correct single bit errors. The advantage of correction is that fewer messages are dropped. The drawback is that three or more bit errors may appear to be a single bit error and the error will be 'corrected' to the wrong thing. The tradeoffs are similar to what was discussed in Section 3.4.1, where we considered using the packet address CRC to correct errors. For example, if a per-message CRC is used and the length of the message is M (including an L bit CRC), then operating in the correction mode decreases the probability of dropped messages due to random bit errors in the data from $M \bullet P_R$ to $\binom{M}{2} P_R^2$. However, the probability of undetected error due to bit errors or burst errors in the data increases from $\binom{M}{3} \frac{1}{4} P_R^4 + P_B \, 2^{-L}$ to $\binom{M}{3} P_R^3 + (M + 1) P_B \, 2^{-L}$.

The first consideration is whether the decrease in message loss rate is significant. If there are other errors that cause the loss rate to be above $M \bullet P_R$ then using the CRC to correct errors has little impact on the overall message loss rate. The second consideration is whether the resulting increase in undetected errors is tolerable. If it is not, then an Extended BCH code CRC of length 2L can be used rather than an Extended Hamming code CRC of length L. The Extended BCH code CRC can be used to correct all single and double bit errors (see Section 2.1.6). If it is used in the correction mode, then the probability of dropped messages is reduced to $\binom{M}{3} P_R^3$. The probability of undetected error due to bit errors or burst errors in the data is: $\binom{M}{4} P_R^4 + (M+ 1) P_B \, 2^{-2L}$.

However, for applications where the integrity of the data is very important, it might be preferred that this extra L bits of overhead be used to further reduce the level of undetected error rather than to provide the ability to correct single and double bit errors (i.e., use the CRC of length 2L in the detection-only mode). Also, if it is not that important to have error-free data (e.g., a video application), then using a CRC to correct single or double bit errors would probably not be worthwhile.

We conclude that for applications that are extremely sensitive to undetected errors, or applications that are relatively insensitive to errors, using a CRC to correct errors does not make sense. Also, if the overall level of dropped messages is not significantly reduced by using the CRC to correct errors (which is likely to be the case if congestion losses are significant), then the correction option should not be used. Finally, if the increase in undetected error rate cannot be tolerated, and it is desired that the length of the CRC not be increased to 2L, then the correction option should not be used. Thus, using a CRC to correct bit errors is appropriate for only certain connections.

3.4.3 Step 3: Consider Additional Correction Options at Layer N-1

The next step is to look at the errors, other than bit errors in the packet data, that can be 'corrected' if dealt with at layer N-1. As stated in Section 3.2.2.1, stray packets, out-of-sequence packets, and duplicate packets are all scenarios that layer N-1 is capable of correcting: stray and duplicate packets can be dropped, and out-of-order packets can be resequenced. One needs an estimate of the likelihood of these scenarios to decide if the decrease in the number of retransmissions that will be accomplished by correcting these scenarios at layer N-1 justifies the overhead that must be added per packet to perform the corrections. In general, if layer N-1 does not handle these errors, it will pass up an errored message to layer N. Layer N may be able to detect the error, but it likely will not be able to correct the error. For example, if packets are put together in the wrong order, layer N will likely detect the error if it uses a per-message CRC, but it would not be capable of correctly resequencing the data. Thus, the message would be dropped. Also note that in general, layer N-2 is not capable of correcting errors involving reconstruction of the message since layer N-2 deals with packets on an individual basis.

Including the destination address or a message ID in the packet are ways of detecting stray packets. Once a stray packet is detected by layer N-1, it can simply be dropped. However,

if the method of preventing misdirected packets is effective enough, it may not be worthwhile to include extra check fields for the purpose of discarding stray packets.

If it is expected that a large number of packets will arrive at the destination out-of-sequence, then it makes sense to include a mechanism at layer N-1 to properly resequence the packets. Especially for a datagram system, where out-of-sequence packets are expected, it is very worthwhile to include some type of packet numbering scheme. Several methods of numbering packets are discussed in the next section. We assume that in addition to the numbering scheme, each packet also contains an ID field to identify which message it belongs to. The combination of the numbering scheme and message ID field can also be used to help identify duplicate packets.

3.4.3.1 Methods for Numbering Packets

3.4.3.1.1 *Fixed Size Packets*

First, we consider options for numbering packets when the packet size is fixed. The most straightforward method is to sequentially number the packets 1, 2, 3, etc. If there is a maximum of N packets per message, then the sequence number field should be $\lceil \log N \rceil$ bits long. In order for there to be resequencing errors, there must be errors in the sequence number of at least two packets. The sequence number field should also help detect lost packets.

Another option would be to sequentially number the packets modulo M, where M is less than N. This does not make sense for re-ordering packets since the ordering cannot be uniquely determined if a message contains more than M packets. Also, such a mechanism is unable to detect the case where a multiple of M consecutive packets is lost.

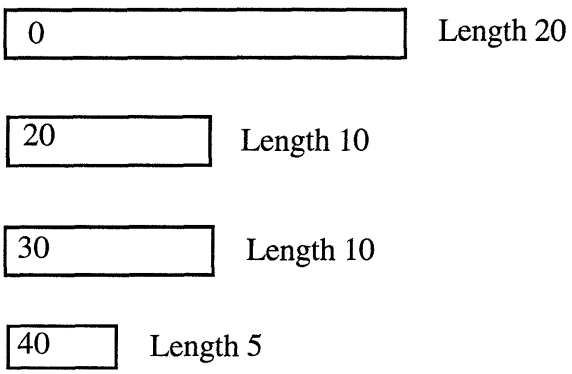
We conclude that the simple consecutive numbering scheme is the better option.

3.4.3.1.2 *Variable Length Packets*

In this section we consider the case where packets can be variable length, and where segmentation may occur at the intermediate nodes. Since all the segmentation does not occur at the source, we do not have the option of numbering the packets sequentially.

One option, which we refer to as Packet Offset Numbering, is to number the packets according to their offset position within the message. We assume each packet contains a field indicating the packet length, and the last packet in the message contains an END flag.

For example, if we have a message comprised of four packets, of length 20, 10, 10, and 5, then the packets will have sequence number 0, 20, 30, and 40, as shown below:



In the packet offset scheme a minimum of two packets must be in error before the order of two packets is interchanged in the message reconstruction. In the example above, if the second packet has an error that changes its sequence number to 30, and the third packet has an error that changes its sequence number to 20, then the errors would not be detected and the order of these two packets would be interchanged. This was possible because these two packets have the same length. Two packets that do not have the same length will not be interchanged unless there is also an error in their length fields.

Errors in the last packet can cause other types of faulty reconstruction. In the example above, if the last packet arrives before the third packet, and the last packet has an error that changes its sequence number to 30, then the last packet will be accepted as the third packet. Since the last packet contains an END flag, it will appear that the message is comprised of only three packets.

The sequence numbers help detect most lost packet scenarios. In order for the lost packet not to be detected, the next packet in the message would have to have an error in its offset and length fields in order to fill in the gap left by the lost packet. (There are other possibilities but they involve errors in more packets.) The sequence numbers, however, do not help detect the case where the last packet is dropped. However, there are other means of detecting such a scenario (e.g., the lack of an END packet).

The packet offset numbering scheme also helps detect errors in the packet length field of a packet. Assume the length of a packet has an error such that its value is increased by L. In order for the error to go undetected, the next packet in the message would have to have an

error such that its offset is increased by L and its length is decreased by L. (There are other possibilities but they involve errors in more packets.) Of course, it does not help detect the case where the last packet has a packet length error; a message length field would be needed to detect this case.

An alternative to packet offset numbering is a Tree Sequencing scheme. In a tree sequencing scheme, rather than using a sequential counter to number each packet, we use the location of the packet on a 'segmentation tree'. Consider splitting a packet into two smaller (not necessarily equally sized) packets. The original packet is referred to as the parent, and the two packets produced are the offspring. The segmentation field of the parent is passed down to both of the offspring. In addition, one of the offspring packets has a '0' appended to its segmentation field and the other offspring packet has a '1' appended to its field. The root of this tree is the original message, with an empty segmentation field. (Since the sequencing field will be variable length, there will also have to be a means of indicating the end of the field.) We assume that the length of each packet is explicitly included in the packet.

Consider an example. Assume the original message is divided into 2 packets, which we call packet A and packet B. Now, assume packet B is further divided into packets C and D. Finally, assume packet C is divided into packets E and F. This produces the tree shown in Figure 3.4.

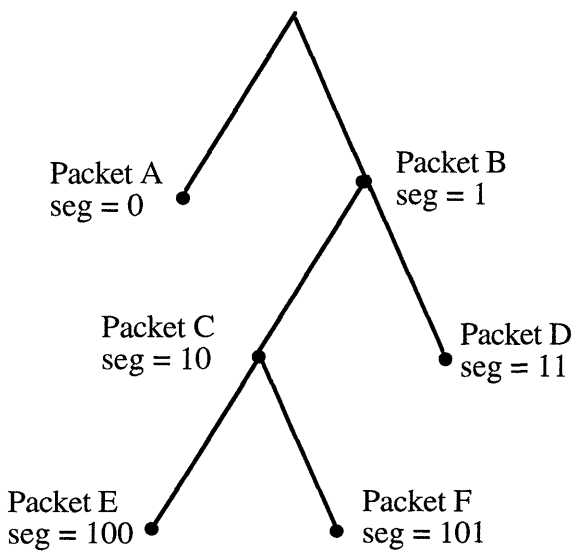


Figure 3.4 Packet segmentation fields in an example of a tree sequencing scheme.

The leaves of the tree, packets A, D, E, and F, are the packets that are expected to arrive at the destination.

Note that even though segmentation may occur at intermediate nodes along the data path, the segmentation field of the parent completely determines the segmentation field of the offspring. Thus, the sequencing can be performed in a distributed fashion.

All intermediate nodes in the segmentation tree (i.e., all nodes except the leaves) should have two children. The destination can check for this property to determine when it has received all packets belonging to the message.

In order to properly resequence the packets at the destination, we can use the following algorithm: at each step, the leaves at the deepest depth of the tree should be combined to form their parent, and then these leaves should be removed from the tree. Eventually we will end up with the root of the tree, which is the original message.

In this scheme, there must be errors in at least two packets before the order of two packets can be interchanged. However, assume a packet has an error in its segmentation field such that it contains the segmentation field of its parent. Assume this packet arrives before its sibling (or before any offspring of its sibling). Then, the tree will have been 'pruned' and too few packets will be used to reconstruct the message.

In order for a lost packet to go undetected, another packet must have an error in it. For instance, in the example above, if packet F is lost and packet E has an error in it that changes its segmentation field to '10', the error will not be detected. Also, with this scheme, errors in the packet length field are not detected.

Let's compare the packet offset scheme and the tree scheme. In general, in both schemes, the order of two packets can be interchanged only if both packets contain an error; thus, both schemes should perform well in resequencing packets. Also, in both schemes, in order for a lost packet to go undetected, there has to be an error in at least one other packet. The major performance difference between the schemes is that the tree method does not detect errors in the packet length field, whereas the packet offset method does. Also, the tree scheme is vulnerable to errors in the segmentation field that result in a pruned tree; the

packet offset method is vulnerable if the last packet in the message contains an error in its offset field.

The packet offset scheme requires more bits of overhead, although the more packets there are for a given size message, the less the difference in overhead per packet. It may be easier to perform resequencing when the packet offset is present, since the destination will know where the packet belongs within the message as soon as it arrives. Another difference is that the presence of the packet offset makes it easier for stray cells to be caught. If the packet lengths on the network are highly variable, then it is not likely a packet will have the precise packet offset value needed in order for it to 'fit in' with the wrong message.

Overall, there is probably a slight advantage to using the packet offset method. The tree scheme may require less overhead than the packet offset method, but it is not quite as robust and may be slightly more difficult to implement.

3.4.4 Step 4: Add Check Fields to Detect Remaining Errors

Refer back to the error diagram in Figure 3.3. Thus far we have dealt with preventing misdirected packets, detecting bit errors in the data, and possibly correcting out-of-sequence or stray packets. The next step is to deal with any remaining error scenarios that have not already been sufficiently handled by the error detection fields chosen in the above steps. Any of the other remaining errors involve problems in reconstructing a message. (The one exception is "Misdirected Message" which is discussed in the next section.) We can deal with these remaining scenarios at either layer N or N-1. In general, we do not have the option of detecting these scenarios at layer N-2, since this layer deals with packets on an individual basis; it does not view a packet in terms of belonging to a message. For example, consider the problem of detecting a lost packet. The only way a lost packet is detected is to view the packet as being one in a sequence of packets or as being one piece of a larger entity, and realize that the lost packet has resulted in a gap. If packets are looked at on strictly an individual basis, as they are in layer N-2, the absence of a packet will go unnoticed.

Error detection at layer N-1 usually involves adding control information to each packet. The effectiveness of this form of error detection greatly depends on the specific circumstances of the error scenario. For example, consider adding a sequence number to each packet to help detect lost packets. Assume packets are expected to travel in sequence.

If a message consists of ten packets, and it is the second packet in the message that is lost, then all eight packets after that in the message would have to have an error in the sequence number field in order for the lost packet to go undetected. However, if it is the last packet in the message that is lost, then the sequence number does not help at all in detecting the error. Thus, the effectiveness of adding error detection fields to each packet may be highly variable. In the best case, there must be a bit error in several packets before the error could go undetected. In the worst case, no bit errors are required for the error to go undetected, which likely necessitates the addition of other error detecting fields.

It would be very tedious to run through all the possible error detection fields that can be added at layer N-1, and analyze how each one performs for a particular error scenario. It makes more sense to look at such fields in the context of individual network systems. We will see specific examples of adding detection fields at layer N-1 in chapter four when we look at the error detection scheme originally proposed by the CCITT for ATM, and in chapter five when we examine error detection in TCP/IP.

Error detection at layer N usually involves a CRC that checks on the integrity of the whole message. A CRC of a given length (say L) is very consistent in its ability to detect errors. Errors involving the reconstruction of a message usually result in the data appearing to be completely random compared to what was originally sent or result in a random set of L bits being interpreted as the CRC. In both cases, we model the CRC as failing to detect the error with probability 2^{-L}.

Based on these observations, it is likely that error detection at layer N should be used. When examining error detection schemes, it is important to consider reasonable worst case scenarios. As we stated in the introduction of this chapter, it is much better to design a scheme that provides sufficient protection against all (or almost all) possible error scenarios than to design a scheme that provides enormous protection against some errors and little protection against others.

3.4.5 Step 5: Consider Single Packet Messages

One test of the effectiveness of an error detection scheme is the rate of undetected error for average sized messages. However, considering just average sized messages may not provide a good measure of the overall effectiveness of the scheme. As we show below, it is easier to detect packet control errors in multi-packet messages than in single packet

messages. Thus, more error protection may be needed to guard against undetected errors in single packet messages.

Consider the case of one packet of a multi-packet message having an error in its control information. It is likely that an inconsistency will occur when the destination attempts to join this packet together with the other packets in the message. Or, if a packet is misdirected, the fact that it is misdirected is more likely to be detected if the 'new' destination attempts to fit this stray packet together with packets that do belong at that destination. Essentially, errors in the control information of one packet of a message may be detected by the other packets in the message.

Single packet messages are more vulnerable to packet control errors. In order to protect single packet messages against these types of errors it is necessary to add information to the message (perhaps implicitly as described in Section 2.3) that can be verified based on the single packet. For example, including the destination address in the message would help detect the case of a single packet message being misdirected. This will be discussed further in the context of ATM and TCP/IP.

3.5 SUMMARY

We summarize the steps for designing an error detection scheme:

Step 1: Reduce Level of Misdirected Data. This usually takes the form of adding a CRC to check on the packet address field.

Step 2: Add Check Fields to Detect Bit Errors in the Data. In most circumstances, adding a CRC to check on the whole message is the most effective way of detecting errors in the data.

Step 3: Consider Additional Correction Options at Layer N-1. The most important consideration is whether to deal with resequencing out-of-sequence packets. We proposed several numbering schemes to accomplish this.

Step 4: Add Check Fields to Detect Remaining Errors. Using a CRC at layer N to detect errors rather than adding many fields at layer N-1 will likely provide more robust error detection.

Step 5: Consider Single Packet Messages. Packet control errors are more difficult to detect in single packet messages; thus more error detection may be needed to handle this special case.

We will apply these design steps to ATM systems in chapter four and TCP/IP systems in chapter five.

CHAPTER 4

ERROR DETECTION IN ATM

4.1 ATM OVERVIEW

Asynchronous Transfer Mode (ATM) is a network protocol currently being designed for use on Broadband Integrated Services Digital Network (B-ISDN) systems. ATM provides a common format for transmitting voice, data, and video over B-ISDN systems. These traffic types have very different requirements in terms of data rate, burstiness, and sensitivity to delay and error. To provide flexibility in handling this wide range of services, the unit of transfer in ATM is a very small, fixed length packet, known as a cell. Users are assigned cells on an as-needed basis. This allows a high degree of multiplexing, without the potential wasted bandwidth associated with synchronous multiplexing schemes such as Time Division Multiplexing (TDM).

There is a natural correspondence between the layers of ATM and the three generic layers discussed in chapter three. The ATM protocol is comprised of the ATM layer and the ATM Adaptation Layer (AAL). The ATM AAL is itself comprised of two sublayers: the Convergence Sublayer (CS) and the Segmentation and Reassembly Sublayer (SAR). The CS is the higher of the two sublayers and operates end-to-end. It corresponds to layer N described in chapter three, except that its PDU is called a frame rather than a message. The SAR layer, which also operates end-to-end, corresponds to Layer N-1. It is responsible for dividing frames into smaller data units at the transmitter; these data units are called segments. At the receiver, the SAR merges the segments together to reconstruct the frame. The ATM layer, which corresponds to layer N-2 and operates node-by-node, is below the SAR layer. It adds routing information to each segment to form the data unit referred to as a cell.

In ATM, we define an undetected error event as occurring when a frame that contains any type of error is accepted by the CS at the receiver as error-free. A realistic goal is to provide enough error checks so that the expected frequency of such an event on any given data line is no more than once per year. To provide some margin in achieving this goal, and to ensure a low rate of error even in the case of multiple lines feeding into a receiver, we use 10^{-3} as our desired maximum annual undetected error frequency per line. We assume that the data rate on a line is 150 Mb/sec.

There are four ATM service classes. We deal specifically with error detection schemes for Class C. This class of service is connection oriented and is characterized by a variable bit rate and no required timing between the source and the destination. Using the error detection design techniques of chapter three, we produce a scheme that is both effective and efficient. We propose to add five bytes of overhead to each frame, including one 34 bit frame CRC.

The CCITT has proposed two error detection schemes for Class C service. One proposal, referred to as AAL Type 3/4 [CCI92c], is quite cumbersome and is weak in detecting certain error scenarios. The other proposed scheme, referred to as AAL Type 5 [CCI92a], is similar to our proposed scheme.

In Section 4.2, we discuss the general properties of ATM networks. Only a few characteristics of ATM networks are relevant to our analysis; the specific details of the protocol are not important. In Section 4.3, we examine the probability of the lowest level errors in ATM networks. In Section 4.4, we step through the design algorithm of chapter three to produce an error detection scheme for ATM systems. This is followed by a discussion of some of the shortcomings of the error protection scheme included in AAL 3/4. Finally, we comment on Synchronous Optical Network (SONET), which serves as an interface between the ATM layer and the actual fiber optic link.

AAL 3/4 is also used for ATM class D traffic which is connectionless. We do not specifically address error detection issues related to connectionless traffic, although much of our analysis is applicable. The main differences are that routing cells and identifying cells that belong to a frame are handled differently for connectionless traffic, so that different mechanisms may be needed to handle routing and identification errors.

4.2 GENERAL PROPERTIES OF ATM NETWORKS

ATM frames can be variable in length, with the maximum length being 65,536 bytes.[Bel90] Frames are broken up at the transmitter into segments; the total length of the segment, including any control information, has been fixed by the CCITT to be 48 bytes. (See Figure 4.1.) A 5 byte header is added to each segment to form a 53 byte cell. The cell header contains a 24 bit virtual channel identifier/virtual path identifier (VCI/VPI) field that is used to route the cell to its destination. The cell header is the only portion of the cell that is examined at intermediate nodes in the network; the frame is only reconstructed at the destination. All cells contain a fixed amount of data, with the possible exception of the last cell in the frame. It is expected that all cells of a frame will follow the same path, and will arrive at the receiver in the same order in which they were sent. Duplicate cells are not expected to occur.

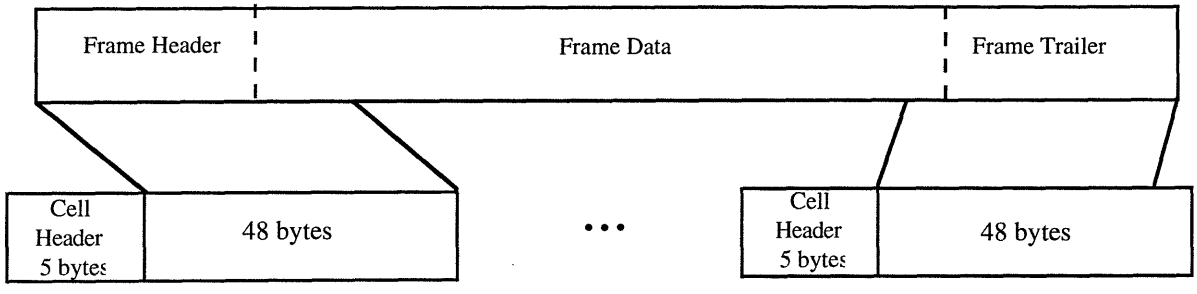


Figure 4.1 The frame, including the frame header and trailer, is partitioned into 48 byte units that are contained in each segment (the last segment of the frame may contain less than 48 bytes). A five byte header is added to each segment to form a cell. In the AAL 3/4 proposal, each segment contains a segment header and trailer, so that only 44 bytes of the frame can be carried in each segment.

There needs to be some method of indicating the last cell of a frame so that the frame can be reconstructed correctly at the destination. It seems natural to address this issue at the SAR layer since it involves the reconstruction of the frame. Indeed, in the proposed AAL 3/4 scheme, there are two bits in the segment that are used to indicate whether a cell is the first cell, one of the middle cells, or the last cell of the frame. As we will see, however, including these bits in the segment is not advantageous from the point of view of error detection. It is better if we include an END flag in the cell header, despite the fact that the ATM layer does not make use of the flag. We will see why this is true in a later section. Unless stated otherwise, we will assume in the analysis below that there is a one bit END flag in the cell header to indicate the last cell of a frame.

4.3 ERROR CHARACTERISTICS OF ATM NETWORKS

In this section we examine the lowest level error characteristics that are the result of physical properties of the network. We assume the ATM network is to be run over fiber optic lines. The three factors of concern are random bit errors, burst errors, and congestion, each of which is discussed below.

4.3.1 Random Bit Errors

We assume independent random bit errors occur on a fiber optic line with probability 10^{-8}. This is probably an overestimate of such bit errors by two or three orders of magnitude. However, our calculations show that even with this conservative estimate, random bit errors are not expected to be the dominant cause of most error scenarios in ATM systems. The notation P_R will be used to represent the probability of random bit errors.

As is noted in the next sub-section, when we consider burst errors, we make reasonable worst case assumptions as to which bits of the cell are actually affected by the burst. Thus, we cover the case where bit errors are correlated rather than independent.

4.3.2 Burst Errors

In one study of a fiber optic system, it was found that the chief cause of burst errors is protection switching.[DCS91] This occurs when a failed repeater causes the data to be switched from the original line to a protection line. During the switching process, the line is essentially open, resulting in a bit error rate (BER) of .5. The study showed that the mean time between these events is approximately four days, and each event results in error bursts of duration 20 to 40 msec. Assuming an average burst length of 30 msec., the fraction of time spent in such bursts is 9×10^{-8}. The notation P_B will be used to represent this probability of burst errors. At a data rate of 150 Mb/sec, about 10600 cells will be affected by a 30 msec. burst.

It is unlikely that this is the only type of burst error we need to consider. In calculating the probability of various error scenarios, we will use P_B as the probability of a cell being hit by a burst error, but we will generally make reasonable worst case assumptions as to which bits of the cell are actually affected by the burst. This should ensure that our proposed error detection scheme is robust.

4.3.3 Congestion

It is very difficult to estimate statistics on the expected congestion in ATM systems due to the highly variable nature of the traffic in the network. However, in general, the design objective is to limit the end-to-end cell loss rate to 10^{-6}.[DCS91] Therefore, we will use 10^{-6} as the probability a cell is dropped due to congestion. This probability will be denoted by P_C.

4.4 DESIGN OF ATM ERROR DETECTION SCHEME

Our goal in this section is to step through the design of an error detection scheme for ATM following the guidelines provided in the previous chapter. Our unit of measure for evaluating the error detection scheme is the expected number of frames, out of those transmitted on a line per year, for which errors are <u>not</u> detected by the combination of the ATM and AAL layers. Our desired maximum annual undetected error frequency per line is 10^{-3}. At a data rate of 150 Mb/sec, about 10^{13} cells can travel over a data line per year. For simplicity, we will assume there are N cells per frame and that about $\frac{10^{13}}{N}$ frames are transmitted per year, per line.

Our calculations assume the random bit error rate, P_R, equals 10^{-8}, the burst error rate, P_B, equals 9×10^{-8}, and the cell loss rate due to congestion, P_C, equals 10^{-6}. As stated earlier, we will generally try to look at reasonable worst case error scenarios.

4.4.1 Step 1: Reduce Level of Misdirected Data

When a connection is first established in an ATM network, all the intermediate nodes through which the cells will travel assign the connection a virtual channel number that is entered in the node's routing table. When a cell arrives at one of the nodes, the node checks the VCI/VPI field in the cell header. If the VCI/VPI value is found in the node's routing table, the cell is forwarded; otherwise, the cell is dropped. A cell will be misdirected if an error occurs in the VCI/VPI field, and the 'new' VCI/VPI matches another entry in the routing table.

Misdirection can be prevented by detecting errors that occur in the VCI/VPI field. Only the cell header is examined at the intermediate nodes; thus any error prevention mechanism must be included per cell, in the cell header. As we saw in Section 3.4.1, a reasonable option is to include a CRC in the cell header that checks on the contents of the header. The number of bits in the header, excluding the CRC, is 32. (There are flow control and

priority bits in addition to the VCI/VPI field.) From equation (2.3), in order for the CRC to be effective, the length of the CRC must be at least 7 bits. The CCITT has chosen the length to be 8 bits, which is a reasonable choice since it results in a cell header of size exactly 5 bytes. For the remainder of our analysis, we assume an 8 bit CRC is present in the cell header. We assume the CRC is an Extended Hamming code CRC, which is capable of detecting all single, double, and triple bit errors, and capable of correcting any single bit error. This provides several options for the operation of the intermediate nodes:

Detection-only option: The CRC is used only to detect errors; if an error is detected in the cell header, the cell is dropped.

Two-state correction/detection option: The default state is that the node uses the CRC to correct any single bit error it detects. The advantage of correcting a single error is that fewer cells will be dropped. The drawback is that three or more bit errors may appear to be a single bit error, in which case the cell header is 'corrected' to the wrong value. The possibility of this occurring is greatest when a burst error has occurred. To counteract this, once the intermediate node detects that a cell has an error in its header (even if it corrects the error), it goes into a detection-only state. It returns to the correction state only after it has received a cell that it perceives as having an error-free header. The state diagram is shown in Figure 4.2. The CCITT has chosen this two-state operation mode for its proposed standard.[CCI90]

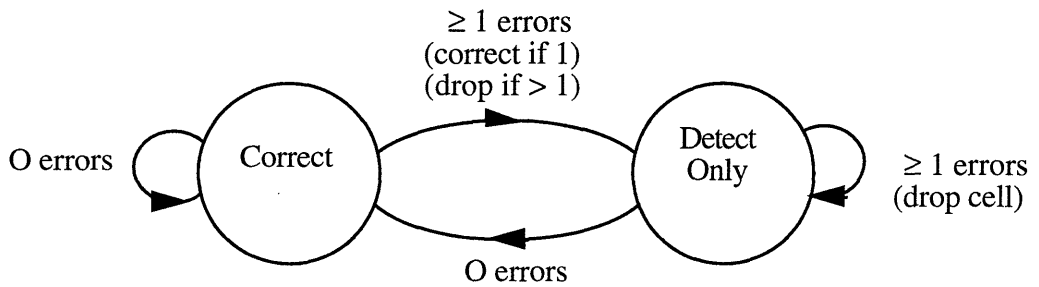


Figure 4.2 State Diagram for 2 state correction/detection option for CRC.

Four-state correction/detection option: This works similarly to the two-state option, except that after two or more errors have been detected in a cell header, or after errors have been detected in two consecutive cell headers, the intermediate node will drop all cells until it receives two consecutive cells that it perceives as having error-free headers. The rationale

for this is that during a burst error, it forces the CRC of two consecutive cells to fail before a cell is accepted. The state diagram is shown in Figure 4.3.

In the next sub-section, we examine the probability of misdirected cells due to burst errors and random bit errors, under these three modes of operation. We make the worst case assumption that all possible VCI/VPI values are in the nodes' routing tables; thus any undetected errors in the VCI/VPI field will cause the cell to be misdirected.

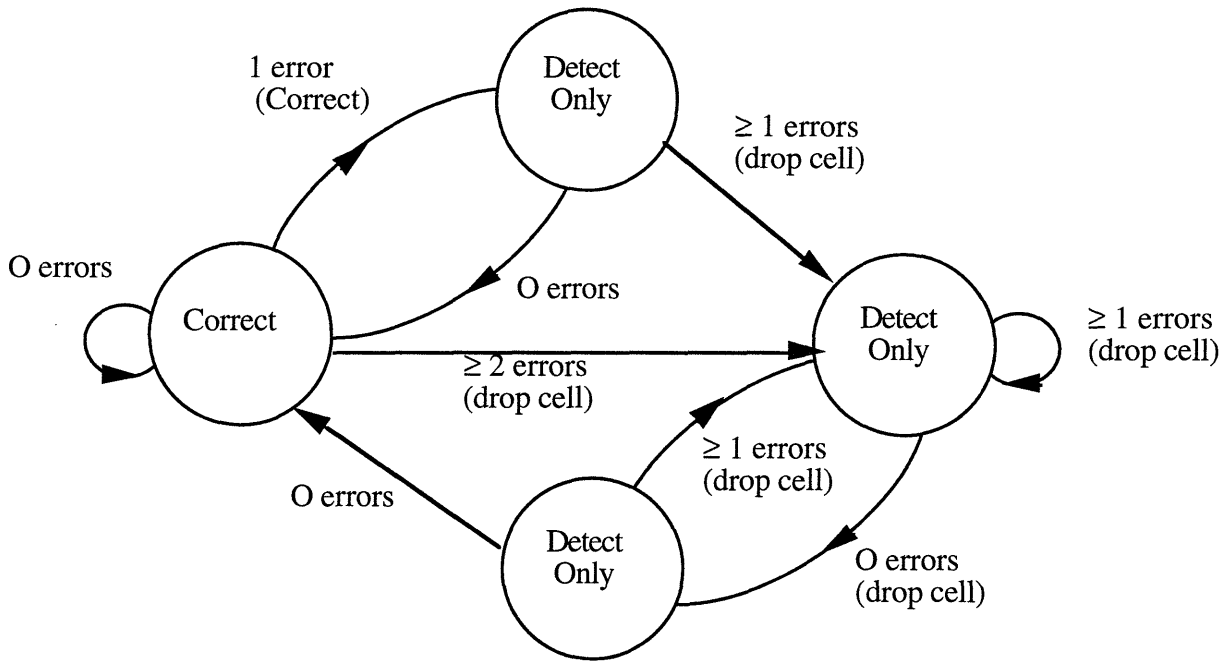


Figure 4.3 State diagram for 4 state correction/detection option for CRC.

4.4.1.1 Burst Errors

First we examine the effectiveness of the three options in detecting burst errors. Misdirection occurs when a cell is hit by a burst error and the CRC does not detect the error. We assume the entire cell header is hit by the burst, resulting in a completely random string. Recall that for a random string, a CRC fails with probability 2^{-L}, where L is the length of the CRC (L equals 8 in this case). We take P_B, the probability a cell is hit by a burst error, to be 9×10^{-8}. We expect about 10600 cells to be hit by the average error burst.

With the detection only option, the probability a cell will be misdirected due to a burst error is about: $P_B \, 2^{-8} = 3 \times 10^{-10}$.

In the two-state CRC option, when the first cell affected by the error burst arrives at the node, the node is likely to be in the correcting state. Using equation (2.13), the first cell in the error burst will be misdirected with probability $41 \cdot 2^{-8}$. After the first cell in the error burst, the node is likely to be in the detect-only state, so the other cells affected by the burst will be misdirected with probability 2^{-8}. Thus, the probability of any given cell being misdirected is about: $P_B\left(\frac{41 \cdot 2^{-8}}{10600} + 2^{-8}\right) \approx P_B\, 2^{-8} = 3\text{x}10^{-10}$.

In the four-state CRC option, the first cell affected by the burst is misdirected with probability $41 \cdot 2^{-8}$. The second cell in the burst is misdirected with probability $(41 \cdot 2^{-16} + 40 \cdot 2^{-16})$. The remaining cells affected by the burst are misdirected with probability about 2^{-16}. Thus, the probability of any given cell being misdirected is about:

$$P_B\left(\frac{41 \cdot 2^{-8}}{10600} + \frac{81 \cdot 2^{-16}}{10600} + 2^{-16}\right) \approx P_B\, 2^{-15} = 2\text{x}10^{-12} \qquad (4.1)$$

4.4.1.2 Random Bit Errors

Random bit errors are less likely to cause misdirected cells. Even in the correction state, where just three bit errors are capable of causing a misdirected cell, the probability can be approximated by:

$$\binom{40}{3} P_R{}^3 \approx 1\text{x}10^{-20} \qquad (4.2)$$

This is negligible compared to the misdirection probabilities due to burst errors.

4.4.1.3 Summary

We expect burst errors to be the chief cause of misdirected cells. The four-state CRC option is the most effective of the three methods in dealing with this error event. Compared with the other two options, the four-state option provides us with a factor of 2^{-7} benefit while incurring only a small penalty in term of complexity.

It is true that the four-state option results in a slightly higher rate of cell loss due to random bit errors than the two-state method, but the difference is insignificant. With the four-state method, a cell will be dropped if there are two or more bit errors in either its header or the previous cell's header, or if there is at least one bit error in both its header and the previous cell's header. The probability of this occurring is on the order of:

$$\binom{(2)(40)}{2} P_R{}^2 \approx 3\text{x}10^{-13} \qquad (4.3)$$

This probability of losing a cell due to random bit errors is negligible compared to the 10^{-6} probability of losing a cell due to congestion.

We will assume for the remainder of our analysis that an 8 bit CRC is present in the cell header, and operates in a four-state correction/detection mode. Thus, the probability of a cell being misdirected is $P_B\ 2^{-15} = 2x10^{-12}$. At a data rate of 150 Mb/sec, about 10^{13} cells can travel over a data line per year. Thus, roughly 20 misdirected cells are expected per year, per line. As will be discussed below, these misdirected cells will be detected with high probability.

4.4.2 Step 2: Add Check Fields to Detect Bit Errors in Data

As discussed in chapter three, the error scenario with the least number of options for error detection is that of bit errors in the data. The only means of detecting the error is to include a redundancy check on the data. Errors in the data can be the result of a burst error or random bit errors.

4.4.2.1 Burst Errors

In Section 4.3.2, we stated that burst errors in ATM are expected to be long; the average burst error is expected to affect about 10600 consecutive cells. However, as we stated in Section 3.4.2.1.3, short error bursts are more difficult to detect since they are likely to produce fewer inconsistencies. Since we can not be sure exactly what type of burst errors to expect, and since robustness is very important in designing an error detection scheme, we will make the worst case assumption that burst errors are short i.e., shorter than the length of a cell. In this section, we will also make the worst case assumption that the burst error does not affect any of the control information in the cell, such as the VCI/VPI field. With these assumptions, the probability a frame will contain data that has been corrupted by a burst error is about NP_B. Thus, the number of frames at the receiver that we expect to contain errored data due to a burst error is $\frac{10^{13}}{N} NP_B = 9x10^5$ per data line per year.

4.4.2.2 Random Bit Errors

The probability a frame of N cells contains a random bit error in the data portion is about $(N)(48)(8)P_R$. The expected number of frames at the receiver containing errored data due to random bit errors is then: $\frac{10^{13}}{N} (N)(48)(8)P_R \approx 4x10^7$, per data line per year.

We assume that we will use a CRC to detect errors in the data. We want a CRC that is powerful enough to reduce the expected annual frequency of frames with undetected bit errors in the data to no more than 10^{-3} per data line. In Section 3.4.2.1, we compared

adding the CRC at layer N-1 to adding the CRC at layer N. In ATM this corresponds to a comparison between a per-segment CRC and a per-frame CRC. In Section 3.4.4, we showed that in general adding the CRC at layer N is preferable; this turns out to be the case for ATM.

4.4.2.3 Per-Segment CRC

The overall size of a segment is 384 bits. Thus, the size of the per-segment CRC must be at least 10 bits in order to satisfy the inequality in (2.3). With a 10 bit per-segment CRC, the expected annual frequency of frames with undetected random bit errors in the data, per line, is about: (from equation (3.2)) $\frac{10^{13}}{N} N 384^3 \frac{1}{24} P_R^4 = 2x10^{-13}$. The expected annual frequency of frames with undetected bit errors in the data due to burst errors, per line, is about: (from equation (3.5)) $\frac{10^{13}}{N} NP_B 2^{-10} = 9x10^2$. Even if just 10% of the bursts are short, the expected number of undetected errors in the data due to burst errors is 90. This does not meet our goal of 10^{-3}. One option is to increase the length of the CRC. We would need to increase the length to 30 bits to meet our goal, which would add a lot of overhead to each segment. We realize, of course, that one reason for this seemingly poor performance of the per-segment CRC is that we are looking at the worst case burst error scenario. However, as we see below, even with this worst case assumption, a per-frame CRC can meet the goal of 10^{-3} without a lot of overhead.

4.4.2.4 Per-Frame CRC

The maximum size of a frame is 65,536 bytes (or 524,288 bits). Thus, the size of the per-frame CRC must be at least 21 bits in order to satisfy the inequality in (2.3). With a 21 bit per-frame CRC, the expected annual frequency of frames with undetected random bit errors in the data, per line, is about: (from equation (3.3)) $\frac{10^{13}}{N} N^3 384^3 \frac{1}{24} P_R^4$.

Letting N equal 1366, which is the maximum number of cells per frame, this frequency equals $4x10^{-7}$, which easily satisfies the goal of 10^{-3}.

The expected annual frequency of frames with undetected bit errors in the data due to burst errors, per line, is about: (from equation (3.6)) $\frac{10^{13}}{N} NP_B 2^{-21} = .4$.

This does not meet our goal of 10^{-3}. However, if we increase the length of the per-frame CRC to 30 bits, we can meet our goal. 30 bits per frame is still not a lot of overhead; thus, this option is feasible.

4.4.2.5 Summary

We conclude that a per-frame CRC of length at least 30 bits should be included as part of the ATM error detection scheme. (Note that this CRC is in addition to the 8 bit CRC in the cell header.) It allows us to meet our undetected error goal of 10^{-3}, even when worst case scenarios are considered. Thus, it should provide very robust error detection capabilities.

4.4.3 Step 3: Consider Correction Options at SAR Layer

Due to the presence of the 8 bit CRC in the cell header, misdirected cells are not expected to occur very frequently. Also, as we are dealing with connection oriented traffic, the cells are expected to arrive at the receiver in the same order in which they are transmitted. Duplicate cells are not expected to occur. Thus, adding special fields to the segment for the purpose of detecting stray or duplicate cells or re-sequencing out-of-order data is not worthwhile.

4.4.4 Step 4: Add Check Fields to Detect Remaining Errors

In this section, we consider the problem of detecting the remaining error scenarios: lost cells, errors in the END flag, and misdirected cells (in Section 4.4.1 we discussed reducing the likelihood of misdirected cells but we did not address the problem of detecting the error if it does occur). We assume a frame CRC of length at least 30 bits is present.

4.4.4.1 Lost Cells

Recall that we assume there is a one-bit flag in the cell header that indicates whether a cell is the last cell in the frame. This leads to two different lost cell scenarios. First, we look at the case where a non-END cell is lost, so that the frame has too few cells. Secondly, we look at the case where an END cell is lost, so that the cells of one frame are merged with the cells of a subsequent frame.

4.4.4.1.1 *Non-END Cell Lost*

We evaluate the number of frames we expect to arrive at the destination missing at least one non-END cell. We assume the END cell is received intact so that frames are not merged together. Congestion, burst errors, and random bit errors all can cause lost cells, but congestion is the dominant cause.

It is likely that congestion will occur in a burst and will result in entire frames being dropped; it is not likely to affect the non-END cells of a frame without affecting the END

cell. Nevertheless, we use the union bound, which shows that the probability of losing at least one non-END cell from a frame of N cells is at most $(N-1)P_C = (N-1)10^{-6}$.

As with congestion, we expect burst errors to affect entire frames. However, if we make the worst case assumption that burst errors are very short (i.e., shorter than the length of one cell) then the probability a frame will lose a non-END cell due to a burst error is about $(N-1)P_B = (N-1)9x10^{-8}$.

Random bit errors in the cell header may also cause a cell to be dropped. The probability of this occurring was approximated in equation (4.3). Thus, a frame loses a non-END cell due to random bit errors with probability $(N-1)\binom{80}{2}P_R^2 = (N-1)\ 3x10^{-13}$.

We conclude that the overall probability that a frame at the receiver is missing at least one of its non-END cells is about $(N-1)10^{-6}$, and the dominant cause is congestion. The expected annual frequency of this event per line is thus: $\frac{10^{13}}{N}(N-1)10^{-6} \approx 10^7$.

Next, we test the ability of the 30 bit CRC to detect this error event. We assume the CRC is in the last cell of a frame. If a non-END cell is lost, we see from equation (2.2) that the CRC calculated by the receiver will essentially be random bits. Thus, the CRC will fail to detect the lost cell with probability 2^{-30}. The expected annual frequency of frames with undetected lost non-END cells, per line, is then: $10^7\ 2^{-30} \approx 9x10^{-3}$. This does not quite meet our goal of 10^{-3}. Increasing the length of the CRC to 34 bits, however, does provide sufficient protection.

4.4.4.1.2 *End Cell Lost*

In general, if the END cells of X consecutive frames are lost, then the cells of as many as X+1 frames are merged together. We will consider the simplest case where X equals 1. Using the above assumptions, an END cell is lost with probability: $P_C + P_B + \binom{80}{2}P_R^2 \approx 10^{-6}$. The expected annual frequency of frames per line losing the END cell is then: $\frac{10^{13}}{N}\ 10^{-6}$. The CRC of the latter of the two frames will be used to check the merged frame (assuming the CRC is contained in the last cell of a frame). It essentially will be checking random data, so a frame CRC of length L will fail to detect the error with probability 2^{-L}. In the worst case, when N is 2 (if N is 1 the whole frame is lost), the length of the frame CRC must be at least 32 bits to meet our goal of 10^{-3}.

Instead of relying solely on the CRC to detect lost cells, we could consider adding a frame length field. However, a frame length field does not detect all lost cell scenarios. Assume that the length field is placed in the last cell of the frame. Then, if the beginning of frame A is merged with the end of frame B, and the resulting merged frame contains the same number of cells as frame B originally contained, then the length field will not help detect the error. We can derive an upper bound for the probability of this occurring as follows. Assume frame A originally contains M cells and frame B originally contains N cells. Assume a burst of congestion hits frames A and B such that the last cell of frame A is lost, and with probability $\frac{1}{\text{M-1}}$ the number of remaining cells in A is i, where i ranges from 1 to M-1, and with probability $\frac{1}{\text{N}}$ the number of cells in B is j, where j ranges from 1 to N (if j equals N then frame B is unaffected by the congestion). We assume the last cell of B is not lost, so that frames A and B are merged together to form a frame. If $\text{M} \leq \text{N}$, then the merged frame will contain exactly N cells if there are i cells remaining in frame A and N-i cells remaining in frame B, for $1 \leq i \leq \text{M-1}$. If $\text{M} > \text{N}$, then the merged frame will contain N cells if there are j cells remaining in frame B and N-j cells remaining in frame A, for $1 \leq j \leq \text{N-1}$. Thus, the probability the merged frame will contain precisely N cells is:

$$\text{if M} \leq \text{N: } (\text{M-1})\left(\frac{1}{\text{N}}\,\frac{1}{\text{M-1}}\right) = \frac{1}{\text{N}} \qquad \text{if M} > \text{N: } (\text{N-1})\left(\frac{1}{\text{N}}\,\frac{1}{\text{M-1}}\right) < \frac{1}{\text{N}}$$

Thus, we can upper bound this probability by $\frac{1}{\text{N}}$. This represents the approximate fraction of lost END cell scenarios that can not be detected by a frame length field.

4.4.4.2 Errors in the END Flag

Next, we consider the scenario where a cell arrives at the correct destination but contains an error in its END flag field. The END flag is a one bit flag in the cell header and is thus protected by the cell header CRC. Gaining the protection of the cell header CRC is the major reason we prefer to include the flag as part of the cell header rather than as part of the segment, despite the fact that the flag is used at the SAR layer and not at the ATM layer.

In order for an error in the END flag to go undetected, the cell header CRC must fail to detect the error. At least three bit errors must occur before the error will go undetected by the CRC, assuming the CRC is in the correction mode. (It is not necessary that one of the three bit errors be in the END flag; the node could make a false 'correction' that results in

an errored END flag.) Thus, the probability the error occurs due to random bit errors and is not caught by the cell header CRC is upper bounded by: $\binom{40}{3} P_R^3 \approx 10^{-20}$.

A burst error hitting the cell header could also cause an error in the END flag. We make the worst case assumption that the address field in the header is unaffected by the burst error so that the cell is not misdirected. Assuming the CRC is in the correction mode, the probability the error occurs due to a burst error and is not caught by the cell header CRC is about: $P_B\ 41\ 2^{-8} = 10^{-8}$. Thus, burst errors are the dominant cause.

First consider the scenario where an END flag is changed to a non-END flag. The errored frame will be merged with the following frame, and the CRC of the next frame will be used to check the resulting frame. It will essentially be checking random bits, and, assuming it is 30 bits long, will fail to detect the error with probability 2^{-30}. Thus, the expected annual frequency of frames with undetected END cell to non-END cell transitions, per line, is: $\frac{10^{13}}{N} 10^{-8}\ 2^{-30}$. This equals 9×10^{-5} for the worst case where N equals 1.

If a non-END cell is changed into an END cell, then the frame is essentially split into two frames. Random bits in the 'false' END cell will be interpreted as the frame CRC for the 'first' frame, and thus will appear to be correct with probability 2^{-30}. The frame CRC in the true END cell will only be checking the latter half of the original frame. Thus, this CRC will also fail with probability about 2^{-30}. Thus, overall, the expected annual frequency of frames with undetected non-END cell to END cell transitions per line is: $\frac{10^{13}}{N}(N-1)\ 10^{-8}\ (2)2^{-30} \approx 2 \times 10^{-4}$.

For either scenario we meet our goal of 10^{-3}.

4.4.4.3 Misdirected Cells

Lastly, we consider the error scenario where a frame contains a stray cell. As shown in Section 4.4.1, the most probable cause of a misdirected cell is a burst error. Due to the presence of the cell header CRC in the four state correction/detection mode, the probability of a cell being misdirected is $P_B\ 2^{-15}$. We make the worst case assumption that every misdirected cell results in one errored frame at the incorrect destination. (Of course, a misdirected cell also results in an errored frame at the correct destination, but we already discussed the lost cell scenario in a previous section.)

If a frame contains a stray non-END cell, the frame CRC will essentially be checking random bits. A 30 bit CRC will fail to detect the error with probability 2^{-30}. The expected annual frequency of frames containing undetected stray non-END cells, per line, is: $10^{13}\ P_B\ 2^{-15}\ 2^{-1}\ 2^{-30} = 1x10^{-8}$. The 2^{-1} term is the probability that a stray cell will not have its END flag set after its header is hit by a burst error.

If a frame contains a stray END cell, then the frame is essentially split into two frames. The frame CRC in the stray END cell will be used to check the 'first' frame. The frame CRC in the true END cell will check the 'second' frame. Either CRC will fail with probability about 2^{-30}. Thus, overall, the expected annual frequency of frames containing undetected stray END cells, per line, is: $10^{13}\ P_B\ 2^{-15}\ 2^{-1}(2)\ 2^{-30} = 3x10^{-8}$.

For either scenario we meet our goal of 10^{-3}.

Consider adding a frame length field to the end of a frame to help detect the scenario where a stray cell is accepted as part of a frame at the wrong destination. Assume the stray cell is really an END cell of a frame that has N cells. If it arrives at the wrong destination such that it is accepted as the N^{th} and final cell of a frame, then the length field in the stray cell would fail to detect this scenario (assuming it was not affected by the error that caused the misdirection). We can approximate the likelihood of this event as follows. Assume the misdirected cell belongs to a connection where all frames are comprised of N cells. Thus, with probability 1/N the stray cell is an END cell; assume the burst error does not affect the END flag. Assume it is misdirected to a destination where the frames are comprised of M cells. There is a 1/M chance that the stray cell will arrive before the i^{th} cell of a frame, for $1 \leq i \leq M$. If $N \leq M$, then the stray END cell will be accepted as the N^{th} cell with probability 1/M. If $N > M$, then it can't be accepted as the N^{th} cell. Thus, with these assumptions, the fraction of misdirected cell scenarios that can't be detected by a frame length field can be loosely upper bounded by $\frac{1}{MN}$.

4.4.4.4 Summary

In summary, we see that a 30 bit frame CRC provides sufficient protection against misdirected cells and errors in the END flag. However, a 34 bit frame CRC is needed to provide sufficient protection against lost cells. This a only a small increase in overhead.

We conclude that a 34 bit frame CRC should be included in the error detection scheme for ATM.

4.4.5 Step 5: Consider Single Cell Frames

Finally, assume a frame is comprised of just a single cell, and assume the header of this cell is hit by a burst error and misdirected. We make the worst case assumption that only the cell header is affected by the burst error; the remainder of the cell is intact. Assume that the cell arrives at the wrong destination immediately after an END cell, and that its own END flag is still set. The receiver will interpret it as being a single cell frame. The frame CRC does not help detect that this frame has been misdirected since the frame information is intact (note that the frame CRC does not check on the cell header). Thus, the frame will be accepted at the wrong destination. In the worst case, every frame is a single cell frame; in this case, the expected annual frequency of this event per line is $10^{13} P_B 2^{-15} \approx 20$. Obviously, not all frames will be single cell frames. However, it is worth noting that control frames (e.g., the frames used to set up calls) often consist of just one cell.

To add greater protection against this error, we can implicitly include the destination address in the frame when calculating the frame CRC. At the transmitter, the frame CRC is calculated as if the address of the desired destination preceded the frame data. At the receiver, the frame CRC is calculated as if the address of the receiver preceded the frame data. If the destination address is less than or equal to 34 bits (the size of the frame CRC), then the error will be detected with certainty. If the address is longer, the frame CRC will fail to detect the error with probability 2^{-34}. (We assume that the incorrect destination address is uncorrelated with the correct destination address; thus, the effect of the misdirection is similar to a burst error hitting the destination field.) Thus, the expected annual frequency of undetected misdirected single cell frames, per line, can be upper bounded by: $10^{13} P_B 2^{-15} 2^{-34} = 2 \times 10^{-9}$. This bound is also valid for the case where the burst error that causes the misdirection carries over into the frame data.

4.5 SUMMARY OF ATM ERROR DETECTION SCHEME

From the discussion above, we conclude that the error detection scheme should consist of:

8 bit cell header CRC in four-state correction/detection mode

34 bit frame CRC

Destination address implicitly checked by the frame CRC

The performance of this scheme is summarized in Table 4.1. We arbitrarily chose N to be 10 for those frequencies that depend on the number of cells per frame. The contrast with AAL 5 is discussed in section 4.7.

TABLE 4.1 **Expected Annual Freq. of Undetected Error**

Error Type	Chief Cause	Expected Annual Freq of Occurrence	Our Scheme	AAL 5
Bit Errors in Data	Burst error	$9x10^5$	$5x10^{-5}$	$2x10^{-4}$
	Random bit errors	$4x10^7$	$2x10^{-11}$	$2x10^{-11}$
Lost non-END Cell	Congestion	10^7	$6x10^{-4}$	$2x10^{-11}$
Lost END Cell	Congestion			
Length Change		$1x10^6$	$6x10^{-5}$	$2x10^{-12}$
No Length Change		$1x10^5$	$6x10^{-6}$	$2x10^{-5}$
Error in END Flag	Burst Error	$5x10^5$	10^{-5}	$7x10^{-10}$
Misdirected Cell	Burst Error			
Length Change		20	$2x10^{-9}$	$4x10^{-14}$
No Length Change		.2	$2x10^{-11}$	$5x10^{-11}$

4.6 IMPLEMENTATION

We assume an Extended Hamming code CRC is used for the frame CRC. The 34 bit frame CRC should be placed at the very end of the last cell of the frame. The last cell or both the last cell and the second to last cell of the frame may contain less than a complete 48 bytes of information. Thus, there needs to be a pad length field immediately preceding the frame CRC to indicate the number of bytes between the end of the frame data and the beginning of this pad length field. The pad length field should be 6 bits long since the pad length is no longer than the length of one segment (i.e., 48 bytes). The frame format is shown in Figure 4.4 below.

Figure 4.4 Format of frame with one per-frame CRC.

The frame CRC should be calculated over the entire frame, including the pad field and the pad length field. This ensures that up to three bit errors in the pad length field are caught

with certainty, assuming there are no other errors in the frame. Also, as discussed in Section 4.4.5, the destination address should be included as a pseudoheader.

4.7 ALTERNATIVE SCHEMES

In the scheme proposed above, we rely on the frame CRC to detect most errors. One drawback to relying solely on a frame CRC is that if the CRC fails to detect a congestion loss, then an undetected error event occurs without there being any type of bit error. A variety of fields could be added to reduce the probability of such an event. However, as we discuss below, even with the addition of these fields, we cannot totally eliminate the possibility of this occurring.

Also, the analysis presented above largely depends on our estimates of the underlying errors in ATM systems. Our estimates of random bit errors and burst errors are probably conservative. However, we are unsure of whether 10^{-6} is a realistic estimate of the cell loss rate due to congestion; thus, it may be desirable to provide greater protection against cell loss. One option is to increase the length of the frame CRC. However, this does not solve the problem of an undetected error occurring without there being any type of bit error.

An alternative is to add other fields to detect the scenarios that involve cell loss. For example, we can take advantage of the fact that most scenarios involving cell loss also result in the length of the frame being changed. Thus, we can replace the pad length field by a 16 bit frame length field, and reduce the frame CRC to 32 bits. (The CRC needed to be 34 bits long to protect against lost cells; however, with the addition of the frame length field, the CRC can be reduced to 32 bits.) This increases the amount of overhead by 8 bits per frame. Note that the CCITT AAL 5 error detection scheme consists of a 16 bit frame length field and a 32 bit frame CRC. The performance of this scheme was shown in Table 4.1.

The overall expected annual frequency of undetected error per line in our proposed scheme and in AAL 5 is approximately the same (7×10^{-4} vs. 2×10^{-4}). The chief cause of undetected error in our scheme is congestion resulting in non-END cells being dropped. Recall that when calculating the frequency of this error, we assumed cells are lost independently due to congestion, which is an extreme worst case assumption. The chief cause of undetected error in AAL 5 is burst errors resulting in bit errors in the data. In calculating the frequency of this error, we assumed cells are hit independently by short

burst errors, which is also an unlikely assumption. Thus, we cannot state definitively which scheme performs better since the performance is tied to the precise nature of the congestion and burst errors.

Note that although AAL 5 provides greater protection against the lost cell scenarios where a length change is involved, it provides less protection against the lost cell scenario where a length change is not involved (i.e., the merged frame scenario discussed in Section 4.4.4.1.2). Only the frame CRC, which is shorter by 2 bits in AAL 5, provides protection against this latter scenario. Thus, this scheme is really not a safeguard against increases in the congestion rate.

One way to provide greater protection against the merged frame scenario is to include a frame ID field in both the first and last cells of a frame. Assume the length of the frame ID field is F bits. Under most circumstances, there would have to be a bit error in one of the ID fields before a merged frame could go undetected. However, if frames A and B are separated by 2^F frames (i.e., they have the same frame ID), and congestion hits resulting in frames A and B being merged, then the frame ID field does not help detect the error. Also, if the merged frame contains the same number of cells that frame B originally contained, then the frame length field does not help detect the error. Again, we have the situation where we totally rely on the frame CRC to detect the error. Thus, we cannot totally eliminate scenarios where an undetected error event occurs without there being a bit error (unless the frame ID field is large enough that it never wraps around).

Note that the frame length field and frame ID fields do not help detect bit errors in the data. Only the frame CRC helps detect this error. Thus, the extra bits in these alternate schemes could be added to the length of the CRC to decrease the frequency of all undetected errors.

4.7.1 AAL 3/4 Proposal

Next, we summarize the error protection scheme originally proposed by the CCITT as part of AAL Type 3/4. Figure 4.5 shows the format of the frame, segment, and cell in this proposal. A 32 bit frame header and frame trailer is added to each frame. Also, each segment contains a header and a trailer as part of the 48 bytes of the segment. There is also a 5 byte cell header.

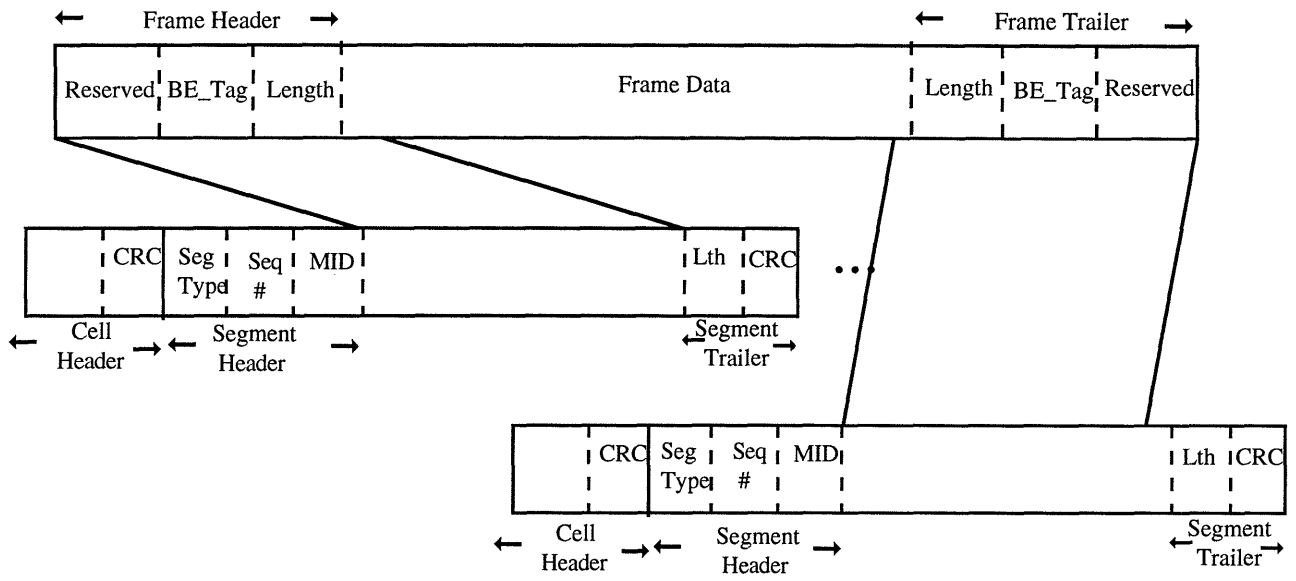


Figure 4.5 Format of frame and cells in the CCITT AAL Type 3/4 proposal.

There is a 16 bit frame length field in both the frame header and trailer to protect against lost cells. (The length field in the frame header is also referred to as the buffer allocation field since it can be used to allocate buffer space at the receiver.) There is an 8 bit frame sequence number, referred to as the Begin/End Tag (BE_Tag), in both the frame header and trailer. This helps protect against merged frames.

The cell header contains the VCI/VPI field, some miscellaneous bits related to routing, and an 8 bit CRC. It is identical to the cell header of our proposal except it does not contain a flag indicating the last cell of a frame. Also, in both AAL Type 3/4 and Type 5, the cell header CRC operates in a two state correction/detection mode rather than a four state mode.

The segment header contains a 2 bit segment type field that indicates whether the cell is the first cell, one of the middle cells, or last cell of the frame. A 4 bit sequence number in the segment header helps protect against lost cells. There is a 10 bit message identifier (MID) field in the segment header that is used for connectionless traffic. This field is not necessary for connection oriented data transfer but the CCITT has assumed it is present in AAL 3/4.

AAL 3/4 also includes a 6 bit field in the segment trailer to indicate the number of bytes contained in each cell. Each cell, except the last cell in the frame, is expected to contain 48

bytes in the information section of the cell. Thus, including a cell length field in each cell is unnecessary. Finally, there is a 10 bit CRC in the segment trailer that protects the contents of the segment.

4.7.1.1 Analysis of AAL 3/4

The major problem with AAL 3/4 is that a per-segment CRC is used rather than a per-frame CRC. A per-segment CRC does not help in detecting lost cells, misdirected cells, or merged frames. This necessitated the addition of fields such as the per-segment sequence number, the frame length field, and the Begin/End Tag. The performance of this scheme very much depends on the characteristics of burst errors in the system. If we assume that burst errors always affect a large number of cells, and that all cells that are hit by the burst will contain completely random bits, then AAL 3/4 provides sufficient protection. If we make the same worst case assumptions that we did in Section 4.4.2, where bursts are very short, then we find that we have a probability of 2^{-10} of not detecting burst errors that affect only the data in the frame. As shown in Section 4.4.2.3, this does not provide sufficient protection. Since we can not be sure exactly what type of burst errors to expect, it makes more sense to use an error detection scheme that is powerful over a wider range of errors. Thus, a per-frame CRC is preferred over a per-segment CRC.

There are some other scenarios that point out the weakness of not having a CRC that checks on the frame as a whole. For example, consider the scenario where a cell is misdirected and is accepted as the END cell of a frame in place of the correct END cell. Assume the misdirection is caused by a burst error and the burst is short enough so that only the cell header is affected. The AAL 3/4 scheme relies totally on the 4 bit cell sequence number and 8 bit BE_TAG to catch the error. In fact, if the cell header CRC is used in a two-state mode as proposed in AAL 3/4, then the expected annual frequency of this error going undetected is about 9×10^{-3} (assuming 10 cells per frame).

Also, consider the scenario where a 'middle' cell is lost from a frame due to congestion and another middle cell is misdirected and becomes part of that same frame. If the misdirection is caused by a short error burst, then only the 4 bit cell sequence number offers protection against this error. If a two-state cell header CRC is used, the probability of this error going undetected is about 2×10^{-4}.

Although neither of these two error scenarios poses major problems, it points out how little protection there is against some of the error scenarios when a per-frame CRC is not

present. (We did not include the performance of the AAL 3/4 scheme in Table 4.1 since there are many additional error scenarios that arise due to errors in the control fields (e.g., errors in the segment type).)

The AAL 3/4 scheme is obviously less efficient than our proposed scheme and the scheme proposed in AAL 5. Assume a frame is comprised of 10 cells. The amount of overhead in the various schemes is:

Our Proposed Scheme	AAL 5 Proposal	AAL 3/4 Proposal
Cell header CRCs: 10 x 8 bits	Cell header CRCs: 10 x 8 bits	Cell header CRCs: 10 x 8 bits
Frame CRC: 34 bits	Frame CRC: 32 bits	Frame header: 32 bits
Pad Field: 6 bits	Frame length field: 16 bits	Frame trailer: 32 bits
TOTAL: 120 bits	TOTAL: 128 bits	Segment header: 10 x 16 bits
		Segment trailer: 10 x 16 bits
		TOTAL: 464 bits

Since much of the overhead in AAL 3/4 is included on a per-cell basis, the difference in overhead between AAL 3/4 and the other two schemes increases as the size of the frame increases. In general, if the number of cells per frame is large, then the overhead per cell is roughly 8 bits in our proposed scheme and in AAL 5, and 40 bits per cell in AAL 3/4. Note that a large portion of the traffic on ATM is expected to consist of video and still images; these applications typically involve very large frames.

We conclude that the AAL 3/4 scheme requires more overhead and provides less protection than our proposed scheme and AAL 5.

4.8 SONET

SONET serves as an interface between the ATM layer and the actual fiber optic link. It includes several parity checks so that the level of bit errors seen by the ATM and AAL layers may be lower than what was assumed above.

From the standpoint of error detection, one of the most important aspects of SONET is the inclusion of a self synchronous scrambler to provide security. If a user transmits the scrambler bit pattern continuously, the output of the scrambler will be a string of zeroes,

which will cause the failure of the SONET system.[DrD91] To counteract this, a second scrambler is added. Let x[n] represent the n^{th} information bit and let y[n] represent the n^{th} bit actually transmitted. The operation of the second scrambler at the transmitter is such that y[n] = x[n] + y[n-43]. The inverse operation is performed at the receiver: x'[n] = y'[n] + y'[n-43]. Thus, if there is one actual transmission error, it will appear at the destination as two bit errors separated by 43 bits.

As shown below, this second scrambler does not significantly affect the ability of the cell header CRC or frame CRC to detect errors. First consider the cell header CRC. The cell header is 40 bits long; thus, if a true bit error hits the cell header, the 'extra' bit error occurs after the cell header. However, any true bit errors that occur in the last 43 bits of the previous cell affect the cell header, as shown in Figure 4.6.

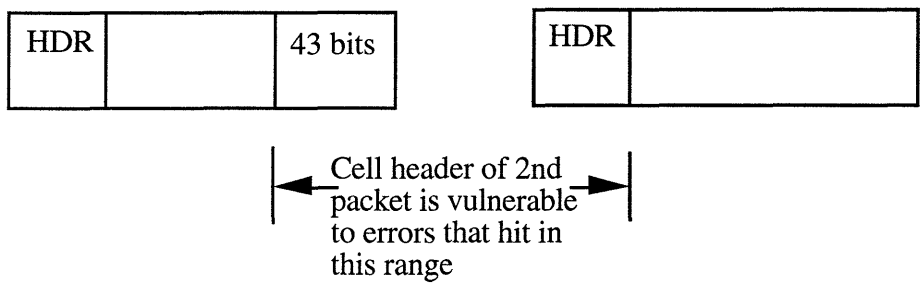


Figure 4.6 True bit errors that occur in the last 43 bits of a cell will result in a second bit error occurring in the next cell header due to the SONET scrambler.

Thus, when calculating the probability of dropped cells and misdirected cells due to random bit errors (either true or 'extra' bit errors) in the cell header, the effective size of the cell header is 83 bits rather than 40 bits. Thus, in equations (4.2) and (4.3), which respectively represent the probability of misdirected cells and lost cells due to random bit errors, 40 should be replaced by 83. This does not significantly affect the results.

Now consider the effect of the scrambler on the error detecting properties of the frame CRC. The length of the frame is greater than 43 bits. Thus, a true bit error that hits a frame may result in two bit errors in the frame. We need to consider whether two true bit errors that hit the frame could possibly go undetected by the frame CRC (the two true bit errors can result in four actual errors in the frame, and the CRC is not guaranteed to detect all four bit error patterns). Much of the terminology used below was described in chapter two.

We assume we are using an Extended Hamming code CRC of length 34 bits. Thus, all valid frames can be considered to be codewords generated by a generator polynomial that is the product of (X + 1) and a primitive polynomial of order 33. Using finite field theory, there exists an element (referred to as a primitive element in the finite field of size 2^{33}) that is a root of the primitive polynomial but that is not a root of $(X^w + 1)$ for any $w < 2^{33}-1$. (Refer to [Gal68] for a more thorough discussion of primitive polynomials and primitive elements.) All valid codewords have the generator polynomial as a factor. Thus, the primitive element must be a root of all valid codewords.

Assume that two true bit errors occur in positions q and s of a frame, where q - s = d. The maximum size of a frame is 2^{19} bits; thus, d must be less than 2^{19}. There are four cases that we need to consider:

1) Assume that the two true bit errors are not in the last 43 bits of any segment (i.e., the extra bit errors occur in the same segment as the corresponding true bit errors), so that there are four bit errors in the frame at positions q, q+43, s, and s+43. The error can be represented by the polynomial:

$$X^q + X^{q+43} + X^s + X^{s+43} = X^s (1 + X^d) (1 + X^{43}) \qquad (4.4)$$

In order for an error not to be detected, the error polynomial must represent a valid codeword. From the property of primitive elements discussed above, we know the primitive element is not a root of $(1 + X^d)$ or $(1 + X^{43})$ (since d is less than $2^{33}-1$); thus, it is not a root of the error polynomial given in equation (4.4). Thus, the error polynomial is not a valid codeword, and the error will be detected.

2) Assume that both of the two true bit errors occur in the last 3 bits of two, not necessarily distinct, segments. Then the corresponding extra errors will occur in the beginning of the next segments, as shown in Figure 4.7. When the segments are joined together to form a frame, each true bit error and its corresponding extra bit error will be separated by three bit positions (the cell headers, which are 40 bits long, are not part of the frame). Thus, the four bit errors in the frame can be represented by the polynomial: $X^q + X^{q+3} + X^s + X^{s+3} = X^s (1 + X^d) (1 + X^3)$

Using the reasoning given for the previous case, a polynomial of this form can not be a codeword; thus, the error will be detected.

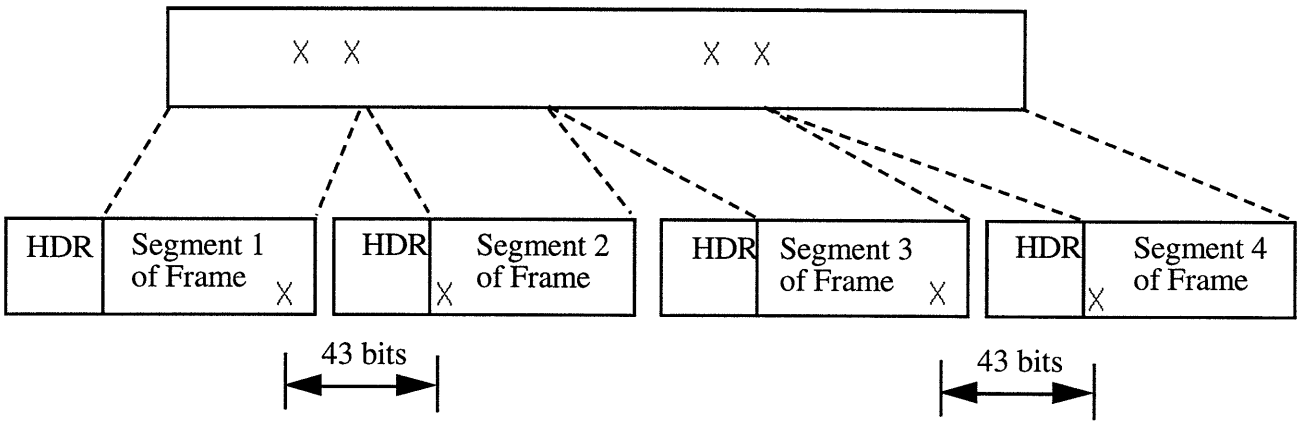


Figure 4.7 If a true bit error occurs in the last 3 bits of a segment, then the extra bit error occurs in the beginning of the next segment. When the segments are put together to form a frame, the true bit error and the extra bit error will be 3 bits apart.

3) If one or both of the two true bit errors occurs in the final 43 bits of a segment, but not in the final 3 bits of the segment, then the extra bit error(s) will occur in the cell header of the next segment. This will result in less than four bit errors in the actual frame, so that the CRC will detect the error.

4) Assume one of the true bit errors occurs in the final 3 bits of a segment, and the second true bit error occurs in the first (384 - 43) bits of a segment. Then one pair of errors in the frame will be separated by 3 bits and the other pair by 43 bits. The error can be represented by the polynomial:

$$X^q + X^{q+3} + X^s + X^{s+43} \qquad (4.5)$$

In order for such an error to go undetected, this polynomial must represent a valid codeword. If the degree 34 generator polynomial is taken to be $(X^{34} + X^{33} + X^{14} + X^{13} + X + 1)$ (obtained from [Pet61]), then by exhaustive search it can be shown that there are no codewords of the form given by (4.5).

Overall, we conclude that two true bit errors are not sufficient to cause the frame CRC to fail to detect the error.

At least three true bit errors must occur before bit errors in the frame will not be detected. There are two different scenarios to consider. First, all three true bit errors could hit in the frame such that the three extra errors also occur in the frame. There would then be a total of 6 bit errors in the frame. Once the position of two of the true bit errors is chosen, there is only one possible position where the third true bit error can occur to result in an

undetected error. If this were not the case, then by the linearity of the code, there would be a codeword with 2 pairs of errors, where the errors are separated by 3 or 43 bits. This was shown above to be impossible. Thus, we can upper bound the expected annual frequency of frames per line where such an error is undetected by: $\frac{10^{13}}{N}\binom{N(48\bullet 8 - 40)}{2} P_R{}^3$. If N is 1366, which is the maximum number of cells per frame, then this frequency is $8x10^{-4}$, which satisfies our goal.

A second scenario is where only one of the three true bit errors occurs in the frame such that its corresponding extra error also hits the frame. (Note that if exactly two of the three true bit errors result in pairs of errors in the frame, there would be a total of 5 bit errors in the frame, which can be detected with certainty by the Extended Hamming code CRC.) Once the position of this error pair and one of the other true bit errors is chosen, there is only one possible position where the third true bit error can occur to result in an undetected error. If this were not the case, then by linearity of the code, there would be a codeword of weight 2, which is impossible (refer to Section 2.1.2). In order for a true bit error to hit such that it only results in one error in the frame, it must occur in the last 43 bits of one of the frame's segments or in the 43 bits prior to the start of the segment, as shown in Figure 4.8. Thus, we can upper bound the expected annual frequency of frames per line where such an error is undetected by: $\frac{10^{13}}{N}(N(48\bullet 8 - 40))\ (N)(2)(43)\ P_R{}^3$. If N is 1366, which is the maximum number of cells per frame, then this frequency is $4x10^{-4}$, which satisfies our goal.

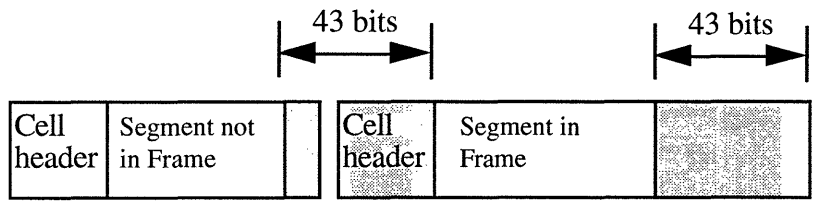


Figure 4.8 In order for a true bit error to only result in one error in the frame, it must hit in the shaded region. This assumes that cells of the frame are interleaved with cells of other frames.

Thus, the scrambler increases the rate of undetected error, but the rate is still acceptable.

4.9 CONCLUSIONS

For the ATM network, a 34 bit frame CRC should provide sufficient protection in attaining our goal of no more than one undetected errored frame per receiver per year. This solution

should be very robust. We also showed that a four-state correction/detection option for the cell header CRC is a simple method of reducing the probability of misdirected data. Our proposed scheme is more effective and efficient than the CCITT AAL Type 3/4 proposal, but is similar to the CCITT AAL Type 5 proposal.

CHAPTER 5

ERROR DETECTION IN TCP/IP

5.1 TCP/IP OVERVIEW

TCP (Transmission Control Protocol) and IP (Internet Protocol) are a set of standards designed for interconnecting computer networks. TCP/IP is the standard communications protocol used by computers connected to the Internet. The Internet is an entity established in the early 1980's by the Defense Advanced Research Projects Agency (DARPA) to provide data communication services between diverse networks distributed worldwide. As of January, 1993, there were an estimated 1.3 million computers on about 8,000 networks interconnected by the Internet.[Con93]

Our reasons for studying TCP/IP are threefold. First, we want to show that the design guidelines presented in chapter three are applicable to TCP/IP as well as ATM, despite the numerous differences between these systems. Second, we use these guidelines as a basis for evaluating the effectiveness of the error detection scheme that is currently used in TCP/IP. Third, we want to determine if the scheme can be made more effective and efficient.

We examine TCP/IP in the context of the Internet. As can be seen in Figure 5.1, the interconnected networks are referred to as subnetworks. The various subnetworks are connected by computers that are called gateway nodes. From the standpoint of TCP/IP, these gateways are the intermediate nodes along the data path. Thus, when considering the low layer errors that occur in TCP/IP, we are actually considering errors that occur within the underlying subnetworks.

Note that although TCP and IP are often referred to as a single unit (i.e., TCP/IP), these two protocols can be implemented independently of each other. For example, TCP can be

used as the transport layer on a single network such as Ethernet[Com91]. IP can be used with the User Datagram Protocol, the Internet Control Message Protocol, or the Exterior Gateway Protocol rather than with TCP (see [Com91] for a description of these other protocols). However, we focus only on the TCP/IP combination in this chapter.

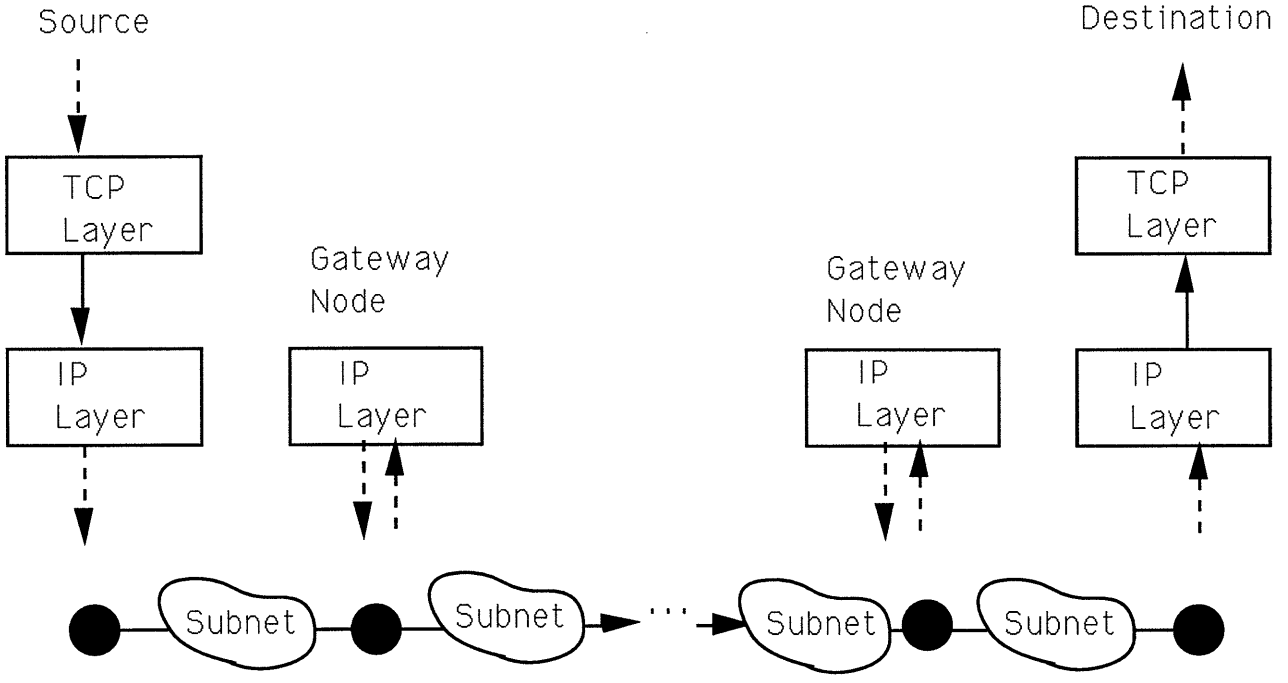


Figure 5.1 Diagram of TCP and IP layers on the Internet.

We can relate the functions performed by TCP and IP to the functions performed by the three layers analyzed in chapter three. The TCP layer corresponds to layer N, although its PDU is called a segment rather than a message. TCP operates on an end-to-end basis, and is responsible for any necessary data retransmission. IP lies below TCP at the source and destination, and is the highest layer at the gateway nodes. It is responsible for dividing a segment up into data units called fragments (rather than packets). Fragmentation can take place at the source as well as at gateway nodes along the data path. The segment, however, is reconstructed only at the destination. IP also handles the routing of fragments from one gateway node to another. Thus, IP can be considered to be a combination of layers N-1 and N-2.

In TCP/IP, we define an undetected error event as occurring when a segment that contains any type of error is accepted by the TCP layer at the destination as error-free. In ATM, we set our goal to be a maximum of one undetected error event per year, per line. The TCP/IP

system is in general much less reliable than ATM, but the lower data rate of TCP/IP results in less transmitted data. Thus, it is reasonable to expect that the annual rate of undetected error can be reduced to this same level in TCP/IP. However, as shown below, the current TCP/IP implementation does not meet this goal. The error detection scheme could be modified so that the goal is met, but at the expense of complexity.

In Section 5.2, we discuss the general properties of TCP/IP and describe the error detection scheme that is currently implemented in TCP/IP. In Section 5.3, we discuss our assumptions about the various low level errors expected in TCP/IP systems. In Section 5.4, we analyze the TCP/IP error detection scheme using the algorithm of chapter three as a guideline. Where appropriate, we suggest improvements to the scheme.

A large portion of the TCP protocol is involved with establishing the connection between the source and destination nodes i.e., sending and acknowledging synchronization messages. We will not analyze the potential problems that arise with this aspect of TCP. (Reference [Hes88] describes some examples of the source and destination becoming unsynchronized.) We will assume the connection has been successfully established and focus on detecting errors that occur in the transmission of data.

5.2 TCP/IP DESCRIPTION

The following description of the TCP/IP protocol was obtained from references [DOD85] and [Com91]. The format of a TCP segment and an IP fragment is shown in Figures 5.2 and 5.3, respectively.

TCP segments can be variable in length, with the maximum length being 65,536 bytes. The segments are divided up into fragments by the IP layer. The size of the fragment depends on the maximum sized datagram that the various encountered subnetworks can accommodate. Fragmentation must be done on 8 byte boundaries. A 16 bit fragment length field in the fragment header indicates the size of the fragment, including the header. The length of the fragment header depends on which options have been set. The length must be a multiple of four bytes, and can range from 20 bytes to 60 bytes. A 4 bit field near the beginning of the fragment header indicates the length of the header, in units of 4 bytes.

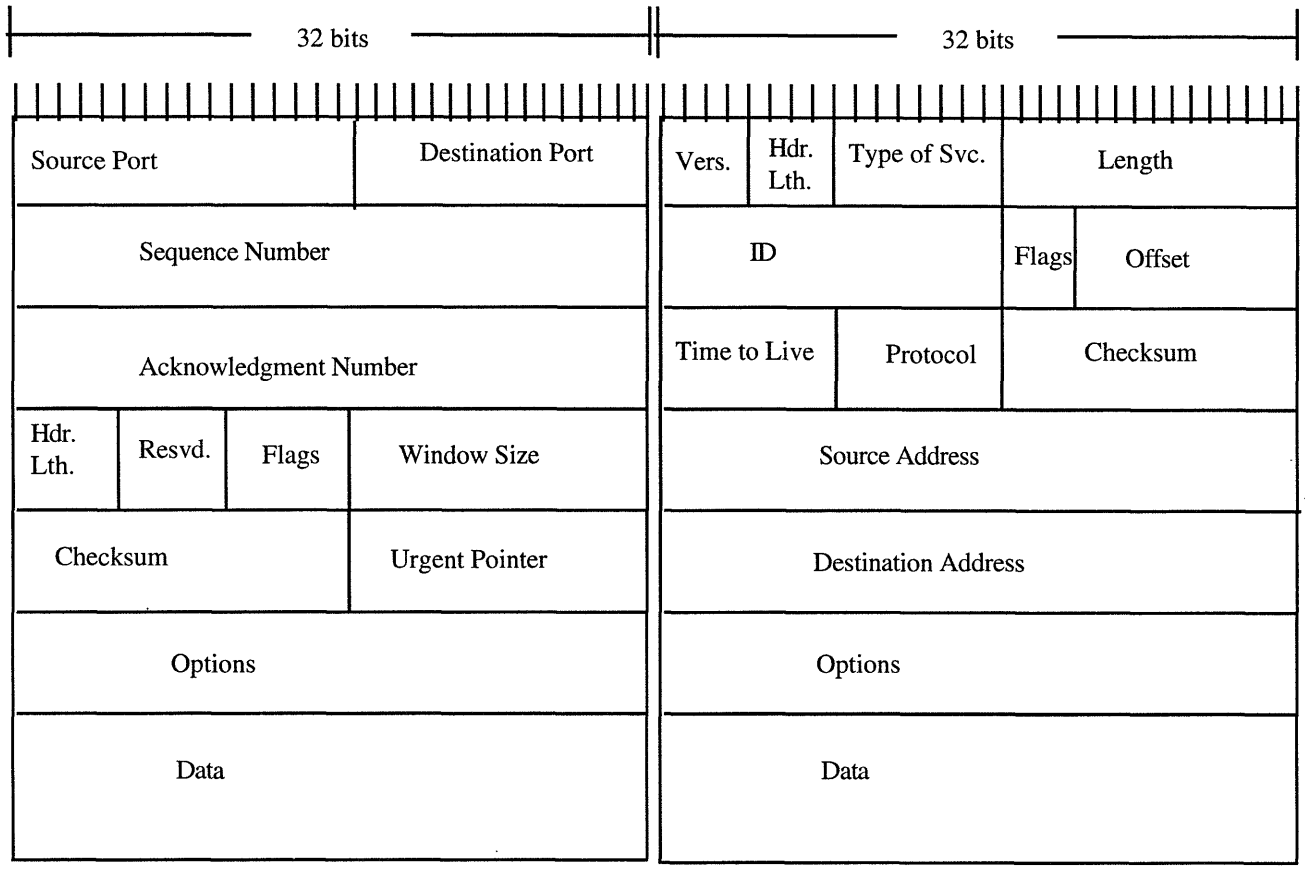


Figure 5.2 TCP Segment.

Figure 5.3 IP Fragment.

Fragments are routed on an individual basis. Each one contains the 32 bit Internet address of the destination, along with the Internet address of the source. There is no guarantee that all fragments belonging to one segment will follow the same path. It is possible that the fragments may arrive at the destination in a different order from which they were sent. This is handled in IP by including a 13 bit offset field in the fragment header that indicates the offset of the fragment within the original TCP segment, in units of 8 bytes. It is also possible that fragments from one segment may arrive intermixed with fragments from another segment. Thus, each fragment includes a 16 bit identification field that identifies the segment to which that fragment belongs. IP also might deliver duplicate fragments. A retransmitted segment may be fragmented differently from the original transmission, but the retransmitted fragments will contain a different 16 bit ID.

Another potential difficulty in reconstructing the segment correctly is that fragments may circulate throughout the network for a long time, so that fragments from a torn-down

connection may be confused with fragments from a current connection. There is an 8 bit Time To Live (TTL) field in each fragment header to help prevent this problem. The field is decremented by one each time a fragment is routed through a gateway. A fragment is discarded if the TTL field becomes 0. This field is more appropriately thought of as a 'Max Gateways to Cross' field, since it does not necessarily have any correlation with time.

The actual length of the original TCP segment is not explicitly contained anywhere. Instead, each fragment has a one bit flag in its header indicating whether the fragment is the last one in the segment or not. The total length of the segment can then be determined as the fragment offset plus the length of the data portion of the final fragment.

The entire fragment header is checked by a 16 bit checksum in the IP header.

TCP also includes two error detecting fields in its header. Each byte of data to be sent in a TCP connection can be thought of as being numbered modulo 2^{32}. The number of the first byte of the segment is contained as a 32 bit sequence number in the header of each segment. This aids in determining when segments are missing at the destination, and also helps to detect segments belonging to torn-down connections. We discuss this further in chapter eight when we analyze the TCP retransmission scheme.

The TCP header also contains a 16 bit checksum, which checks on the entire TCP segment. The checksum also checks on a pseudoheader comprised of the destination address, source address, segment length, and protocol indicator (which should be set to TCP). IP determines the values for these fields based on what it has received, and passes these values to TCP. The TCP checksum is calculated as if these fields were part of the TCP segment. (See chapter two for a discussion of pseudoheaders.)

The length of the segment header depends on which options have been set. The length must be a multiple of four bytes, and can range from 20 bytes to 60 bytes. A 4 bit field near the beginning of the segment header indicates the length of the header, in units of 4 bytes.

Table 5.1 compares the characteristics of TCP/IP to those of ATM Service Class C. TCP corresponds to the Convergence Sublayer of the AAL, and IP corresponds to a

combination of the Segmentation and Reassembly Sublayer and the ATM Layer. Below, when we use the term message, we are referring to a segment in TCP/IP systems and a frame in ATM systems. Table 5.2 compares the mechanisms included for error detection and reconstruction of messages in TCP/IP, the CCITT AAL 3/4 Proposal, and our ATM proposal. TCP/IP and our proposed ATM scheme are similar in that both schemes include a redundancy check on the whole message. However, our scheme overall has many fewer checks than TCP/IP, which is reasonable since ATM systems are expected to be much more reliable than TCP/IP systems. Many of the checks that are included in TCP/IP are also included in the AAL 3/4 proposal. In TCP/IP these checks are needed due to the unreliable nature of IP; in the AAL 3/4 proposal, these checks are needed since there is no redundancy check on the frame as a whole.

Table 5.1 Characteristics of TCP/IP vs. ATM

TCP/IP	ATM (Service Class C)
Variable length segments	Variable length frames
Max length of segment is 2^{16} bytes	Max length of frame is 2^{16} bytes
Variable length segment header	Fixed length frame header and trailer
Variable length fragments	Fixed length cells
Variable length fragment header	Fixed length cell header
Fragments are not expected to travel in order; duplicate fragments are expected	Cells expected to travel in order; duplicate cells not expected
Reassembly generally only done at destination. Fragmentation can occur at any gateway along the data path.	Reassembly generally only done at destination. Segmentation of frames occurs only at the source.

Table 5.2 Mechanisms for Error Detection and Reconstruction of Messages

TCP/IP	AAL 3/4 Proposal	Our ATM Proposal
1 bit flag in fragment header to indicate last fragment of segment	2 bit field in cell payload indicates BOM, MOM, EOM	1 bit flag in cell header to indicate last cell of frame
2 byte fragment ID to identify which fragments belong to the same segment	10 bit message ID (MID) in cell. Set to a default value for connection-oriented traffic.	No ID field in cells
Checksum on whole segment	No check on whole frame	CRC on whole frame
No per-fragment check on fragment data	CRC on cell payload	No per-cell check on cell data
Checksum on fragment header	CRC on cell header	CRC on cell header
Fragment length field included in all fragments	Cell length field included in all cells	No cell length field; pad length field at end of frame
Offset within segment included in each fragment	Cells are numbered mod-16	Cells not numbered
Segment length field is not explicitly included in the segment. It is part of the pseudoheader checked by the segment checksum.	Length field in both frame header and trailer	No length field in segment
32 bit sequence number for TCP segment; increases by the number of bytes in the segment.	8 bit frame ID in both frame header and trailer	No frame ID field
Destination address of segment checked implicitly by segment checksum.	Destination address not checked	Destination address of frame implicitly checked by frame CRC

5.3 ASSUMPTIONS

5.3.1 Error Characteristics of TCP/IP

TCP/IP is run over a wide variety of systems, with very different error characteristics. We can expect random bit errors, burst errors, and congestion. We use the variables P_R, P_B, and P_C to represent the random bit error rate, burst error rate, and congestion drop rate, respectively. It is difficult to come up with meaningful estimates of the likelihood of these different events since the underlying subnetworks may have very different characteristics. However, it is likely that the TCP/IP system will be less reliable than the ATM system. In ATM, we assume P_R equals 10^{-8}, P_B equals 9×10^{-8}, and P_C equals 10^{-6}. Our approach will be to use probabilities that are about 100 times more likely than this, which is obviously somewhat arbitrary. Thus, we use $P_R = 10^{-6}$, $P_B = 10^{-5}$, and $P_C = 10^{-4}$.

Throughout our analysis, we comment on when these assumptions significantly affect our results.

Note that systems that require high reliability are likely to have additional error checks in addition to the checks included in TCP/IP. For example, the subnets could be local area networks that are likely to include CRCs that check on the validity of the data. Thus, the bit error rate seen at the TCP/IP level is likely to be smaller than the value we are using.

5.3.2 Annual Traffic

The central network of the Internet is the NSF backbone net, which is made up of T1 lines running at 1.5 Mb/sec[Com91]. (This is being updated to T3 lines running at 45 Mb/sec.) Local data links are likely to be running at 56 Kb/sec. However, we will use 1.5 Mb/sec as the data rate for the whole data path since that will provide some margin in attaining our goal of no more than one undetected error event per year per line.

The size of a fragment can be variable, but we will perform our analysis using the default fragment size of 576 bytes. If our calculations call for a segment size, we will arbitrarily use 1024 bytes as the size. Both the IP header and the TCP header can range in size from 20 to 60 bytes, but we will assume the size is 20 bytes, which is the typical size in practice.

Using these assumptions, we estimate that about 10^{10} fragments and 5×10^9 segments travel per line, per year.

5.4 ANALYSIS OF TCP/IP ERROR PROTECTION SCHEME

We use the five step algorithm presented in chapter three to analyze the error protection provided in the current implementation of TCP/IP. TCP and IP both make use of a 16 bit checksum rather than a CRC, since a checksum is easier to implement in software. It is worthwhile to review the error detecting properties of a checksum. If N is the number of 16 bit words being checked by the checksum, then the probability of two random bit errors occurring and going undetected by the checksum is given by equation (2.15): $16 \binom{N}{2} \frac{1}{2} P_R^2$. In the analysis below, we are frequently concerned with the case where a random bit error occurs in a particular M bit field. If M is less than or equal to 16, then the probability of the error occurring and not being detected is:

$$M\,(N-1)\,\frac{1}{2}P_R^2 \quad (5.1)$$

We can also use equation (5.1) to upper bound the probability of the error occurring and not being detected for the case where M is greater than 16. Throughout the analysis, we assume the probability of undetected errors due to more than two random bit errors is negligible.

Referring back to Table 3.1, the errors that need to be handled in TCP/IP are:

TABLE 5.3: List of Error Scenarios in TCP/IP

1. Stray fragments
   - Fragments arrive at the wrong destination
   - Fragments remain from an old connection
2. Out-of-sequence fragments
3. Duplicate fragments
4. Lost fragments
5. Fragments from more than one segment are merged into one segment
   - Error in ID field
   - END fragment lost
6. Bit errors in the data
7. Bit errors in the control information
   - Error in END field
   - Error in offset field
   - Error in fragment length field
   - Error in fragment header length field
   - Error in segment header length field

For completeness, we will examine each of these error scenarios, using the guidelines of chapter three. Our analysis will show that bit errors in the data is by far the most common type of undetected error event that can be expected. The results of the analysis are summarized in Table 5.4 at the end of the section.

5.4.1 Step 1: Prevention of Misdirected Fragments

In IP, routing of the fragments is performed based on the value of the 32 bit destination address in the IP fragment header. Undetected errors in the address field can result in misdirected data. To protect against stray data, IP includes a 16 bit checksum that checks on the fragment header.

Let's calculate the protection that is provided by the IP header checksum. We make the worst case assumption that any undetected errors in the address field will result in a misdirected fragment. First, consider undetected random bit errors. At least 2 bit errors must occur in the 20 byte header before the error could go undetected, and at least one of the bit errors must occur somewhere in the 32 bit address. Using equation (5.1), the probability of this is roughly: $(32)(10\text{-}1)\frac{1}{2}P_R^2 = 10^{-10}$. Thus, the number of misdirected fragments due to random bit errors per year, per line is about 1. If P_R were 10^{-5} rather than 10^{-6}, the frequency of stray fragments per year, per line would be about 100. Note that in ATM, with a cell header CRC for protection, we expect about 20 misdirected cells per year, per line.

Next we look at the probability of undetected error if the fragment is hit by a burst error. We assume the header checksum fails to detect the error with probability 2^{-16}. Thus, with P_B equal to 10^{-5}, the probability of stray fragments due to burst errors is: $P_B\ 2^{-16} = 10^{-10}$. Again, this leads to about 1 misdirected fragment per year, per line.

We conclude that a 16 bit checksum provides sufficient protection against misdirected fragments. It provides less protection against random bit errors than a CRC, but the protection is still sufficient. As for burst errors, the length of the redundancy check is the key factor. Thus, the 16 bit checksum is actually more effective than an 8 bit CRC against burst errors.

For the remainder of the analysis, we assume that the probability of a fragment being misdirected is about $2\text{x}10^{-10}$.

5.4.2 Step 2: Detection of Bit Errors in Data

The only protection against bit errors in the data is the 16 bit checksum in the TCP segment. First, consider random bit errors. Assuming the segment size is 1024 bytes (i.e., 512 16-bit words), the expected annual frequency of undetected random bit errors in the data, per line is about: $(5\text{x}10^9)\ 16 \binom{512}{2} \frac{1}{2} P_R^2 = 5\text{x}10^3$. This does not come close to meeting the goal of no more than one undetected error per year, per line. If the underlying system happens to be very reliable (e.g., the underlying media is reliable or the subnets include their own error protection mechanisms), and P_R is on the order of 10^{-8} rather than 10^{-6}, then the annual frequency of undetected error would be .5. Thus, under very good

conditions, the checksum provides sufficient protection. However, it is desirable that the error detection scheme of TCP/IP provide enough protection over a range of systems.

One improvement would be to increase the length of the checksum. For every doubling of the length of the checksum, the rate of undetected error decreases by a factor of 2. This is not an efficient means of decreasing the undetected error rate. Another option would be to use a different type of checksum. There has been some research done on alternate ways of calculating a checksum so that undetected errors are less likely. For example, in [Fle82] a checksum is described that is capable of detecting more two bit error patterns than the checksum used in TCP.

Ideally a CRC would be used instead of a checksum. A CRC provides better error protection than a checksum, but it is more complex to implement in software. However, it is currently possible to implement a CRC in inexpensive hardware; thus, it would not be unreasonable to include a CRC as part of the protocol. Assume a 32 bit Extended Hamming code CRC were used. Then, using equation (2.10), the rate of undetected bit errors per year, per line would be: $(5\text{x}10^9)\binom{8192}{3}\frac{1}{4}P_R{}^4 = 10^{-4}$.

Next, consider the protection the 16 bit segment checksum provides against burst errors. We make the worst case assumption that only the data portions of the fragments are affected by the burst error. If fragments are independently hit by a burst error with probability P_B, then the expected number of segments containing undetected bit errors in the data due to burst errors, per year per line, is: $(10^{10})\,P_B\,2^{-16} = 1$. To provide some margin in attaining the goal of no more than one undetected error per year, per line, it would be preferable to have a redundancy check of length 32 bits.

Thus, we see that a 16 bit segment checksum is barely adequate in detecting burst errors and is inadequate in detecting random bit errors in the data (assuming the random bit error rate is 10^{-6}). A CRC of length at least 32 bits should be used. For the remainder of our analysis, however, we will continue to assume a 16 bit checksum is present since that is what is specified in the TCP protocol. We will see that undetected random bit errors in the data dominates all other undetected error events.

5.4.3 Step 3: Correction Options at the IP Layer

There are four types of problems that, if dealt with at the IP layer, can result in fewer dropped segments. Misdirected, duplicate, or old fragments can be detected and discarded, and out-of-sequence fragments can be reordered. The goal of this section is to determine the likelihood of these four error events, and to determine if it is worthwhile to add fields at the IP layer for the purpose of 'correcting' these errors.

5.4.3.1 Misdirected Fragments

Due to the presence of the fragment header checksum, the number of misdirected fragments is expected to be small. Furthermore, each fragment contains a source address and destination address field so that the destination can identify to which connection the fragment belongs. (Recall that IP is a connectionless layer, so that virtual channels are not established as they are in the connection-oriented service class of ATM.) When IP attempts to merge fragments together to form a segment, it checks to make sure that each of the fragments contains the same source address and destination address. It is likely the source address of a misdirected fragment will not be acceptable, so that the fragment will be dropped. Also, each fragment contains a 16 bit ID field that identifies the segment to which it belongs. It is unlikely that the ID field of a misdirected fragment would match the ID field of fragments at the incorrect destination (this is discussed further in Section 5.4.3.4). Due to the presence of these mechanisms, it is not worthwhile to add additional fields specifically for the purpose of detecting misdirected fragments at the IP layer.

5.4.3.2 Out-of-Sequence Fragments

Out-of-sequence fragments are common in IP; thus, it is worthwhile to include a sequencing mechanism so the fragments can be properly reordered by the destination. The sequencing mechanism also helps detect duplicate and old fragments.

Fragments are variable length, and fragmentation can occur at gateway nodes along the data path. Section 3.4.3.1.2 described two sequencing mechanisms that would be appropriate for IP: offset numbering and tree sequencing. Offset numbering is more robust, but tree sequencing requires less overhead. IP makes use of the offset numbering scheme; there is a 13 bit field in the fragment header that indicates the offset of the fragment within the TCP segment, in units of 8 bytes. (Segments have a maximum size of 2^{16} bytes.) The choice of the offset numbering scheme makes sense. Using the tree sequencing method would reduce the average amount of fragment header overhead by about 1 byte, but this is a

relatively small savings considering the fragment header can be anywhere from 20 bytes to 60 bytes long.

The offset field by itself is not sufficient to reorder fragments, since fragments belonging to different segments may be interleaved. For example, assume segment A and segment B are comprised of two fragments, both of size 512 bytes. If the fragments arrived in the following order the IP layer would not know how to properly resequence them: Segment B fragment, offset 512; Segment A fragment, offset 512; Segment B fragment, offset 0; Segment A fragment, offset 0. Thus, it is necessary to include a field that identifies which segment the fragment has come from. IP includes a 16 bit ID field for this purpose. The ID field is also helpful in detecting out-of-date fragments, as will be discussed in Section 5.4.4.

In addition to the offset field and ID field, IP also includes a fragment length field in each fragment. As described in Section 3.4.3.1, given the presence of these three fields, the worst case resequencing scenario occurs when the last fragment arrives before the second-to-last fragment, and the last fragment contains an error in its offset field so that its offset equals the offset of the second-to-last fragment. The last fragment would in effect take the place of the second-to-last fragment, and the second-to-last fragment would be ignored. The expected annual frequency of such an error occurring due to random bit errors in the 13 bit offset field and going undetected at the IP layer can be upper bounded by:
$(5\text{x}10^9)\ 13\ (10 - 1)\ \frac{1}{2}\ P_R{}^2 = .3.$

We can also consider a burst error hitting the header of the last fragment and changing the offset field and the checksum field (but leaving the other fields intact). (Note that if the offset field were the only field affected by the burst error, then the error would be detected with certainty, since the 16 bit checksum can detect error bursts up to 15 bits.) After being hit by a burst error, the offset field will equal the offset of the second-to-last fragment with probability 2^{-13}. The IP checksum will not detect the error with probability 2^{-16}. Thus, the expected annual frequency of this error occurring due to burst errors and going undetected at the IP layer is: $(5\text{x}10^9)\ P_B\ 2^{-13}\ 2^{-16} = 9\text{x}10^{-5}$.

Thus, the expected rate of resequencing errors at the IP layer is .3 per year, per line. As will be discussed below, the segment checksum helps detect resequencing errors that do

occur. Thus, we conclude that it is not worthwhile to add further fields at the IP layer for the purpose of resequencing fragments.

5.4.3.3 Duplicate Fragments

Duplicate fragments should be detected by the combination of the fragment offset field and the ID field. There is not a need to add further checks to detect duplicate fragments.

Note that when segments are retransmitted, a new ID is used for the fragments. This is necessary since segments may be fragmented differently when they are retransmitted, and the fragments from earlier transmissions may still be present in the system.

5.4.3.4 Old Fragments

Routing errors may occur such that a fragment circulates throughout the network for a long time before being delivered to the destination. This can lead to errors if the out-of-date fragment is not distinguishable from current fragments. One weak form of protection against this scenario is the Time To Live field in the fragment header. As described in Section 5.2, this field upper bounds the number of gateway nodes a fragment can cross. However, it does not necessarily limit the amount of time a fragment can spend in the network. For instance, a fragment can circulate through one subnetwork for a long time; during this time it does not cross any gateway nodes.

Thus, it can be expected that out-of-date fragments will arrive at a destination. The 16 bit ID field included in the fragment can be used by IP to detect old fragments. When a fragment that is not a complete segment arrives at the destination, the destination checks to see if the ID field matches the ID of any other fragments it is currently processing. If it does not match, the destination sets a reassembly timer corresponding to this new ID. If another fragment with this ID is not received by the expiration time of the reassembly timer, the fragment is dropped. Thus, the likelihood of the old fragment appearing to be part of a current segment depends on the length of the timer. The longer the timer is, the more likely the old fragment will be accepted. If the data rate of a connection is 1.5Mb/sec and segments are 1024 bytes long, then the connection cycles through the 16 bit ID field in about 9 minutes. Thus, the timer should be set for a time significantly less than this in order for the ID field to be effective in detecting old fragments. In [DOD85] the suggested timer setting is 15 seconds. Also, in order for the old fragment to be accepted, it would have to contain the proper offset and length field to fit in with fragments at the destination.

Thus, we conclude that it is not worthwhile to add further checks at the IP layer for the purpose of detecting old segments.

5.4.4 Step 4: Examine Remaining Error Scenarios

Thus far, we have demonstrated the usefulness of the following error detection fields: a checksum on the fragment header, a fragment offset field, a fragment length field, a fragment ID field, and a redundancy check on the segment data. Ideally, the redundancy check on the segment data would be a CRC. However, TCP specifies that a checksum be used rather than a CRC. This is a very critical decision. As discussed in chapter three, only the data redundancy check is capable of detecting bit errors in the data. As shown in Section 5.4.2, the TCP checksum does not provide a lot of protection against random bit errors; the expected annual undetected error rate per line due to random bit errors in the data is $5x10^3$ (assuming P_R equals 10^{-6}). Thus, given that no additional data redundancy checks are used, the overall rate of expected undetected errors per year per line can be no smaller that $5x10^3$.

In this section, we run through the remaining error scenarios and show that with the current TCP/IP error protection scheme, the undetected error rate due to these other errors is negligible compared to $5x10^3$. Thus, it does not make sense to include additional fields to detect these scenarios, since the overall performance of the scheme will not improve. The data redundancy check field essentially establishes a lower bound to the undetected error rate.

The error scenarios below involve fragments being put together incorrectly to form a segment. In most of these scenarios, in order for the error not to be detected by the IP layer, some M bit field in one of the fragment headers must contain an error such that after the error it contains some specific value, say V. We will make the worse case assumption that just one bit error changes the field in such a way that it will contain the value V. Since the fragment header is protected by a checksum, a second bit error must occur somewhere else in the header in order for the checksum not to detect the error. Thus, if random bit errors cause the error in the M bit field, then the probability of the two bit errors occurring and not being detected by the checksum is (using equation 5.1):
$M\,(10\text{-}1)\,\frac{1}{2}P_R^2 = M(5x10^{-12})$.

The error in the M bit field could also be caused by a burst error. We will assume that a burst error hits such that only this M bit field and the header checksum are affected by the error. (If only the M bit field were affected, and M is less than 16, then the error would be detected with certainty by the checksum; if fields other than the M bit field and the checksum were affected by the error, then other inconsistencies would likely be produced). After the burst error, we assume the M bit field contains the value V with probability 2^{-M}. The error will not be detected by the header checksum with probability 2^{-16}. Thus, if a burst error causes the error in the M bit field, then the probability of the burst error occurring and not being detected by the checksum is $P_B\, 2^{-M}\, 2^{-16} = 2\text{x}10^{-10}\, 2^{-M}$.

Thus, in general, the overall probability of the error in the M bit field occurring and not being detected is:

$$M\,(10\text{-}1)\,\frac{1}{2}P_R^2 + P_B\,2^{-M}\,2^{-16} \qquad (5.2)$$

For M greater than 5, the term due to random bit errors dominates.

Throughout this section, we assume segments are comprised of multiple fragments, and that 10^{10} fragments comprising $5\text{x}10^9$ segments are transmitted per year, per line.

5.4.4.1 Stray Fragments

As shown in Section 5.4.1, the fragment header checksum reduces the probability that a fragment will be misdirected to $2\text{x}10^{-10}$. If a fragment is misdirected, it could be accepted as part of a segment at the wrong destination. In order for the misdirected fragment to be accepted as part of a segment, its source address must match the source address of the other fragments in the segment. We'll make the worst case assumption that it does match. Also, the ID of the misdirected fragment must match the ID of the other fragments in the segment. It will match with probability about 2^{-16}. (There might be more than one segment being constructed for a given connection so the probability of the ID of the misdirected fragment matching one of them will be higher than 2^{-16}. As mentioned in Section 5.4.3.4, this will depend on the length of the reassembly timer. We will ignore this.) In addition, the offset and length of the misdirected fragment must be precisely the right values in order for the stray fragment to fit in with the other fragments of the segment. We will make the worst case assumption that it does fit in. (Note that stray fragments are more likely to appear acceptable if the fragment size throughout the network is the same; errors are more likely to be detected if the fragment size is highly variable.) If the IP layer fails to detect the misdirected fragment, it will pass up to TCP a segment that has incorrect data in it. The

segment checksum will fail to detect the error with probability 2^{-16}. Assuming each misdirected fragment results in one errored segment, then the expected annual frequency of segments containing undetected misdirected fragments per line is: $10^{10}\ 2x10^{-10}\ 2^{-16}\ 2^{-16} = 5x10^{-10}$. This is negligible compared to the expected frequency of undetected bit errors in the segment data.

Note that the problem of old fragments circulating in the network most likely results from a routing error. Thus, we assume the calculation above for misdirected fragments also takes into account the scenario of old fragments arriving at a destination

5.4.4.2 Out-of-Sequence Fragments

As discussed in Section 5.4.3.2, IP includes many mechanisms for the purpose of correctly resequencing fragments that arrive out-of-order. Each fragment includes a fragment offset field, a fragment length field, and an ID field; these fields are all protected by the fragment header checksum. The expected number of undetected sequencing errors at the IP layer is $3x10^{-1}$, per year, per line. The TCP checksum will fail to detect the error with probability 2^{-16}. Thus, the overall expected number of segments containing undetected out-of-sequence fragments per year, per line is $3x10^{-1}\ 2^{-16} = 5x10^{-6}$. This is negligible compared to the $5x10^{3}$ rate of undetected errors due to bit errors in the data.

5.4.4.3 Lost Fragments

Congestion, burst errors, and random bit errors can all cause lost fragments. We make the worst case assumption that fragments are dropped independently due to congestion with probability P_C. We also assume fragments are independently hit by a burst error and dropped with probability about P_B. One random bit error in the fragment header will result in the fragment being dropped. The overall probability a fragment will be lost is then:
$P_C + P_B + (20)(8)P_R \approx 3x10^{-4}$.

There is a one bit flag in the fragment header that indicates whether a fragment is the last fragment of the segment. We look at the case where a non-END fragment is lost, so that the segment has too few fragments. If an END fragment is lost, fragments from multiple segments are merged together. Merged segments are discussed in the next section.

Each fragment contains a fragment offset field and a length field. The destination can use these fields to determine if there are any gaps in reconstructing the segment. Assume a segment is comprised of two fragments and assume the first one is dropped. There must

be a bit error in the 13 bit fragment offset field of the second fragment in order for the lost fragment not to be detected by IP. Using equation 5.2, the probability of the error occurring in the fragment offset field and not being detected by the fragment header checksum is: $13\,(10\text{-}1)\frac{1}{2}P_R{}^2 + P_B\,2^{-13}\,2^{-16}$.

If the IP layer does not detect the lost fragment, an incorrect segment will be passed up to TCP. In general, lost fragments will result in the calculation of the TCP checksum essentially being random. Thus, the segment checksum will fail to detect the error with probability 2^{-16}. Overall, the expected annual frequency of segments containing undetected lost non-END fragments per line is about:
$5\text{x}10^{9}\,(3\text{x}10^{-4})\left(13\,(10\text{-}1)\,\frac{1}{2}P_R{}^2 + P_B\,2^{-13}\,2^{-16}\right)2^{-16} = 1\text{x}10^{-9}$. This is negligible compared to $5\text{x}10^{3}$.

Notice that including the segment length in the pseudoheader does not help detect the error. The segment checksum fails to detect the error with probability 2^{-16} whether or not the segment length is included in the pseudoheader.

5.4.4.4 Merged Segments

There are many error scenarios that could result in the fragments of multiple segments being merged together at the destination. The most likely scenario is described here.

Consider two segments, which we refer to as A and B, that are each comprised of two fragments. Assume all four fragments are the same size. Assume the first fragment of segment B arrives, followed by the second fragment of segment A. If the 16 bit ID field of the B fragment has an error in it such that it matches the ID of the A fragment, then the two fragments will be merged together to form a segment.

Using equation (5.2), the probability of the error occurring in the ID field and not being detected by the header checksum is: $(16\,(10\text{-}1)\frac{1}{2}P_R{}^2 + P_B\,2^{-16}\,2^{-16})$. If the error is not detected by the IP layer, then the segment checksum (which will come from the B fragment) will essentially be checking random bits; it will fail to detect the error with probability 2^{-16}. Thus, the expected annual frequency of undetected merged segments per line, is: $5\text{x}10^{9}\left(16\,(10\,\text{-}1)\,\frac{1}{2}P_R{}^2 + P_B\,2^{-16}\,2^{-16}\right)2^{-16} = 5\text{x}10^{-6}$. Again, we see that

the undetected error rate of this error is negligible compared to the undetected error rate of random bit errors in the data.

5.4.4.5 Errors in the END Flag

There is a one bit flag in the fragment header that indicates whether the fragment is the last fragment of the segment. The flag is protected by the header checksum so that at least one other bit error must occur in the header in order for an error in the END flag not to be detected. Since the flag is only one bit, the probability of the error occurring and not being detected by the header checksum is: $(10\text{-}1)\frac{1}{2}P_R{}^2 + P_B\ 2^{-1}\ 2^{-16} = 8\text{x}10^{-11}$.

If a bit error occurs such that the END fragment of a segment is changed to a non-END fragment, then the fragments of this segment will be merged with the fragments of the next segment. Merged segments were already discussed in the previous section, so there is no need to discuss this scenario further.

If a bit error changes a non-END fragment to an END fragment, then the resulting segment will be too short. (The portion of the original segment that follows the errored fragment should be recognized by the receiver as an invalid segment since there will not be a fragment with segment offset 0.) The segment checksum will fail to detect the error with probability 2^{-16}. Thus, the expected annual frequency of segments containing undetected non-END to END transitions per line is: $5\text{x}10^9\left((10\text{-}1)\frac{1}{2}P_R{}^2 + P_B\ 2^{-1}\ 2^{-16}\right)\ 2^{-16} = 6\text{x}10^{-6}$. This is negligible compared to $5\text{x}10^3$.

5.4.4.6 Offset Errors

Consider two consecutive fragments belonging to the same segment; call them fragment 1 and fragment 2. In order for the destination to join these two fragments together, the offset of fragment 2 must precisely equal the offset of fragment 1 plus the data length of fragment 1. (The length of the data portion of a fragment is determined by the fragment length field minus the fragment header length field.) Thus, in order for an error in the 13 bit offset field of fragment 2 to go undetected, there must also be errors in the header of fragment 1 (in either the 4 bit header length field or the 16 bit fragment length field). The fragment headers are each protected by a checksum; thus, overall at least two errors must occur in both fragments 1 and 2. If the error is not detected by IP, then an incorrectly reconstructed segment will be passed up to TCP. The segment checksum will fail to detect the error with

probability 2^{-16}. Thus, the expected annual frequency of segments containing undetected fragment offset errors per line is upper bounded by:

$$5x10^9\left(13\ (10\text{-}1)\ \frac{1}{2}P_R{}^2 + P_B 2^{-13} 2^{-16}\right)\left((16+4)(10\text{-}1)\ \frac{1}{2}P_R{}^2 + P_B\ (2^{-16}+2^{-4})2^{-16}\right)2^{-16}$$

$$= 4x10^{-16}$$

This is negligible compared to $5x10^3$.

5.4.4.7 Fragment Length Errors

Errors in the fragment length field or fragment header length field are least likely to be detected if they occur in the last fragment of a segment. In order for such an error not to be detected, there must be at least two bit errors in the header of the last fragment; errors are not needed in any other fragment header. If the header checksum fails to detect the error, a segment of incorrect length will be passed up to TCP. The segment checksum will fail to detect the error with probability 2^{-16}. Thus, the expected annual frequency of segments containing undetected length errors, per line, is about:

$5x10^9\left((16+4)\ (10\text{-}1)\ \frac{1}{2}P_R{}^2 + P_B\ (2^{-16} + 2^{-4})\ 2^{-16}\right)2^{-16} = 7x10^{-6}$. This is negligible compared to the expected frequency of undetected bit errors in the segment data.

5.4.4.8 Errors in Segment Header Length Field

The segment header is variable length. There is a 4 bit field in a fixed location in the segment header that indicates the size of the header. The length of the data portion of the segment is the length of the entire segment, as passed up to TCP by IP, minus the length of the header, as specified in the 4 bit field. If there is an undetected error in the header length field, the wrong amount of data will be accepted by the destination.

The segment checksum checks the entire segment, including the segment header. (The location of the checksum in the header is fixed, and is thus unaffected by an error in the header length field.) In order for the error in the segment header length not to be detected, there must be at least one other bit error in the segment. If the segment length is 1024 bytes, the probability of an error in the header length occurring and not being detected by the segment checksum is: $(4)\ (512\ \text{-}1)\ \frac{1}{2}P_R{}^2 + P_B\ 2^{-16} = 10^{-9}$. The expected annual frequency of segments with undetected segment header length errors per line is then: $5x10^9\ 10^{-9} = 5$.

Note that the pseudoheader plays no role in this error scenario. The length field included in the pseudoheader is the length of the entire segment [Com91]; thus, it is unaffected by the value of the header length field.

This scenario is very similar to the scenario where bit errors in the data are not detected by the checksum. In both cases, the segment checksum is not powerful enough to detect random bit errors in the segment. As stated earlier, a more powerful checksum or a CRC should be used.

5.4.4.9 Summary

In the sections above, we showed that bit errors in the data is the dominant cause of undetected errors. The expected annual frequency of segments containing undetected bit errors in the data is $5x10^{3}$, per line. All other error scenarios yield an expected annual frequency of undetected error of less than 10^{-5}. (We consider undetected bit errors in the segment header length field to fall under the same category as bit errors in the data.) Thus, the weak segment checksum severely affects the performance of the error detection scheme.

5.4.5 Step 5: Examine Single Fragment Segments

The last step in analyzing the error detection scheme is to examine error scenarios involving segments comprised of only one fragment. Specifically, we need to consider the case where the single fragment is misdirected or where it contains an error in its length field or its header length field.

5.4.5.1 Misdirected Fragment

If a single fragment segment is misdirected, the 16 bit ID field does not help in detecting the misdirection. The only mechanism for detecting the misdirected segment is the 32 bit destination address included in the pseudoheader. The pseudoheader is checked by the segment checksum. Thus, if the segment is misdirected, the checksum will essentially be checking random bits for the destination address. Thus, the segment checksum fails to detect the error with probability 2^{-16}.

Due to the fragment header checksum, however, misdirection is not a frequent occurrence. As shown in Section 5.4.1, we expect no more than two misdirected fragments per year, per line. Thus, we can upper bound the annual frequency of undetected misdirected single fragment segments by: (2) $2^{-16} = 3x10^{-5}$. This may be a very loose upper bound depending on the number of segments that are comprised of just one fragment.

Regardless, this undetected error rate is negligible compared to the expected frequency of undetected bit errors in the segment data.

5.4.5.2 Error in Length Field

Consider a bit error in the length field or header length field of the single fragment. Since the fragment is not being merged together with other fragments, IP can not rely on the offset fields of other fragments to help detect this error. If the error is not caught by the header checksum, the wrong sized segment will be passed up to TCP. Actually, this scenario is no different from the scenario considered in Section 5.4.4.7, where the last fragment of a segment has an error in its length field or header length field. As shown there, the fragment header checksum and segment checksum provide enough protection against this error scenario.

5.4.6 Summary of Error Protection in TCP/IP

Table 5.4 summarizes the above analysis of the current TCP/IP implementation. The figures in the table assume the following: data rate of 1.5 Mb/sec, segment length of 1024 bytes, fragment length of 576 bytes, segment header length and fragment header length of 20 bytes each. The probability of the underlying errors is assumed to be: $P_R = 10^{-6}$, $P_B = 10^{-5}$, and $P_C = 10^{-4}$.

5.5 CONCLUSIONS

Overall, the TCP/IP scheme follows most of the principles laid out in chapter three. A fragment checksum is used to prevent misdirection, a redundancy check is placed on the segment rather than on individual fragments, and fragment offset and ID fields are included so that fragments can be properly resequenced and stray fragments can be dropped. However, the choice of a 16 bit checksum as a redundancy check on the segment is not effective; it does not provide satisfactory protection against bit errors in the data. With our assumption that the random bit error rate is 10^{-6}, the current TCP/IP implementation does not succeed in reducing the number of undetected error events per year, per line to one or less. Rather, $5x10^3$ undetected error events can be expected per year, per line. A more powerful segment checksum should be used or a segment CRC should be used. We showed that a 32 bit CRC would provide sufficient protection. The drawback of the CRC is its software implementation complexity.

Another way of looking at this is if reliability is an important issue, then there needs to be more error protection than what TCP/IP provides. In practice, the subnetworks over which

Table 5.4

Error Type	Chief Cause	Expected Annual Freq. of Occurrence	Expected Annual Freq. of Undetected Error Using Current TCP/IP Error Detection Scheme
Bit Errors in Data	Burst error	10^5	1
	Random Bit Errors	$4x10^7$	$5x10^3$
Misdirected Fragment	Burst Error, Random Bit Error	2	$5x10^{-10}$
Out-of-sequence Fragments (after IP attempts to resequence)	Random Bit Error	.3	$5x10^{-6}$
Lost non-END Fragment	Congestion, Random Bit Error, Burst Error	$2x10^6$	10^{-9}
Merged Segments	Random Bit Error	$8x10^4$	$5x10^{-6}$
Error in END Flag	Random Bit Error, Burst Error	$3x10^4$	$6x10^{-6}$
Error in Fragment Offset	Random Bit Error	10^5	$4x10^{-16}$
Error in Frag. Length or Hdr. Length	Random Bit Error	$2x10^5$	$7x10^{-6}$
Error in Segment Header Length	Random Bit Error, Burst Error	$7x10^4$	5

the TCP/IP connection is run may provide their own error detection mechanisms that effectively reduce the bit error rate at the TCP/IP level. For example, local area networks, which often serve as a subnet in a TCP/IP connection, typically include a CRC as a

redundancy check. If the random bit error rate were 10^{-8} rather than 10^{-6}, then the goal of no more than undetected error event per year, per line is met.

Random bit errors play a more important role in TCP/IP than they do in ATM for two reasons. First, ATM systems are expected to be run over fiber optic systems, where the random bit error probability is very small. TCP/IP is designed to run on any type of network, so the bit error rate can be much higher. Secondly, in ATM, Extended Hamming code CRCs are used for redundancy checks on the cell header and the frame data; four bit errors must occur before the CRC will fail to detect the error. In TCP/IP, checksums are used for redundancy checks on the fragment header and segment data; just two bit errors can result in an undetected error. As is shown in Table 5.4 the chief cause of most errors in TCP/IP is random bit errors.

The TCP/IP scheme provides very good protection against all errors except bit errors in the data. Thus, if the TCP/IP scheme were modified to include a segment CRC, it would provide sufficient protection over a wide range of systems.

CHAPTER 6

RETRANSMISSION SCHEMES

6.1 INTRODUCTION

The previous three chapters dealt with designing effective error detection schemes. We specifically examined error detection at three layers of a network, which we call layers N, N-1, and N-2. Layer N operates end-to-end and is concerned with the integrity of the message as a whole. Layer N-1 also operates end-to-end but deals with errors on a packet level; it is responsible for dividing a message into packets at the source, and reconstructing the message at the destination. Layer N-2 is involved with routing packets on a node-by-node basis and deals with preventing routing errors. Using the error detection techniques discussed in these chapters, we assume that any message that is accepted by layer N at the destination will with very high probability be error-free.

In addition to error detection, a network protocol must provide a means of recovering from errors. Error recovery is the subject of the remaining chapters. This chapter analyzes retransmission schemes, where the source retransmits data that has not been successfully received by the destination. The overall goal of this chapter is to determine how the characteristics and resources of a network should affect the choice of retransmission schemes. Chapters seven and eight apply this analysis to ATM and TCP/IP systems, respectively.

It should be noted that not all connections require recovery from errors. For example, if a connection carries video or voice, it is not crucial that all messages be received intact. However, if a connection carries critical data, such as bank transactions, it is important that the destination be able to reproduce exactly what was sent by the source with very high probability.

Throughout this chapter, it is assumed that error recovery is performed only at layer N. The reasons for recovering from errors at layer N were enumerated in Section 3.2. For simplicity, we will consider a message to be the unit of retransmission. The actual unit of retransmission is a layer N SDU (a message is the layer N PDU), since different control information may be added when data is retransmitted. Unless otherwise noted, we use the term 'message' to refer to data messages, not control messages.

There are three specific scenarios that layer N needs to handle. First, a message may never reach layer N at the destination. This may be due to the message being lost during transmission, or due to layer N-1 dropping the message after detecting an error in reconstructing the message. Second, layer N itself may detect an error in a message. Third, messages may be passed up to layer N in a different order from which they were sent by the source.

If a message is not received by layer N, then the destination must inform the source of this so that the data can be retransmitted. The destination can send a control message indicating the message is lost; this is known as a negative acknowledgement (NACK). Or, the destination can implicitly indicate to the source that the message has been lost; for example, the lack of a positive acknowledgement (ACK) can be interpreted as a NACK. A large portion of this chapter focuses on various NACK and ACK strategies.

If layer N at the destination detects that a message contains an error, it has two options. It can drop the message and request that layer N at the source retransmit the data. Alternatively, layer N could attempt to correct the error. For example, it could use a CRC to correct bit errors in the data. However, as discussed in Section 3.4.2.2, using a CRC to correct errors results in a higher expected rate of undetected error. Correction of errors may be a viable alternative for certain applications, but we will not consider this option further.

There are two options commonly used to deal with messages delivered out-of-sequence. In one option, known as Go Back N, the receiver drops out-of-sequence messages, thereby necessitating their retransmission. In the selective repeat option, out-of-sequence messages are buffered at the receiver. When the missing messages arrive, the messages are properly resequenced before layer N delivers them to a higher layer. In Section 6.6 we discuss the tradeoffs involved in using Go Back N and selective repeat.

Throughout this chapter we assume that messages contain a sequence number so that lost or out-of-sequence messages can be detected. Also, ACKs contain the sequence numbers of the messages that are being acknowledged, and NACKs contain the sequence numbers of the messages that need to be retransmitted. Throughout this chapter, it is assumed that undetected errors will not occur in the sequence number of a message or the sequence numbers contained in ACKs or NACKs. Chapter nine briefly discusses the consequences if such undetected errors do occur.

6.1.1 Overview of Retransmission Schemes

There are two broad classes of retransmission schemes that we consider in this chapter: timer-based schemes and poll-based schemes. Such a classification is perhaps somewhat artificial since there are schemes that have aspects of both classes, as will be discussed in Section 6.5. Nevertheless, it is helpful to define this dichotomy so that we have a basis for comparing and contrasting many of the retransmission schemes that are currently in use or that have been proposed for future systems.

In timer-based schemes, the only information that the destination uses to determine if a message has been received properly is the message itself. The receiver does not use the arrival of one message to determine the status of any other message. Also, the receiver does not use control messages sent by the source to determine the status of data messages. In the most simple implementation of this type of scheme, the receiver sends an ACK to the source whenever it receives a message that is acceptable i.e., error-free, and in the case of Go Back N, in sequence. The source maintains a timer that indicates when it expects to receive an ACK for a particular message. If the timer expires without an ACK having been received, the source assumes the message has been dropped, and retransmits the message.

In poll-based schemes, information other than the message itself can be used to determine the status of the message. As with timer-based schemes, the arrival of message X may be used to generate an ACK of message X. In addition, however, message X may be used to determine the status of other messages. For example, assume we have a system where messages are expected to arrive in order. If message X arrives without message X-1 having arrived, the receiver can send a NACK of message X-1 since it is likely that message X-1 has been lost. In some sense, message X can be considered by the receiver to be a poll questioning the arrival of all messages with sequence number less than X. Another characteristic of poll-based schemes is that the source may send a control message

(i.e., an explicit poll message) indicating which messages it has sent and requesting that the receiver send a status message indicating which messages it has received.

High speed protocols currently under development, such as ATM and Xpress Transport Protocol (XTP), have chosen to use poll-based schemes.[CCI92b],[Pro89],[Saw90] Despite these proposals, very little analysis of the performance of these schemes has been done (or at least little has been published). We present a detailed analysis of poll-based retransmission schemes in Section 6.2. The main advantage of these schemes is that the source and receiver are more in synch with each other. The control messages exchanged between the source and destination can be used to eliminate some ambiguity as to what has been sent and what has been received. In some systems, it is possible to use this information to prevent unnecessary retransmissions. The disadvantages are the overhead involved with sending the control messages, and the delay in waiting for the control messages to arrive.

In Section 6.3, we present an analysis of timer-based schemes. These schemes are more commonplace, and have been analyzed more thoroughly in the literature.[Sch87], [Zha86] Thus, our analysis will be more qualitative, and will focus on comparisons with the poll-based schemes. The advantage of timer-based schemes is that they potentially may result in low delay. The disadvantage is that their performance depends on the source having a good estimate of the round trip delay between the source and destination; an overestimate may lead to increased delay and an underestimate may lead to an avalanche of unnecessary retransmissions.

6.1.2 Evaluation Criteria

The major evaluation criteria for retransmission schemes are delay and overhead. In the sections below, we use these criteria to evaluate poll-based schemes and timer-based schemes.

6.1.2.1 Delay

Given that a message is transmitted error-free, it incurs a delay consisting of the transmission delay in sending the message and the propagation delay from source to receiver. If the message is not received error-free and it needs to be retransmitted, then it also incurs a retransmission delay. The minimum retransmission delay consists of: the transmission delay in sending a control message (e.g., a NACK) from the receiver back to the source, the propagation delay from receiver to source, the transmission delay in

retransmitting the message, and the propagation delay from source to receiver. Given the assumption of end-to-end retransmissions, any retransmitted message will experience at least this much delay. Note that if the application is such that the data will be useless after this much delay, then there is no sense in trying to retransmit it.

In addition to this minimum retransmission delay, the actual retransmission delay also depends on the time it takes for the receiver to determine whether or not a data message needs to be retransmitted, and the time until the receiver sends a control message to the source indicating the status of the data message. As will be shown in the sections below, this additional delay, which we refer to as the excess delay, greatly depends on our choice of retransmission scheme.

6.1.2.2 Overhead

The second criterion we use to evaluate the various retransmission schemes is overhead. Again, this is comprised of several components. First, there is the overhead involved with sending control messages e.g., ACKs, poll messages, etc.. Second, there is the overhead due to unnecessary retransmissions. Third, there is the processing overhead at the source and destination. For example, there is the processing involved with maintaining timers, or involved with maintaining a resequencing buffer.

6.2 POLL-BASED RETRANSMISSION SCHEMES

In this section, we analyze retransmission schemes based on a polling mechanism. In the basic polling scheme, the source periodically sends poll messages to the receiver. The receiver responds to the polls by transmitting a status message indicating which data messages it has received. The source, in turn, uses the status message to determine which data messages can be released from the retransmission buffer and which data messages need to be retransmitted. We start off by analyzing this simple scheme, and then later on consider adding various options.

6.2.1 Analysis of Simple Polling Scheme

6.2.1.1 Assumptions

We look at the transmission of data messages from one source to one destination and consider the problem of retransmitting data that does not arrive correctly at the destination. We consider the round trip delay (RTD) of the connection to be the total propagation delay, queueing delay, and processing delay from source to destination and back. (It does not include delays due to the retransmission protocol; these will be discussed below.) In high

speed systems such as ATM, the RTD is expected to be almost constant. However, in low speed systems such as TCP/IP, the RTD may be highly variable. (This is discussed further in Section 6.3.1 below.) However, for our initial analysis, we will assume the RTD is fixed and that the delay from source to receiver is the same as from receiver to source.

We assume that the data rate of the connection is constant, and that the source always has data to send. Let R be the data rate of the connection, in terms of the number of packets transmitted per RTD. For example, a data rate of 100 Kb/sec, an RTD of 50 msec., and a packet size of 500 bits correspond to an R of 10. We assume data messages and control messages are transmitted at the same speed, and arrive at the destination in the same order in which they were sent by the source. As a point of reference, we refer to networks with an R of about 1 or smaller as low speed networks; networks with an R on the order of 100 or higher can be considered to be high speed networks.

We assume all data messages in the connection are comprised of N packets and that polls and status messages are comprised of one packet. We assume that the destination must wait until all packets of a data message have been received before it can decide whether the message is valid. Finally, we assume that once a poll is received, the receiver immediately responds with a status message.

6.2.1.2 Overhead and Delay of Polling Scheme

We let I equal the number of data messages that are sent in between sending polls. The I messages can be original transmissions or retransmissions. We examine how various choices of I affect the overhead and delay involved with the polling scheme.

A poll is comprised of one packet and one poll is sent after every NI data packets. Letting V equal the fraction of traffic from source to destination that is dedicated to polling, we see that:

$$V = \frac{1}{NI + 1} \qquad (6.1)$$

(We ignore the overhead in the reverse direction, but for each poll arriving at the destination there is a status message sent back to the source.)

Alternatively, we could look at the efficiency of the system, where efficiency is defined as 1-V. It is important to consider efficiency since this represents an upper limit on throughput. Thus, even if there are no losses in the system, the fraction of packets sent by the source that are ultimately delivered to the user as data can be no more than 1-V.

From the point of view of efficiency, we see that for a given N, we would like I to be as large as possible. The tradeoff, of course, is that the larger I is, the longer the delay until a lost message is retransmitted. Below, we further analyze the tradeoff between delay and efficiency.

In our analysis, the unit of time is the amount of time it takes to transmit one packet. Consider any data message and assume the transmission of the message starts at time $\mathcal{T}$. Since we assume the transmission rate and RTD are fixed, we know that the complete message is not expected to arrive at the destination until time $\mathcal{T}$+ N + .5R. (N is the transmission delay in sending the N packets of the message, and .5R is the delay from source to destination.) Since we assume that the destination cannot make a decision as to whether or not a message needs to be retransmitted until it receives the entire message, we see that $\mathcal{T}$+ N + .5R represents the earliest time it could make such a decision. (Since data is assumed to travel at a fixed rate, a lost message can actually be detected earlier than time $\mathcal{T}$+ N + .5R; for simplicity, we assume that the receiver must always wait until this time before deciding the message needs to be retransmitted.)

Let $\mathbb{E}$ be the delay from time $\mathcal{T}$+ N + .5R until the time a status message is sent from the receiver to the source indicating the status of this data message. For the simple polling scheme we described above, a status message will not be sent until a poll is received. $\mathbb{E}$ is not the same for all data messages; it depends on how soon a poll is sent after the particular data message. Figure 6.1 depicts $\mathbb{E}$ for an arbitrary data message.

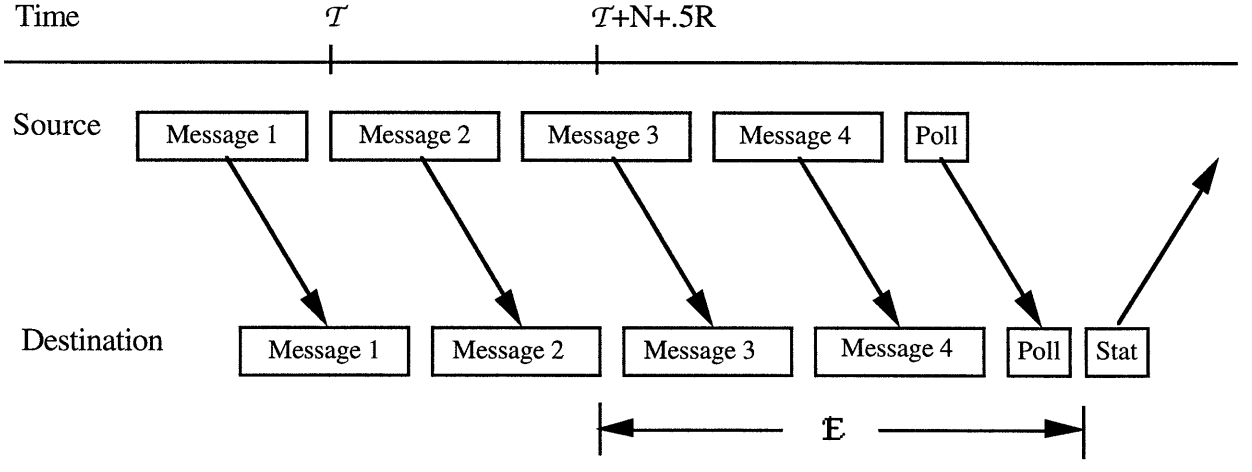


Figure 6.1 $\mathbb{E}$ represents the delay from the arrival time of message 2 to the time the destination begins to transmit a status message indicating the status of message 2.

The expected value of $\mathcal{E}$ for the simple polling scheme, assuming that poll messages are never dropped, is:

$$E = \text{Expected Value of } \mathcal{E} = \frac{(I-1)N}{2} + 1 \quad (6.2)$$

The '1' term represents the transmission delay of sending the poll message.

Now let's specifically consider the case where the data message needs to be retransmitted. At time $\mathcal{T} + N + .5R + \mathcal{E}$, a status message is sent NACKing the errored data message. Due to the transmission delay of the status message and the delay from receiver to source, the NACK arrives at the source at time $\mathcal{T} + N + R + \mathcal{E} + 1$. We assume the source retransmits the data message as soon as it receives the NACK. (If more than one message is NACKed in the status message then obviously only one of them can be retransmitted immediately; the others will incur greater delay. We discuss this further in the section on burst losses.) The retransmitted message is expected to arrive at the destination at time $\mathcal{T} + 2N + 1.5R + \mathcal{E} + 1$. If the original transmission of the message had not been dropped, it would have arrived at the destination at time $\mathcal{T} + N + .5R$. Thus, we see that the total delay due to retransmission is:

$(\mathcal{T} + 2N + 1.5R + \mathcal{E} + 1) - (\mathcal{T} + N + .5R) = N + R + \mathcal{E} + 1.$

We refer to this as the retransmission delay.

Given that the transmission rate and RTD are fixed, then the minimum possible retransmission delay is: $N + R + 1$ i.e., $\mathcal{E}$ equals 0. This could potentially be achieved, for example, by a timer-based scheme if the source knows the RTD precisely: when transmission of a message begins, a timer is initialized to 0; if an ACK of the message has not been received after $N + R + 1$ time units (the '1' term is due to the transmission of the ACK) the message is retransmitted.

In the polling scheme, $\mathcal{E}$ is not 0. $\mathcal{E}$ can be viewed as the excess retransmission delay. The actual value of $\mathcal{E}$ is not the crucial factor to consider, but rather the relative value of $\mathcal{E}$ compared to the minimum retransmission delay. Thus, we define the relative excess delay $\mathcal{D}$ to be:

$$\mathcal{D} = \frac{\mathcal{E}}{N + R + 1} \quad (6.3)$$

Before proceeding, we discuss why the relative excess delay is what is important. First, consider a high speed network. High speed networks such as ATM and XTP are likely to use a selective repeat scheme, thus necessitating a resequencing buffer at the destination. The retransmission delay affects how large the buffer should be. The relative excess delay $\mathfrak{D}$ affects the relative extra buffer space that is needed. Consider a simplistic example where if $\mathfrak{D}$ were 0, the buffer space would need to be 10 Mb. and if $\mathfrak{D}$ were 10%, the buffer space would need to be 11Mb. Even though the size of the increment, 1Mb., is large, there is not that much difference in requiring a 11Mb. buffer over a 10Mb. buffer. From equation (6.3), we see that as the data rate increases, $\mathfrak{D}$ tends toward $\frac{\mathfrak{E}}{R}$. Thus, at very high speeds, we are essentially looking at the excess delay relative to the round trip delay.

Next, consider a low speed network. Typically, on low speed networks, Go Back N is used rather than selective repeat. One of the chief concerns on a low speed network is the amount of bandwidth that is used. If a data message is lost and Go Back N is used, then the minimum amount of wasted bandwidth will be N + R + 1 packet transmissions. $\mathfrak{D}$ represents the relative additional amount of wasted bandwidth. We see from equation (6.3) that if R is very small compared to N, then $\mathfrak{D}$ is close to $\frac{\mathfrak{E}}{N}$. Thus, at very low speeds, we are essentially looking at the excess delay compared to the transmission delay of one message.

By looking at the relative excess delay, we favor schemes where $\mathfrak{E}$ is small compared to the minimum retransmission delay. This may not be the most important criterion, however, for connections carrying delay sensitive traffic. For these connections, overall delay is the primary concern; how the delay is apportioned between the minimum retransmission delay and the excess delay is of secondary importance. For example, a scheme where the excess delay and the minimum retransmission delay are equal and small in magnitude may be appropriate for delay sensitive traffic. However, it would not appear to be a good scheme using the relative excess delay criterion. Nevertheless, for many connections, the relative excess delay criterion is a meaningful measure of performance.

Returning to our analysis, we combine equations (6.2) and (6.3) to derive the expected relative excess delay for the polling scheme:

$$D = \text{Expected Value of } \mathbb{D} = \frac{(I-1)N + 2}{2N + 2R + 2} \qquad (6.4)$$

In practice, it makes sense to use only integral values of I. A poll sent in the middle of a data message generates the same status information as a poll sent at the beginning of the data message, but generates it at a later time. Thus, only those values of D that correspond to integral values of I can be attained. The minimum possible relative excess delay is attained when I equals 1, which corresponds to sending a poll after every message.

To see the tradeoff between expected relative excess delay and overhead as a function of N and R, we combine equations (6.1) and (6.4) to get:

$$V = \frac{1}{D(2N + 2R + 2) + N - 1} \qquad (6.5)$$

This tradeoff is plotted in Graphs 6.1, 6.2, and 6.3 for three values of R. As stated above, only certain values of D can be attained due to the integer constraint on I. For R equal to .1 and R equal to 10, we plotted the discrete points that correspond to integral values of I. For R equal to 1000, the discrete points were close enough together that we plotted a continuous curve.

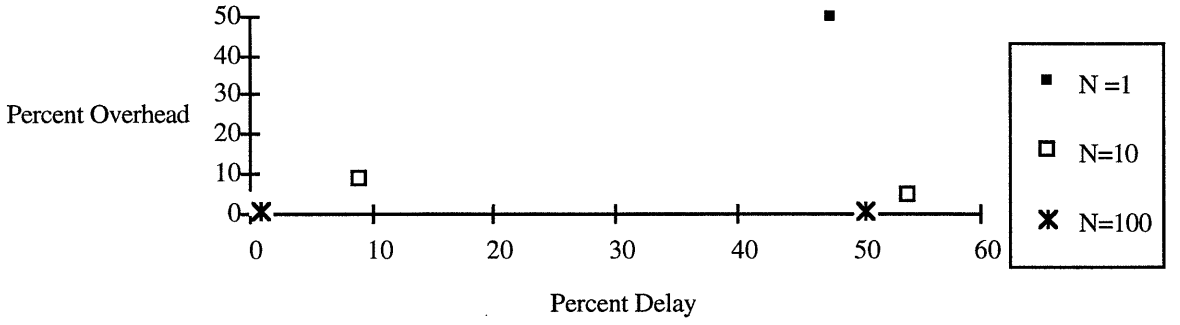
For R equal to .1 and N equal to 1, the minimum expected relative excess delay attainable is roughly 50%. (If the RTD is 50 msec, and the packet size is 50 bytes, then an R of .1 corresponds to a data rate of 850 bits/sec.) This is not surprising since for these parameters, the size of the poll and the size of the data message are the same, and the RTD is relatively negligible. For R equal to .1 and N equal to 10, we can achieve no better than about a 10% expected relative excess delay. We see that if N is as large as 100, then it is possible to attain a small expected relative excess delay. Based on Graph 6.1, we conclude that for very low speed networks, and very small messages, a polling scheme probably will not provide good delay performance during times of retransmission.

For R equal to 10, we can attain reasonable expected values of relative excess delay (i.e., 10% or less) for any value of N. The overhead for the case of N equal to 1 is still high.

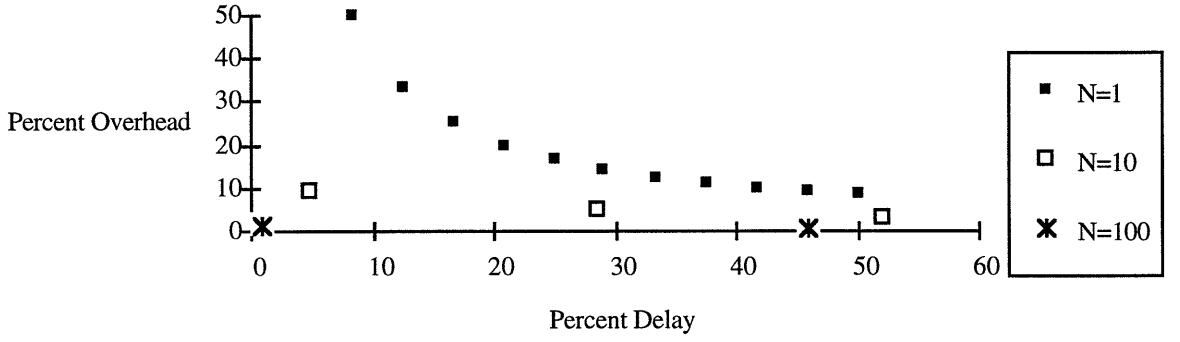
As R gets larger, the overhead V tends towards $\frac{1}{2RD}$ and is thus independent of N. This is illustrated on Graph 6.3 where R equals 1000. For this value of R, we see that we can

attain a 5% expected relative excess delay with less than 1% overhead, regardless of the size of the message.

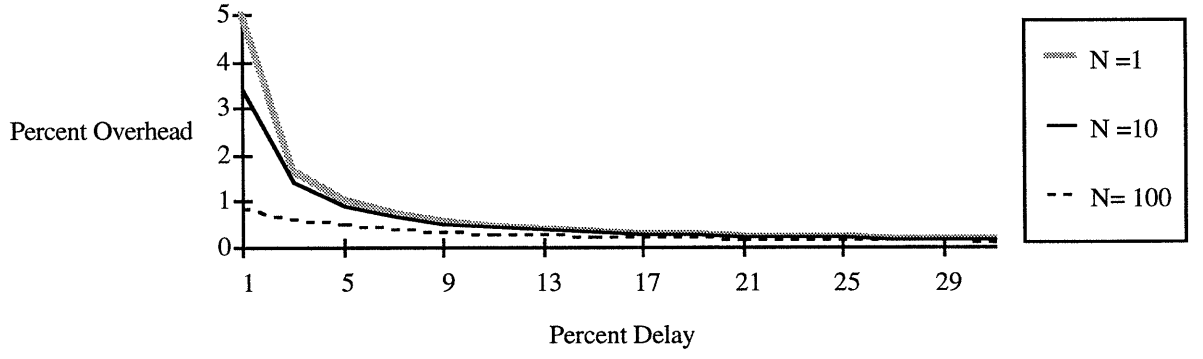
Graph 6.1 Percent Overhead vs. Percent Expected Relative Excess Delay R=.1



Graph 6.2 Percent Overhead vs. Percent Expected Relative Excess Delay R=10



Graph 6.3 Percent Overhead vs. Percent Expected Relative Excess Delay R=1000



Note that the polling overhead may not be a critical factor, since, as will be discussed in section 6.2.3, it may be possible to piggyback the polls on data messages so that the overhead is reduced to almost zero.

6.2.1.3 Choosing Polling Parameters

Assume a data connection requires that the expected relative excess delay be no more than some D_{max}. Rearranging equation (6.4), the source can calculate the appropriate value for I by:

$$I = \left\lfloor \frac{D_{max}(2N + 2R + 2) + N - 2}{N} \right\rfloor \qquad (6.6)$$

(where $\lfloor \bullet \rfloor$ indicates the integer portion)

Some of the polling scheme proposals refer to sending a certain number of polls per RTD rather than sending a poll after every I data messages. Given the target value D_{max}, and letting the number of polls that should be sent per RTD be F, then:

$$F = \frac{R}{N\left\lfloor \frac{D_{max}(2N + 2R + 2) + N - 2}{N} \right\rfloor + 1} \qquad (6.7)$$

As R increases for a fixed N, we see that I increases toward $\frac{2D_{max}R}{N}$ and F increases toward $\frac{1}{2D_{max}}$. Thus, as R gets large, more data messages are sent in between poll transmissions but more polls are sent per RTD. Also, the polling rate in terms of polls per RTD becomes independent of N.

If the connection is such that the total delay in terms of seconds is the important criterion, then the target value for the expected relative excess delay would change as R varies. The expected relative excess delay can be expressed as:

$$D = \frac{\text{Expected Excess Delay}}{\text{RTD + Message Transmission Time + Packet Transmission Time}}$$

where the unit of time is seconds. (In equation 6.3, all times were expressed in units of 'time to transmit one packet'.) If the data rate of the connection increases, then the transmission time of a packet (and of a message) decreases. Thus, a larger expected excess delay can be tolerated without an increase in the overall retransmission delay (we assume the RTD is fixed.) We conclude that for delay sensitive connections, as R increases the maximum allowable D also increases.

6.2.1.4 Worst Case vs. Average Case

Above we considered the average excess delay; we should also consider the worst case excess delay. Let E equal the expected excess delay as defined in equation (6.2). Assuming polls and status messages are not lost, then the worst case scenario in terms of delay is when the message transmitted immediately after a poll is lost. In this case, the delay is approximately double that of the average case. This is shown in Figure 6.2.

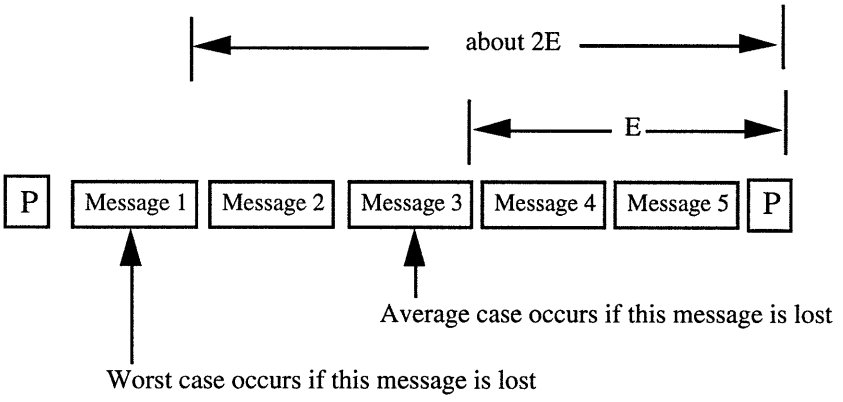


Figure 6.2 Average and worst case delay if poll and status message are not lost.

We should also consider the possibility of polls and status messages being dropped. Again, let E equal the expected excess delay as defined in equation (6.2). Define a cycle to be the time between poll transmissions; thus, a cycle is approximately equal to 2E. If either a poll or status message is lost, then the source needs to wait an extra cycle before it can determine the status of the receiver. The average delay (where the average is taken over the cycle before the lost poll or status message) increases to about 3E, and the worst case delay (given that just 1 poll or status message is lost) increases to about 4E. This is depicted in Figure 6.3. Obviously, the delay increases even more if two or more consecutive poll or status message are lost.

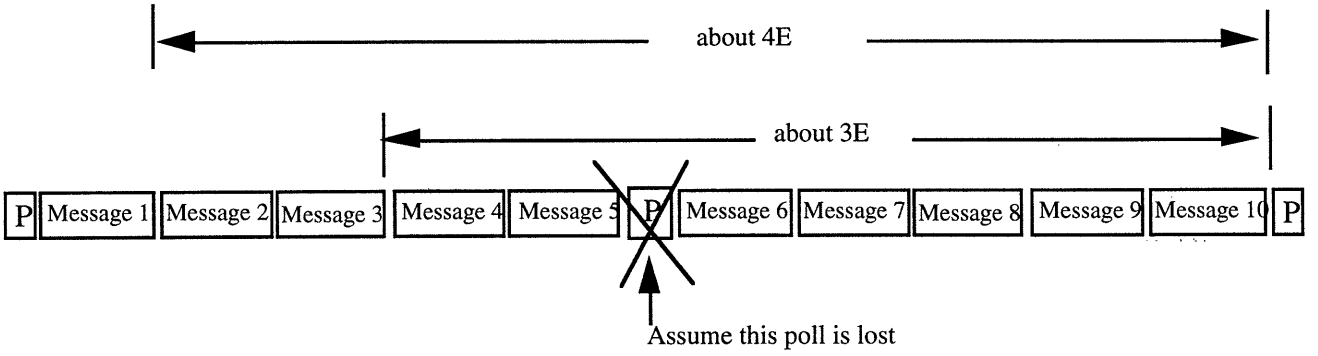


Figure 6.3 Average and worst case delay if one poll or status message is lost.

6.2.1.5 Burst Losses

In the above analysis, we assumed that the loss statistics were such that a status message never generated more than one retransmission. Thus, when the status message is received at the source, any NACKed message can be immediately retransmitted. In this section, we consider burst losses, where many consecutive messages are lost. First, consider the case where message X and message X+1 are both NACKed in the same status message. Message X+1 will not be retransmitted immediately; it will be delayed by an extra N time units while message X is retransmitted before it.

In the worst case, all I data messages that are sent in between poll transmissions are dropped. Refer to Figure 6.4, where we assume I equals 3. Assume the poll that follows the dropped messages is received correctly, and the corresponding status message is received at the source. This status message will NACK all I messages. From the figure, we see that for all I messages, there will be a delay of NI + R + 2 time units from the time the message is originally transmitted until the time it is retransmitted. Since the minimum possible retransmission delay is N + R + 1, each message suffers an excess delay of (I-1)N + 1. Above, where we assumed only one of the I messages is lost, the expected excess delay is $\frac{(I-1)N}{2} + 1$.

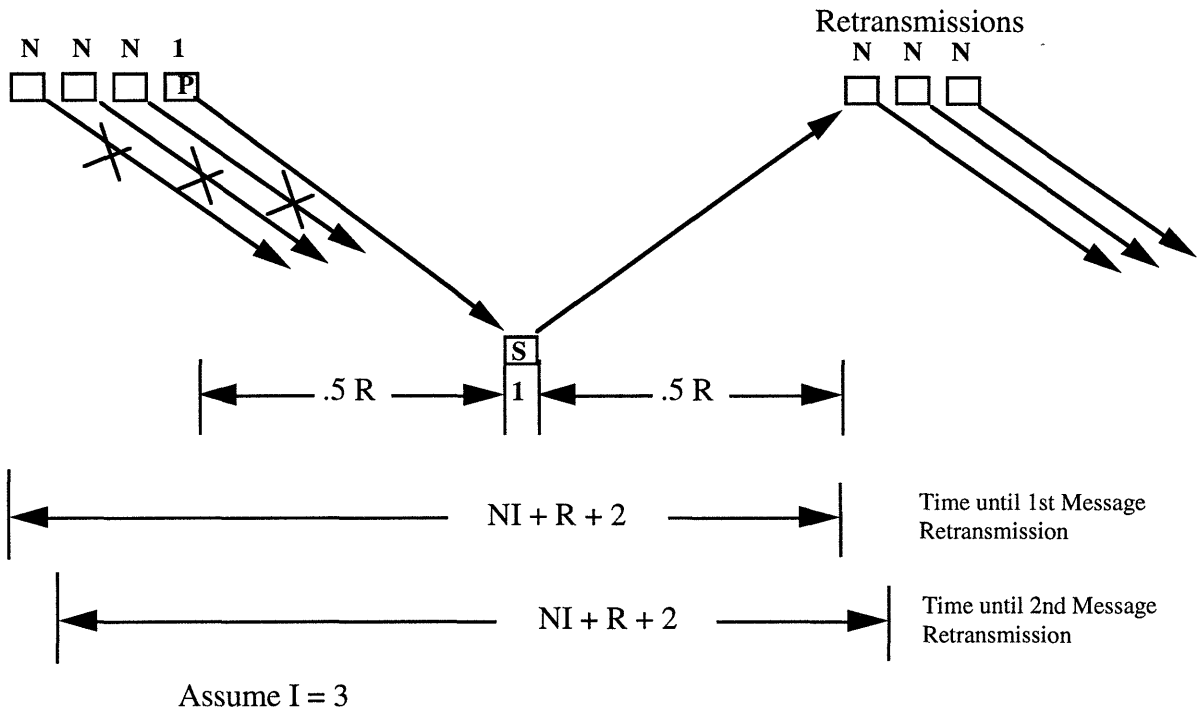


Figure 6.4 Delay if all I data messages are lost.

If the burst loss is due to a burst error along the data path, then it is likely the poll messages will be lost also. Or, if the burst loss is due to congestion, and polls do not have a higher priority than data, then it is likely that polls will be dropped. If no polls or data messages arrive at the receiver, then no status messages will be sent. The source cannot determine which messages have been lost until a status message arrives. Thus, the lack of status messages will delay the retransmission process.

Assume a burst loss starts at time $\mathcal{T}$ and ends at time $\mathcal{T}$+ B, where B is less than R + N + 1, i.e., all transmissions from time $\mathcal{T}$ to $\mathcal{T}$+ B are dropped. Define a successful poll as one that arrives successfully at the receiver and whose corresponding status messages arrives successfully at the source. Assume the first successful poll after the burst ends is sent at time $\mathcal{T}$+ B + δ, as shown in Figure 6.5. The first retransmission occurs at time $\mathcal{T}$+ B + δ + R + 2 (the '2' term is due to the transmission time of the poll and of the status message). If the excess delay had been 0, the first retransmission would have been sent at time $\mathcal{T}$+ R + N + 1. Thus, in this scenario, the polling scheme results in an excess delay of:

$$B + \delta + 1 - N \qquad (6.8)$$

and a relative excess delay of:

$$\frac{B + \delta + 1 - N}{R + N + 1} \qquad (6.9)$$

If the first poll sent after the burst ends is successful, then δ lies between 0 and NI.

We see that the excess delay in this scenario has two main components: the delay until the burst error ends (i.e., B) and the delay until a poll is sent (i.e., δ). In Section 6.2.1.2, where we discussed the scenario of just a single message being dropped (as opposed to a burst of messages being lost), the excess delay was composed of the delay until a poll is sent. Thus, when burst errors occur, and the length of the burst error is shorter than R + N + 1, the excess delay essentially increases by the length of the burst error. (The actual amount of the increase in excess delay will depend on where in the polling 'cycle' the burst error starts and ends.)

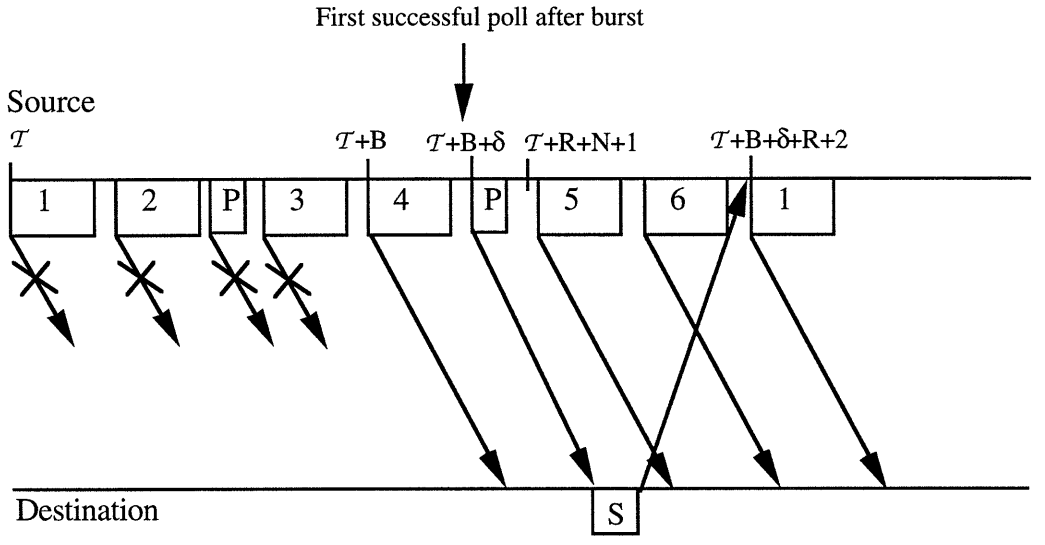


Figure 6.5 Ideally, message 1 would be retransmitted at time $\mathcal{T}$+ R + N + 1. However, message 1 will not be retransmitted until a poll sent after it reaches the destination successfully and triggers a successful status message. Thus, the retransmission mechanism is susceptible to error in either direction.

Next, consider the case where B is greater than R + N + 1. It would not be desirable to retransmit the first lost message at time $\mathcal{T}$+ R + N + 1, since it would be lost again as part of the burst loss. Ideally the first retransmission would be sent at time $\mathcal{T}$+ B; thus, the minimum retransmission delay is B time units. (Of course, it would be difficult to attain this minimum delay since the source would have to know precisely when the burst error is going to end, and start retransmitting at that time.) Refer to Figure 6.6. Assume that the

first successful poll after the burst ends is sent at time $\mathcal{T}$+ B + δ. The first retransmission will occur at time $\mathcal{T}$+ B + δ + R + 2 rather than at time $\mathcal{T}$+ B. Thus, in this scenario, the excess delay is:

$$\delta + R + 2 \qquad (6.10)$$

and the relative excess delay, according to the modified definition, is:

$$\frac{\delta + R + 2}{B} \qquad (6.11)$$

If the first poll sent after the burst ends is successful, then δ lies between 0 and NI.

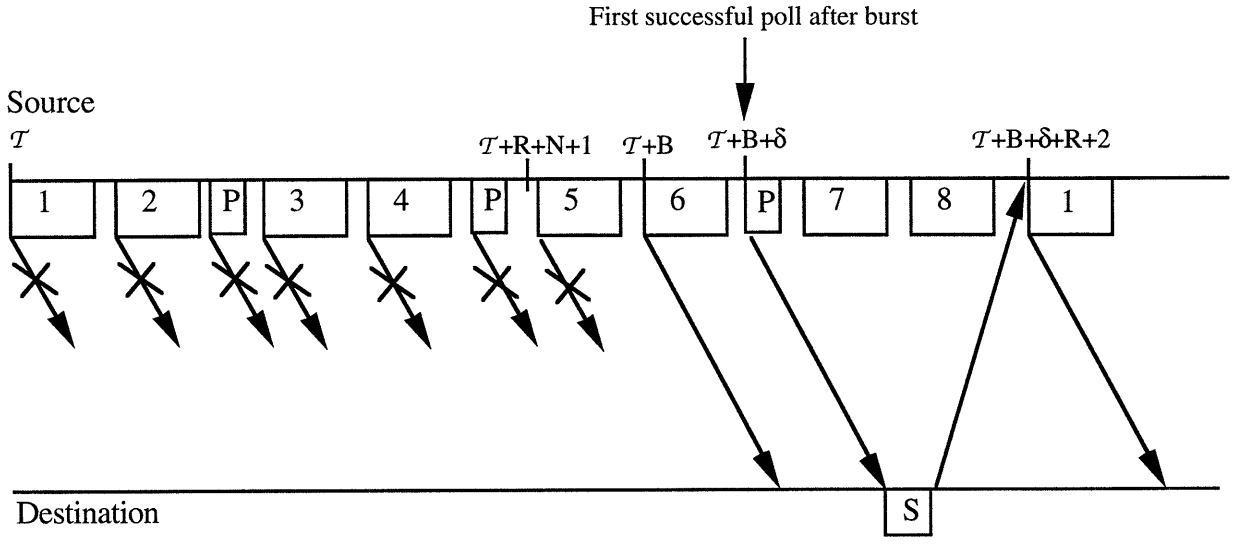


Figure 6.6 It is not desirable to retransmit message 1 at time $\mathcal{T}$+ R + N + 1, since it would be dropped again. Ideally, it would be retransmitted at time $\mathcal{T}$+ B. However, the retransmission will not occur until a poll reaches the destination and its corresponding status message reaches the source.

The excess delay in this scenario has two main components: the delay until a poll is sent (i.e., δ) and the round trip delay until the corresponding status message is received (i.e., R). In all of the other scenarios discussed thus far, the delay in waiting for the status message to arrive was part of the minimum retransmission delay and not the excess delay (i.e., in other scenarios the source needed to wait a minimum of one round trip delay in order to allow feedback from the destination time to arrive; in this scenario, we assume that the duration of the burst error provides enough time for any feedback to reach the source so that ideally the source would start to retransmit as soon as the burst error ends.) Thus, compared to the scenario where single messages are lost, the excess delay for the long burst error scenario essentially increases by the amount of one round trip delay. (The

actual amount of the increase in excess delay will depend on where in the polling 'cycle' the burst error starts and ends.)

As we can see from these examples, burst losses result in an increase in the excess delay. This increase in excess delay is desirable in the following situation: assume the burst loss is due to congestion and assume window-based flow control is used. Window-based flow control does not exercise control over retransmissions; the source dumps a retransmission into the network whenever a NACK arrives. If status messages are not triggered in the polling scheme, then NACKs will not be received by the source. This provides more time for the congestion to dissipate.

If rate-based flow control is used, however, the increase in excess delay is not desirable. In rate-based flow control, the source sends α messages per second, for some α, whether the messages are original transmissions or retransmissions. Thus, the lack of status messages will not decrease the traffic being sent into the network. It just means that until status messages arrive NACKing the lost data, the data messages that are sent by the source will be original transmissions rather than the necessary retransmissions. Of course, it is possible the source will decrease its data rate to β messages per second after it has not received a status message for a long time. But given that it is sending β messages per second, it would be preferable from the standpoint of delay that these be the necessary retransmissions rather than new transmissions (since layer N at the receiver must deliver the messages in proper sequence). Thus, the fact that status messages are not being sent by the receiver is undesirable in this scenario. (However, note that a rate-based scheme would likely be implemented with some sort of restriction on the transmission window so that the lack of NACKs in the polling scheme would eventually lead to the end of the send window being reached; at this point the polling scheme would be beneficial if congestion is still present in the network.)

If the burst loss is due to a burst error along the data path, then the long delay until a status message is sent is undesirable.

6.2.1.6 Adjusting the Polling Rate

In the analysis above, we assumed that the data traffic was steady. Now let's consider the case where the data traffic is bursty. It seems reasonable to send a poll at the end of the data burst rather than waiting for the poll timer to expire. After sending the end-of-burst

poll, subsequent polls should probably be sent at a lower frequency until data starts to be sent again. If the end-of-burst poll is lost and subsequent polls are sent infrequently, then the excess retransmission delay will increase. However, since no new data messages are being sent, the increased delay will not lead to an overflow of the resequencing buffer in the case of selective repeat or wasted packet transmissions in the case of Go Back N. If the connection is delay sensitive, then the poll frequency probably should not be decreased. Once all data messages have been ACKed and delivered at the destination, no further polls need to be sent until more data is transmitted.

6.2.1.7 Preventing Unnecessary Retransmissions

As mentioned in the introduction of the chapter, one of the chief advantages of using a polling scheme is that unnecessary retransmissions can be prevented. This is only possible if messages arrive at the receiver in the same order in which they are transmitted. The general idea is that if a poll is sent immediately after message number X, then by the time the poll arrives at the destination, all messages with sequence number less than or equal to X should have arrived already. Thus, the receiver knows that if any of these messages have not arrived, they need to be retransmitted. (It is assumed the poll contains some indication that all messages up to sequence number X have been transmitted.)

Of course, a second poll could arrive at the destination before the retransmissions have been received. This would cause a second NACK of the lost messages, which could lead to unnecessary retransmissions. To avoid this, there must be some way of associating a NACK with the poll that generated it, and there must be some way of determining whether a retransmission has been sent before or after the poll. Poll sequence numbers can be used to accomplish this. In the next chapter when we discuss the proposed ATM poll-based scheme, we will provide the full details of how poll sequence numbers can be used to prevent unnecessary retransmissions.

6.2.1.8 Estimating the RTD

From equation (6.7), we see that in order to determine the frequency of polls, the source needs to know the value of R, where R is defined as the number of packets transmitted in one RTD. Above, we assumed that the RTD remains constant throughout the connection. In reality, the RTD will change due to varying queueing delays along the data path. Thus, the source needs to monitor the delay between sending a poll and receiving an associated status message in order to estimate the RTD. (Note that for high speed networks such as

ATM, queueing delays are expected to be small relative to the propagation delay so that the RTD is approximately constant.)

If the estimate of the RTD is too high, the excess delay will be higher than intended. If the estimate is too low, the overhead will be higher than necessary. It is important to note that even if the source incorrectly estimates the RTD, unnecessary retransmissions can be prevented assuming data travels in sequence. As will be shown in Section 6.3, this is not the case for timer-based schemes.

6.2.2 Unsolicited Status Messages

One feature that has been included with several of the proposed polling schemes (e.g., ATM and Xpress Transport Protocol) is that in addition to sending status messages in response to polls, the receiver is permitted to send a status message whenever it determines that a data message has been lost. We will adopt the terminology of ATM and refer to these additional status messages as unsolicited STATs. From this point on, we will refer to status messages sent in response to polls as solicited STATs.

The following example demonstrates why unsolicited STATs may be desirable. Consider a network where data is expected to travel in order. If message number 2 arrives and message number 1 has not arrived, the receiver can assume message number 1 has been lost. With the unsolicited STAT option, the receiver can immediately send a status message requesting the retransmission of message number 1 rather than waiting for a poll to arrive.

Another situation in which an unsolicited STAT potentially could be sent is if a message arrives at the destination and layer N detects an error in the message (or layer N-1 detects an error and informs layer N). The destination could conceivably send an unsolicited STAT NACKing this errored message. However, since the message is in error, we assume that the destination cannot reliably determine the correct message sequence number; thus, it may NACK the wrong message. Thus, we assume that the destination does not send an unsolicited STAT in this situation. It must wait for the next correctly received message or poll before sending a STAT.

6.2.2.1 Expected Delay When Using Unsolicited Status Messages

In this section we calculate the average value of E for a lost message when unsolicited STATs are implemented, where E is the excess retransmission delay as defined in Section 6.2.1.2 above. We make the following assumptions: the data message following the lost

message arrives correctly at the receiver, data is being sent at a steady rate, and polls and status messages are not lost. The value of E depends on which message has been lost (see Figure 6.7). If the lost data message is immediately followed by a poll, then the lost message is NACKed in a solicited STAT and E equals one (the unit of time is the time to transmit one packet). In all other cases, the lost data message will be initially NACKed in an unsolicited STAT. The unsolicited STAT is sent after the destination receives and reassembles the next data message; thus, E equals N.

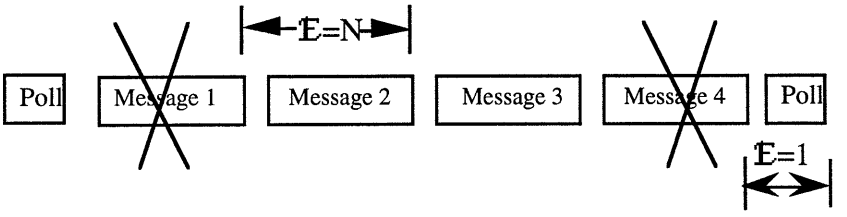


Figure 6.7 Excess retransmission delay when unsolicited STATs are used. If message 1 is lost, it will be NACKed in an unsolicited STAT; if message 4 is lost, it will be NACKed in a solicited STAT.

The expected value of E is thus:

$$E = \text{Expected Value of } E = \frac{(I-1)N}{I} + \frac{1}{I} \qquad (6.12)$$

Again, we are interested in the expected relative excess delay, D. Combining equations (6.3) and (6.12), we get:

$$D = \frac{(I-1)N + 1}{I(N + R + 1)} \qquad (6.13)$$

6.2.2.2 Benefit of Using Unsolicited Status Messages

The two potential benefits of using unsolicited STATs are decrease in retransmission delay and decrease in polling overhead. These are discussed below.

6.2.2.2.1 *Decrease in Delay*

First, we examine using unsolicited STATs to decrease the excess retransmission delay. Assume polls are sent after every I data messages. Comparing equations (6.4) and (6.13), we see that the delay benefit in using unsolicited STATs is greater the larger the value of I. This can be seen on Graphs 6.4 and 6.5.

If I equals 1, a poll arrives after each data message; thus, the unsolicited STAT option would be unnecessary (unless a poll were lost). For some low to medium speed connections, I may need to be 1 in order to meet a certain delay goal. For example, assume it is desired that the expected relative excess delay be no more than 20%, and assume R is 10 and N is 10. From Graph 6.4, we see that I needs to be 1 in order to keep the expected relative excess delay below 20%. There would be no need to implement unsolicited STATs. In general, given the desired maximum expected relative excess delay D_{max}, and the size of the message N, we can determine how high R must be before unsolicited STATs are beneficial. Setting I equal to 2 and solving for R in equation (6.13) yields:

$$R \geq (N+1)\left(\frac{1}{2D_{max}} - 1\right) \quad (6.14)$$

The larger N is, the larger R needs to be in order for unsolicited STATs to provide a delay benefit. This is expected: as N increases, the transmission delay in sending a message increases (for a fixed R); thus, the delay in waiting for message X+1 to arrive in order to determine that message X has been lost increases.

At high speeds, even as I increases the relative excess delay remains small (see Graph 6.5); thus, it is likely the polling mechanism would be implemented with I greater than 1. Thus, unsolicited STATs would provide a delay benefit for such connections. However, the reduction may be very small, e.g., on Graph 6.5, if I equals 6, the reduction in delay is less than 2%.

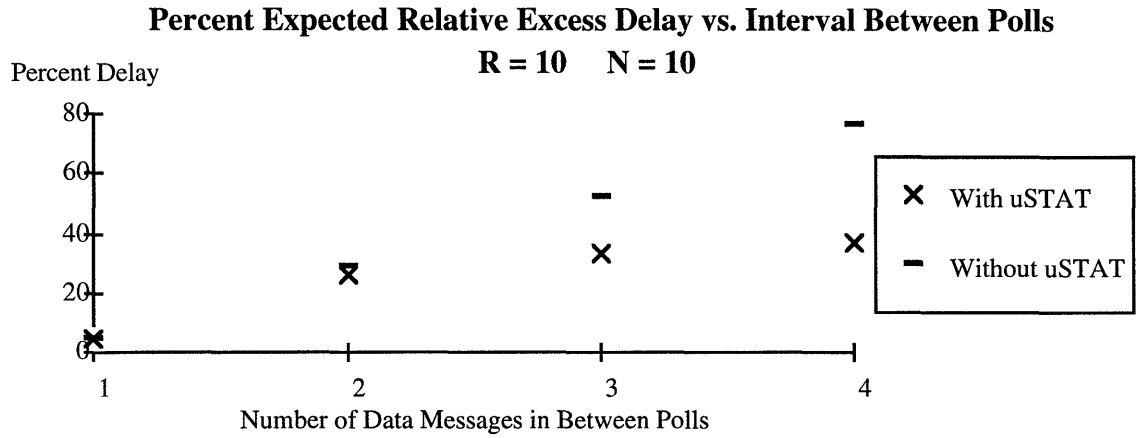
Note that our calculations are based on the assumption that the message following the lost message arrives intact. The larger N is, the higher the probability a data message will be lost. Thus, this assumption is less likely to hold as N increases, thereby mitigating some of the delay benefit when N is large.

Also, note that equations (6.12) and (6.13) pertain to the case where random losses occur. If a burst of messages is lost, the excess delay will increase, as was described in Section 6.2.1.5. A retransmission will not occur until either a data or poll message gets through to the destination and its corresponding status message arrives at the source.

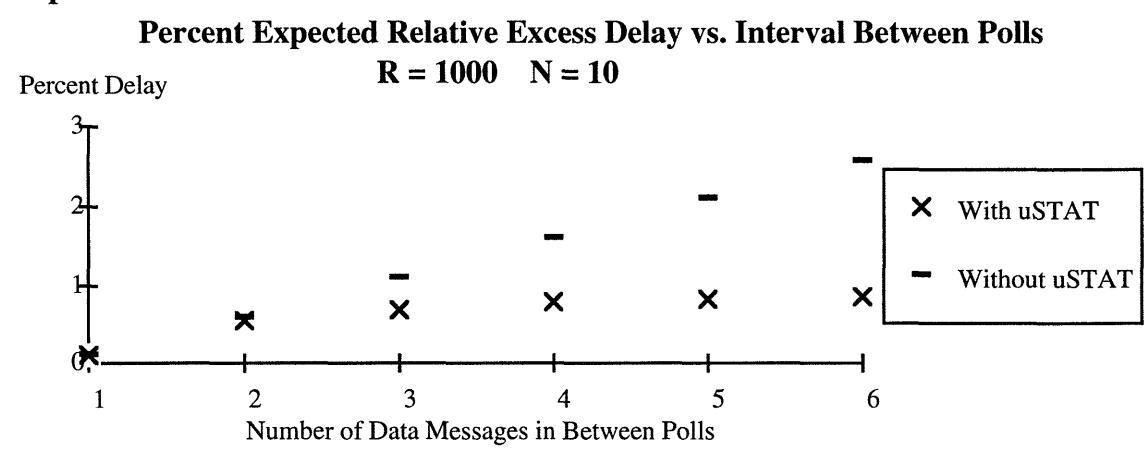
If unsolicited STATs are added to a scheme and the polling frequency is not decreased, then the overhead will be higher compared to a solicited STAT-only scheme since more STATs will be sent. In the worst case, every other data message is lost and a STAT would

be sent after each received data message. In most cases, however, we expect the overhead increase to be small.

Graph 6.4



Graph 6.5



6.2.2.2.2 *Decrease in Overhead*

Unsolicited STATs can also be viewed as a means of reducing the polling overhead. For example, assume R is 10 and N is 10 and assume the desired maximum expected relative excess delay is 40%. Then, as shown on Graph 6.4, the interval between polls can be increased from 2 to 4 messages if unsolicited STATs are used.

However, there are problems with reducing the polling frequency too much. We assume that unsolicited STATs only NACK messages that have not been NACKed previously (this is how they work in ATM). Thus, if an unsolicited STAT is lost, the messages it NACKed

will not be NACKed again until a solicited STAT is sent. Thus, we cannot increase I by too much or else the relative excess delay will be very large if an unsolicited STAT is lost.

To determine how much to decrease the polling frequency, we could use the following criterion: if an unsolicited STAT is lost, we want the expected excess delay to be no larger than the expected excess delay if a solicited STAT is lost in a solicited STAT-only scheme. Refer to Figure 6.8. Let $\mathfrak{F}$ be the polling frequency in the solicited STAT-only scheme, and let E be the expected excess delay if no polls or status messages are lost. If a solicited STAT is lost in the solicited STAT-only scheme, the expected delay becomes about 3E (this was discussed in Section 6.2.1.4.). In order for the unsolicited STAT scheme to achieve this same expected excess delay in the case of an unsolicited STAT being lost, we need the polling frequency in the unsolicited STAT scheme to be at least about $\frac{\mathfrak{F}}{3}$. Thus, we can expect no more than a threefold decrease in polling overhead when using unsolicited STATs.

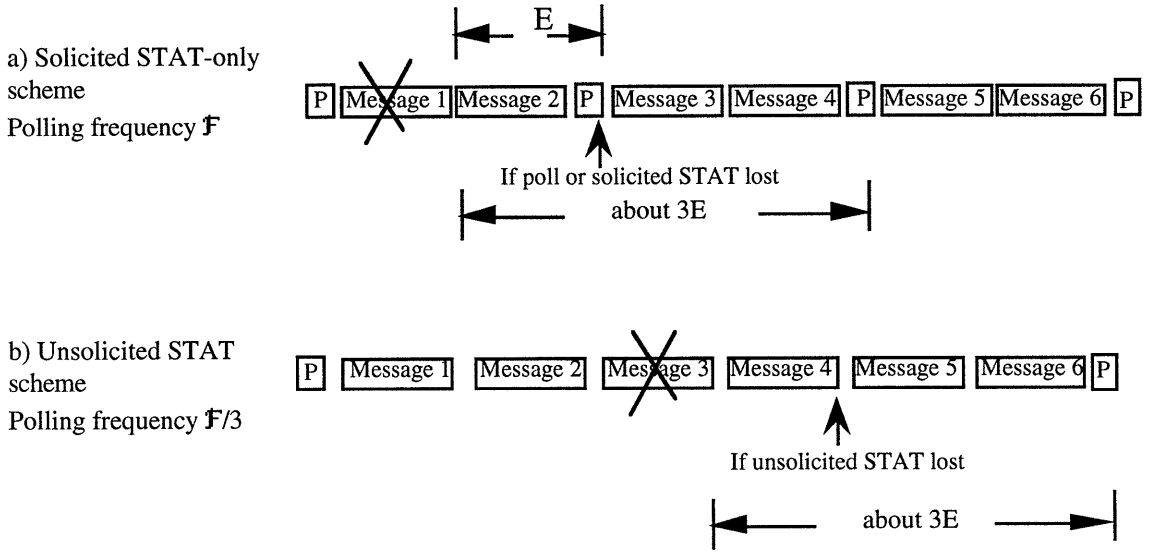


Figure 6.8 Excess delay if: (a) a solicited STAT is lost in a solicited STAT-only scheme, and (b) an unsolicited STAT is lost in an unsolicited STAT scheme.

Another drawback to decreasing the rate of polls and relying on unsolicited STATs is that the traffic may be bursty. If the last message in a burst is lost, it may be a long time before the next data message reaches the destination. Thus, for bursty traffic, polls should be sent at the end of the bursts, regardless of whether or not unsolicited STATs are implemented.

A third drawback to decreasing the rate of polling is that in many schemes the status messages are also used to ACK data so that the source can release ACKed messages. Since unsolicited STATs are only sent when a message is lost, they cannot be relied upon for releasing data at the source. A lower polling rate would therefore necessitate larger buffers at the source.

We conclude that it is not worth implementing unsolicited STATs for the purpose of decreasing overhead - the benefits are likely to be small.

6.2.2.3 Conclusions on Using Unsolicited STATs

If unsolicited STATs are used, they should be used as a means of reducing delay and not as a method of reducing polling overhead. Unsolicited STATs will result in decreased delay only for certain values of R and N. The larger N is, the larger R should be before an unsolicited STAT scheme is implemented. The drawback of using unsolicited STATs is the increased complexity of the polling scheme. The receiver must send two different types of status messages; these two types of status messages may need to be processed differently by the source (as is the case in ATM).

6.2.3 Piggybacked Polls

In this section we consider a solicited STAT-only scheme where the polls are 'piggybacked' on data messages. It is assumed each data message contains a field that indicates whether the message also serves as a poll. The advantage of this option is that separate poll messages may not have to be sent, thus reducing the polling overhead.

Before discussing the drawbacks, we should comment on the implementation of piggybacked polls. The poll indicator field could be included in any packet of the data message. Nevertheless, we assume that the entire message must be received and reassembled at the destination before the poll can generate a status message. From the point of view of delay, it would be better if the poll could generate a status message as soon as the packet that contains it arrives. However, this violates the principle of isolating layers within the network, since we assume retransmissions are handled only at the layer that deals with messages. Also, if, as discussed in chapter three, a per-message CRC is included rather than a per-packet CRC, then the integrity of the poll-carrying packet can only be checked once the whole message has been reassembled.

One drawback of using piggybacked polls is that they may lead to greater delay. Consider the scenario where a data message is lost but the following message is received intact. If we are using a piggybacked poll scheme, and each data message is marked as a poll, then the excess retransmission delay is N time units, i.e., the time for the data message following the lost one to arrive. In a scheme where explicit poll messages are sent after every data message, the excess delay is only 1 time unit.

In fact, the difference in delay may be even greater. The longer a message is, the greater the probability it will be lost; thus, a poll piggybacked onto a long data message is more likely to be dropped than a one packet explicit poll message. In general, we see that the larger N is, the greater the increase in expected excess delay when using piggybacked polls.

The third drawback of using piggybacked polls is that it may not be possible to totally rely on them as the sole means of polling. If there is no more data to be sent or if the data traffic is very bursty then explicit polls may need to be sent. This adds slightly to the complexity of the scheme, since the source has to monitor whether or not explicit poll messages are needed.

6.2.3.1 Conclusions on using Piggybacked Polls

Using piggybacked polls makes the most sense when N is relatively small. With N small, the savings in overhead can potentially be large and the delay penalty relatively small. For example, from Graph 6.2, we see that for R equal to 10 and N equal to 1, attaining a 10% relative excess delay requires that the overhead be close to 50%. Using piggybacked polls, we can decrease the overhead (if the data traffic is steady the polling overhead can be reduced to zero) without affecting the delay.

6.2.4 Receiver Generated Status Messages

In the poll-based schemes discussed thus far, the source sends polls to the receiver (perhaps piggybacked on data messages) and the receiver immediately responds to the poll by sending a status message. Another option is to have the receiver control when status messages are sent. In this section, we assume that the receiver relies solely on receiving a message with sequence number greater than X in order to determine that message X has been lost; we assume that neither the receiver nor the source maintains a timer for the purpose of generating retransmissions. This is similar to the unsolicited status message option discussed earlier except that the receiver controls when it sends the status message

(i.e., it can batch the NACKs). As with the schemes discussed above, it is possible to eliminate unnecessary retransmissions in receiver-controlled schemes, assuming that data travels in sequence. An example of such a scheme is the Checkpoint Mode Protocol [BKK88] (this is actually a node-by-node retransmission protocol at the Data Link Control layer; nevertheless the same technique can be used on an end-to-end basis to eliminate unnecessary retransmissions).

There are several potential problems in relying on the receiver to generate status messages if the receiver does not also maintain a timer. For example, if the last message of a burst is lost, the receiver will have no way of knowing the message was ever sent, and thus will not send a status message NACKing it until data transmission resumes. The other problem scenarios are tied in with window-based flow control.[BKK88] Assume that the window size is W; if the oldest unacknowledged message is X, then the source can't send messages past X+W-1. If W consecutive messages are lost, the protocol will deadlock. In another scenario, assume the first message is lost, the next W-1 messages are received correctly, and the retransmission of the first message is lost. Again, the protocol will deadlock.

There are several solutions to these problems, all of which add to the complexity of the protocol. The solution proposed in [BKK88] is to send 'empty' data messages that are not subject to flow control restrictions. Empty messages could be sent if the source has no more data to send or if it has reached the end of its send window. Alternatively, the source could send a poll message rather than an empty data message. The poll would indicate the highest message number transmitted thus far.

Another possible solution is that once the source runs out of data to send or reaches the end of its send window, it keeps retransmitting the oldest unacknowledged message. (If selective repeat is being used it probably makes more sense to send the oldest message, followed by the second oldest, etc., rather than continually sending the oldest message.) The advantage of this method is that if this particular message has been lost, the source does not have to wait for a NACK to arrive before retransmitting the message. The disadvantage is it will produce unnecessary retransmissions.

Note that in general timers can be used to avoid the deadlock problems enumerated above. For example, the source can maintain a timer to indicate when it expects to receive an ACK for a given message. If an ACK is not received by this time, the source retransmits the message. Alternatively, the receiver can maintain a timer indicating when it expects to

receive a message. If the message is not received by this time, a NACK is sent to the source. The drawback of timers is that unnecessary retransmissions may occur. Schemes that combine both a polling mechanism and a timer will be discussed more thoroughly in Section 6.5.

6.2.4.1 Comparisons with Source-Controlled Schemes

Now let's compare the performance of receiver-controlled schemes (without any timers) to schemes where polls trigger the status messages from the receiver. Overhead should be less in receiver-based schemes. The receiver does not have to send a status message until it has something to NACK, or until it needs to free up space in the retransmission buffer at the source. Thus, fewer status messages would be sent in a system where the message loss rate is not high. Also, explicit poll messages do not have to be sent, except for the situations enumerated above.

The minimum delay before a lost message is NACKed is N time units, i.e., the time for the next message to arrive. Thus, as N gets larger, the excess delay in receiver-controlled schemes increases.

The receiver-controlled scheme provides greater flexibility by allowing the receiver to send status messages when it wants. Other than that, it is not significantly different from a polling scheme where piggybacked polls and unsolicited STATs are allowed.

6.3 TIMER-BASED RETRANSMISSION SCHEMES

In this section, we examine schemes where retransmissions are triggered by a timer. In general, if a timer expires before an ACK has been received for a particular message, then the message is retransmitted. Timer-based schemes are used in both TCP and ISO TP4.[DOD85], [Sch87], [DDK90]

We make similar assumptions as in Section 6.2: packets are fixed size, messages are comprised of N packets, and the data rate is fixed. The unit of time is the time to transmit one packet. The RTD is assumed to be R time units.

6.3.1. Delay Analysis

Assume a timer is maintained for each message that has been transmitted but not acknowledged. Only one timer is actually needed - we discuss the implementation of the

timer in Section 6.3.2. Also, assume the receiver sends an ACK after each successful arrival.

First, consider the ideal case where R is known precisely. If a message is initially transmitted at time $\mathcal{T}$, then it is expected to arrive at the receiver at time $\mathcal{T}$+ N + .5R. If the message arrives error-free and is accepted by the receiver, a one packet ACK is sent immediately (the transmission of ACKs is discussed in more detail below). The ACK should arrive at the source at time $\mathcal{T}$+ N + R + 1. (The '1' term is due to the transmission delay in sending the ACK.) Thus, the timer for the message should be set to expire at time $\mathcal{T}$+ N + R + 1. If an ACK has not been received by this time, the source assumes the message did not reach the destination successfully and retransmits the message.

Thus, in this ideal case where R is known, the excess delay (as defined in Section 6.2.1.2) is 0. Realistically, R is not known precisely. Let R_T be the true RTD and let R_E be the estimate of the RTD. If $R_E > R_T$, then the excess delay equals $R_E - R_T$. However, if $R_E < R_T$, then the timer will expire before the ACK could possibly have reached the source; this may result in an unnecessary retransmission.

Compared to the poll-based scheme, the excess delay of the timer-based scheme is less 'quantized'. For example, consider the piggyback poll scheme where polls are included as part of the data messages. If data messages consist of N packets, then the excess delay for any lost message is some multiple of N. In the timer scheme, the excess delay depends on the value of $R_E - R_T$.

In contrast to poll-based schemes, the performance of the timer-based scheme heavily relies on how well the source can estimate the RTD. If the RTD is not highly variable then the source should have a good estimate of the RTD, so that there is little excess delay. If the RTD does vary a lot, the inability of the source to accurately estimate the RTD may lead to large excess delay, or a large number of unnecessary retransmissions. The variability of the RTD depends largely on the queueing delays along the data path. Queueing delays are caused by packets arriving at an intermediate node faster than the node can process them, or by many packets at a node contending for the same output line. In either case, the packets must wait in a buffer until the node can process them (or if the buffer is full, the packets will be dropped). The time spent waiting in a buffer constitutes queueing delay and increases the effective RTD. For a given packet size and buffer size, queueing delays are potentially longer the lower the data rate of the network (assuming flow control is equally

effective in both cases). Thus, the variability of the RTD is expected to be larger with low speed networks than with high speed networks.

Using relative excess delay as a performance criterion, we see that the performance of timer-based schemes improves with larger propagation delay. Let R_P denote the propagation delay portion of the RTD; assume the source knows R_P exactly. Let R_{QT} denote the true queueing delay along the data path and let R_{QE} be the source's estimate of the queueing delay. Then the relative excess delay, assuming $R_{QE} > R_{QT}$, is:

$$\frac{R_{QE} - R_{QT}}{N + R_P + R_{QT} + 1} \qquad (6.15)$$

If R_P is large, then an overestimate of the queueing delay will not significantly affect the performance of the scheme. Thus, the source can be more conservative in estimating the RTD (i.e., use a larger estimate) and thus reduce the number of unnecessary retransmissions.

If the connection is delay sensitive, then the larger the propagation delay, the smaller the excess delay should be. The source would be less likely to pad its estimate of the RTD. If the RTD is highly variable this could lead to many unnecessary retransmissions.

6.3.2 Overhead Analysis

Now let's consider processing overhead. In general, only one retransmission timer is necessary; an actual timer does not need to be maintained for each outstanding message. After a message is transmitted, the time of transmission should be stored along with the message in the retransmission buffer. As ACKs arrive, the source should continually update its estimate of the RTD. If the estimate of the RTD added to the timestamp of the oldest unacknowledged message is less than the current time, then the message should be retransmitted. Thus, the processing overhead involves maintaining one timer, estimating the RTD, and performing time comparisons with the oldest message on the retransmission buffer.

As for traffic overhead, in the source to receiver direction, the only overhead is due to unnecessary retransmissions. The number of unnecessary retransmissions will depend on how conservative an estimate of the RTD is used by the source. Thus, as with poll-based schemes where polls are not piggybacked, there is a tradeoff between delay and overhead, but the tradeoff is more difficult to quantify in the timer-based scheme.

In the receiver to source direction, there is the overhead due to the transmission of ACKs. If an ACK is sent immediately after each successful arrival, then the overhead in the reverse direction is $\frac{1}{N+1}$ (this assumes the data rate is the same in both directions). If N is small, the overhead will be very large.

We should make a few comments about the transmission of ACKs. Our calculations have implicitly assumed that an ACK is sent immediately upon the successful arrival of a message, i.e., the ACK pre-empts any reverse data traffic going from receiver to source. (We made this same assumption for status messages in poll-based schemes.) Another option is that the ACK be piggybacked onto the tail-end of a message traveling in the reverse direction (an ACK is simply a message number). This would reduce the amount of overhead due to sending ACKs, but it would increase the delay. If the ACKs are piggybacked, then the timer should be set to expire at time $\mathcal{T} + N + R + N_R$, where N_R is the size of messages in the reverse direction. If N_R is small, it makes a lot of sense to piggyback the ACKs: the overhead due to ACKs is reduced to 0 (or near 0) and the delay is only increased a small amount (i.e., N_R-1 time units). Also, if N_R is small, then the probability of the data message carrying the ACK being dropped is not much greater than the probability of a one packet ACK message being dropped.

6.3.3 Burst Losses

In Section 6.2.1.5, it was shown that in poll-based schemes, excess delay increases when a burst of messages is lost. In timer-based schemes, however, retransmissions are not delayed when burst losses occur. The key difference is that in poll-based schemes, the retransmission mechanism depends on a message getting through to the receiver (either data or a poll) and a status message getting through to the source. Timer-based schemes rely on a timer rather than on a message reception to trigger a retransmission.

Assume the first message lost in the burst is originally transmitted at time $\mathcal{T}$, and assume messages are lost for the next B time units. First, consider the case where B is less than R + N + 1. Assuming the source knows the RTD precisely, it will begin to retransmit the lost messages at time $\mathcal{T} + R + N + 1$; thus, the relative excess delay will be 0. Thus, timer-based schemes are relatively unaffected by burst errors shorter than one round trip delay. As shown in Section 6.2.1.5, if a poll-based scheme is used, the expected excess delay increases by approximately the length of the burst as is shown in equation (6.8).

Next, consider the scenario where B satisfies: $K(R + N + 1) < B \leq (K+1)(R + N + 1)$ for some $K \geq 1$. The first lost message will be retransmitted at times $\mathcal{T}+ i(R + N + 1)$ for $1 \leq i \leq (K+1)$. The first K retransmissions will be lost as part of the burst; we assume the $(K+1)^{st}$ retransmission is successful. The excess delay will be $(K+1)(R + N + 1) - B$ and the relative excess delay will be $\frac{(K+1)(R+N+1) - B}{B}$. Thus, the relative excess delay can be bounded by:

$$0 \leq \frac{(K+1)(R + N + 1) - B}{B} < \frac{R + N + 1}{B} < 100\% \qquad (6.16)$$

Of course, this depends on the source having an accurate estimate of the RTD. For the poll-based scheme, the relative excess delay is given in (6.11):

$$\frac{\delta + R + 2}{B}$$

where δ lies between 0 and NI, assuming the first poll sent after the burst ends is successful.

The drawback of the timer-based scheme during a long error burst is that the lost messages are unnecessarily retransmitted K times. This could be detrimental if the burst loss is due to congestion and window-based flow control is used (refer back to the discussion in Section 6.2.1.5).

6.4 POLL-BASED SCHEMES VS. TIMER-BASED SCHEMES

In this section we examine whether a timer-based scheme or a poll-based scheme is more appropriate for a given network. (In Section 6.5, we discuss schemes that include both a poll mechanism and a timer.) Throughout this section it is assumed that packets travel in order. Recall that we defined R to be the number of packets that are sent per RTD; thus, R increases as the data speed of the network increases. Recall from Section 6.2.1.1 that we refer to networks with an R of about 1 or smaller as low speed networks and networks with an R on the order of 100 or higher as high speed networks.

6.4.1 Comparison of Poll-Based and Timer-Based Schemes

In the sections above, we showed that using relative excess delay as a delay criterion, the performance of both poll-based and timer-based schemes improves as R increases, assuming random messages are lost. If bursts of messages are lost, however, the two schemes perform quite differently. If the burst loss is shorter than one round trip delay, then the excess delay in the poll-based scheme increases by about the duration of the burst;

the excess delay in the timer-based scheme is relatively unaffected. If the burst loss is longer than one round trip delay, then the excess delay increases in both schemes, although the increase in poll-based schemes is likely to be greater; however, timer-based schemes trigger retransmissions too quickly in this scenario so that some retransmissions will be lost as part of the burst loss.

The increase in excess delay that accompanies poll-based schemes is advantageous during bursts of congestion if window-based flow control is used and if the congestion is persistent; under these same conditions, the fact that timer-based schemes trigger retransmissions too quickly is detrimental. In other situations, the greater expediency in timer-based schemes is preferable. Note that window-based flow control is often used on systems with small R and rate-based flow control is likely to be used on systems with large R (since window-based schemes depend more on timely feedback from the receiver to the source).

The amount of processing necessary in the two types of schemes is likely to be about the same. Both poll-based and timer-based schemes require one timer to be maintained at the source. Timer-based schemes are characterized by the receiver sending ACKs and the source checking whether the ACK timer has expired. Poll-based schemes require that the source periodically send polls, that the receiver respond to polls with status messages, and that the source perform some type of sequence number comparison upon receiving a NACK to guarantee the property of no unnecessary retransmissions.

Both schemes need to maintain an estimate of the RTD. An overestimate of the RTD in either type of scheme results in greater delay. An underestimate of the RTD in poll-based schemes results in smaller delay, but more overhead per RTD. An underestimate of the RTD in timer-based schemes results in unnecessary retransmissions.

The key difference between the schemes is that timer-based schemes are capable of low excess delay, but at the expense of unnecessary retransmissions. Poll-based schemes do not produce unnecessary retransmissions (assuming some sort of sequencing scheme is used), but may result in a large excess delay. Below, we compare the performance of the two types of schemes on low speed and high speed networks.

6.4.2 Low Speed Networks

First, let's consider a low speed system where the message size is small. Neither type of scheme performs well in this environment. In poll-based schemes, the transmission time of the poll (whether or not it is piggybacked) represents a significant portion of the total retransmission time. In timer-based schemes, the low speed of the network may result in highly variable queueing delays; the inability to properly estimate the RTD could potentially result in large delay or in many unnecessary retransmissions.

Since efficient bandwidth usage is likely to be a major concern on low speed networks, poll-based schemes are preferred since they do not produce unnecessary retransmissions. Note that if a poll-based scheme is used, the polls should be piggybacked to reduce the polling overhead; it is not worthwhile to implement unsolicited status messages since the data speed is low.

One could argue that if buffers at the intermediate nodes are kept very small, then the queueing delays would be less variable, enabling a timer-based scheme to perform better. However, small buffers would likely lead to bursts of messages being dropped due to congestion. Assuming window-based flow control is used, the extra retransmission delay in poll-based schemes during times of congestion would be desirable. Thus, even in this situation, we see that poll-based schemes are preferred.

For low speed networks where the message size is large, poll-based schemes provide small relative excess delay (for random losses) and low overhead, in addition to being able to prevent unnecessary retransmissions. Thus, their advantage over timer-based schemes is even more pronounced in this environment. Note that in such an environment, polls should not be piggybacked and unsolicited status messages should not be implemented.

In general, we conclude that for low speed systems, poll-based schemes are preferable to timer-based schemes.

6.4.3 High Speed Networks

First, we can consider the case where all speeds in a network are scaled up by the same factor, while the packet size and buffer size remain fixed. As shown in Section 6.2 on Graph 6.3, poll-based schemes perform well at high data rates (assuming bursts of messages are not lost). Also, as the data rates increase, queueing delays decrease, assuming flow control remains equally effective. Thus, timer-based schemes also perform

well. In general, both types of schemes should provide low relative excess delay for the scenario where all connections are running at high speed.

High speed networks, however, are likely to carry a wide range of traffic types; the data rates and burstiness of the various connections may span a wide range. Thus, despite the fact that the data links are running at high speed, some of the individual connections may be running at low speed. This mix of traffic favors the use of timer-based schemes. The high speed of the links will result in low queueing delay. Thus, the estimates of the RTD should be fairly accurate. For the low speed connections in the system, the delay in waiting for a poll to arrive may be relatively large (due to the transmission delay in sending a poll); thus, these connections may suffer a large relative excess delay if poll-based schemes are used.

Some type of rate-based flow control is likely to be used on these integrated service systems, which again favors the use of timer-based schemes. In rate-based flow control, a certain number of messages will be sent in a given time period, regardless of whether the messages are new transmissions or retransmissions. Thus, if the system is congested, delaying retransmissions does not help alleviate the congestion (unless an upper limit on the send window is reached). Thus, from the standpoint of delay, the retransmissions should be sent as quickly as possible, which favors the use of timer-based schemes.

One strategy that might be used on high speed networks is to make the buffers at the intermediate nodes very large. Memory is supposedly very cheap, so that large buffers are not costly. Flow control would still be implemented with the goal of keeping queueing delays small. However, if severe congestion develops, packets will be queued in buffers at the intermediate nodes rather than being dropped. The rationale for this is that queueing a message in a buffer will probably result in less overall delay than dropping the message and retransmitting it. (Delay sensitive traffic would be given priority in the buffers so that the queueing delay of such traffic would not be large.) In such a scheme, losses due to congestion would be less likely; also, the queueing delays may be more variable, especially if the flow-control mechanism is not effective. Thus, the performance difference between timer-based and poll-based schemes would be less significant in such an environment. However, if long burst errors are expected, then again the increase in excess delay which accompanies burst losses in poll-based schemes would be undesirable.

Overall, timer-based schemes are better able to provide low delay in a high speed, integrated service environment.

6.4.4 Conclusions on Poll-Based vs. Timer-Based Schemes

Timer-based schemes have traditionally been used on low-speed networks, such as TCP/IP. Such schemes have not performed well, as our analysis predicts. They lead to many unnecessary retransmissions, which is a serious problem on low speed networks where efficient bandwidth usage is desirable. Poll-based schemes essentially eliminate unnecessary retransmissions, but do so at the expense of increased retransmission delay. In a high speed, integrated service environment, where low delay is more critical than efficient bandwidth usage, timer-based schemes are generally better than poll-based schemes. In the next section, we discuss schemes that combine polling and timers.

6.5 COMBINATION OF POLL-BASED AND TIMER-BASED SCHEMES

6.5.1 Overview of Poll/Timer Scheme

It is possible to use both a polling mechanism and a timer to generate retransmissions. The advantage of a poll-based scheme is that it can eliminate unnecessary retransmissions; the drawback is that the excess delay increases when bursts of messages are lost. The advantage of a timer-based scheme is it imposes a firm upper bound on the excess delay; the disadvantage is the performance of the scheme heavily relies on the source's ability to estimate the round trip delay. The two types of schemes potentially can be combined so as to take advantage of their best features. The polling mechanism should be relied upon as the primary means of generating retransmissions since it will not produce spurious retransmissions. The timer mechanism should be used to provide an upper limit to the excess delay.

Consider implementing the following scheme on a network where we assume data travels in sequence. (Another scheme that combines timers and polls is described in [NRS90].) The polling portion of the scheme is similar to the scheme described in Section 6.2. Periodically, the source sends a poll to the receiver indicating which messages it has sent. The receiver responds with a status message. Any NACKed messages are retransmitted by the source, subject to whatever criterion ensures the retransmission is necessary. As in a normal timer scheme, whenever a message is received successfully at the destination, the destination sends an ACK of the message back to the source. However, the source sets its retransmission timer based on the transmission time of a poll, rather than on the transmission time of a message. Assume poll P is transmitted at time $\mathcal{T}$, and assume the source's estimate of the RTD is R_E. The timer corresponding to poll P should be set to expire at time $\mathcal{T} + 2 + \beta R_E$, where $\beta > 1$ permits some error in estimating R_E and where 2

represents the transmission time of the poll and the corresponding status message. If the timer corresponding to poll P expires without the source receiving a status message corresponding to poll P or corresponding to a poll sent after poll P, the source retransmits all unacknowledged messages sent prior to poll P.

6.5.2 Performance of Poll/Timer Scheme

Let's analyze how such a scheme performs. Consider some data message X; assume poll P is the next poll transmission after this message. First, assume message X is transmitted successfully. An ACK of message X will be sent by the destination. Poll P should trigger a status message that also ACKs the message. Thus, assuming the source's estimate of the RTD is accurate, there should be at least two opportunities for the data message to be ACKed before the retransmission timer corresponding to poll P expires. If β is chosen large enough, then there will be time for additional status messages to arrive at the source prior to the expiration of the timer. This increases the likelihood that an ACK of the data message will arrive successfully at the source. Also, a large β is helpful if the source has underestimated the RTD. As in a typical timer scheme, the larger β is, the smaller the probability of an unnecessary retransmission. However, the larger β is, the larger the maximum allowable excess delay.

Next, assume message X is lost. Poll P will generate a NACK of the message. If the status message corresponding to poll P arrives at the source, message X will be retransmitted approximately one RTD after poll P was sent. If poll P or its corresponding status message is lost, the retransmission will be delayed. If β is close to 1 and R_E is an accurate estimate of the RTD, then the expiration of the timer will trigger the retransmission; if β is large or R_E is an overestimate, then a status message corresponding to a poll sent after poll P will likely generate the retransmission.

Finally, consider a burst of data messages and polls being dropped. If a poll-based scheme were used without a timer, then no retransmissions would occur until a poll successfully arrived at the destination, and its corresponding status message arrived at the source. In the combined poll/timer scheme, the expiration of the timer can also trigger the retransmission. The presence of the timer puts a firm upper bound on the maximum delay until a lost message is retransmitted (assuming polls are sent regularly).

The drawback of a combination polling/timer scheme is the added complexity. The source has to deal with sending polls and setting timers. The destination has to send ACKs as well as status messages.

In the next sections, we analyze in what environments a combined poll/timer scheme would be appropriate.

6.5.2.1 Low Speed Networks

On a low speed network, the source is likely to have difficulty estimating the RTD due to the highly varying queueing delays. Thus, to reduce the number of unnecessary retransmissions, the source will use a large β when setting the retransmission timer. In effect, the scheme will be a pure polling scheme with the timer mechanism only coming into play during very long burst losses. If the burst losses are due to congestion, it is preferable to delay the retransmissions; thus, having the timer mechanism is not worthwhile.

6.5.2.1 High Speed Networks

On a high speed network, the source is likely to have very accurate estimates of the RTD. Thus, with the timer scheme alone, the excess delay should be small and the number of unnecessary retransmissions should be small. There really is not a reason to implement the polling mechanism in addition to the timer.

6.5.2.1 Intermediate Networks

A combination poll/timer scheme is more appropriate for a network that has characteristics that are somewhere 'in between' low speed and high speed. The RTD should be somewhat varying such that relying on a timer scheme alone would produce too many retransmissions. However, the RTD should not be so difficult to estimate that β needs to be set very high in order to avoid many retransmissions (which essentially renders the timer mechanism almost useless). Low retransmission delay should be somewhat of a concern, to make it worthwhile to implement the timer scheme in order to upper bound the excess delay. Bandwidth efficiency should be somewhat of an important issue, otherwise there would be no need to implement polls.

The more closely the characteristics of the system resemble those of a high speed network, i.e., the better the source can estimate the RTD and the more important low delay is, the smaller β should be. The smaller β is, the larger role the timer mechanism plays.

Conversely, the lower the speed of the system, the larger β should be; this increases the role of the polls and status messages.

6.5.3 Receiver Controlled Poll/Timer Schemes

In Section 6.2.4, we discussed retransmission schemes where the receiver controls when status messages are sent without the use of timers. If data is expected to travel in order, then if message X arrives at the destination before message X-1 arrives, the destination assumes that message X-1 has been lost. The destination periodically will send a status message to the source indicating which messages need to be retransmitted. However, as discussed in Section 6.2.4, the scheme potentially deadlocks when the source has no more data to send or when the end of the 'send window' is reached. We discussed several solutions to this problem, one of which was to include a timer mechanism. Thus, if the message at the end of a burst is dropped, the expiration of the timer will trigger the retransmission. The advantage of such a scheme is that the source does not have to transmit polls, and the receiver has more control over when status messages are sent. Of course, the receiver can not delay too long in sending a status message since the timer might unnecessarily expire (also status messages are used to free up space in the retransmission buffer at the source).

The delay performance of such a scheme is probably not that different from a source-controlled polling scheme. The advantage of the receiver controlled scheme is that possibly fewer status messages might be sent. Using a timer mechanism limits the excess delay in case several consecutive data messages or status messages are lost.

It is interesting to note that a scheme similar to this can be used in X.25 at the Data Link Control Layer.[CCI81],[Tan88] Whenever a frame arrives correctly at a node, it is ACKed. If frame X arrives before frame X-1 arrives, a REJECT message can be sent by the destination. However, only one REJECT message can be sent for a given frame. A retransmission timer is maintained by the source in case a REJECT message is lost. If the timer expires without a message being ACKed, it is retransmitted. In some implementations, a node is allowed to send explicit poll message in order to generate status messages.

6.5.4 Summary of Poll/Timer Schemes

Retransmission schemes that combine polls and timers are useful in certain situations. Implementing both mechanisms allows the source a lot of flexibility in establishing the

tradeoff between low delay and bandwidth efficiency. Implementing timers is also a simple way of fixing the deadlock problems of receiver controlled polling schemes.

6.6 SELECTIVE REPEAT VS. GO BACK N

As mentioned in the introduction of this chapter, there are two commonly used options to deal with messages that arrive at the destination out-of-sequence. In Go Back N (GBN), the receiver drops out-of-sequence messages. GBN is simple to implement. The receiver only needs to keep track of which message number it expects to receive next. Typically, the receiver indicates this next_to_receive number in control messages sent to the source. The source then knows that all messages up to but not including this message number have been successfully received, but that messages with sequence number greater than or equal to next_to_receive have not been received (or at least not by the time the control message was sent by the receiver). If the source retransmits a particular message, then it makes sense that it also retransmit all messages that follow it. Essentially, the source 'goes back' to the message indicated by next_to_receive, and starts transmitting from that point forward.

The second option is referred to as selective repeat (SR). In SR, the receiver maintains a resequencing buffer for messages that arrive out-of-sequence. A message is kept in the buffer until all messages with earlier sequence numbers arrive. The receiver indicates to the source which message numbers are missing and which have successfully arrived. The source then only needs to retransmit those messages that are missing.

SR will generally result in fewer retransmissions. One drawback to SR, however, is the need for a resequencing buffer at the destination. In general, if the oldest outstanding message is lost K times, the resequencing buffer must be able to hold all the data transmitted in K RTDs, or else the efficiency of the SR scheme decreases. At the very minimum, the buffer should be large enough to hold one RTD worth of data. A second and more important drawback is the complexity of implementing an SR scheme. The receiver must keep track of the status of all messages it receives, and must deal with resequencing messages it receives out of order.

The performance of GBN and SR have been studied in detail (for example, see [RoS89], [Kon80], [AnP86], [Sch87], [BeG92]). Here, we review the results from [BeG92] on bandwidth efficiency in the two schemes. We also briefly discuss the effect of burst losses in GBN and SR systems, and the expected delay in the two schemes. This provides some

insight into which scheme is better suited for a particular system. One of the motivations for comparing the two systems is the ongoing debate of whether ATM should implement GBN or SR. This will be examined in more detail in chapter seven.

6.6.1 Efficiency Analysis

We define γ as the expected number of transmitted messages from source to destination per successfully accepted message at the destination. Efficiency, which we represent by η, is then defined as $1/\gamma$.

We make the following assumptions: messages are fixed length and comprised of N packets; the RTD is fixed and R packets are transmitted per RTD; the source always has data to send. If a message is initially sent at time $\mathcal{T}$, the source learns the status of the message at time $\mathcal{T}$+ N + R + 1 (i.e., throughout this section we assume the excess delay is 0). When a NACK arrives at the source indicating a message needs to be retransmitted, the source finishes transmitting the message it is currently sending, and then performs the retransmission. Thus, the source sends $\left\lceil \frac{R+1}{N} \right\rceil$ messages in between the original transmission of a message and the retransmission of a message. We assume that packets are dropped randomly with probability Q_R.

In the SR system, we assume that the resequencing buffer is large enough that it never overflows. In an actual SR system, overflows are possible; thus, our calculations for the efficiency of an SR system are actually upper bounds. An SR system performs no worse than a GBN system in terms of efficiency, so the GBN results are loose lower bounds for the SR system.

6.6.1.1 Random Losses

If packets are dropped independently with probability Q_R, then a message is dropped with probability $1 - (1 - Q_R)^N$, which is approximately equal to $N Q_R$. Following the derivation in [BeG92], we see that for a GBN system:

$$\eta = \frac{1 - N Q_R}{1 + N Q_R \left\lceil \frac{R+1}{N} \right\rceil} \qquad (6.17)$$

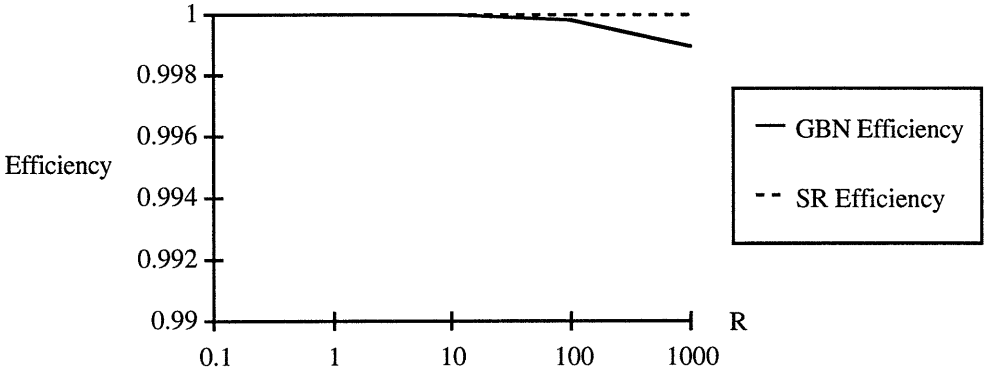
For an SR system:

$$\eta = 1 - N\,Q_R \qquad (6.18)$$

We see that GBN becomes relatively worse as Q_R or R increases. This can be seen on Graphs 6.6. and 6.7. (The plots do not significantly change if N is varied.) SR systems are often proposed for satellite systems, where Q_R is relatively high, and the RTD is long.

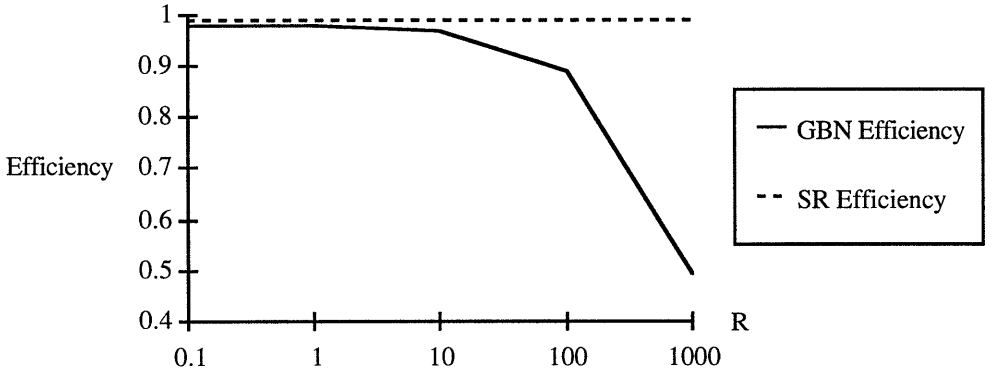
Graph 6.6 Efficiency of GBN and SR

(N =10 and Packet Loss Rate = 1E-6)



Graph 6.7 Efficiency of GBN and SR

(N =10 and Packet Loss Rate = 1E-3)



6.6.1.2 Burst Losses

Consider the case where packets are dropped in bursts, rather than independently. A GBN system performs better if losses occur in bursts rather than independently. This is because after a packet is dropped, the GBN system will end up dropping a whole RTD worth of packets anyway. Thus, ideally, the packets losses would occur in bursts of size R+1 (or a multiple of R+1). If this were the case, there would be no loss of efficiency due to the

need for the source to 'go back' and retransmit a whole RTD worth of packets. An SR system is relatively unaffected by the burstiness of the losses, as long as the resequencing buffer is very large.

We conclude that the performance gap between GBN and SR systems narrows on systems where packet losses tend to occur in bursts.

6.6.2 Delay Analysis

A lost message also affects the next RTD worth of messages. In an SR scheme, these messages must wait in the resequencing buffer before being delivered to a higher layer; in GBN, these messages are dropped and must be retransmitted. In this section, we analyze the minimum delay that these RTD worth of messages must suffer in GBN and SR schemes.

First, consider the case of no losses in the system. If a message starts to be transmitted at time $\mathcal{T}$, then it should arrive at the destination at time $\mathcal{T}$+ .5R + N. It should be delivered to the higher layer at this time. Thus, the total delay is .5R + N.

Now let's assume one message is lost. We assume the message is originally transmitted at time $\mathcal{T}$, and at time $\mathcal{T}$+ N + R + 1 the source is informed the message has been lost. In a GBN scheme, if a message is lost, the next RTD worth of messages is dropped also. The source will begin to retransmit the messages at time $\mathcal{T}$+ N + R + 1, beginning with the first one that was lost. Assuming there are no additional losses, then each one of the retransmitted messages will arrive at the destination 1.5R + 2N + 1 time units after it is originally sent. The retransmitted messages should arrive in sequence so they can be delivered to the higher layer upon arrival at the destination. Thus, the lost message causes an extra delay of R + N + 1 for a whole RTD worth of messages.

In an SR scheme, after a message is lost, the messages that follow it will be accepted at the destination, but they cannot be delivered until the lost message is retransmitted and arrives. These messages must wait in the resequencing buffer while the lost message is retransmitted. The amount of time spent in the resequencing buffer is not the same for all affected messages, as shown in Figure 6.9. The time ranges from R+1 time units for the first message affected after the lost one, to N time units, for the last message affected. The average time in the resequencing buffer is about .5 (R + N + 1) time units. In GBN, all affected messages suffer an extra delay of R + N + 1. In either scheme, note that the initial

lost message is delayed the same amount, i.e., R + N + 1 time units. Thus, for the assumptions we have made, the average extra delay is smaller by a factor of 2 in SR, but the worst case delay is the same.

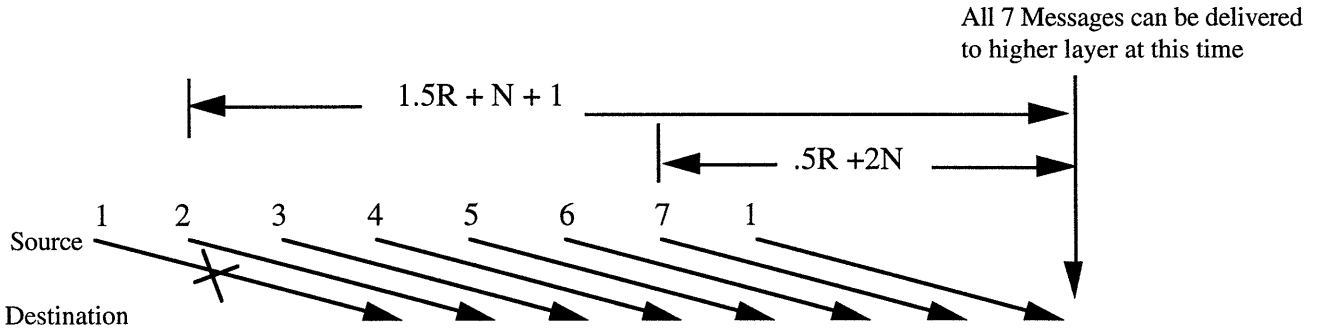


Figure 6.9 Time spent in resequencing buffer at the destination for one RTD worth of messages. All messages can be delivered to the higher layer as soon as the missing message is retransmitted and arrives.

6.6.3 Conclusions

For small R and low packet loss rate (Q_R), the performance difference between GBN and SR is not likely to be significant. As R or Q_R increases, the SR scheme becomes relatively more efficient. The chief drawback of SR is the additional complexity of buffering and resequencing messages at the destination. Thus, for systems with large R or Q_R, which scheme to use depends on whether bandwidth efficiency or complexity is the more important issue.

CHAPTER 7

RETRANSMISSION IN ATM

7.1 INTRODUCTION

Chapter six presented a general analysis of poll-based and timer-based retransmission schemes. In this chapter, we focus on the poll-based scheme that has been proposed by the ATM Standards Committee. The basic scheme consists of the source periodically sending polls to the destination indicating which frames have been sent, and the destination responding with a status message indicating which of these frames have not been received. However, many additional features, such as unsolicited status messages, have been included in the proposed scheme. Also, a major design goal was to ensure that the scheme does not produce any unnecessary retransmissions. In order to accomplish this, the source and destination must maintain several variables. With the added complexity of these features, it is not readily apparent whether the scheme generates the necessary retransmissions.

Further details of the scheme are provided in Section 7.2. In Section 7.3, we formally prove that the scheme does eventually generate a retransmission of any lost frame without producing any unnecessary retransmissions, assuming that certain reasonable conditions hold. In proving the correctness of the scheme, we also further define the protocol. In Section 7.4, we examine how the protocol can fail if the conditions for proper operation are not met. Finally, in Section 7.5 we analyze whether a poll-based scheme is appropriate for the ATM environment.

As described in chapter four, ATM uses different terminology than we have adopted in our general analysis. The Convergence Sublayer (CS) of the ATM Adaptation Layer (AAL) corresponds to layer N; the PDU at this layer is referred to as a frame rather than a

message. Frames are variable length, with the maximum length being 65,536 bytes. Retransmissions in ATM are performed end-to-end at the CS layer, and a frame is the unit of retransmission. The CS layer at the source is responsible for sending poll messages and the CS layer at the destination is responsible for sending status messages. (Note that the term 'message' rather than 'frame' is used for poll and status messages.) The Segmentation and Reassembly Sublayer of the AAL is analogous to layer N-1; the PDU at this layer is a 48 byte data unit called a segment. The ATM layer corresponds to layer N-2. The ATM layer PDU is referred to as a cell rather than a packet, and consists of 53 bytes (i.e., a segment plus a 5 byte header).

7.2 DESCRIPTION OF PROPOSED RETRANSMISSION SCHEME

The following description of the proposed ATM retransmission scheme was obtained from references [T1S92],[CCI92b], and [CCI93]. In Section 7.2.5, an example is given that illustrates many of the details of the protocol.

The protocol allows both selective repeat and Go Back N options. (For a description of these options, refer back to Section 6.6.) The standards committee assumes selective repeat will be used in almost all connections, and, in general, our discussion assumes that selective repeat is implemented. Go Back N operation is summarized in Section 7.2.6. In Section 7.5.3, we analyze whether selective repeat makes sense in an ATM environment.

7.2.1 Sequence Numbers

Each data frame in a connection is numbered sequentially modulo 256. (There is also an option to number frames modulo 2^{24}[T1S91].) Also, each poll message is numbered sequentially, independently of the data frames. Frames and poll messages are expected to arrive at the destination in the order in which they are sent.

The source maintains the variable SEQ to indicate that all data frames up to but not including SEQ have been transmitted at least once, and the variable PSEQ to indicate the sequence number of the last poll message transmitted. SEQ and PSEQ are maintained independently of one another. After a data frame is transmitted, the frame is stored in a retransmission buffer, along with the current value of PSEQ.

The destination maintains the variable SEQR to indicate that all frames up to but not including SEQR have been received. It also maintains the variable SEQH to keep track of one higher than the highest numbered frame that it knows the source has transmitted. Thus,

if frame number X arrives, and X is greater than or equal to SEQH, then SEQH is updated to equal X+1. Polls also cause the value of SEQH to be updated, as described in the next section.

7.2.2 Poll Messages

The frequency with which poll messages are sent is determined by the transmitter. At this time, the standards committee has not yet specified guidelines for determining the polling frequency. PSEQ is incremented just before a poll is sent. A poll contains the current value of both PSEQ and SEQ. If the value of SEQ contained in a poll is greater than SEQH, the destination updates SEQH to equal SEQ. Rather than sending explicit poll messages, the source can piggyback the polling information on a data frame.

7.2.3 Status Messages

There are two types of status messages sent by the receiver: solicited and unsolicited. These are referred to in the protocol description as solicited STATs and unsolicited STATs. A solicited STAT is sent in response to a poll message. A solicited STAT indicates the value of SEQR and provides the status of all frames between the value of SEQR and SEQ-1, inclusive, where SEQ is indicated in the poll message. It also includes the value of PSEQ contained in the incoming poll message.

The receiver sends an unsolicited STAT whenever it receives a frame with sequence number X greater than SEQH which shows that there are missing frames with sequence numbers greater than or equal to SEQH and less than X. (Note that the check for missing frames is performed before updating SEQH to the value of X+1.) An unsolicited STAT only NACKs frames with sequence numbers that fall within this range. Due to the rules for updating SEQH, this guarantees that any frame NACKed in an unsolicited STAT has not previously been NACKed in another status message. The one exception to this is that the protocol provides the option for the receiver to send two identical unsolicited STATs rather than just one. This provides some protection against the STAT being lost.

A poll can optionally be used to generate an unsolicited STAT in addition to a solicited STAT. If a poll arrives with SEQ equal to X and the receiver determines that there are missing frames with sequence numbers greater than or equal to SEQH and less than X, the receiver has the option of sending an unsolicited STAT that NACKs only those frames that fall within this range. (Note that the check for missing frames is performed before updating SEQH to the value of SEQ contained in the poll.) Even if the receiver sends an

unsolicited STAT in response to a poll, it also sends a solicited STAT that provides the complete status of the receiver. As we see in the next section, unsolicited STATs are easier to process; thus, NACKing a frame with an unsolicited STAT before sending a solicited STAT may result in the lost frame being retransmitted slightly sooner. It also provides some redundancy.

7.2.4 Retransmission

When the source receives a solicited status message, it retransmits all NACKed frames as long as the value of PSEQ stored along with the frame in the retransmission buffer is less than the value of PSEQ indicated in the status message. The value of PSEQ in the status message is the sequence number of the poll that generated the status message. The comparison of PSEQ values is done to prevent unnecessary retransmissions. Essentially the retransmission criterion is: retransmit any NACKed frames that were sent before the poll that triggered the NACK. This is discussed further in Section 7.3. When a data frame is retransmitted, the current value of PSEQ is stored in the retransmission buffer along with the frame.

There is a logical variable stored with each transmitted data frame, called RTS. This is set to FALSE when a frame is first transmitted. When the source receives an unsolicited status message, it retransmits all NACKed frames as long as the corresponding RTS value is FALSE. No comparisons of PSEQ are done. Once a frame is retransmitted due to an unsolicited STAT, RTS is set to TRUE. The purpose of the RTS variable is to prevent duplicate retransmissions when the receiver sends two identical unsolicited STATs.

With either type of status message, the source removes from the buffer all frames up to but not including SEQR, as indicated in the status message.

7.2.5 Example of Selective Repeat Operation

An example of the polling scheme operation is shown in Figure 7.1. The important points are described below.

Frames 1 and 2 are delivered successfully. Frame 3 is lost, and Poll 1 generates a solicited STAT that NACKs frame 3. (We assume the receiver chooses not to also send an unsolicited STAT in response to the poll.) Note that the arrival of the poll also causes SEQH to be updated to 4 (i.e., due to the poll, the destination knows that all frames through 3 have been sent at least once).

Frame 4 is lost; frame 5 is received successfully and triggers an unsolicited STAT that NACKs frame 4.

The PSEQ value stored with frame 3 is 0; thus, solicited STAT 1 triggers the retransmission of frame 3 (since 0 < 1). The unsolicited STAT of frame 4 automatically triggers the retransmission of frame 4 without any PSEQ comparisons being necessary.

Before the retransmissions of frames 3 and 4 are sent, Poll 2 is sent, which generates a solicited STAT that NACKs both frames 3 and 4. Solicited STAT 2 triggers no retransmissions since the PSEQ value stored with frames 3 and 4 is now 2.

The retransmission of frame 3 is lost, but the retransmission of frame 4 is successful. Even though the destination receives frame 4 and not frame 3, it does not send an unsolicited STAT, since SEQH is 6.

Finally, Poll 3 generates solicited STAT 3 that NACKs frame 3. This triggers the successful retransmission of frame 3.

7.2.6 Go Back N Operation

If a connection is operating in the Go Back N (GBN) mode, only frames that arrive in sequential order are accepted. If a frame arrives out-of-sequence, the frame is dropped and an unsolicited status message is sent. The status message only needs to indicate SEQR. No further unsolicited status messages are sent until after the frame with sequence number SEQR arrives.

When a source in the GBN mode receives an unsolicited status message, the source retransmits all frames beginning with the frame numbered SEQR. When a solicited status message is received, the source compares the PSEQ value of the incoming status message to the PSEQ value associated with the frame numbered SEQR. If the PSEQ value associated with frame SEQR is smaller than the value of PSEQ contained in the status message, then all frames in the retransmission buffer from SEQR onward are retransmitted. Of course, if SEQR is equal to SEQ then no retransmissions are necessary.

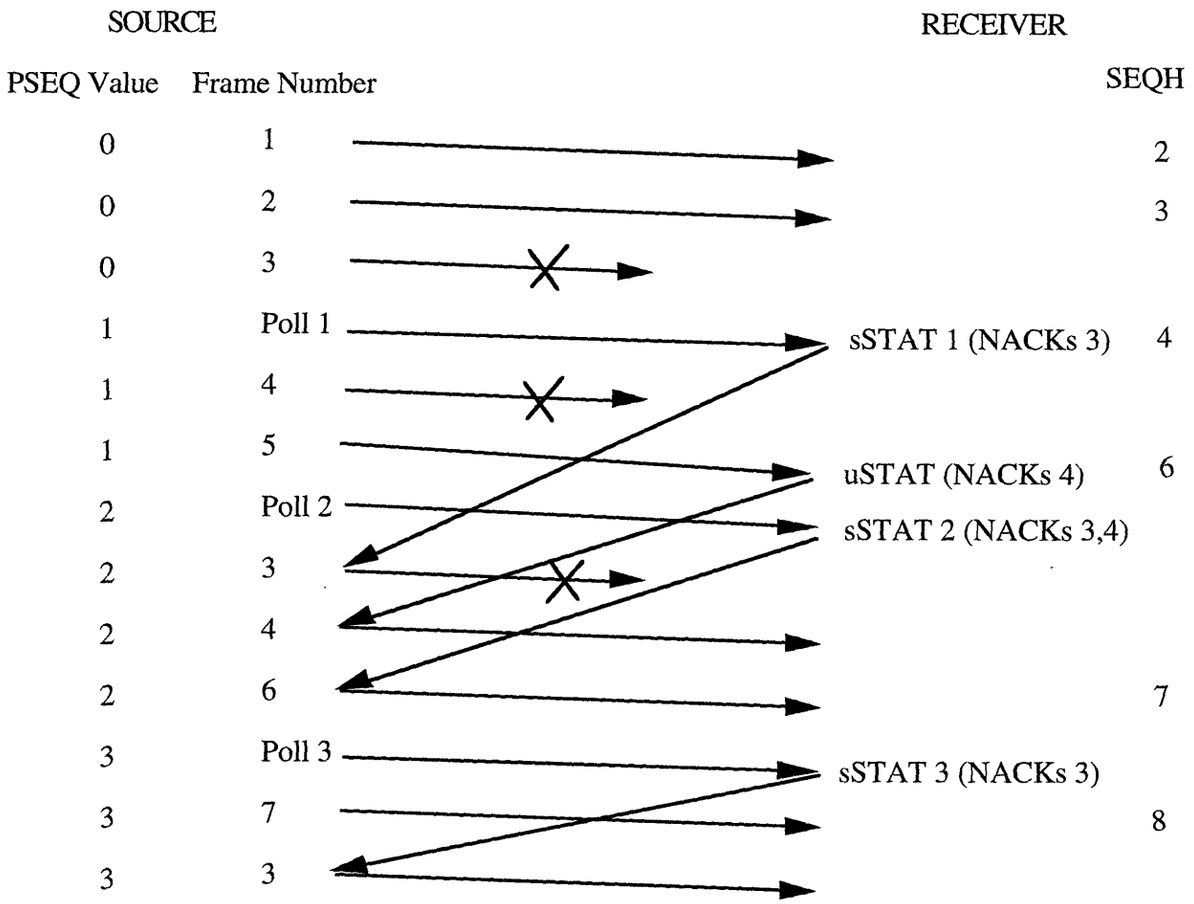


Figure 7.1 Operation of ATM retransmission scheme, under selective repeat.

7.3 PROOF OF PROPER OPERATION

As can be seen from the previous section, the proposed ATM retransmission algorithm requires the maintenance of several variables, and involves many special cases and ad hoc additions. Thus far, the standards committee has not formally proved that the retransmission scheme works. The goal of this section is to prove the correctness of the protocol. We must show that the source can continue forever to accept frames for transmission from the higher layer, and that all frames are eventually delivered in proper sequence at the destination. We also need to show that the claim of no unnecessary retransmissions is true.

7.3.1 Conditions of Normal Operation

It is easy to come up with special circumstances where the proposed ATM retransmission scheme does not work correctly. Thus, we first must define the conditions of "normal operation". Under these conditions, we can prove the correctness of the protocol. The conditions of normal operation are:

1) Frames (data or control) travel in order on the links.

2) Undetected errors do not occur.

3) State information at the source and receiver is not lost.

4) The connection does not go down.

5) There is some $q > 0$ such that each frame (data or control) is received error-free with probability at least q.

6) All transmission and propagation delays are finite.

7) If polls are numbered modulo M_P, then no more than M_P-2 consecutive poll/status message combinations are lost (i.e., one out of every M_P-1 consecutive polls arrives successfully at the destination, and the corresponding status message arrives successfully at the source).

8) If frames are numbered modulo M_F, and the oldest unacknowledged frame at the source is SEQA, then the source does not transmit past frame SEQA + M_F - 2. (Due to the flow control window, the upper limit of the send window may be much smaller than this.)

9) If polls are numbered modulo M_P, and the solicited STAT most recently received by the source contained a poll sequence number of PSEQL, then the source does not transmit past poll number PSEQL + M_P - 1.

The first seven conditions deal with properties of the network that must hold to guarantee proper operation. The last two conditions should really be specified as part of the protocol. Also, we make the obvious assumption that a frame cannot be received before it is sent. We also assume that in the initial state of the system, there are no frames on any of the links.

Below, we prove the proper operation of the protocol given the above conditions. We first prove the protocol works if all sequence numbers are integers that can increase without bound. In Section 7.3.6, we consider the case where frame sequence numbers are integers modulo M_F and poll sequence numbers are integers modulo M_P. The methodology of the proof closely follows that used in [BeG92] to prove the correctness of general Go Back N schemes.

7.3.2 Source and Destination Algorithms

First, let's review the definitions of the various variables, and precisely state the algorithms at the source and destination. In this section, we assume all variables and sequence numbers are ordinary integers.

SEQA = oldest unacknowledged frame at source
SEQ = one greater than highest numbered frame transmitted by source
PSEQ = sequence number of poll most recently sent by source
PSEQL = poll sequence number contained in solicited STAT most recently received by source
RTS = logical variable associated with each frame transmitted by source. It is initialized to FALSE, and set to TRUE if the frame is retransmitted due to an unsolicited STAT
SEQR = lowest consecutive frame that the destination hasn't correctly received
SEQH = one greater than the highest numbered frame that the destination knows the source has sent

We also use the following conventions:

$PSEQ_P$ = PSEQ number sent in a poll
SEQ_P = SEQ number sent in a poll
$PSEQ_S$ = PSEQ number contained in a solicited STAT
$SEQR_S$ = SEQR number contained in a STAT (either solicited or unsolicited STAT)
P_X = PSEQ stamp of frame number X (stored in retransmission buffer)

Algorithm at Source

1. Initialize SEQA, SEQ, PSEQ, and PSEQL to 0. The first transmitted data frame contains sequence number 0. The first transmitted poll contains poll sequence number 1.

2. Do steps 3 through 8 repeatedly. Steps 5 through 8 are performed whenever the conditions are met. Steps 3 and 4 are done repeatedly within finite intervals chosen by the source.

3. If SEQ < SEQA + M_F - 1, and a frame is available from the higher layer, assign frame number SEQ to the frame, and increment SEQ by one (again, this may not be possible due to the flow control window). Transmit frame, and store frame in retransmission buffer with current value of PSEQ, and set the corresponding RTS variable to FALSE.

4. If PSEQ < PSEQL + M_P -1, increment PSEQ by one. Transmit a poll containing poll number PSEQ and the current value of SEQ.

5. If a STAT is received with $SEQR_S$ > SEQA, increase SEQA to $SEQR_S$.

6. When a solicited STAT is received, set PSEQL to $PSEQ_S$.

7. If a NACK of frame number X is received in a solicited STAT containing $PSEQ_S$, then retransmit frame X if P_X < $PSEQ_S$. If frame X is retransmitted, update P_X to current value of PSEQ.

8. If a NACK of frame number X is received in an unsolicited STAT, retransmit X if the corresponding RTS variable equals FALSE. If frame X is retransmitted, set P_X to current value of PSEQ, and set RTS to TRUE.

Algorithm at Destination

1. Initialize SEQR and SEQH to 0. Do steps 2, 3, and 4 repeatedly.

2. If receive a frame with sequence number X equal to SEQR, accept the frame, and increment SEQR. Perform the following loop:

```
While (SEQR equals the sequence number of a frame already in resequencing buffer) {
        increment SEQR
        }
```

Pass up to the higher layer all frames from X through the new value of SEQR-1 (in proper sequence). If X ≥ SEQH, update SEQH to X + 1.

3. If receive a frame with sequence number X such that SEQR < X ≤ SEQR + M_F - 2, store the frame in the resequencing buffer. (The receive window may actually be smaller due to flow control restrictions.) If X > SEQH, NACK all missing frames between SEQH and X-1, inclusive (if any), in an unsolicited STAT. Set $SEQR_S$ in the STAT to the current value of SEQR. As an option, two identical unsolicited STATs can be sent. If X ≥ SEQH, update SEQH to X + 1.

4. If receive a poll with SEQ_P > SEQH, optionally NACK all missing frames between SEQH and SEQ_P - 1, inclusive (if any), in an unsolicited STAT. Set $SEQR_S$ in the STAT to the current value of SEQR. Also, if SEQ_P > SEQH, update SEQH to SEQ_P. Regardless of whether the unsolicited STAT is sent, send a solicited STAT containing $PSEQ_S$ equal to $PSEQ_P$, containing $SEQR_S$ equal to the current value of SEQR, and NACKing all missing frames with sequence number less than SEQ_P.

Throughout our discussion, it is assumed that each of the steps in the above algorithms at the source and destination can be viewed as an indivisible operation i.e., once a given step is started, no other operations are performed until the entire step is finished. When we refer to transmissions at the source and destination, we assume the frame or control message is passed down to a transmit queue at a lower layer, which is served in First In First Out order.

To prove the correctness of this protocol, we must show *safety* and *liveness*.[BeG92] We also need to show that the algorithm does not produce unnecessary retransmissions.

7.3.3 Safety Condition

The algorithm is safe if it never delivers an out-of-sequence frame to the higher layer at the destination. From step 2 of the algorithm at the destination, we see that frames must be passed up in sequence, so that the algorithm is safe.

7.3.4 Liveness Condition

The algorithm is live if the source can continue forever to accept packets from the higher layer, and the destination continues to deliver them to the higher layer.

Let SEQA(t), SEQ(t), PSEQ(t), SEQR(t), and SEQH(t) represent the value of these five variables at time t. From the algorithm statement, it can be seen that all five must be non-

decreasing in t. Define a **successful poll** as a poll that gets to the destination error-free, and whose corresponding solicited STAT gets to the source error-free.

Consider any transmitted frame with sequence number X. Refer to Figure 7.2. Let t_{Ti} equal the transmission time of the i^{th} copy of frame X. Polls are sent periodically within finite intervals. Due to condition (5), some poll sent after time t_{Ti} must be successful. Let t_{Pi} be the transmission time of the first successful poll sent after time t_{Ti}. Let t_{Ri} be the time this poll arrives at the destination. Let t_{Si} equal the time the STAT corresponding to this poll arrives at the source. Thus, $t_{Ti} < t_{Pi} \leq t_{Ri} \leq t_{Si}$. Since we assume finite delays, and because of condition 5, t_{Ri}, and t_{Si} are finite.

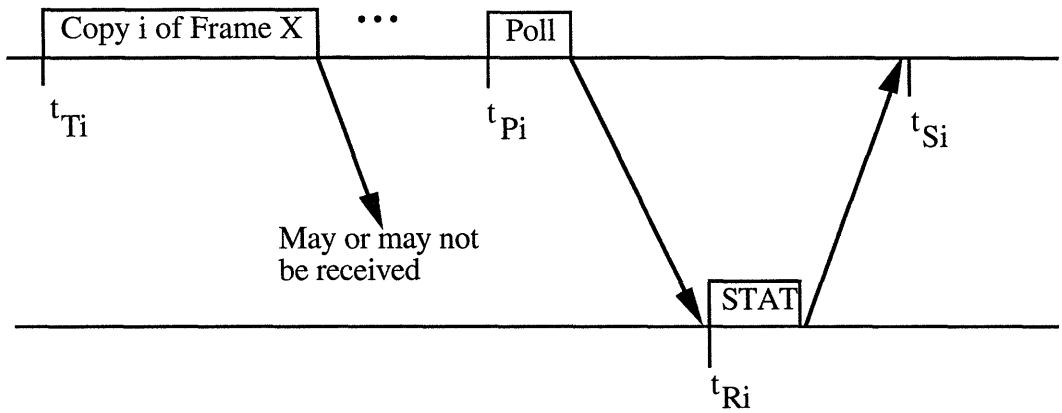


Figure 7.2 Copy i of frame X is transmitted at time t_{Ti}. Some poll that it is transmitted after time t_{Ti} will be successful. We assume such a poll is sent at time t_{Pi} and received at the destination at time t_{Ri}. The corresponding solicited status message is received at the source at time t_{Si}.

Let $P_X(t)$ represent the PSEQ stamp of frame X at time t. Then:

$$P_X(t) = PSEQ(t_{Ti}) \quad \text{for } t_{Ti} \leq t < t_{T(i+1)} \qquad (7.1)$$

where we take $t_{T(i+1)}$ to be ∞ if X is not transmitted for the $(i+1)^{th}$ time. Also, since SEQ is one greater than any transmitted frame:

$$SEQ(t) > X \quad \text{for } t > t_{Ti} \qquad (7.2)$$

We want to show that if the i^{th} copy of frame X has been transmitted, either this copy will be received successfully at the destination, or copy i+1 of frame X will be transmitted. (In the next section, we show that both events do not occur.)

The successful poll sent at time t_{Pi} contains $PSEQ_P = PSEQ(t_{Pi}^-) + 1$ and $SEQ_P = SEQ(t_{Pi}^-)$, where t_{Pi}^- represents the start of the poll transmission operation, before PSEQ is updated. Since $t_{Pi}^- > t_{Ti}$, from (7.2), we have:

$$SEQ_P > X \qquad (7.3)$$

Also, the nondecreasing property of PSEQ(t) implies that:

$$PSEQ_P > PSEQ(t_{Ti}) \qquad (7.4)$$

At time t_{Ri}, one of the following 2 conditions must hold:

a) Frame X has been received by the destination. Since t_{Ri} is finite, this implies frame X has been received in finite time.

b) Frame X has not been received by the destination. Since the poll was sent after frame X, and it arrives before frame X, it means frame X is lost (since frames travel in order). The poll triggers a solicited STAT NACKing frame X since, as shown above, $SEQ_P > X$. The solicited STAT, with $PSEQ_S$ equal to $PSEQ_P$, arrives at the source at time t_{Si}. It is possible an unsolicited STAT NACKing copy i of frame X has already been received by the source prior to t_{Si}, in which case copy i+1 may already have been sent. If not, then, from equation (1), $P_X(t_{Si}) = PSEQ(t_{Ti})$. Combining this with equation (7.4) yields:

$$P_X(t_{Si}) = PSEQ(t_{Ti}) < PSEQ_P = PSEQ_S$$

Thus, since $P_X(t_{Si}) < PSEQ_S$, the condition of step 7 in the algorithm at the source is satisfied, and frame X is retransmitted.

We conclude that, given copy i of frame X has been sent, either copy i will be successfully received, or copy i+1 will be transmitted. From condition 5, we assume that frames are successfully received with probability greater than some non-zero q. Thus, eventually frame X will be received successfully. (Note that even without unsolicited STATs, frame X is eventually received successfully; only solicited STATs are needed to satisfy liveness.)

Now, let Y be the value of SEQA at any time t. We know that all frames before Y have been received at the destination and have been ACKed; thus, frame Y must fall within the receive window. From what was shown above, frame Y will eventually be received at the destination in finite time. At the time Y is accepted at the destination, SEQR will be incremented beyond Y. The destination will be able to deliver to the higher layer all frames through at least frame Y. The next successful poll sent after the successful transmission of Y will generate a solicited STAT with $SEQR_S > Y$. Thus, at the time this STAT arrives at the source, SEQA will be incremented beyond Y.

We conclude that the value of SEQA is always incremented after some finite time (assuming there is data to send). This allows the higher layer at the source to continue to submit frames.

We have shown that the algorithm is live: the higher layer at the source can continue to submit frames, and the destination will continue to deliver them. Next, we need to show the algorithm does not produce any unnecessary retransmissions.

7.3.5 No Unnecessary Retransmissions

We define an unnecessary retransmission as occurring when copy j of a frame is sent even though copy i, for some $i < j$, is not lost. If copy i+1 is not sent unless copy i is lost, then we know that copy j, for all $j > i$, will not be sent unless copy i is lost. Thus, we just need to consider copies i and i+1.

In the discussion below, we examine whether copy i+1 of an arbitrary frame, say frame X, could ever be unnecessarily sent. In order for copy i+1 to be unnecessary it must be true that copy i of frame X does arrive (and is accepted) at the destination.

There are 3 possible ways a NACK of frame X can be generated (for simplicity, SEQH is used rather than SEQH(t)):

1) Frame Y arrives at the destination, and at the time of its arrival X has not arrived and been accepted, and the following holds: $Y > X \geq SEQH$. An unsolicited NACK of frame X is sent and SEQH is updated to Y+1.

2) A poll arrives at the destination, and at the time of its arrival X has not arrived and been accepted, and the following holds: $SEQ_P > X \geq SEQH$. An unsolicited NACK of frame X is optionally sent. A solicited NACK of frame X must be sent. After the NACKs are sent, SEQH is updated to SEQ_P.

3) A poll arrives at the destination, and at the time of its arrival X has not arrived and been accepted, and the following holds: $SEQ_P > X$ and $SEQH > X$. A solicited NACK of frame X is sent. If $SEQ_P > SEQH$, then after the NACK is sent, SEQH is updated to SEQ_P.

First, consider the case where $i = 1$. Assume copy 1 of frame X arrives (and is accepted) at the destination. Copy 1 of frame X must be sent before any copy of frame Y, where Y >

X, and must be sent before a poll with SEQ_P, where $SEQ_P > X$. Given that frames travel in order, such a frame Y or such a poll cannot arrive before frame X. Thus, the conditions in the three procedures above are not satisfied; copy 1 of frame X will not be NACKed, so no further copies of frame X will be sent.

Next, consider the case where $i > 1$. Since frame X has been sent more than once, and since frames are only retransmitted in response to NACKs, it must be true that frame X was NACKed at least once. From statements 1, 2, and 3 above, it must be true that after frame X is NACKed, SEQH is updated to a value greater than X. (In statement 3, SEQH is already greater than X.) SEQH is nondecreasing. Thus, after frame X has been NACKed once, statements 1 and 2 above can never be satisfied, since they require $X \geq$ SEQH (i.e., an unsolicited STAT can only NACK copy 1 of a frame.) (For now, ignore the case where two identical unsolicited STATs are sent.)

Thus, we only need to consider statement 3 above. Let t_i be the time that copy i of frame X is transmitted. Let P_{Xi} be the PSEQ stamp associated with copy i of frame X. Then $P_{Xi} = PSEQ(t_i)$. We assume that copy i is received and accepted by the destination. We consider whether any poll can trigger an unnecessary transmission of copy i+1 of frame X.

a) First, consider a poll sent before t_i. Its poll sequence number, $PSEQ_P$, satisfies $PSEQ_P \leq PSEQ(t_i)$. Thus, the corresponding solicited STAT would contain $PSEQ_S \leq PSEQ(t_i) = P_{Xi}$. Thus, copy i+1 of frame X would not be transmitted since P_{Xi} is not less than $PSEQ_S$.

b) Next, consider a poll sent after t_i. If copy i of frame X gets to the destination, then it must arrive before such a poll (since frames travel in sequence). Thus, this poll will not NACK frame X.

Thus, a poll will not trigger an unnecessary retransmission of frame X.

The last case we need to consider is where copy 1 of frame X is lost, and frame X is retransmitted due to an unsolicited STAT. The destination has the option of sending two identical unsolicited STATs. However, if frame X is retransmitted due to an unsolicited STAT, the variable RTS_X is set to TRUE, so that all future NACKs of frame X contained in unsolicited STATs are ignored. Thus, if both unsolicited STATs arrive at the source, the second one will be ignored.

Overall, we see that given copy i of frame X is received, copy i+1 will not be sent. Thus, no unnecessary retransmissions are produced.

7.3.6 Sequence Numbers with Modulus

In the sections above, we showed that the protocol works correctly if sequence numbers can increase without bound. In this section, we show that the protocol continues to work if frame sequence numbers are treated modulo M_F, and poll sequence numbers are treated modulo M_P.

7.3.6.1 Received Frame Sequence Numbers

First, continue to assume that sequence numbers are integers increasing without bound. Consider the successful transmission of an arbitrary frame sent by the source. Assume it is transmitted at time t_1 and received at the destination at time t_2. Obviously, $t_2 \geq t_1$. The sequence number of the frame, say X, must lie in the source's send window at time t_1. Thus:

$$SEQA(t_1) \leq X \leq SEQA(t_1) + M_F - 2 \quad (7.5)$$

From step 5 of the algorithm at the source, it must be true that $SEQA(t) \leq SEQR(t)$ for all t. (If $SEQA(t) > SEQR(t)$, it would mean the source received an ACK for a frame that was not received by the destination.) Thus, for any $t_1 \leq t_2$,

$$SEQA(t_1) \leq SEQR(t_1) \leq SEQR(t_2) \quad (7.6)$$

We showed above that the protocol does not produce unnecessary retransmissions. Thus, the transmission of frame X must be necessary. Thus, $SEQR(t_2) \leq X$. (If $SEQR(t_2)$ were greater than X, it would mean frame X has already been received by time t_2.) Combining this with equations (7.5) and (7.6), yields:

$$SEQA(t_1) \leq SEQR(t_2) \leq X \leq SEQA(t_1) + M_F - 2 \quad (7.7)$$

The destination accepts frames in the range from $SEQR(t_2)$ to $SEQR(t_2) + M_F - 2$. (The acceptance window may actually be smaller for flow control purposes.) We have: $0 \leq (X - SEQR(t_2)) \leq M_F - 2$. Thus, X falling in the range $SEQR(t_2)$ to $SEQR(t_2) + M_F - 2$ is equivalent to X mod M_F falling in the range $SEQR(t_2)$ mod M_F to $(SEQR(t_2) + M_F -$

2) mod M_F. Thus, treating the sequence numbers as integers modulo M_F does not affect the acceptance policy at the destination.

Note that we need to define what it means to "fall in the range" of A and B, where A and B are numbers modulo M_F. One can envision the numbers from 0 to M_F-1 on a circle, increasing clockwise. Then for any two numbers A and B, we define the region that extends clockwise from A to B as representing the numbers that fall between A and B. This is shown in Figure 7.3.

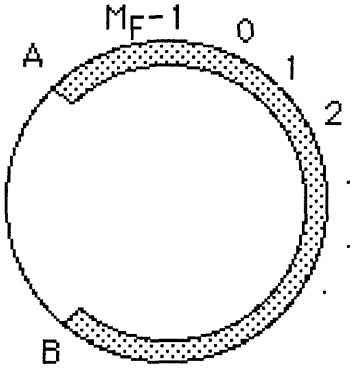


Figure 7.3 The shaded region represents the numbers that fall between A and B.

Note that the property of no unnecessary retransmissions is important in the above argument. In general selective repeat systems, where unnecessary retransmissions can occur, if the modulus is M_F, then ambiguity can occur if the source transmits past SEQA+M_F/2 - 1 (as opposed to SEQA + M_F - 2 for the ATM scheme).[BeG92]

7.3.6.2 Generating Solicited Status Messages

Again, assume increasing integers are used for sequence numbers. Assume a poll is transmitted at time t_1, and arrives at the destination at time t_2. Obviously, $t_1 \leq t_2$. Assume the poll contains SEQ_P, where $SEQ_P = SEQ(t_1)$. Thus, at time t_1, all frames through SEQ_P - 1 have been transmitted. Due to the restriction on the send window, we know that:

$$SEQ_P - 1 \leq SEQA(t_1) + M_F - 2 \qquad (7.8)$$

Since $SEQ(t) \geq SEQA(t)$ for all t, we have:

$$SEQA(t_1) \leq SEQ(t_1) = SEQ_P \leq SEQA(t_1) + M_F - 1 \quad (7.9)$$

In equation (7.6), we showed $SEQA(t_1) \leq SEQR(t_2)$ for any $t_1 \leq t_2$. The value of $SEQR(t_2)$ indicates that the source must have sent the frame with sequence number $SEQR(t_2) - 1$ at some time prior to t_1, say at time t_0. (If the frame were sent after t_1, it could not have arrived prior to the poll that was sent at t_1.) Using the restriction on the send window at the source, $SEQR(t_2) - 1 \leq SEQA(t_0) + M_F - 2$. Since SEQA(t) is non-decreasing in t, we arrive at:

$$SEQA(t_1) \leq SEQR(t_2) \leq SEQA(t_1) + M_F - 1 \quad (7.10)$$

We have shown that both SEQ_P and $SEQR(t_2)$ lie between $SEQA(t_1)$ and $SEQA(t_1) + M_F - 1$. A poll arriving at time t_2 generates NACKs of frames between $SEQR(t_2)$ and $SEQ_P - 1$, inclusive. Thus, the sequence numbers can be treated modulo M_F without causing ambiguity with NACKing frames in solicited STATs.

7.3.6.3 Generating Unsolicited Status Messages

Again, assume increasing integers are used for sequence numbers. Unsolicited STATs can be generated by the arrival of a frame or a poll. An unsolicited STAT is always sent if a frame with sequence number X arrives at time t before a frame with sequence number between SEQH(t) and X-1, inclusive (where SEQH(t) is the value of SEQH before it is updated due to the arrival at time t). An unsolicited STAT is optionally sent if a poll containing SEQ_P arrives at time t before a frame with sequence number between SEQH(t) and $SEQ_P - 1$, inclusive. Let t_1 be the transmission time at the source of the frame or poll that generates the unsolicited STAT, and let t_2 be the arrival time of the frame or poll at the destination.

From the definition of the protocol, it must be true that $SEQR(t) \leq SEQH(t)$ for all t. Thus, using equation (7.6), we know $SEQA(t_1) \leq SEQR(t_2) \leq SEQH(t_2)$. The value of $SEQH(t_2)$ indicates that the source must have sent the frame with sequence number $SEQH(t_2) - 1$ at some time prior to t_1, say at time t_0. Using the restriction on the send window at the source, $SEQH(t_2) - 1 \leq SEQA(t_0) + M_F - 2$. Since SEQA(t) is non-decreasing in t, we arrive at:

$$SEQA(t_1) \leq SEQH(t_2) \leq SEQA(t_1) + M_F - 1 \quad (7.11)$$

From the restriction on the send window, we know that if frame X is sent at time t_1, then $X \leq SEQA(t_1) + M_F - 2$. Or, if a poll with SEQ_P is sent at time t_1, then $SEQ_P - 1 \leq SEQA(t_1) + M_F - 2$. Thus, using the rule for generating NACKs, any frame Y NACKed in an unsolicited STAT satisfies:

$$SEQA(t_1) \leq SEQH(t_2) \leq Y \leq SEQA(t_1) + M_F - 2 \qquad (7.12)$$

Therefore, sequence numbers can be treated modulo M_F without causing ambiguity with frames NACKed by unsolicited STATs.

7.3.6.4 Receiving Status Messages at Source

Again, assume increasing integers are used for sequence numbers. Let t_1 be the time a STAT (either solicited or unsolicited) is sent by the destination, and let t_2 be the time the STAT arrives at the source. Let $SEQR_S$ be the value of SEQR contained in the STAT. Let $SEQA(t_2)$ be the value of SEQA at the time the STAT arrives (i.e., before SEQA is updated to $SEQR_S$). Thus, all frames through $SEQR_S - 1$ are being ACKed by this STAT.

A STAT ACKing frame $SEQR_S$ cannot be received before a STAT that ACKs only through $SEQR_S - 1$ (since SEQR(t) is non-decreasing in t and frames travel in sequence). Thus, $SEQA(t_2) \leq SEQR_S$. (If $SEQA(t_2)$ were greater than $SEQR_S$, it would mean that $SEQR_S$ had been ACKed by time t_2.) Also, a STAT cannot ACK a frame that has not been sent. Due to the restriction on the send window, $SEQR_S - 1 \leq SEQA(t_2) + M_F - 2$. Thus:

$$SEQA(t_2) \leq SEQR_S \leq SEQA(t_2) + M_F - 1 \qquad (7.13)$$

Thus, SEQA can be updated to $SEQR_S$ without ambiguity.

Now consider any NACKs contained in the STATs, and consider the sequence numbers as ordinary integers again. Let Y equal the sequence number of a NACKed frame in the STAT. By definition of the protocol, we know $Y \geq SEQR_S$.

First, consider the case where the STAT is an unsolicited STAT triggered by the arrival at the destination of a frame with sequence number X. By definition of the protocol, $Y < X$. We assumed the STAT is generated at time t_1; thus, we know that frame X has been sent by the source prior to t_1, say at time t_0. Due to the restriction on the send window, we

know that: $X \leq SEQA(t_0) + M_F - 2$. Thus, using the fact that SEQA(t) is non-decreasing in t:

$$Y < X \leq SEQA(t_0) + M_F - 2 \leq SEQA(t_2) + M_F - 2 \quad (7.14)$$

Combining this with equation (7.13) and the fact that $Y \geq SEQR_S$ yields:

$$SEQA(t_2) \leq Y \leq SEQA(t_2) + M_F - 2 \quad (7.15)$$

Next, consider the case where the STAT is triggered by a poll containing SEQ_P (the STAT can be solicited or unsolicited). By definition of the protocol, $Y < SEQ_P$. The STAT is generated at time t_1; thus, we know that a poll containing SEQ_P has been sent by the source prior to t_1. Thus, we also know that frame SEQ_P -1 must have been sent by the source before t_1, say at time t_0. Due to the restriction on the send window, we know that: $SEQ_P - 1 \leq SEQA(t_0) + M_F - 2$. Thus, using the fact that SEQA(t) is non-decreasing in t:

$$Y \leq SEQ_P - 1 \leq SEQA(t_0) + M_F - 2 \leq SEQA(t_2) + M_F - 2 \quad (7.16)$$

Combining this with equation (7.13) and the fact that $Y \geq SEQR_S$ yields:

$$SEQA(t_2) \leq Y \leq SEQA(t_2) + M_F - 2 \quad (7.17)$$

Thus, at the arrival time t_2 of any STAT, $SEQR_S$ and any NACK contained in the STAT fall between $SEQA(t_2)$ and $SEQA(t_2) + M_F - 2$.

Combining the last four sections, we see that for any time t, any STAT that arrives at the source at time t will contain NACKs and $SEQR_S$ that lie in the region SEQA(t) to $SEQA(t) + M_F - 1$. Also, SEQ(t) lies in this region due to the restriction on the send window. Also, consider any frame or poll that is transmitted at time t_1 and received at the destination at time t_2. Then, at t_2, the sequence number contained in the frame lies in the region $SEQA(t_1)$ to $SEQA(t_1) + M_F - 1$; or, if it's a poll that is received, the value of SEQ_P contained in the poll lies in this region. We also showed $SEQR(t_2)$ and $SEQH(t_2)$ lie in this region.

Thus, all frame sequence numbers and SEQA, SEQ, SEQR, and SEQH can be kept modulo M_F without affecting the operation of the protocol.

7.3.6.5 Poll Sequence Numbers

First consider poll sequence numbers as ordinary increasing integers. Let t_1 be the time a poll is transmitted by the source. Assume the poll contains $PSEQ_P$. Let t_2 be the time the corresponding solicited STAT gets back to the source (if it does). The STAT carries $PSEQ_S$ equal to $PSEQ_P$. PSEQL(t) represents the poll sequence number contained in the last received solicited STAT as a function of time. PSEQL(t) is non-decreasing in t. Since the STAT carrying sequence number $PSEQ_S$ does not arrive until time t_2, and since frames travel in sequence, $PSEQL(t_2) < PSEQ_S$ (we assume $PSEQL(t_2)$ represents the value of PSEQL at the arrival time of the STAT, before it is updated to $PSEQ_S$). At time t_1, $PSEQ_P$ must fall within the send window for polls. Thus, $PSEQ_P \leq PSEQL(t_1) + M_P - 1$. Since PSEQL(t) is non-decreasing in t, $PSEQ_P \leq PSEQL(t_2) + M_P - 1$. Thus, overall, we have:

$$PSEQL(t_2) < PSEQ_S = PSEQ_P \leq PSEQL(t_2) + M_P - 1 \qquad (7.18)$$

Thus, any solicited STAT that arrives at the source at time t contains a PSEQ value that is within the range from PSEQL(t)+1 to $PSEQL(t) + M_P - 1$, inclusive.

Now, let's consider the PSEQ stamps in the retransmission buffer. Consider any time t_2 when a solicited STAT arrives at the source. Obviously, the poll with $PSEQ_P$ equal to $PSEQ_S$ was sent before time t_2. Thus, $PSEQ(t_2) \geq PSEQ_P = PSEQ_S$.

At time t_2, the source examines each frame in the retransmission buffer:

- if a frame is ACKed by the STAT, the source removes it from the buffer
- if a frame is NACKed by the STAT and it is retransmitted then the PSEQ stamp of the frame is updated to $PSEQ(t_2)$.
- if a frame is NACKed by the STAT but it is not retransmitted, then the PSEQ stamp of the frame must have been greater than or equal to $PSEQ_S$.
- if a frame is in the buffer but not ACKed or NACKed by the STAT, then it must have been sent after the poll carrying $PSEQ_P$. Thus, the PSEQ stamp must be greater than or equal to $PSEQ_S$.

After the source finishes servicing the STAT, PSEQL is updated to $PSEQ_S$. From the above discussion we see that all frames in the retransmission buffer at this time will have a

stamp greater than or equal to $PSEQ_S$. Thus, due to the restriction on the poll send window, after the STAT is serviced, all PSEQ stamps in the retransmission buffer lie between PSEQL and PSEQL+ M_P-1, inclusive.

Thus, the poll sequence numbers can be treated modulo M_P without ambiguity problems.

7.3.7 Out-of-Sequence Retransmissions

From the discussion above, it would seem that for any two frames, X and X+1, copy i of frame X+1 is never sent before copy i of frame X. However, this is not true even if the conditions stated in Section 7.3.1 hold. Consider the following example, which is depicted in Figure 7.4 Assume the destination sends a solicited STAT NACKing frame X, and later sends an unsolicited STAT NACKing frame X+1. The unsolicited STAT will not NACK frame X since unsolicited STATs can only NACK frames that have not been NACKed previously. Assume the solicited STAT is lost. When the unsolicited STAT arrives, frames X+1 will be retransmitted before frame X is retransmitted. The protocol still functions properly as shown by our proof (e.g., this scenario does not produce unnecessary retransmissions), but it is a peculiar feature. It arises because unsolicited status messages do not contain the full status of the receiver.

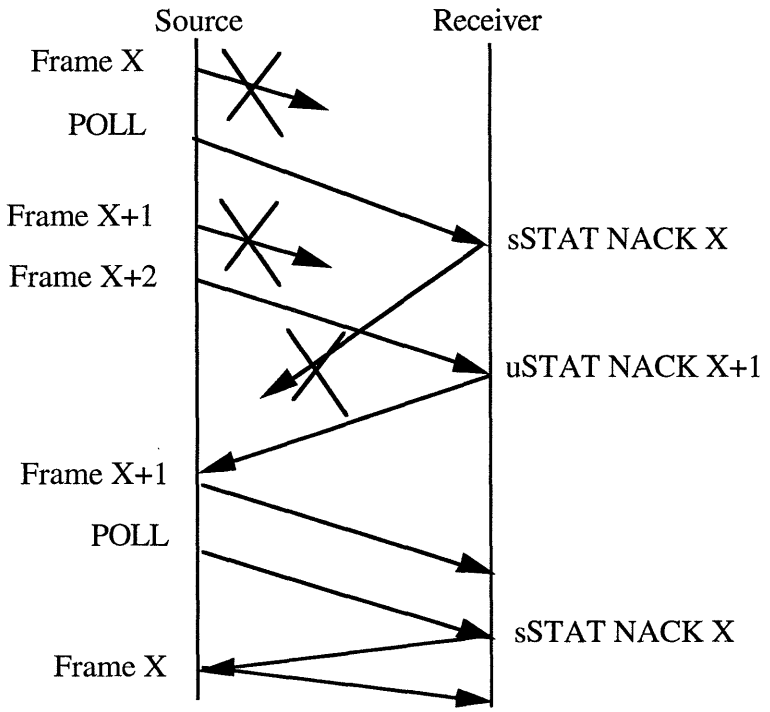


Figure 7.4 An unsolicited STAT NACKing frame X+1 arrives at the source before a solicited STAT NACKing frame X. Thus the second copy of frame X+1 is sent before the second copy of frame X.

7.4 POTENTIAL PROBLEMS

In the previous section, we examined the proposed ATM retransmission scheme under normal operating conditions. Here, we consider the problems that can arise if some of the conditions enumerated in Section 7.3.1 do not hold.

7.4.1 Limit on Outstanding Number of Frames

If SEQA is the oldest unacknowledged frame in the retransmission buffer, then the source is not permitted to transmit past frame (SEQA + M_F - 2)mod M_F, where we assume the frames are numbered modulo M_F. This was condition (8) in Section 7.3.1.

Assume there are no outstanding frames in a connection, and SEQA, SEQ, SEQR, and SEQH all equal S. Assume the source then violates the restriction on the number of outstanding frames, and transmits M_F frames. After sending precisely M_F frames, SEQ will again equal S. Assume all M_F frames are lost. If a poll is sent out after the M_F frames, it will contain SEQ equal to S. Upon receiving this poll, the destination responds with a status message containing SEQR equal to S. The source could interpret this as an ACK of all M_F frames. Thus, these M_F frames would not be retransmitted and would never be delivered to the destination.

7.4.2 Limit on Outstanding Number of Polls

If the solicited STAT most recently received by the source contained a poll sequence number of PSEQL, then the source is not permitted to transmit past poll number (PSEQL + M_P - 1)mod M_P. This was condition (9) in Section 7.3.1.

In Section 7.3.6.5, we showed that the stamp of any frame in the retransmission buffer must correspond to poll PSEQL or to a poll that was sent after poll PSEQL. Thus, due to condition (7), stamps in the retransmission buffer correspond to no more than M_P polls. If the source were to transmit poll number (PSEQL + M_P)mod M_P (which obviously carries sequence number PSEQL), then both the oldest frames in the buffer and the newest frames in the buffer could be stamped with PSEQL. This could lead to unnecessary retransmissions, as shown in Figure 7.5. In this example, frame 1 is lost and retransmitted. When it is retransmitted, the value of PSEQ has wrapped around to 0 again. Thus, the status message corresponding to poll 1 causes the unnecessary retransmission of frame 1. This is a serious problem since the correctness of the protocol relies on the fact that there are no unnecessary retransmissions.

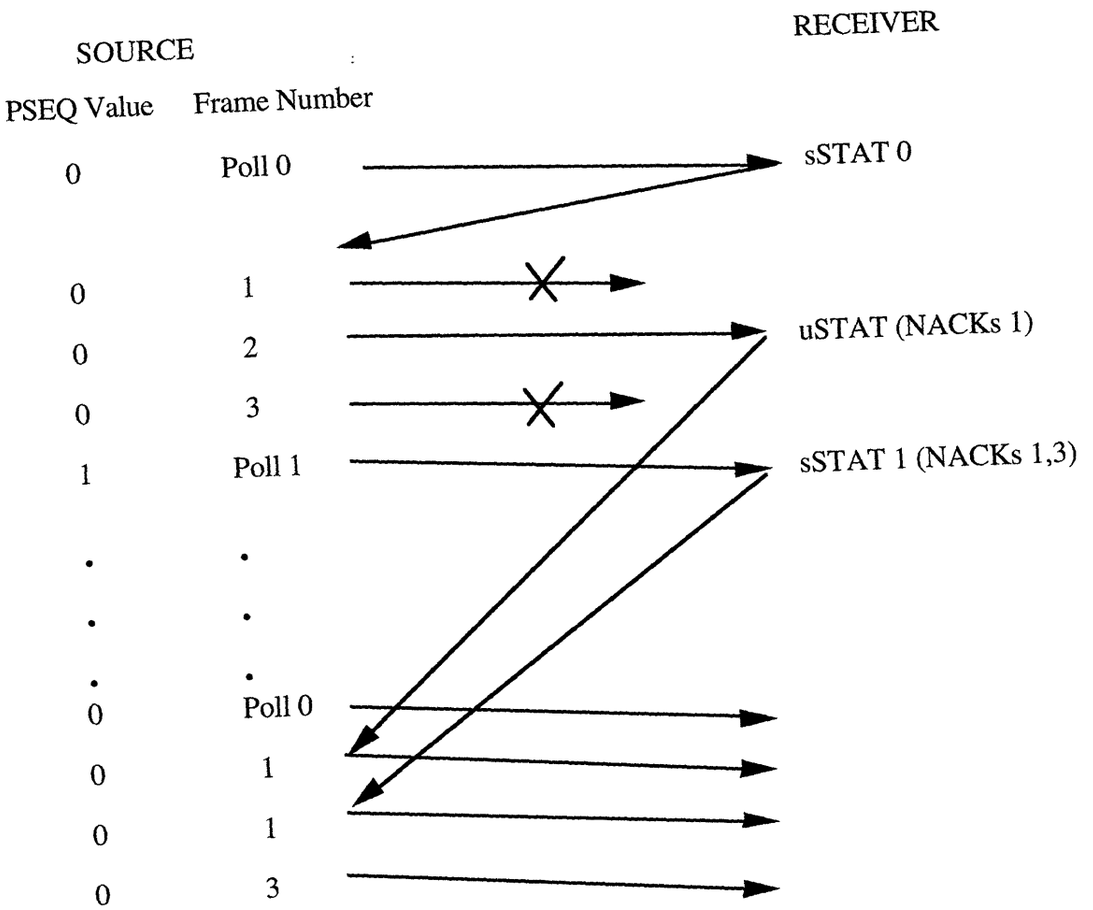


Figure 7.5 Unnecessary retransmission caused by the source violating the restriction on the number of outstanding polls.

7.4.3 Consecutive Lost Polls

Condition (7) in Section 7.3.1 stated that no more than M_P-2 consecutive poll/status message combinations are lost. If this condition does not hold, and the source adheres to condition (9), then the source would be unable to send any more polls. The source may be able to partially rely on unsolicited STATs for NACKs. However, if the last frames of a connection are lost, or if an unsolicited STAT is lost, then some frames will never be retransmitted. Or, if M_F-1 consecutive frames are successfully transmitted, the destination will not be able to ACK these frames due to the lack of polls. Due to the restriction of condition (8), the source will be unable to transmit any more frames. The protocol will be deadlocked.

To deal with this situation, the proposed scheme includes a variable Timer_NO_RESPONSE. If this timer expires without the source receiving a solicited STAT, the connection is terminated.

7.4.4 Out-of-Sequence Traffic

In this section, we consider problems that can arise if frames (either data frames or control frames) arrive out-of-sequence. As in typical systems, if a frame, say frame X, arrives out-of-sequence and very much delayed, it may be interpreted by the destination as being frame $X+M_F$. In addition, in the ATM scheme, out-of-sequence frames can result in unnecessary retransmissions, which in turn may lead to serious failures of the protocol. In Section 7.3, where we proved that the retransmission protocol works properly, it was assumed that unnecessary retransmissions do not occur. With this assumption, the destination was guaranteed not to receive frames prior to SEQR, since SEQR indicates the destination has received all frames through SEQR-1. Without this assumption, the protocol will likely fail, as is demonstrated in the following example.

Assume SEQA equals 0 and SEQ equals 100. Assume frames 0 through 50 have been received by the destination. Thus, SEQR and SEQH equal 51. Next, assume frame number 5 is retransmitted unnecessarily. This extra copy of frame 5 will be stored in the resequencing buffer at the destination, and will be treated as frame 5 in the 'next cycle' of 224 frames. Thus, an incorrect frame will be passed up to the higher layer at the destination. Also, when frame 5 arrives at the destination, the receiver will interpret this as being a frame 'greater than' SEQH. Thus, it will send an unsolicited STAT NACKing frames 'between' 51 and 4, inclusive, and it will set SEQH to 5. This erroneous NACK could result in more unnecessary retransmissions (although if the source realizes that the status message does not make sense, it would ignore it). The value of SEQH will be incorrect so that the unsolicited STAT mechanism will be corrupt.

Below we consider the various scenarios where frames travel out-of-sequence.

Scenario 1: Copy 1 of frame X+1 is transmitted after copy 1 of frame X, but arrives before it. An unsolicited STAT will be sent NACKing frame X, and frame X will be unnecessarily retransmitted.

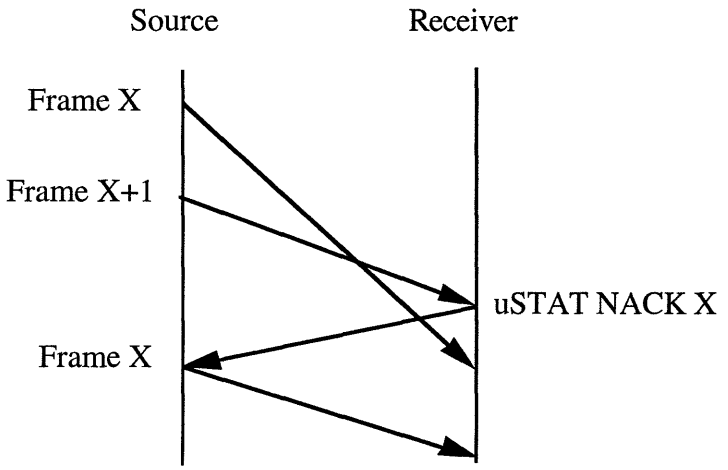


Figure 7.6 Frame X+1 arrives before frame X, triggering an unsolicited STAT of frame X.

Scenario 2: A poll message containing SEQ equal to X+1 (indicating frames up to and including sequence number X have been sent before this poll) arrives before frame X. The solicited STAT will NACK frame X and will result in an unnecessary retransmission.

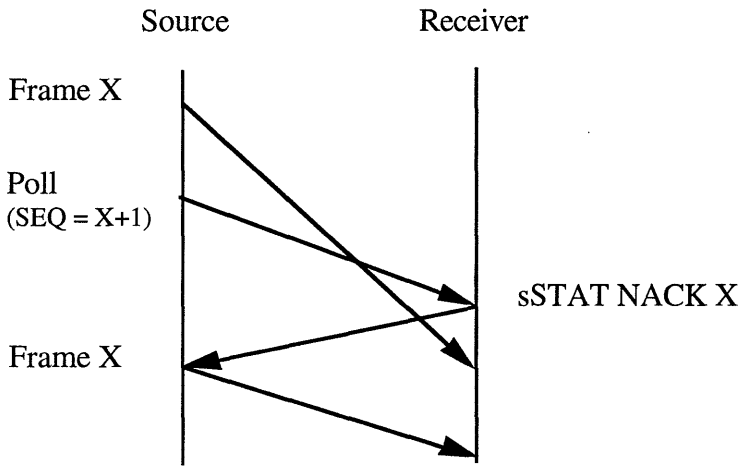


Figure 7.7 The poll arrives before a frame that was sent before it.

Scenario 3: Poll P-1 arrives at the destination before poll P, but the solicited STAT in response to poll P arrives at the source before the solicited STAT in response to poll P-1. After STAT P is serviced, PSEQL will be updated to P. There are several situations that can cause problems; we'll consider just one. Assume frame X is NACKed in STAT P-1. If frame X was also NACKed by STAT P and was retransmitted, then the PSEQ stamp of X must have been updated to a value that lies 'between' P and P-2. Since PSEQL will be updated to P, P-1 will be treated as if it is greater than the PSEQ stamp of X. Thus, frame X will be retransmitted unnecessarily due to STAT P-1.

Scenario 4: A solicited STAT arrives at the source before an unsolicited STAT that was sent earlier by the receiver. (Refer to Figure 7.8 below.) If the solicited STAT NACKs a frame that is also NACKed by the unsolicited STAT, then an unnecessary retransmission will result.

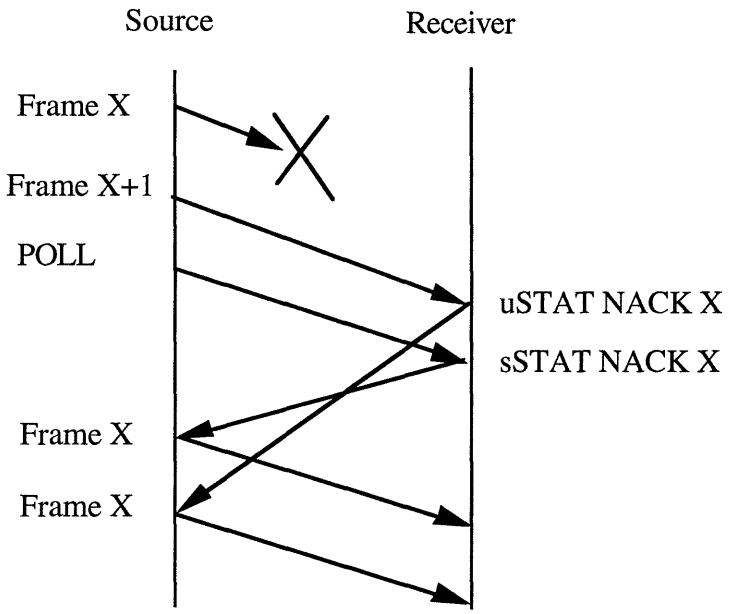


Figure 7.8 The solicited STAT arrives before the unsolicited STAT that was sent before it. No PSEQ comparisons are done when the unsolicited STAT arrives, so that frame X is unnecessarily retransmitted.

Scenario 5: An unsolicited STAT arrives at the source before a solicited STAT that was sent earlier by the receiver. The solicited STAT will not NACK anything NACKed by the unsolicited STAT (an unsolicited STAT cannot NACK a frame that has been previously NACKed). Thus, no problems are caused by this scenario.

To summarize, scenarios 1, 2, 3, and 4 may result in unnecessary retransmissions, which can cause serious problems with the protocol as indicated in the beginning of this section.

7.5 POLL-BASED SCHEME IN ATM ENVIRONMENT

In the previous sections, we analyzed the operation of the proposed ATM retransmission scheme. Here, we consider whether such a poll-based scheme is appropriate for the ATM environment.

7.5.1 Burst Losses

ATM is a high speed, integrated service network. The potential for low delay is one of the important advertised features of ATM. However, the analysis of chapter six showed that the weakness of poll-based schemes is that the retransmission delay may be quite large. The most serious problem is that poll-based schemes perform poorly when burst losses occur. If all frames sent in between poll transmissions are lost (but the polls are not), the expected excess retransmission delay doubles as compared to the expected delay if just a single frame is lost between polls (see Section 6.2.1.4). If the polls or status messages are lost also, then the expected excess retransmission delay can increase significantly more as shown in Section 6.2.1.5.

7.5.2 Unsolicited STATs

The proposed ATM protocol makes use of unsolicited STATs. If frames are comprised of N cells, and D_{max} is the desired maximum expected relative excess delay (assuming non-burst losses), then from equation 6.14 unsolicited STATs provide a delay benefit if:

$$R \geq (N+1)\left(\frac{1}{2D_{max}} - 1\right)$$

Assume D_{max} is 5% and the RTD is 50 msec. If the number of cells per frame, N, is 1, then unsolicited STATs provide a delay benefit if the data rate is greater than about 150 Kb/sec; if N is 100, then they are beneficial if the data rate is greater than about 8Mb/sec.

It obviously depends on the specific parameters of the connection whether unsolicited STATs are beneficial. Even if they do decrease the delay, it is not clear that this feature should be implemented. For example, as shown in Graph 6.5, at high speeds, low excess delay can be achieved even without the use of unsolicited STATs. Second, as we saw in Section 7.3, unsolicited STATs add a lot of complexity to the protocol. Thus, the protocol is more vulnerable to failure.

7.5.3 Selective Repeat vs. Go Back N

As discussed in Section 7.2, the proposed ATM retransmission protocol provides two options when a frame is received out-of-sequence. In the selective repeat (SR) option, the receiver stores out-of-sequence frames in a resequencing buffer, and requests that the source retransmit the missing frames. In the Go Back N (GBN) option, the receiver drops out-of-sequence frames.

The designers of ATM assume that almost all connections will use the SR option[T1S92]. In this section we analyze whether SR is a sensible option. As discussed in Section 6.6, the chief advantage of SR over GBN is greater bandwidth efficiency. The main disadvantage of SR is that the receiver must keep track of the status of each message it receives, and must resequence frames that arrive out of order.

Efficiency, is defined as $1/\gamma$, where γ is the expected number of transmitted data frames from source to destination per successfully accepted data frame at the destination. If cells are dropped independently with probability Q_R, and frames are comprised of N cells, and R cells are sent per RTD, then, from equation 6.17 and 6.18:

$$\text{GBN efficiency} = \frac{1 - N\,Q_R}{1 + N\,Q_R \left\lceil \frac{R+1}{N} \right\rceil}$$

$$\text{SR efficiency} = 1 - N\,Q_R$$

In ATM, however, we expect burst cell losses as well as random cell losses. As discussed in Section 6.6.1.2, the performance of GBN improves if cells are dropped in bursts rather than randomly, whereas the performance of SR is relatively unaffected by the burstiness of the losses (assuming the resequencing buffer is large).

Our goal is to determine whether the greater efficiency of SR is significant enough to justify the additional complexity. We will consider the case which produces the largest efficiency difference between SR and GBN, i.e., the case where cells are dropped randomly. We make the worst case assumption that any cell affected by a random bit error or a burst error is dropped. As usual, let P_R be the probability of a random bit error, P_B be the probability of a cell being hit by a burst error, and P_C be the probability a cell is dropped due to congestion. Then Q_R equals $((53)(8)\,P_R + P_B + P_C)$. If P_R equals 10^{-8}, P_B equals 10^{-7}, and P_C equals 10^{-6}, as assumed in chapter four, then Q_R equals $5\text{x}10^{-6}$. Graph 7.1 plots the efficiency of SR and GBN for this value of Q_R. The difference in performance is very small. (The graphs change very little if N is varied.)

As stated in chapter four, the congestion loss rate may be greater than 10^{-6}, depending on how well the flow control works. Graph 7.2 plots the efficiency if P_C increases to 10^{-4} (Q_R would also increase to 10^{-4}). Again, the difference is not significant. Recall that the

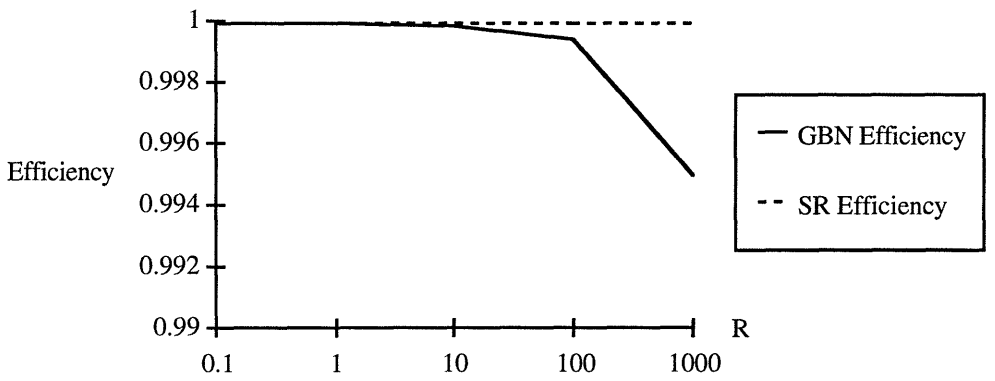
difference will be even less if the cells are dropped in bursts, which is likely to be the case if the congestion drop rate is high.

As an extreme case, we could consider P_C (and Q_R) equal to 10^{-3}. The efficiency of SR and GBN for this value of Q_R is shown on Graph 7.3. The efficiency difference between SR and GBN is significant only for R greater than 100. If the RTD is 50 msec, an R of 100 corresponds to a data rate of 850 Kb/sec.

Thus, only at this extremely high rate of cell loss does SR provide a significant improvement over GBN. Certainly, a cell loss rate of 10^{-3} is not expected. Thus, it does not appear that SR is warranted.

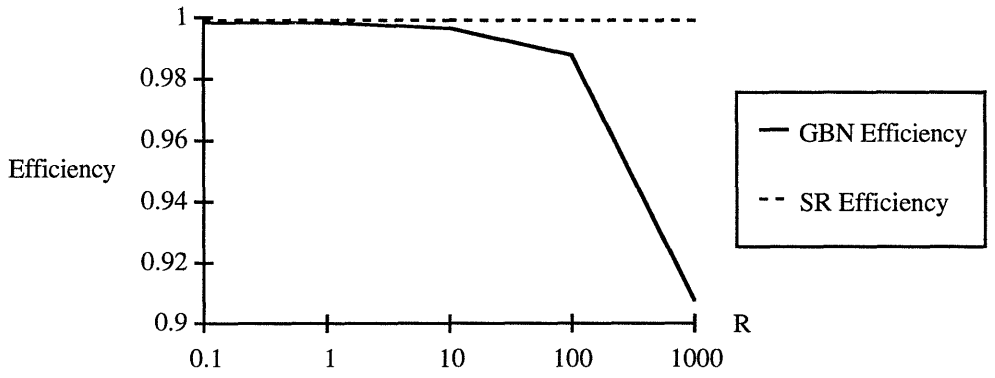
Graph 7.1 Efficiency of GBN and SR

(N =10 and Cell Loss Rate = 5E-6)

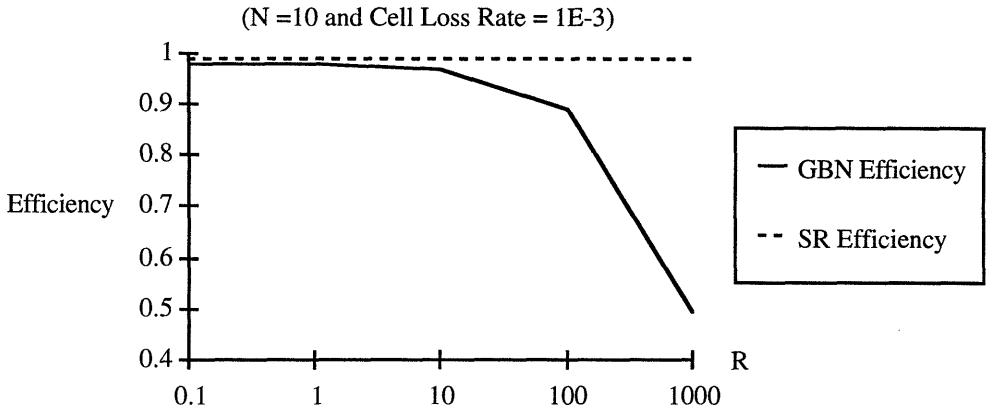


Graph 7.2 Efficiency of GBN and SR

(N =10 and Cell Loss Rate = 1E-4)



Graph 7.3 Efficiency of GBN and SR



7.5.4 Timer-Based Schemes

As discussed above, the chief problem with a poll-based scheme in an ATM environment is the poor performance during burst losses. Here, we consider whether timer-based schemes are a better option. In timer-based schemes, the source virtually maintains a timer for each transmitted frame. If an ACK of a frame is not received by the time its associated timer expires, the frame is retransmitted. The source needs to estimate the RTD in order to determine how to set the timer.

The high speed of the data links in ATM means that queueing delays should be small relative to propagation delay. Thus, the RTD should not vary much, allowing the source to accurately set its timers. As discussed in Section 6.3.1, this will result in low excess retransmission delay.

The retransmission process in timer-based schemes is generally unaffected by burst losses. The key difference is that timer-based schemes produce retransmissions if an ACK is not received; poll-based schemes rely on a poll or a data frame getting through to the destination, and a status message getting through to the source in order to generate a retransmission. If the system is in a state where it is dropping many frames, it is not expedient to rely on a successful transmission in order to generate retransmissions. The tradeoff of using a timer based scheme, of course, is that unnecessary retransmissions may occur. Given that delay is more important than bandwidth efficiency in ATM systems, this is a sensible tradeoff.

7.6 Conclusions

In this chapter, we have analyzed the proposed ATM retransmission scheme. We have proved that under reasonable conditions, it will generate retransmissions of lost frames without producing unnecessary retransmissions. However, if frames travel out-of-order, which is not expected in ATM, the protocol will fail. Also, if many consecutive poll/status messages are lost, the protocol will deadlock, or the connection will be declared dead.

We showed that poll-based retransmission schemes are not the best choice for ATM. Dropped frames in ATM are likely to occur in bursts, due to burst errors and congestion. With poll-based schemes, the retransmission delay may significantly increase when burst losses occur. Timer-based schemes are more effective at providing low excess retransmission delay. Given the importance of being able to provide low delay in ATM, timer-based schemes are more appropriate.

Also, as discussed in Section 6.3.1, delay in poll-based schemes is very quantized. One of the reasons the transmission method in ATM is asynchronous is to provide the flexibility to deal with a wide range of traffic types. Poll-based schemes are inherently quantized which removes some of this flexibility. Timer-based schemes allow a much more continuous form of control. The source can be flexible in adjusting its estimate of the RTD to achieve the right balance between delay and unnecessary retransmissions.

CHAPTER 8

RETRANSMISSION IN TCP/IP

8.1 INTRODUCTION

In this chapter, we analyze the TCP timer-based retransmission scheme. As indicated in chapter six, the performance of timer-based schemes relies heavily on how well the source can estimate the round trip delay from source to destination and back to the source. Obtaining an accurate estimate of the round trip delay is quite difficult in a TCP/IP environment. A large portion of this chapter is devoted to analyzing why the estimation of the round trip delay is difficult, and what modifications can be made to the TCP scheme to improve its performance.

In Section 8.2, we describe the TCP retransmission scheme. The protocol provides several retransmission options; in Section 8.3, we examine the advantages and disadvantages of the various options. In Section 8.4, we discuss the difficulties involved with estimating the round trip delay; we also suggest some improvements that could be implemented. Finally, in Section 8.5, we analyze how a poll-based scheme might perform in a TCP/IP environment.

As described in chapter five, TCP/IP uses different terminology than we have adopted in our general analysis. The TCP layer corresponds to layer N; the PDU at this layer is referred to as a segment rather than a message. Segments are variable length, with the maximum length being 65,536 bytes. Retransmissions are performed end-to-end at the TCP layer, and a segment is the unit of retransmission. The IP layer performs the functions of both layers N-1 and N-2. The IP PDU is variable length and is called a fragment.

8.2 DESCRIPTION OF TCP RETRANSMISSION SCHEME

8.2.1 Sequence Numbers

Each byte of data in a TCP connection can be thought of as being numbered sequentially modulo 2^{32}. There is a 32 bit field in the segment header that indicates the sequence number of the first byte of data in the segment.

The sequence number for the very first byte of data for a connection is obtained from the current value of a sequence number counter. The counter, incremented every 4 μsec, cycles through all 2^{32} sequence numbers in about 4.5 hours. The counter runs independently of the data being sent. The purpose of the counter is to prevent old data which may still be circulating in the network from having a sequence number that overlaps with the current connection. If the sequence numbers of old data and current data overlap, then it is possible that the receiver may not be able to distinguish which segments are current. Obviously, the counter is not guaranteed to prevent this problem. For example, data may circulate within a subnetwork for more than 4.5 hours, so that its sequence number appears current. (Note that IP fragments have a 'Time To Live' field, but this is only decremented when a gateway is crossed.) The counter is tied to an external clock, so even if the source node fails, the counter keeps running.

8.2.2 Acceptance Policies

The receiver keeps track of which bytes of data have been received. It maintains the variable RECV_NEXT to indicate that all bytes up to but not including RECV_NEXT have been received successfully. The receive window extends from RECV_NEXT to RECV_NEXT + WND_SIZE -1, where WND_SIZE is the size of the window as specified by the layer above TCP. A receiver may only accept data that falls within the receive window.

There are two possible acceptance policies that a receiver may implement. The in-order acceptance policy specifies that a segment will only be accepted if its sequence number precisely matches RECV_NEXT. This is similar to the acceptance policy of a Go Back N scheme. If the segment extends past the end of the receive window, only that portion of the segment that falls within the window is accepted. In the in-window acceptance policy, any portion of a segment that falls within the receive window will be accepted. This is similar to the acceptance policy of a selective repeat scheme. Note that with either policy, it is possible that only portions of a segment may lie within the receive window and be accepted.

8.2.3 ACK Policies

TCP segments also serve as ACK messages. There is a one bit flag in the segment header that indicates whether or not the segment contains a valid ACK. There is a 32 bit field in the header to indicate the ACK number. Regardless of the acceptance policy used, the receiver sets the ACK field to RECV_NEXT. There are two acceptable policies for sending ACKs. In one option, the receiver sends an ACK as soon as it receives a valid segment. If the receiver has no data to send in the reverse direction, it sends an empty data segment with the ACK flag set to TRUE. In the second option, the receiver can wait until it needs to send a data segment in the reverse direction and piggyback the ACK on that segment. With this option, the receiver must maintain a timer so that if no data is sent in the reverse direction, the ACK will eventually be sent.

The receiver never sends NACKs. Even if the receiver determines a segment is in error and drops it, it does not send a NACK.

8.2.4 Round Trip Delay Estimate and Timer Implementation

The retransmission options that will be described in the next section rely on maintaining retransmission timers for the transmitted segments. In practice, however, there is no need to maintain an actual timer for a segment. The source just needs to keep track of the current time. We assume that whenever a segment is transmitted, it is placed at the end of the retransmission buffer and 'stamped' with the current time. Checking whether the timer for a segment has expired means that the current timeout value is added to the timestamp of the segment, and the result is compared to the current time. If the calculated time is greater than the current time, then the timer has expired. Resetting the timer refers to updating the timestamp to the current time.

In general, the source sets the timeout value to β R, where R is the current estimate of the round trip delay (RTD), and β is chosen to be greater than 1. The RTD estimate includes all propagation delays, queueing delays, and processing delays from the source to the destination and back to the source. The RTD for a segment is typically calculated by noting the time difference between the timestamp of a segment and the arrival time of the ACK for that segment. (There are many problems with such a calculation, as will be discussed in Section 8.4.) Based on this RTD sample, the source updates its estimate of the RTD as follows:

$$\text{New RTD Estimate} = (1-\alpha)\ \text{RTD Estimate} + \alpha\ \text{RTD Sample} \quad (8.1)$$

The factor α is chosen to be between 0 and 1. The larger α is, the heavier the new RTD sample is weighted.

The timeout value is then set for β times the RTD estimate. β is typically chosen to be 2, although some implementations of TCP allow β to be adjusted dynamically [Com91]. The larger β is, the less likely underestimates of the RTD will lead to unnecessary retransmissions.

8.2.5 Retransmission Policies

After a segment has been sent, it is placed at the end of a retransmission buffer. A segment is removed from the buffer when an ACK is received with an ACK number greater than the sequence number of the segment's last byte of data. There are three possible retransmission policies at the source. When a connection is established, any of the three retransmission policies can be chosen to be used with either the in-order or in-window acceptance policy at the receiver, but only certain combinations make sense.

In one retransmission policy, which we refer to as Method A, a timer is kept for the segment at the head of the retransmission buffer. If this segment has not been ACKed before the timer expires, the entire buffer is retransmitted and the timer restarted. This option is compatible with the in-order acceptance policy at the receiver. The combination of this retransmission policy and the in-order acceptance policy is effectively a Go Back N scheme.

In the second option, which we refer to as Method B, a timer is kept for the segment at the head of the buffer. If this segment has not been ACKed before the timer expires, this one segment is retransmitted and the timer is restarted i.e., the timestamp of this segment is updated to the current time. (We assume that the retransmitted segment remains at the head of the buffer although the protocol specification is ambiguous as to whether this is the case.) This scheme will be more fully explained in Section 8.3.2. Method B, combined with the in-window acceptance policy, is similar to a selective repeat scheme.

In the third option, which we refer to as Method C, a timer is kept for the segment at the head of the buffer. If this segment has not been ACKed before the timer expires, this one segment is retransmitted and then placed at the end of buffer (again, it is ambiguous at to whether this is how the protocol is actually implemented). The timer of the new segment at

the head of the buffer is not restarted, which necessitates that a timer backoff strategy such as Karn's Algorithm be used. This will be described more fully in Section 8.3.3.

8.2.5.1 Repackaging Option

Another option at the source involves whether the data in the retransmission buffer is repackaged into different sized segments when it is retransmitted. When the source sends a segment, it can only send as much data as the receiver allows (incoming segments indicate the maximum number of bytes that can be sent in the reverse direction). Thus, the source may be forced to send very small segments during times of congestion. When retransmissions are necessary, the source has the option of dividing up the data into different sized segments. Thus, a retransmission of a segment may actually mean that only part of the segment is being retransmitted, or that it is being retransmitted along with data from one of the adjacent segments in the buffer.

8.3 RETRANSMISSION POLICY ANALYSIS

8.3.1 Selective Repeat vs. Go Back N

The combination of retransmission Method A and the in-order acceptance policy is effectively a Go Back N (GBN) scheme. Retransmission Method B, combined with the in-window acceptance policy, is similar to a selective repeat (SR) system. In this section, we compare the efficiency of GBN and SR in a TCP/IP environment. Efficiency is defined as $1/\gamma$, where γ is the expected number of transmitted segments from source to destination per successfully accepted segment at the destination. Assume fragments are dropped independently with probability Q_R, and segments are comprised of N fragments. Assume the timeout value at the source is βR. If a timer expires and a retransmission is necessary, the source finishes transmitting the message it is currently sending, and then performs the retransmission. Thus, the source sends $\left\lceil \frac{\beta R+1}{N} \right\rceil$ segments in between the original transmission of a segment and the retransmission of the segment. Then, from equations 6.17 and 6.18:

$$\text{GBN efficiency} = \frac{1 - N\,Q_R}{1 + N\,Q_R \left\lceil \frac{\beta R+1}{N} \right\rceil} \qquad (8.2)$$

$$\text{SR efficiency} = 1 - N\,Q_R \qquad (8.3)$$

Assume that fragments are dropped independently due to random bit errors, burst errors, and congestion. As usual, let P_R be the probability of a random bit error, P_B be the probability of a fragment being hit by a burst error, and P_C be the probability a fragment is dropped due to congestion. Assume the size of a fragment is 576 bytes (this is the default size). Then Q_R equals $((576)(8)\ P_R + P_B + P_C)$. If P_R equals 10^{-6}, P_B equals 10^{-5}, and P_C equals 10^{-4}, as assumed in chapter five, then Q_R equals $5x10^{-3}$. Note that Q_R is dominated by the term due to random bit errors. (However, as pointed out in chapter five, on connections where reliability is important, it is likely that there are additional checks at other layers so that the bit error rate is lower than what we assumed.)

Equations 8.2 and 8.3 assume that retransmissions occur only when a segment has been dropped. However, in TCP/IP, unnecessary retransmissions are also likely to occur due to the inability of the source to accurately track the RTD. Unnecessary retransmissions result in lower efficiency for both GBN and SR schemes. For simplicity, however, we will ignore this.

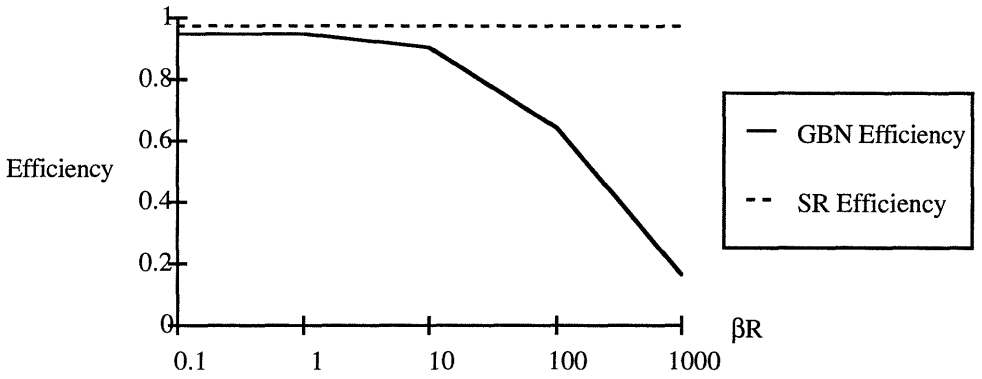
On Graph 8.1, we plot the efficiency of SR and GBN as βR varies, for Q_R equal to $5x10^{-3}$ and N equal to 5. There is not a significant difference between the curves until βR increases past 10. Given the size of the fragment (i.e., 576 bytes), the value of R depends on the speed of the connection and the length of the RTD. The maximum connection speed in TCP/IP was limited to 1.5 Mb/sec, although 45 Mb lines are being introduced. At 1.5 Mb/sec, with β equal to 2, a βR of 10 corresponds to an RTD of 15 msec. If the connection speed is only 56Kb/sec, then a βR of 10 corresponds to an RTD of 400 msec. The RTDs in TCP/IP span a very wide range. They are likely to fall within the range of 50 msec. to 10 sec. Thus, it depends on the parameters of the connection whether SR provides a significant benefit in terms of efficiency.

The drawback of SR is that the destination must maintain a resequencing buffer. Given that some of the computers in the TCP/IP environment are quite primitive, re-ordering segments at the destination may pose a significant processing burden. Also, in deriving the equation for the efficiency of an SR system, we assumed that the buffer at the destination never overflows. If the destination is unable to allocate a buffer size large enough to hold two or three RTDs worth of data, then the efficiency of the SR system will be less than that shown on the graph. Thus, the decision of whether to use GBN or SR depends not only on the parameters of the connections, but also on the capabilities of the receiver.

Note that the efficiency of both a GBN and an SR system is relatively unaffected by the fragment size, assuming the size of the segment is unchanged and assuming that βR is much larger than 1. This is because Q_R is approximately proportional to the size of the fragment, and N and R are inversely proportional to the fragment size. Thus, equations 8.2 and 8.3 are close to constant as the fragment size changes. If the segment size were to increase, then both efficiency curves would be shifted downward.

Graph 8.1 Efficiency of GBN and SR

(N =5 and Fragment Loss Rate = 5E-3)



8.3.2 Analysis of Retransmission Method B

The combination of retransmission method A and the in-order acceptance policy is a typical GBN scheme. (However, most GBN schemes are implemented on systems where data is expected to arrive in order; in TCP/IP segments can arrive out-of-sequence. A very old segment can confuse the GBN system i.e., the old segment may contain a sequence number that makes it appear to be a current segment.) We will not analyze retransmission method A further. In this section, we analyze the combination of retransmission method B and the in-window acceptance policy.

Let the unit of time be the time to transmit one fragment (assume fragments are fixed size), and let the source's estimate of the RTD be R time units. Assume the source is transmitting data at a steady rate, and assume segments are 1000 bytes long and comprised of N fragments. Thus, one segment is transmitted every N time units.

We will use the convention that a segment is timestamped and placed at the end of the retransmission buffer after it has been completely transmitted, and that its timer is set to expire βR time units later. Assume that segment 1000 is at the head of the retransmission buffer, and assume it has a timestamp of $\mathcal{T}$. Assume that an ACK for segment 1000 is not

received by the expiration time (i.e., $\mathcal{T}+ \beta R$) , and the segment is retransmitted. The timer for segment 1000 is now set to expire at time $\mathcal{T}+ N + 2\beta R$.

In this scheme, the source cannot possibly retransmit segment 2000 until segment 1000 is ACKed. This makes sense from the point of view of reducing the number of unnecessary retransmissions. ACKs only indicate the value of RECV_NEXT at the receiver. Thus, a segment cannot be ACKed until all previous segments have been ACKed. Thus, in the example above, if segment 1000 is really lost, then the source must wait an additional RTD after it is retransmitted before it could possibly receive an ACK for segment 1000 or for any segments after it.

For simplicity, assume that βR accurately represents the total delay from source to destination and back. Then, assuming the first transmission of segment 1000 is lost but the retransmission is successful, the ACK for segment 1000 arrives at time $\mathcal{T}+ N + 2\beta R$. Then, assuming segment 2000 has also been lost, segment 2000 will be retransmitted at this time. In general, if *i* consecutive segments are lost (but the retransmissions are not lost), then the $i+1^{st}$ segment will be retransmitted at time $\mathcal{T}+ iN + (i+1)\beta R$, under Method B. If the policy at the destination was to send ACKs whenever a segment was successfully received and to indicate in the ACK which segment it was that triggered the ACK, then the timers could be set such the $i+1^{st}$ segment would be retransmitted at time $\mathcal{T}+ iN + \beta R$ (i.e., the timers could be set to expire one round trip delay after the segment is transmitted). Thus, because of the lack of information in the ACK, Method B increases the expiration time of the $i+1^{st}$ segment by i βR time units.

We see that Method B slows down the retransmission process. In the case where the losses are due to burst errors, slowing down retransmissions would only be advantageous if the burst error were still affecting segments even after one RTD. If the losses are due to congestion, then there are two reasons why slowing down the retransmission process could be beneficial. First, it would give the congestion more time to dissipate. Second, consider the case where congestion causes the RTD to increase but does not cause the segments to be dropped. In this case, segment 1000 will be retransmitted since its timer will expire (the estimate of the RTD will be too small). However, segment 2000 will not be retransmitted immediately after segment 1000. This gives more time for the ACK of segment 2000 to arrive (i.e., it has until time $\mathcal{T}+ N + 2\beta R$ rather than $\mathcal{T}+ N + \beta R$). This will prevent the unnecessary retransmission of segment 2000. This possibly could prevent the 'retransmission avalanche' that often accompanies congestion.

8.3.3 Analysis of Retransmission Method C

In retransmission Method C, one timer is kept for the segment at the head of the retransmission buffer. If this timer expires, the segment is retransmitted and placed at the end of the retransmission buffer. Again refer to the example given in the previous section. Assume segment 1000 is lost, and then retransmitted at time $\mathcal{T} + \beta R$. The timer for segment 2000 (the new head of the buffer) will be set to expire at time $\mathcal{T} + N + \beta R$. As discussed above, an ACK for segment 2000 cannot be sent by the receiver until both segment 1000 and segment 2000 are received. Thus, the earliest an ACK for segment 2000 could be received is $\mathcal{T} + N + 2\beta R$ (assuming βR accurately represents the total RTD). Using this scheme, the timer for segment 2000 will expire, and segment 2000 will be retransmitted. In fact, one whole RTD worth of segments will be retransmitted. If Method C is used with the in-order acceptance policy, then the scheme is similar to GBN - whenever the segment at the head of the buffer is lost, one RTD worth of segments will be retransmitted. If Method C is used with the in-window acceptance policy, then this RTD worth of retransmissions may be unnecessary, unless there has been a burst loss.

This does not appear to be a very effective scheme. However, in practice, Method C is used with a timer backoff scheme. A commonly used scheme is Karn's Algorithm[Com91], [KaP87]. The source starts out by using its estimate of the RTD to determine the timeout value for the retransmission timers. However, whenever a segment is retransmitted, the source increases the timeout value as follows:

new_timeout = γ * timeout

Typically, γ is chosen to be 2. Thus, the timeout value is doubled each time there is a retransmission. In practice, the timeout value is not increased past some maximum value. Whenever the source successfully receives an ACK for a segment that has only been transmitted once, it reverts back to using its RTD estimate for the timeout value.

In general, if *i* consecutive segments are lost and retransmitted, the timer of the $i+1^{st}$ segment will expire at time $\mathcal{T} + iN + \gamma^i \beta R$. As shown in the previous section, under Method B, if *i* consecutive segments are lost, the timer of the $i+1^{st}$ segment will effectively expire at time $\mathcal{T} + iN + (i+1)\beta R$. Thus, under Karn's algorithm, the increase in timeout value is exponential. Under Method B, the effective increase in timeout value is linear.

Karn's Algorithm is more flexible than Method B. In some TCP implementations, the value of γ can be changed with each retransmission. Also, since Karn's algorithm is a

timeout policy rather than a retransmission policy, it can also be used with retransmission Method A.

As with Method B, the tradeoff in Karn's Algorithm is between excess delay and unnecessary retransmissions. Assume a segment is lost. If the next segment is also lost and needs to be retransmitted, these methods will slow down that retransmission. This may be advantageous for congestion losses, but not for losses due to random errors or burst errors (unless the burst error lasts for more than one RTD). If the next segment does not need to be retransmitted and the estimate of the RTD is equal to or greater than the true RTD, then the increase in timeout value essentially has no effect. If the next segment does not need to be retransmitted but the source's current estimate of the RTD is too small, then both Method B and Karn's Algorithm provide extra time for the ACK to arrive, so that unnecessary retransmissions can be avoided.

8.3.3.1 Timer Backoff and Congestion Avoidance

As shown above, under Karn's Algorithm, if i consecutive segments are lost and retransmitted, the timer of the $i+1^{st}$ segment will expire at time $\mathcal{T} + iN + \gamma^i\beta R$. Thus, the timeout value grows exponentially. After an ACK is successfully received for a segment that has been transmitted only once, the RTD is calculated, the source updates its estimate of the RTD, and the timeout value reverts back to β times the RTD estimate. Thus, the timeout value jumps back down at this point. (Note that the current version of TCP ties the value of β to the estimated variance; thus, the larger the increase in the RTD estimate, the larger the value of β.[COM91])

Retransmissions also affect the flow control mechanism in TCP. Segments traveling from the destination to the source indicate the size of the receive window at the destination. The source uses this to determine the size of its send window. However, whenever a timer expires and a segment is retransmitted, the source cuts the size of its send window in half. If the send window is smaller than the receive window, then whenever a segment is received successfully, the source increases the send window by the size of one segment. When the send window reaches half of its original size, the source only increases the window when an ACK is received for the segment at the end of the current send window. Thus, the cutback in the flow control window is exponential; this is similar to Karn's Algorithm, where the increase in timeout value is exponential. Under Karn's Algorithm, however, the timeout value may suddenly decrease. The increase of the flow control window is more gradual. However, the increase in the flow control window occurs

whenever an ACK is received. Karn's Algorithm decreases the timeout value only after an ACK is received for a segment that has never been retransmitted.

Let's examine the interaction between these two mechanisms. Refer to Figure 8.1. Assume the original timeout value is βR and the original size of the send window is 8 segments. Assume 4 segments can be sent in one RTD. Assume the first four segments are lost. At time $\mathcal{T} + \beta R$, the retransmission timer for segment #1 expires, and the segment is retransmitted. The increase in the timeout value to $2\beta R$ throttles the retransmission process (i.e., segment #2 isn't sent until $\mathcal{T} + 2\beta R+N$), and the decrease in the send window to 4 segments throttles the transmission of new segments (i.e., segment #5 cannot be sent after the retransmission of segment #1 since the window has been decreased to 4 segments).

After the retransmission of segment #2 at time $\mathcal{T} + 2\beta R+N$, the send window (which is now halved to 2 segments long) is smaller than the number of outstanding segments. Thus, despite the fact that the timer for segment #3 expires at time $\mathcal{T} + 4\beta R+2N$, the send window of size 2 segments prevents its retransmission. Note that, according to [Com91], the timeout value isn't increased unless the segment is actually retransmitted; thus, even though the timer for segment #3 expires, the timeout value is not changed at that time. Segment #1 is retransmitted again at time $\mathcal{T} + 5\beta R$; the send window is halved to 1 segment. Assume this retransmission is successful and the ACK for segment #1 is received. We assume the actual RTD for segment #1 is $4\beta R$. When the ACK arrives, the window is increased to 2 segments, so that segment #3 can be retransmitted (i.e., the send window now consists of segment #2 and segment #3). The timeout value of $8\beta R$ prevents segment #2 from being retransmitted again until time $10\beta R+N$.

The ACK of segment #2 also ACKs segment #3. We assume that the send window is increased by 2 segments when this ACK arrives. Finally, segment #5 can be sent. After segment #5 is successfully ACKed, the timeout value will be set to $\beta^* R^*$, where β^* and R^* take into account the RTD estimate for segment #5. (Note that α in equation 8.1 is often taken to be 1/8 [Com91], which does not weight the new RTD sample very heavily.) The send window will be 4 segments, so that several segments can be transmitted immediately after one another. If other sources begin dumping segments into the network, the congestion may start to build up again. The timeout value of $\beta^* R^*$ may not be high enough, leading to unnecessary retransmissions. This would trigger the timeout backoff

and window cutback mechanism again. A more gradual decrease of the timeout value might be more appropriate.

8.3.4 Problems With Resized Segments

TCP allows the source to repackage data into different sized segments when the data is retransmitted. This potentially could lead to confusion at the receiver, as shown in the following example. Assume the receiver is implementing an in-window acceptance policy, and assume the receive window extends from sequence number 100 to 200. Assume a segment (call it segment B) with sequence number 150 and length 50 arrives at the receiver. The whole segment falls within the receive window, so it will be accepted in its entirety by the receiver. Now assume that another segment (call it segment A) with sequence number 100 and length 75 arrives at the receiver. Thus, segments A and B overlap. It is not specified in the protocol what the acceptance policy is in this situation.

It is possible that this situation can arise from normal, error-free operation. For example, assume the initial segments were each 50 bytes long, and had sequence numbers 100 and 150. Assume the source does not receive an ACK for segment 100 before its retransmission timer expires. Now, at the time of retransmission, assume that the flow control mechanism allows the source to send a segment of length 75 bytes. The source might repackage the data in the retransmission buffer, and send segment 100 of length 75 bytes. If the original segment with sequence number 150 arrives at the receiver, followed by the retransmitted segment numbered 100, we arrive at the situation described above.

It is also possible that undetected errors could lead to this situation. Segment B could be correct, and segment A could have been affected by an undetected error event that resulted in it having the wrong length. Alternatively, segment A could be correct, and segment B could have an undetected error in its sequence number field. (A third option is that either segment A or B is a stray segment from an old connection that happens to have a sequence number that overlaps with the current connection. We will assume this scenario is less likely than the other two scenarios.)

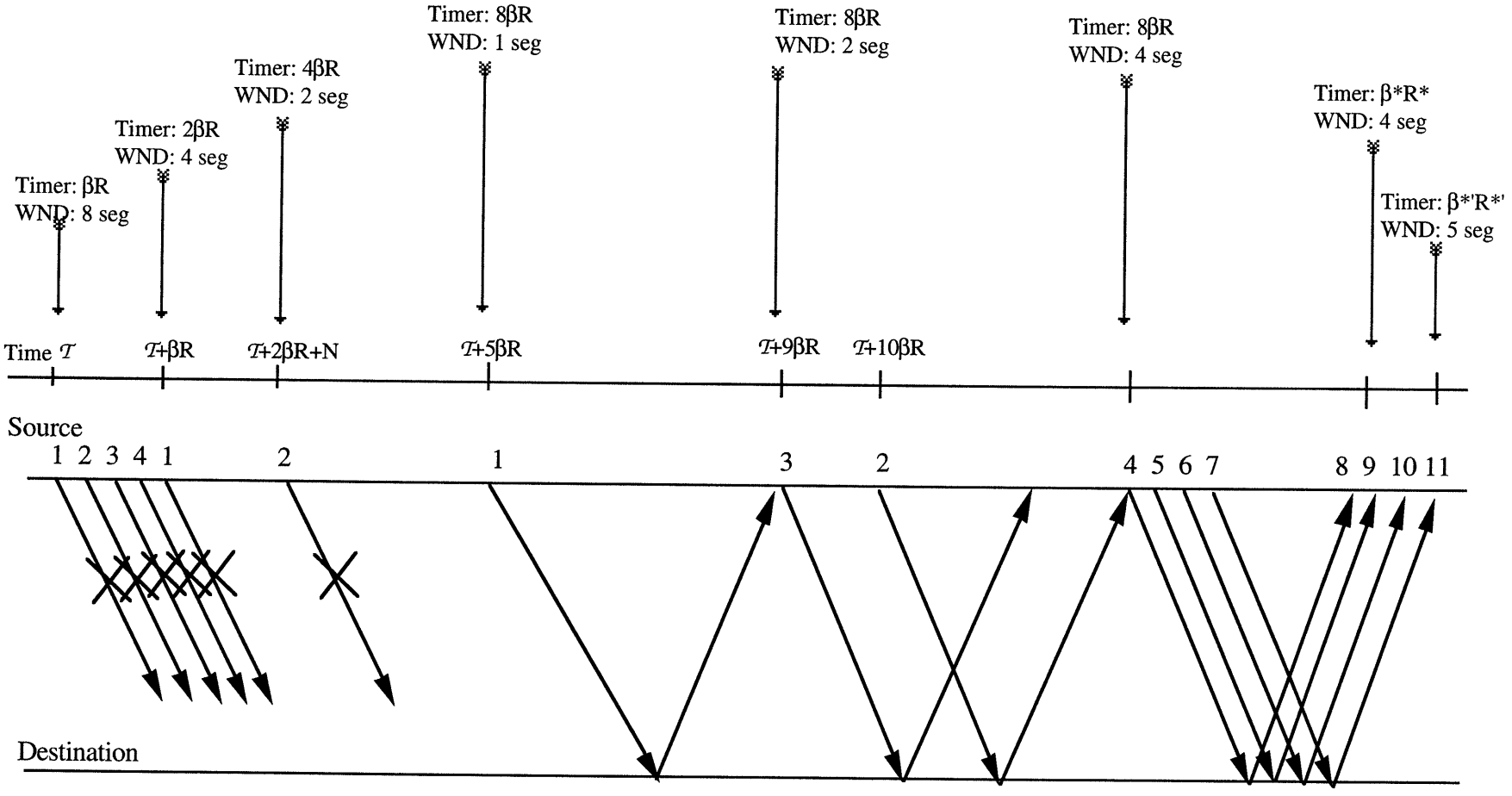


Figure 8.1 The timer backoff scheme throttles the retransmissions; the window flow control scheme throttles the new transmissions. Once transmissions are successful, the send window begins to grow. After the transmission of segment #5 is successful, the timeout value drops to β*R*. After the transmission of segment #7 is successful, the send window increases to 5 segments.

In this situation the receiver has five different options:

i) Accept segment A and drop segment B

ii) Accept segment B and drop segment A

iii) Drop segments A and B

iv) Accept all of segment B and the first portion of segment A

v) Accept all of segment A and the final portion of segment B

Since it is possible that undetected errors have led to this situation, the safest alternative is to drop both segments A and B. However, as described above, this situation could arise from normal, error-free operation. Thus, dropping both segments could unnecessarily decrease the throughput of the system. The best policy to choose depends on the expected rate of undetected errors.

First consider the possibility that segment A has an undetected error that results in it having the wrong length. The most likely scenario is that the segment header length field of segment A has an undetected error in it. The destination subtracts the header length from the overall segment length (which is passed up by IP) to determine the size of the data portion of the segment. The segment header can be anywhere from 20 bytes to 60 bytes long; thus, errors in the header length field can account for differences in data length up to 40 bytes. The header length field is a four bit field that is only protected by the segment checksum. From the analysis in Section 5.4.4.8, the expected annual frequency of undetected header length errors per line is 5.

Next consider the scenario where segment B has an error in its sequence number field. The sequence number field is a 32 bit field in the segment header that is also only protected by the segment checksum. The expected annual frequency of undetected sequence number errors per line is about 40.

Thus, the expected frequency of these undetected errors is not negligible. Assume that the receiver's policy is to treat segments A and B as if they are error-free (i.e., the receiver assumes the sequence number overlap has occurred due to resegmentation of the data). Then the expected frequency with which this policy will result in the receiver accepting incorrect data is approximately 45 times per year per line. However, in Section 5.4.2, we showed that the expected annual frequency of undetected bit errors in the data portion of a segment is 5×10^3 per line. (This assumed that the random bit error rate was 10^{-6}. It is

likely that additional error detection mechanisms will be used within the various sublayers so that the effective error rate at the TCP/IP layer would be smaller.) Thus, bit errors in the data is still more likely to result in the destination accepting incorrect data. Given this fact, the reliability of the system is not significantly affected if the receiver assumes that segments A and B overlap due to proper operation.

Note that the segment checksum is the only means of detecting errors in the segment header length and in the segment sequence number, as well as the only means of detecting errors in the data. As discussed in Section 5.4.2, a checksum is not a powerful error detection mechanism. If a segment CRC were used, the expected frequency of these undetected errors would be reduced by several orders of magnitude.

8.4 ESTIMATION OF ROUND TRIP DELAY

The performance of a timer-based retransmission scheme heavily depends on the ability of the source to estimate the RTD. We discuss this further in Section 8.4.1. Unfortunately, the TCP/IP environment and the TCP protocol make it very difficult for the source to accurately estimate the RTD. We discuss the reasons for this, as well as suggest improvements to the scheme, in Section 8.4.2.

8.4.1 Performance

Throughout this section, let the unit of time be the time to transmit one fragment. Let R_T be the actual RTD for a given segment, including propagation, queueing, and processing delays. Let R_E be the source's current estimate of the RTD. Consider the situation where the source finishes transmitting a segment at time τ, and sets the segment's retransmission timer to expire at time $\tau + \beta R_E$. If $\beta R_E < R_T$, then the timer will expire too soon. Even if the segment is received successfully, the ACK will not be received by the source before the expiration time. Thus, an unnecessary retransmission can occur. Note that this is most detrimental for retransmission Method A, which is essentially a GBN scheme. If the timer of the segment at the head of the buffer expires, the whole buffer is retransmitted. Thus, underestimating the RTD can lead to a burst of unnecessary retransmissions. (Of course, it is possible that if the ACK is received before all the segments are retransmitted, then the source need not retransmit these remaining segments. However, this option is not explicitly included as part of the protocol description in [DOD85].) In Methods B and C (which are similar to SR schemes), the one segment will be unnecessarily retransmitted, but the timeout value increases after this to prevent (or at least reduce the probability of) more unnecessary retransmissions.

If $\beta R_E > R_T$, then the source gives adequate time for the ACK to be received. However, if the segment is lost, then $\beta R_E - R_T$ represents the excess retransmission delay (refer to Section 6.3). If one of the SR schemes is implemented, then an overestimate of the RTD leads to greater delay in delivering segments under error conditions to the higher layer at the destination, and may lead to the resequencing buffer filling up. In the GBN scheme, an overestimate of the RTD also leads to a delay in delivering segments at the destination; in addition, it results in greater inefficiency in delivering fragments. Recall that the formula for efficiency in the GBN system is:

$$\text{GBN efficiency} = \frac{1 - N\,Q_R}{1 + N\,Q_R \left\lceil \frac{\beta R + 1}{N} \right\rceil}$$

The βR in the denominator represents how many fragments are sent in between the transmission of a segment and the expiration of the segment's timer. Thus, this is actually βR_E. The larger βR_E is, the lower the efficiency of the GBN system. Thus, overestimates of the RTD degrade the efficiency of a GBN system.

We conclude that the performance of the GBN system (i.e., Method A) is more sensitive to how well the source can estimate the RTD. In any of the retransmission schemes, underestimates of the RTD can degrade the performance of the system when segments have been received successfully; overestimates are harmful if segments need to be retransmitted. Thus, the expected loss statistics of the system should influence the policy the source uses to estimate the RTD.

8.4.2 Problems Estimating the RTD

8.4.2.1 Queueing Delays

The TCP/IP environment makes it very difficult to estimate the RTD. One problem is that the queueing delays along a given path can be highly variable. Since the data rate of the links may be low, congestion at a node may result in fragments being buffered for a long time before being serviced. Thus, queueing delays can be appreciable relative to the overall RTD.

8.4.2.2 Multiple Data Paths

A second problem arises from the fact that fragments are routed individually at the IP layer. Thus, two consecutive segments may travel over different paths. The RTD can potentially be very different over the two data paths.

8.4.2.3 Updating RTD Estimate

In general, when the source receives an ACK for a segment, it calculates the RTD for that segment. There are problems with calculating the RTD for a segment as discussed below, but for now, assume it is accurate. As mentioned in Section 8.2.4, the source uses the RTD sample to update its estimate of the RTD:

$$\text{New RTD Estimate} = (1-\alpha)\ \text{RTD Estimate} + \alpha\ \text{RTD Sample}$$

The factor α is chosen to be between 0 and 1. The larger α is, the heavier the new RTD sample is weighted. Assume the most recent RTD sample is very different from the old estimate of the RTD. The source has no way of knowing whether this is an anomaly or whether the RTD will remain at this new value for a long time. Thus, it is difficult for the source to know how to choose α. In general, the source sets the timeout value of the retransmissions timers to β times the current estimate of the RTD, for $\beta > 1$, to provide some leeway in its estimate of the RTD.

8.4.2.4 Inaccurate RTD Samples

The RTD sample itself may not be very accurate due to the way ACKs are sent in TCP. Whenever a segment is received at the destination, an ACK is sent indicating the value of RECV_NEXT. Thus, if the in-window acceptance policy is used, the ACK may not actually be acknowledging the segment that triggered the ACK. Consider the following example. Assume all segments have length 50. Assume segments with sequence number 100, 150, 200, 250, and 300 are transmitted. Assume the transmission of segment 300 is completed at time $\mathcal{T}_{300}$. Assume segment 100 travels over one path where the delay is D, and assume the other four segments travel over another path where the delay is D'. Assume that $D' << D$, so that the four latter segments arrive before segment 100. When segments 150 through 300 arrive, the destination will send an ACK with RECV_NEXT set to 100. After the arrival of segment 100, the destination will send an ACK with RECV_NEXT equal to 350. Assume the source receives this ACK at time $\mathcal{T}_A$. The RTD sample for segment 300 will be calculated to be $\mathcal{T}_A - \mathcal{T}_{300}$. However, the RTD sample corresponding to segment 300 should really be much smaller than this. Thus, the RTD samples calculated by the source may not be very accurate.

One possible improvement that could be implemented is that the ACK should also indicate which segment number triggered the ACK. (This was also suggested in [Zha86].) In the example above, this would allow the source to obtain five accurate samples of the RTD.

8.4.2.5 Ambiguous ACKs

Another problem arises when a segment has been transmitted multiple times. When an ACK is received for the segment, the source does not know which copy of the segment is being ACKed. This is referred to in [Com91] as the 'ambiguous ACK' problem. First, assume that the source's policy is to associate the ACK with the earliest copy of the segment that has been sent. Assume copies 1 and 2 of a segment have been transmitted, and assume an ACK is received for the segment. If the original transmission were lost and the ACK was actually sent in response to the retransmission, the source's policy would result in an overestimate of the RTD. As described in [COM91], if every segment were lost at least once, this policy would result in the RTD estimate growing without bound.

Next, assume that the source's policy is to associate the ACK with the latest copy of the segment that has been transmitted. Assume the RTD increases so that the ACK for the first copy of a segment is not received before the timer expires, forcing there to be a retransmission. When the ACK for the first copy does arrive, it will be interpreted as being the ACK of the second copy. Thus, the source's estimate of the RTD for this segment will be too small. In fact, this problem will persist; the source will continue to underestimate the RTD so that each segment will end up being retransmitted.

We see that neither policy will work for all cases. The solution adopted in TCP is to not calculate RTD samples for ACKs that are received for segments that have been transmitted more than once. Of course, this in itself is not an acceptable solution. If the RTD increases, and segments are retransmitted because their timers expire, the source would not generate RTD samples for these segments. Thus, it would never increase its estimate of the RTD to the proper level, resulting in more unnecessary retransmissions. To take care of this scenario, the source must increase its effective timeout value after a retransmission, so that eventually an ACK will be received before the timer expires. After this ACK is received, the source can increase its estimate of the RTD (assuming the ACK is for a segment that was never retransmitted). Note that increasing the effective timeout value is precisely what occurs in both Karn's Algorithm and retransmission Method B. (This also implies that Karn's algorithm, or a similar type of timeout backoff strategy, must be used with retransmission Methods A and C to handle this scenario of increasing RTD.)

Another possible solution to the ambiguous ACK problem would be to associate an ID number with each transmission of a segment. Thus, the first transmission of a segment would contain one ID, the retransmission of the segment would contain a different ID. This ID number would be carried in the segment, so that the destination can include it in the ACK. The source would then be able to associate the ACK with the correct transmission, so that the ACK would not be ambiguous. There is already a method in place in TCP/IP to implement this. When TCP requests that the IP layer transmit a segment, it has the option of passing down a 16 bit ID number. This ID number is then used as the ID field in each of the fragments comprising the segment. (If TCP does not pass an ID number down, IP generates one using a counter.) This ID field could be passed up to the TCP layer at the destination so that it could be included in the ACK. The drawback is that this method adds 16 bits to the length of the ACK field, for a total of 48 bits. The ACK field is part of the segment header, which currently ranges in size from 20 bytes to 60 bytes. (If the suggestion presented in Section 8.4.2.4 is used, where the sequence number of the segment triggering the ACK is also included in the ACK, the length of the ACK field would be 96 bits.) An alternative would be to have a 'copy number' field included in the segment header and in the ACK field. This would probably only need to be about 4 bits long.

8.5 POLL-BASED SCHEME IN TCP/IP

Poll-based retransmission schemes were analyzed in detail in chapter six. The analysis showed that one of the main advantages of poll-based schemes is that if data travels in order, then it is possible to implement the scheme such that there are no unnecessary retransmissions. This was demonstrated in the context of ATM in chapter seven. The major disadvantage is that poll-based schemes may result in large excess retransmission delay when bursts of data are lost. In TCP/IP systems, bandwidth efficiency is of greater concern than low delay. Thus, the ability to eliminate unnecessary retransmissions would be desirable. Of course, data does not travel in sequence in TCP/IP systems, nor is the delivery time of a segment bounded. Thus, a polling scheme can not eliminate unnecessary retransmissions. Nevertheless, as we show below, such a scheme may perform favorably compared to the current TCP timer-based scheme.

Assume that the TCP layer at the source sends out polls periodically. As discussed in chapter six, the poll could actually be piggybacked on a data segment that is being transmitted. The poll contains a poll sequence number and the value of SEQ, which we

define to be one higher than the highest data byte thus far sent by the source. Note that polls are numbered independently of segments. As in the proposed ATM scheme, the source would maintain the variable PSEQ to indicate the most recent poll that has been sent. Whenever a segment is transmitted or retransmitted, it is stored in the retransmission buffer and 'stamped' with the current value of PSEQ. As described in detail in chapter seven, this allows the source to determine whether a data segment was sent before or after a poll. Since segments may be resized or only a portions of a segment may be accepted at the destination, the PSEQ stamp should be associated with all bytes of the segment. If only a portion of a segment is retransmitted, then only that portion has its PSEQ stamp updated.

A status message is sent by the TCP layer at the destination in response to a poll. The status message contains the poll sequence number of the poll that triggered it, the current value of RECV_NEXT (let $RECV\_NEXT_S$ represent this value), and the status of all segments from RECV_NEXT through SEQ, where SEQ is indicated in the poll.

When the source receives a status message, it removes from the retransmission buffer all data through $RECV\_NEXT_S$-1. If GBN is being implemented, it checks whether the PSEQ stamp of the byte numbered $RECV\_NEXT_S$ is less than the poll sequence number indicated in the status message. If so, it begins to retransmit from sequence number $RECV\_NEXT_S$ onward. If SR is being implemented, then any NACKed segment that has a PSEQ stamp less than the poll sequence number contained in the status message is retransmitted.

Essentially the retransmission criterion is: if the poll arrives at the destination before data that was sent prior to the poll, then retransmit the data. In a system where data travels in order, such as ATM, this guarantees that there are no unnecessary retransmissions. Below, we analyze how this scheme would work in a TCP/IP environment, where data does not travel in sequence. (Note that since unnecessary retransmissions can occur, the send window must be half the size it was in the ATM scheme; refer to section 7.3.6.1).

8.5.1 Poll-Based Schemes vs. Timer-Based Schemes in TCP/IP

First, let's review the problems enumerated in Section 8.4 in timer-based schemes. One major problem was estimating the RTD based on the ACKs that were received. There are many situations where it is ambiguous which segment triggered the ACK, thus resulting in inaccurate RTD calculations. In poll-based schemes, having an estimate of the RTD is not crucial. It is only needed to get an idea of how often to send polls. The source can

estimate the RTD as the time between the transmission of a poll and the reception of the status message corresponding to that poll. Due to the poll sequence number, there should be no ambiguity as to which poll a status messages corresponds.

A second problem in timer-based schemes is that when queueing delays increase, the RTD increases, so that the retransmission timer of a segment may expire prematurely. In a poll-based scheme, the poll is delayed in the queue as well as the data. Thus, an increase in the queueing delay would not cause the poll to unnecessarily NACK a segment, assuming the data and poll travel over the same route.

The RTD may also fluctuate due to segments being transmitted over different routes, although this is not a common occurrence. Consider the case where a poll is transmitted over a path that takes longer than the path taken by segments that were sent before it. If any of these segments need to be retransmitted, then the extra delay along the poll's route would increase the excess retransmission delay, but it would not trigger unnecessary retransmissions. If a poll is transmitted over a path that is shorter than the path taken by segments transmitted before it, then unnecessary retransmissions may occur. This is opposite to what happens in timer-based schemes: increases in the RTD tend to result in unnecessary retransmissions; decreases in RTD result in increases in the excess retransmission delay (since the source's estimate of the RTD is too high).

As described in Section 6.2.1.5, the excess retransmission delay may greatly increase when burst losses occur in poll-based schemes, since the polls also might be lost. However, this increase in delay happens to some degree in the current implementation of TCP/IP regardless of the retransmission scheme, due to the fact that ACKs only contain the value of RECV_NEXT. Assume an SR scheme is implemented, and assume segments 100 and 200 are transmitted and lost. Segment 200 cannot possibly be ACKed until segment 100 is ACKed. Thus, after segment 100 is retransmitted, the source must wait one whole RTD before retransmitting segment 200, to give the ACK of segment 100 a chance to arrive. Thus, there is a minimum excess retransmission delay of about one RTD.

We conclude that poll-based schemes cure some of the problems inherent in a timer-based scheme: there is no need to accurately estimate the RTD, and increases in queueing delay do not result in unnecessary retransmissions. Neither type of scheme deals well with the scenario where the RTD fluctuates due to segments traveling over different paths. During burst losses, a poll-based scheme results in an increase in the excess delay, as does the

TCP timer-based schemes. Overall, the performance of a poll-based scheme should be superior to that of a timer-based scheme.

8.6 CONCLUSIONS

In this chapter, we reviewed the operation of the retransmission scheme included in the TCP protocol. We focused on the more unusual facets of the protocol, such as the option at the source to resize the data segments when they are retransmitted. We analyzed the difficulties involved with estimating the RTD in a TCP/IP environment, and suggested methods to improve the scheme. Finally, we showed that a poll-based scheme would eliminate some of the problems resulting from the varying nature of the RTD.

CHAPTER 9

CONCLUSIONS

9.1 MAJOR CONTRIBUTIONS

We have demonstrated that it is possible to approach the design of error protection protocols in a systematic fashion. In chapter three, we presented a five step algorithm for designing effective and efficient error detection schemes. We brought out the following points: certain errors can be prevented if dealt with at one layer rather than another layer; certain errors can be fixed if detected at one layer as opposed to another layer; certain errors are much harder to detect than others; given that a message is the unit of retransmission, it is often more effective and efficient to include error detection mechanisms on a per-message basis, rather than on a per-packet basis.

In chapter four, we applied these techniques to the correction oriented services of ATM. We developed an error protection scheme that is more effective and efficient than the AAL Type 3/4 scheme proposed by the ATM Standards Committee. The chief difference is that our proposed scheme includes a per-frame CRC, while the AAL Type 3/4 proposal includes a per-cell CRC. The per-cell CRC is weak in detecting burst errors; also its inability to detect many other errors necessitated the addition of several other error detection fields. Our proposal is very similar to the AAL Type 5 proposal.

In chapter five, we showed that the major weakness in the error detection scheme of TCP/IP is the use of a checksum as a redundancy check on the segment data, rather than a CRC. This results in a high expected rate of undetected bit errors in the data. All other types of errors have an expected undetected error rate that is several orders of magnitude lower. Thus, the weak data redundancy check significantly impairs the overall performance of the TCP/IP error detection scheme. This reinforces the notion that the effectiveness of an error detection scheme is measured by the error scenario that results in

the most undetected errors. It also points out that if reliability is an important issue, then there needs to be more error protection than what TCP/IP provides.

In addition to examining error protection protocols, we also analyzed data retransmission schemes. To gain insight into the performance of various retransmission schemes, we developed a delay criterion that we referred to as relative excess delay. We used this criterion as one means of evaluating poll-based retransmission schemes. Our comprehensive analysis of poll-based schemes showed that the major advantage of such schemes is that they can be implemented such that there are no unnecessary retransmissions, assuming that data travels in sequence. The chief drawback of poll-based schemes is that they lead to an increase in the excess retransmission delay when bursts of messages are lost. We showed that timer-based schemes can potentially lead to low delay. However, the performance of timer-based schemes heavily relies on how well the source can estimate the round trip delay (RTD) from source to destination and back. Underestimates of the RTD can lead to many unnecessary retransmissions. The higher the speed of the data links, the smaller the queueing delays should be relative to the propagation delay; this should enable the source to better estimate the RTD. Based on our analysis, we showed that poll-based schemes are best suited for systems where bandwidth efficiency is more important than low delay. Timer-based systems are more appropriate for high speed networks than they are for low speed networks.

In chapter seven, we analyzed the poll-based retransmission scheme that has been proposed for ATM. We formally proved the correctness of the protocol, given certain reasonable conditions. We also showed that a timer-based scheme is probably more appropriate than a poll-based scheme in the high speed, integrated service environment of ATM.

In chapter eight, we discussed some of the interesting features of the TCP timer-based retransmission scheme. We analyzed why such a scheme performs poorly in the TCP/IP environment. Our analysis showed that a poll-based scheme would likely perform better.

One topic which we did not explore is the stabilization properties of retransmission schemes. We briefly outline this topic in the next section. This is an area for future research.

9.2 STABILIZATION PROPERTIES OF RETRANSMISSION SCHEMES

We use the term self stabilizing to describe a retransmission scheme that eventually returns to proper operation regardless of the encountered error (unless the severity of the error causes the connection to be terminated). Proper operation refers to the source, the destination, and the path between them being in a valid state, where validity depends on the specific scheme.

There are many different failures that can cause a retransmission scheme to perform incorrectly. We can divide them into three broad categories. First, there are undetected errors, which occur when the error detection scheme fails. Second, there are node failures, which can lead to the inability to deliver some data or can cause the loss of state information. Third, there can be failures of the underlying network system. For example, we could design a scheme around the assumption that data arrives at the receiver in the same order in which it is sent; a network failure could result in this not being the case.

Whenever any of these failures causes the destination to deliver incorrect or out-of-sequence data or causes data transmission to halt due to the protocol being deadlocked, we can say the connection is in an error phase. We might say a scheme is self stabilizing if it is capable of eventually terminating the error phase, and, if no other failures were to occur, the connection would not enter another error phase. However, as the examples below show, this definition of self stabilization may not be acceptable.

9.2.1 Examples

Consider the following example. Assume packets are numbered modulo M. Assume that the RTD becomes abnormally large so that the source sends a burst of over M packets within one RTD. Assume packet 1 arrives at the destination, but the next M packets are lost. All packets after this are received correctly. Assume the ACK of packet (X+M) mod M is interpreted by the source as ACKing packet X. Thus, the missing M packets will never be retransmitted, and the destination will not realize that they are missing. It is unclear whether we should consider such behavior as self-synchronizing. Although M packets are lost and the loss is not detected by the receiver, all packets after this are delivered correctly in sequence (assuming no other errors occur). However, the situation persists that packet X+M is interpreted at the receiver as being packet X. From an absolute numbering point of view, every packet is incorrect and we may not want to describe the protocol as having stabilized.

The following example points out further ambiguities in defining self stabilization. Assume we are dealing with a scheme that is designed not to produce any unnecessary retransmissions. Assume a failure occurs that results in a persistent problem that produces unnecessary retransmissions. Assume it does not interfere with data being delivered correctly and in sequence at the destination. However, the scheme has not resumed normal operation. If our goal is to produce schemes that return to proper operation, and not just correct operation from the point of view of the end user, then we need to broaden our interpretation of self stabilization.

In the third example, assume a source sends packets 1 and 2, but the destination only receives packet 2. Assume the destination has an in-window acceptance policy so that packet 2 is accepted. An ACK of packet 2 is transmitted back to the source. Assume an undetected error occurs in the ACK so that it appears to the source to be an ACK of packet 1. The source releases packet 1 from its retransmission buffer. Assume the size of the receive window is 2, so that the destination cannot accept any more packets until packet 1 arrives. However, the source has already released packet 1 so it is unable to retransmit it. The connection will be deadlocked. We need to provide the retransmission scheme a means of breaking the deadlock.

In the fourth example, assume the destination node fails in such a way that the data in its receive buffer is lost and status information, such as the next packet number it is expecting, is arbitrarily set to some value. Assume the destination node is unaware that this has happened so does not try to reinitialize the connection. We need to provide the retransmission scheme with a means of resynchronizing its status information based on the incoming data. For a detailed study of this, refer to [Var92].

9.2.2 Future Study

The scenarios listed above do not exhaust the possible problems that may occur. One goal of our future research is to develop more structured representations of the various possible errored states. A second goal is to develop a more precise definition of self-stabilization. Using this, hopefully we will be able to generalize the properties a retransmission scheme must possess in order to be self stabilizing.

REFERENCES

[AnP86] Anagnostou, M. and Protonotarios, E., "Performance Analysis of the Selective Repeat ARQ Protocol", *IEEE Transactions on Communications*, 34:127-135, 1986.

[BeG92] Bertsekas, D. and Gallager, R., *Data Networks*, Second Edition, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1992.

[Bel90] Bellcore, "Preliminary Report on Broadband ISDN Transfer Protocols," SR-NWT-001763, Issue 1, December 1990.

[BKK88] Bux, W., Kermani, P., and Kleinoeder, W., "Performance of an improved data link control protocol," *Proceedings ICCC'88*, Tel Aviv, Oct. 30-Nov. 3, 1988, pp. 251-257.

[CCI81] CCITT VIIth Plenary Assembly, "Data Communications Networks: Recommendations X.1-X.29, Volume VIII, Fascicle VIII.2, Geneva, 1981.

[CCI90] CCITT Recommendation I.432, "B-ISDN User-Network Interface - Physical Layer Specification", Matsuyama, December, 1990.

[CCI92a] CCITT Study Group XVIII/8-5, "AAL Type 5, Draft Recommendation Text For Section 6 of I.363," Copenhagen, October, 1992.

[CCI92b] CCITT Study Group XI Temporary Document XI/2-24, "Results of September 1992 XI/2 discussions on Q.SAAL," Geneva, September, 1992.

[CCI92c] CCITT Recommendation I.363, "B-ISDN ATM Adaptation Layer (AAL) Specification, Geneva, 1992.

[CCI93] CCITT Study Group XI Document DT/11/3-28, "Service specific connection oriented (SSCOP) specification," May 23, 1993.

[Com91] Comer, D., *Internetworking with TCP/IP Volume I*, Second Edition, Prentice-Hall, New Jersey, 1991.

[Con93] "Internet Domain Survey - January 1993," *ConneXions*, 7:44-45, March, 1993.

[DCS91] Dravida, S., Cheng, Y., and Saksena, V.R., "Error performance of broadband ISDN networks," submitted to *IEEE Transactions on Communications*, 1991.

[DDK90] Doeringer, W., Dykeman, D., Kaiserwerth, M., Meister, B., Rudin, H., and Williamson, R., "A survey of light-weight transport protocols for high-speed networks," *IEEE Transactions on Communications*, 38:2025-2039, 1990.

[DOD85] DDN Protocol Handbook, Volume 1, DOD Military Standard Protocols, December, 1985.

[DrD91] Dravida, S. and Damodaram, R., "Error detection and correction options for data services in B-ISDN," *IEEE Journal on Selected Areas in Communication*, 9:1484-1495, 1991.

[Dra90] Dravida, S., "Error control aspects of high speed networks," Int. Teletraffic Cong. Symp. Broadband Technologies, Oct. 9-11, 1990.

[FKL89] Fujiwara, T., Kasami, T., and Lin, S., "Error detecting capabilities of the shortened Hamming codes adopted for error detection in IEEE Standard 802.3," *IEEE Transactions on Communications*, 37:986-989, 1989.

[Fle82] Fletcher, J., "An arithmetic checksum for serial transmission," *IEEE Transactions on Communications*, 30: 247-251, 1982.

[Gal68] Gallager, R., *Information Theory and Reliable Communication*, John Wiley & Sons, New York, 1968.

[Hes88] Hess, R., "TCP and Error Detection and Correction", Bachelors Thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, MA, 1988.

[KaP87] Karn, P. and Partridge, C., "Improving round-trip time estimates in reliable transport protocols", *Proceedings of ACM SIGCOMM '87*, Stowe, VT, August 11-13, 1987, pp. 2-7.

[Kon80] Konheim, A., "A queueing analysis of two ARQ Protocols," *IEEE Transactions on Communications*, 28: 1004-1014, 1980.

[LaM91] LaPorta, T, and Schwartz, M., "Architectures, Features, and Implementation of High-Speed Transport Protocols," *IEEE Network Magazine*, May, 1991.

[Lyo91] Lyon, T., "Simple and Efficient Adaptation Layer (SEAL)," *T1S1 Contribution No. T1S1.5/91-292*, August, 1991.

[MeP82] Meijer, A. and Peeters, P., *Computer Network Architectures*, Computer Science Press, 1982.

[NRS90] Netravali, A., Roome, W., and Sabnani, K., "Design and implementation of a high-speed transport protocol," *IEEE Transactions on Communications*, 38:2010-2024, 1990.

[Pet61] Peterson, W., *Error Correcting Codes*, MIT Press and John Wiley & Sons, Inc, 1961.

[Pro89] Protocol Engines, Inc., "XTP Protocol Definition," Revision 3.4, 1989.

[RoS89] Rosberg, Z., and Shacham, N., "Resequencing delay and buffer occupancy under the selective-repeat ARQ," *IEEE Transactions on Information Theory*, 35:166-173, 1989.

[Sch87] Schwartz, M., *Telecommunication Networks*, Addison-Wesley Publishing Co., Reading, MA, 1987.

[Tan88] Tanenbaum, A., *Computer Networks*, Second Edition, Prentice-Hall, Inc., Englewood Cliffs, 1988.

[T1S91] "A proposed convergence protocol to serve class C and D users," *T1S1 Contribution No. T1S1.5/91-188,* May, 1991.

[T1S92] "Description of service specific connection oriented protocol (SSCOP)," *T1S1 Contribution No. T1S1.5/92-237* , June, 1992.

[Var92] Varghese, G., *Self-Stabilization by Local Checking and Correction*, PhD Thesis, MIT, October, 1992.

[VaV89] Vanstone, S., and van Oorschoot, P., *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic Publishers, Boston, 1989.

[Zha86] Zhang, L., "Why TCP Timers Don't Work Well," *ACM SIGCOMM '86 Symposium*, Stowe, VT, Aug. 5-7, 1986, pp. 397-405.

BIOGRAPHY

The author was born in Middletown, NJ, on September 24, 1963. In 1981, she was named a US Presidential Scholar and the NJ Female Scholar-Athlete. In 1985, she was graduated summa cum laude from Princeton University with a BSE in EECS. At Princeton, she was a National Merit Scholar and a participant in the Bell Labs Engineering Scholarship Program, and was awarded the G. David Forney Award for excellence in electrical engineering. After Princeton, she worked three years at a software development company. While at MIT, she has been an NSF Fellow and a DARPA Fellow.