On the Use of Erasure Codes in Unreliable Data Networks

by

Salil Parikh

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2001

Author……..…….……..…
Department of Electrical Engineering and Computer Science
May 21, 2001

Certified by……..……
Professor Vincent W. S. Chan
Joan and Irwin Jacobs Professor of
Electrical Engineering and Computer Science &
Aeronautics and Astronautics
Director, Laboratory for Information and Decision Systems.
Thesis Supervisor

Accepted by…….……...
Arthur C. Smith
Chairman, Department Committee on Graduate Students

On the Use of Erasure Codes in Unreliable Data Networks

by

Salil Parikh

Submitted to the Department of Electrical Engineering and Computer Science on May 21, 2001, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science.

Abstract

Modern data networks are approaching the state where a large number of independent and heterogeneous paths are available between a source node and destination node. In this work, we explore the case where each path has an independent level of reliability characterized by a probability of path failure. Instead of simply repeating the message across all the paths, we use the path diversity to achieve reliable transmission of messages by using a coding technique known as an erasure correcting code. We develop a model of the network and present an analysis of the system that invokes the Central Limit Theorem to approximate the total number of bits received from all the paths. We then optimize the number of bits to send over each path in order to maximize the probability of receiving a sufficient number of bits at the destination to reconstruct the message using the erasure correcting code. Three cases are investigated: when the paths are very reliable, when the paths are very unreliable, and when the paths have a probability of failure within an interval surrounding 0.5. We present an overview of the mechanics of an erasure coding process applicable to packet-based transactions. Finally, as avenues for further research, we discuss several applications of erasure coding in networks that have only a single path between source and destination: for latency reduction in interactive web sessions; as a transport layer for critical messaging; and an application layer protocol for high-bandwidth networks.

Thesis Supervisor: Professor Vincent W. S. Chan
Title: Joan and Irwin Jacobs Professor of
Electrical Engineering and Computer Science &
Aeronautics and Astronautics
Director, Laboratory for Information and Decision Systems.

Acknowledgements

It is impossible to adequately express my deepest gratitude to my advisor, Professor Vincent Chan, for his tremendous support, unwavering encouragement, and infinite patience. Without him, I would not have even tried.

I am extraordinarily fortunate to be surrounded by the minds and souls that I have met at MIT: Mr. Doug Marquis, Professor Muriel Medard, Dr. John Moores, and Ms. Sue Patterson. They have been my finest mentors, teachers, and friends, generously sharing their strength, wisdom, and unbridled humor.

Contents

1 Introduction

The pursuit of reliable transmission of data through a network to ensure correct reception at the destination is not a new problem. Numerous techniques have been developed, refined, implemented, and widely deployed to arrive at a reliable data delivery service.

Forward error correction (FEC) is a popular means of overcoming physical channel impairments, such as signal attenuation over unamplified long-haul links (or satellite links), or burst errors due to inherent, environmental, or interference causes. FEC is used even in modern low-loss high-reliability fiber-optic networks to stretch signal budgets in 2.5 Gbps (OC-48) to 40 Gbps (OC-768) long-haul and undersea transmission systems.

Adding redundancy to data for the sake of providing error correction is often considered too expensive in terms of overhead. Instead, classes of protocols have been developed which, in essence, have the sender wait for an acknowledgement of packet receipt before sending additional data packets. These include schemes such as Positive Acknowledgement with Retransmission (PAR) or Automatic Repeat Request (ARQ), with the widely deployed Transmission Control Protocol (TCP) perhaps being the most popular realization [Tan96]. These protocols typically use a framing system that uses simple and conventional error detection methods to identify corrupted packets, but rely on retransmission of the entire packet instead of recovering from bit errors as is done FEC systems, thus reducing overall overhead and improving efficiency, at the cost of adding retransmission delays. Modern ARQ protocols such as TCP integrate ARQ with flow-control and congestion-control enhancements [Stev94].

To overcome the problem of link outages, routing protocols have been extensively studied, culminating in variants of Link-State or Distance-Vector classes of protocols to reroute traffic around broken links or recalculate paths in a network undergoing changes to its topology [BG87]. While widely deployed, especially in the Internet, these routing protocols can have long convergence times (on the order of minutes), and a message may suffer long end-to-end delays over a network undergoing rapid, recurrent, or pathological topology reconfiguration events – as is the case when the network is undergoing intense attack.

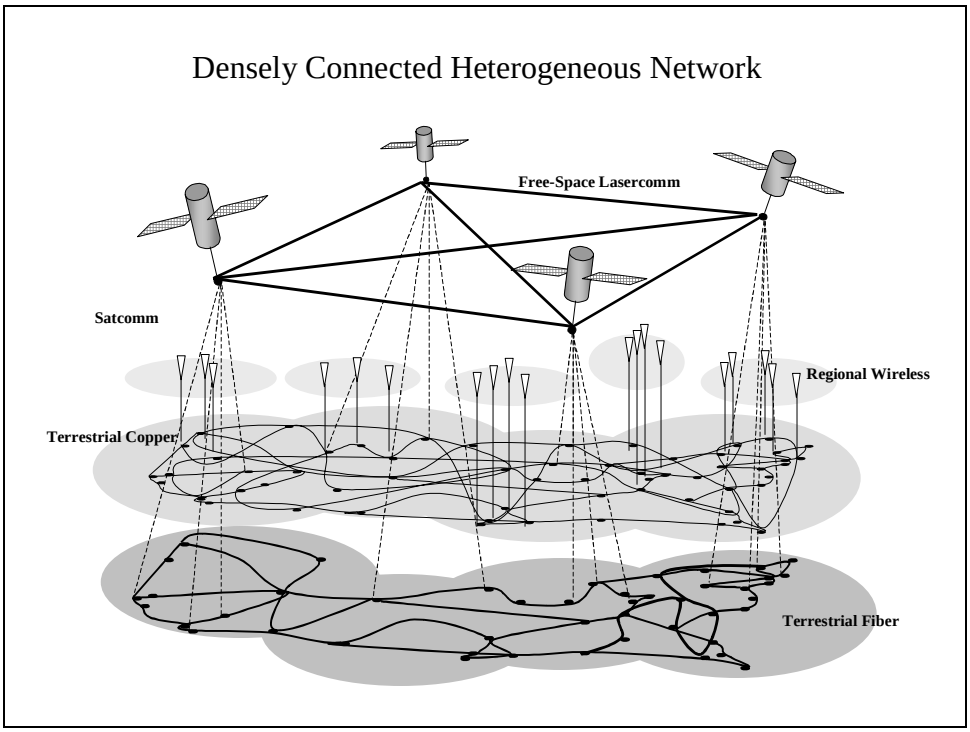

Figure 1-1

Consider the modern global-reach packet-data communication network depicted in Figure 1-1. The overall network is an interconnection of multiple heterogeneous networks, each made of different media types (such as fiber optic cables, wireless connections, satellite links, and copper cables), each can belong to different administrative domains, each can use differing network management and fault-recovery systems, and each can have varying levels of security and protection against attack. (When considering military networks, we can add battlefield radio and "ad-hoc" networks to the list of media modalities [Chan01].) In this thesis, we will consider only links that are independent of each other in terms of failure. These independent links are interconnected by packet routers, switching relays, or time division multiplexers and demultiplexers. These links and networks have multiple modes of failure, ranging from inherent physical characteristics, physical damage ("backhoe fade"), configuration errors, software failures, congestion (due to oversubscription or anomalous traffic transients), and malicious attack (in the form of physical disruption or "denial of service" attacks which have become common in recent times). The source and destination nodes usually have multiple independent, and possibly heterogeneous, interfaces into the network, and have multiple paths through the network between each other. Because of the many sources of vulnerabilities, we must consider individual segments of such a network to be inherently unreliable [Chan97].

The problem we consider in this work is how to reliably, quickly, and efficiently send a *critical* message from source to destination in such an unreliable network. We emphasize the word critical to imply delay-sensitivity and limited opportunity for retransmission. Such a scenario is encountered, for example, in defense networks, where building a ubiquitous and robust network that is under a single administrative, management, and security domain (a "stove-pipe" system) is an expensive and impractical proposition [Chan97]. Furthermore, many transmissions can be classified as critical to the success of an operation with severe time delivery requirements. Examples are early warning messages, intel updates, and air tasking orders.

In particular, we explore the use of the available path diversity in the network to deliver critical (and usually short length) delay-sensitive messages. In a highly connected network topology, there are often multiple independent paths between the source and destination. In critical cases, diversity can be a network design criterion and be designed into the topology when the network is deployed. This path diversity can be used to improve the reliability of message transfer. For short messages on the order of only a few packets, one obvious way to proceed is to broadcast the packets on every path through the network, but a single packet can also be broken into tiny fragments and each fragment sent via embedding in free bytes in other IP packets. However, for longer messages, repeating the same message over multiple paths results in a very inefficient use of network resources, and can further exacerbate network failures. Instead of simply repeating the message, one approach is to introduce some redundancy into the message, and send a subset of the message over each of the paths. At the receiver, the added redundancy will compensate for any lost portions of the message through the decoding process.

In principal, we can use block error correcting codes to introduce the redundancy [Lin83]. These codes can correct for varying numbers of bit errors in a message. However, for data networks, the error handling is usually such that an entire portion (a frame or packet) is discarded when errors are detected, instead of attempting to correct the corrupted bits. The corrupted packet is not delivered to the end application; instead, upon detecting an error, the entire packet is silently discarded in a protocol like TCP. For example, in an Ethernet, a frame that contains corrupted bits will fail the cyclic redundancy check (CRC), and will be discarded by the network adapter before it is seen by higher-layer network protocol stacks--bad packets are discarded and treated as packet losses. Thus, a packet that arrives corrupted is indistinguishable from a packet that never arrived at all, as far as the higher layer protocols are concerned. In the scenario of a defense network, some of the available paths may be under denial-of-service attacks, and the packets in those paths can be irretrievably lost or altered.

For this loss model, an *erasure* correcting code is a more suitable choice than a general error correcting code, as well as easier to deal with, since the exact position of the error (rather, the missing data) is known. An erasure code takes source data consisting of (K) bits, and adds redundant bits to form an encoded set of (N) bits, which are then transmitted. The receiver can then reconstruct the message from any (K) of the transmitted bits. The packet-level analogy is depicted in Figure 1-2 [Riz97].

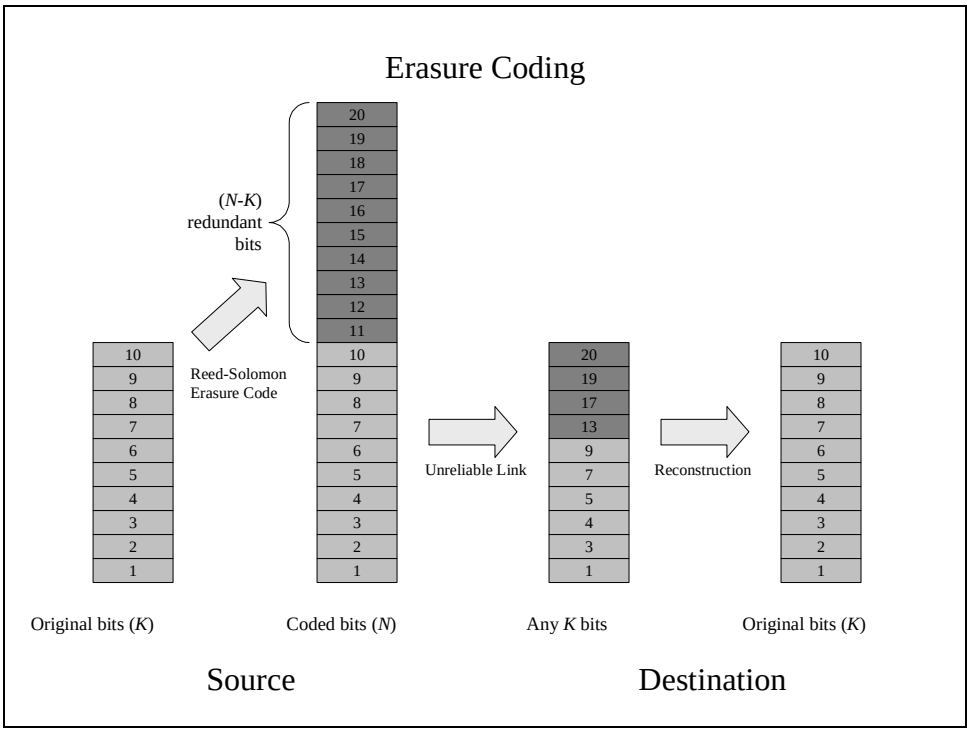

Figure 1-2

Consider the abstracted network in Figure 1-3. Each path can be characterized by an independent probability of failure (p_i) which is a composite of all the factors that affect the overall reliability of the path. There are (L) independent paths between source and destination, and each path is either successful or unsuccessful for the entire duration of the message transmission session. There are (K) source bits to be transmitted. These (K) bits are coded into (N) bits in such a way that the entire message can be reconstructed as long as any (K) coded bits are received at the destination.

The problem we examine in this work is to determine how many coded bits (N_i) to send over each path given the a-priori probability of failure of each path (p_i), in order to maximize the probability of receiving enough bits $M \geq K$ to reconstruct the message. We model the problem such that if a path fails, all bits sent on that link are lost.

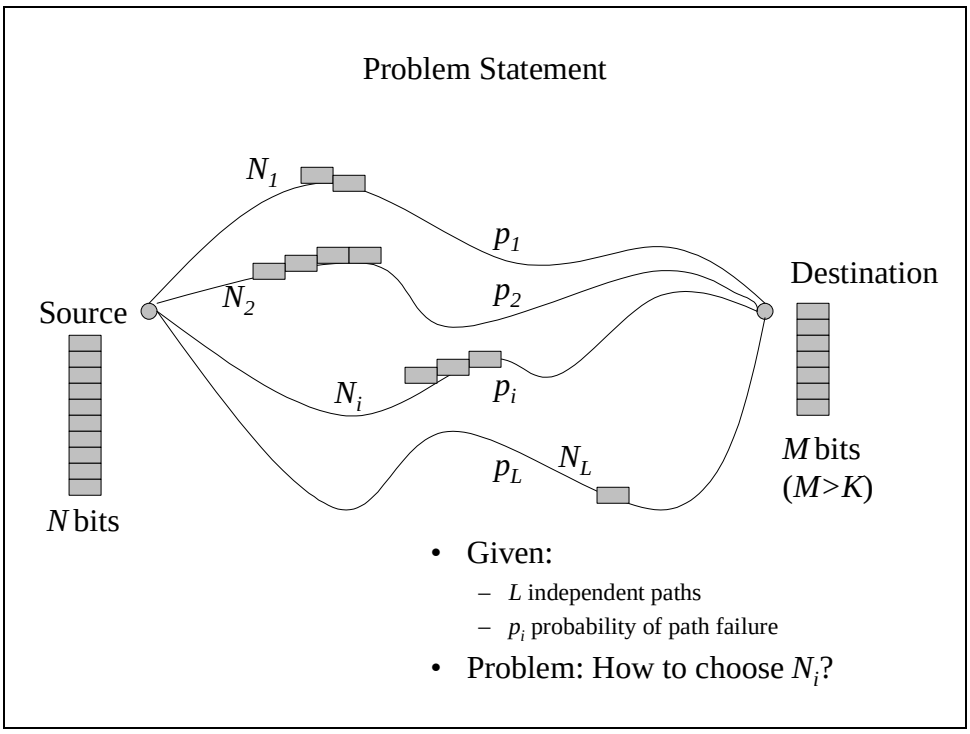

Figure 1-3

The issue of actually finding independent paths in a network is not trivial. For example, it is not sufficient to examine an IP *traceroute* probe, which reveals the identities of intermediate routers along a path [Gibb2K]. Even though two paths may not have any IP routers in common, they could share the same physical link via a shared ATM cell switch because ATM circuits (permanent virtual circuits or switched virtual circuits) can share the same SONET paths. For example, a link may be a fiber-optic point-to-point SONET link that uses ATM to provide virtual links between multiple pairs of routers. The routers have no knowledge that they are sharing the same link. The architecture of the current Internet has this property. In recent years, with the widespread deployment of WDM systems, it is common for private unshared links to share the same physical links for portions of an end-to-end path, with each link being carried on different wavelengths in parallel within the same fiber. At the wide-area network (WAN) level of the network, there is significant consolidation by the use of WDM, and furthermore with the practice

of pulling dozens of individual fibers in fiber conduits. However, in the metropolitan and local areas (including optical distribution networks and passive optical networks), fiber networks are not so collapsed, and we are observing dense connectivity that can afford multiple independent paths. With the addition of interconnected heterogeneous network links, our model of a densely connected high-bandwidth mesh becomes more plausible [Chan01]. Further increasing connection density is the recent commercialization and deployment of short-haul open-air point-to-point laser links that have 155 Mbps and higher capacities, even higher capacity orbital laser communication systems currently in development, and satellite access networks that usually have multiple satellites in view of the users and multiple crosslinks and downlinks.

In most IP networks, routing protocols usually ensure that only one path is used between a source and destination (however, there are often asymmetries in round-trip paths). If an intermediate link fails, the route is recalculated, and delivery of packets resumes. Control of the routing of a message in an IP network is usually out of reach to the end users, but system administrators have sufficient control to establish routes manually. For example, in ATM networks, permanent virtual circuits (PVC's) can be manually configured to produce independent paths through a network. To address the case of the IP network where the user cannot influence the routes directly, we can establish relay nodes to artificially force packets to travel along certain paths. Doing this requires detailed topology and configuration information on the underlying network, and secure relay nodes in strategic locations, but it can in principal be done. Messages are sent to intermediate nodes in the network, which may further bounce the message off of other relay nodes to force the message to traverse a designated path in the network, in order to overcome the network's end-to-end determination of route. This strategy can be improved if the relay host is itself a router between multiple networks. In the case of a relay host that is an end-station, packets may have to traverse a set of links twice as they reflect off a relay host, once to arrive at the relay, and once again as it egresses the relay back to the multiply-connected router to which the relay is attached. In the near future, new protocols such as MPLS (Multi-Protocol Label Switching) and MPλS (Multi-Protocol Lambda Switching) can be used to establish diverse paths from edge-router to edge-router [Chan01].

The problem of using a diversity of paths between source and destination has been investigated by [Rab89] and [Max93]. Our work builds upon the work of [Chan97] in particular, and extends the thinking of [Byer98] and [Byer99]. Rabin's work on Information Dispersal algorithms is similar in spirit and motivation to our work, but we detail a different analysis of the situation. Maxemchuck spreads data across several paths through a network to decrease the fraction of the

capacity used on one path, thus allowing each path to support more users. He defines "redundant dispersity routing" in which a linear combination of bits is used to create additional messages to place on the paths in such a way so that the original message can be reconstructed from a subset of the messages sent on the paths. We use a related coding scheme in this work, to add redundancy to a message that has been partitioned for placement on the multiple paths. Byer's work on "digital fountains" uses an erasure coding scheme called "Tornado Codes" to enable a user to download a large file faster from multiple sources (not necessarily over different paths) without having to establish individual file transfer sessions with each server.

The organization for the remaining chapters is as follows. Chapter 2 presents a coding technique suitable for packet data networks. Chapter 3 presents analytical findings and simulation results about the diversity-path transmission technique in general. Finally, chapter 4 discusses additional environments and applications in which such techniques might be useful, and serves to introduce potential avenues for future research and implementation.

2 Packet-Level Erasure Coding

In the preceding chapter, we applied the motivation and procedures to message portions organized as bits. Most data networks, however, work with data in units of frames or packets. For example, Ethernet uses a cyclic redundancy check (CRC) to detect errors in a frame; corrupted frames are discarded by intermediate routers or by the end station's media access controller (MAC), and are never seen by higher-layer protocol stacks. Sometimes, an IP packet is fragmented into smaller packets by an intermediate router to allow transmission on a link with a lower maximum transmission unit (MTU) size. Since IP does not track or retransmit fragments, a lost fragment results in the entire IP packet being discarded by the end-station. Similarly, a corrupted ATM cell, UDP packet, or TCP segment will be discarded with no notification to upper-layer protocols (except perhaps to network management counters). The point is that packet-oriented data networks are well described as an erasure channel.

In this chapter, we present one example of an erasure coding procedure suitable for packets. The procedure, known as Reed-Solomon Erasure Coding, can be mechanically cast as a linear algebra problem, which is what we do here to avoid delving into the details of the deeper theory behind the code. The following is a practical distillation of the procedures discussed in [Mac90, Riz97]. One should consult [Lin83] for a more complete and theoretical treatment. This technique has been used successfully in research implementations [Non98], and very similar methods are used in RAID systems to generate the "checksum" words on redundant disks [Pla99].

For simplicity, first assume that each "packet" is one byte. We will see soon how to extend the procedure to multiple-byte packets. To construct the coded bytes Y, we apply a generator matrix G to a vector of K source bytes S:

$$Y = G \times S .$$

The matrix multiplication must be carried out using finite field arithmetic. Figure 3-1 indicates the dimensions of the matrices and vectors.

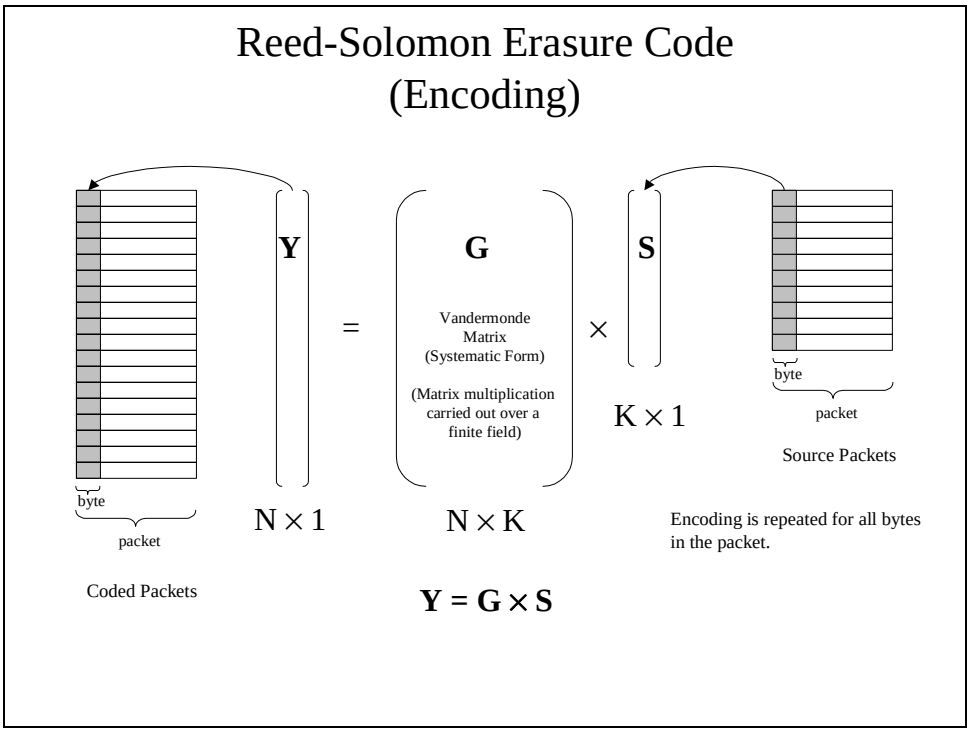

Figure 2-1

At the destination, we take any distinct K of the received bytes, and form the received vector Y'. The decoding matrix G' is composed of the rows of G that correspond to the received bytes in Y'. (See Figure 3-2). To find the original source bytes, we solve for S in:

$$Y' = G' \times S$$

This requires us to invert G':

$$S = (G')^{-1} \times Y'.$$

The solution can be found by standard Gauss-Jordan techniques, but with the reminder that all arithmetic operations must be executed within a finite field. (One cannot simply feed the matrices into most standard matrix solving routines.)

To produce the coded (redundant) packets, we apply the same matrix to successive bytes of the packets, repeating the encoding procedure on every byte in the packet. Corresponding bytes across the packets are fed through the generator matrix to produce encoded bytes of the redundant packet. That is, the second byte of each packet is taken as a vector and multiplied by the

generator matrix to result in a vector of the second coded bytes of the redundant packets. Since the same code is applied to all the bytes in the packet, arbitrarily long packets can be supported.

The generator matrix G is organized to produce a "systematic code". The top portion of G is the identity matrix (of rank K), and the lower portion is a Vandermonde matrix V that has the structure to be discussed next.

$$G = \begin{bmatrix} I \\ V \end{bmatrix}$$

Thus, the first K coded bytes of Y are simply the original K source bytes, and the next $N-K$ bytes are the redundant bytes.

The matrix V is composed of elements of the finite extension Galois Field GF(2^8). The elements[1] of V are $v_{i,j} = x_i^j$, where x_i is an element of the finite-field. Details on the construction and arithmetic of the field are presented in below. Since we are working with 8-bit bytes, we choose to operate over GF(2^8). Thus each row of V can have up to 2^8-1 elements.

A consequence of the Vandermonde structure in G is that any subset of rows from G that forms a square matrix (that is, any K rows) is guaranteed to be linearly independent, and therefore invertible [Riz97, Pla99]. Thus the inversion of G' is guaranteed.

[1] Note that we index our rows and columns starting with zero.

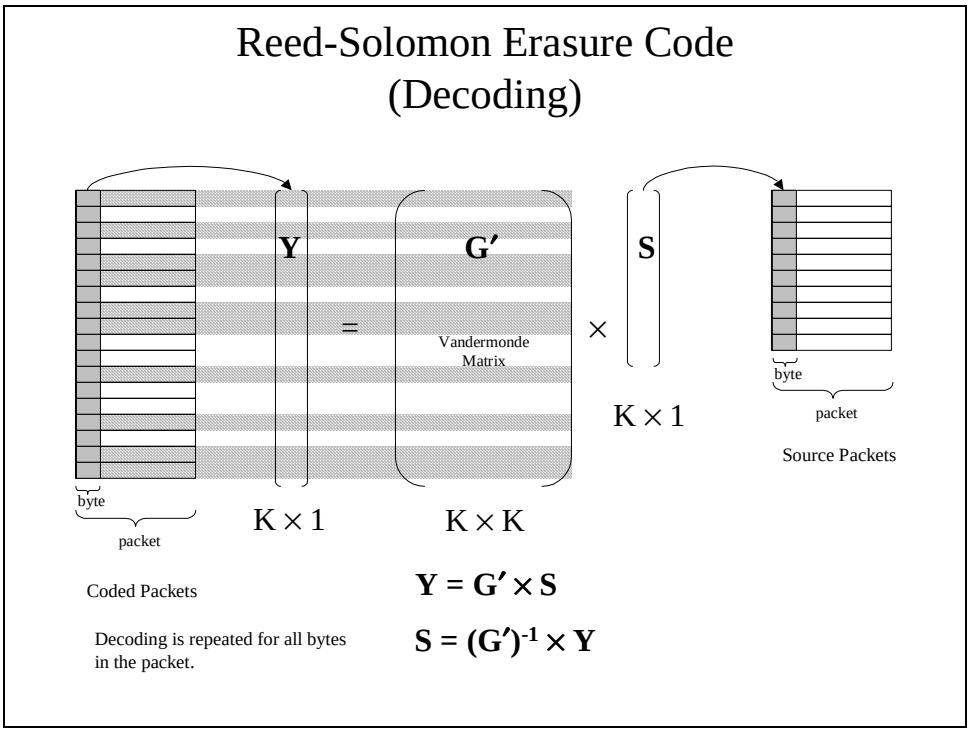

Figure 2-2

2.1 Finite Field Arithmetic

To construct the elements of the finite field GF(2^8) we use a primitive polynomial[2] of degree 8 whose coefficients are in GF(2): $x^8 + x^4 + x^3 + x^2 + 1$. Following the procedure detailed in [Lin83] we enumerate all the elements of the field by successive exponentiation of the primitive element $\alpha = 2$. Thus $x_i = \alpha^i$, and the Vandermonde matrix V component of G becomes:

$$V = \begin{bmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \cdots & \alpha^k \\ \alpha^0 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2k} \\ \alpha^0 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3k} \\ \vdots & \ddots & & & \vdots \\ \alpha^0 & \alpha^k & \alpha^{2k} & \cdots & \alpha^{k^2} \end{bmatrix}$$

[2] A polynomial $p(X)$ over $GF(2)$ of degree m is said to be *irreducible* over $GF(2)$ if $p(X)$ is not divisible by any polynomial over $GF(2)$ of degree less than m but greater than zero. An irreducible polynomial $p(X)$ of degree m is said to be *primitive* if the smallest positive integer n for which $p(X)$ divides $X^n + 1$ is $n = 2^m - 1$. Among their interesting and useful properties, primitive polynomials can be used to methodically construct the elements of Galois Fields $GF(2^m)$. Primitive polynomials are tabulated in [Lin83].

In the process of constructing the field elements, we construct tables of exponents and logarithms. The exponent table maps the exponent i to the corresponding field element; the logarithm table maps the field element to the corresponding exponent of the primitive element. These tables are used to perform multiplication (or division) of field elements by adding (or subtracting) their logarithms, and then converting back to the field element.

For illustration and reference, Appendix A includes a procedure to generate the field elements and the accompanying exponent and logarithm tables. Also included are procedures that provide multiplication and division operations for the field elements.

- `gfgen()` generates the field elements from a primitive polynomial, and also computes tables of exponents (gfexp[]), logarithms (gflog[]), and multiplicative inverses (gfinv[]) for use in field computations. The function `gftest()` is provided to conveniently view the tables.

- `gfmult()` implements finite-field multiplication, and must be used instead of regular decimal multiplication

- decimal addition should be replaced by bit-level `XOR`.

We also provide the simple function `RseEncode()` that implements the finite-field matrix multiplication that encodes packets made of 8-bit bytes, in order to illustrate how multiple bytes in a packet are coded. Because we repeat the code for every byte in the packet, packets of any length can be used, so long as all packets are of the same length. See [Riz97] for an efficient implementation of the decoder, which uses ordinary Gauss-Jordan Elimination to invert G'. [NumRec92] provides a space-efficient Gauss-Jordan Elimination routine that can be used if modifications are made to replace addition and multiplication operations with their appropriate finite-field counterparts.

In our work, we choose to use GF(2^8) because we are working with 8-bit symbols. If the situation was such that 16-bit symbols were the natural unit (as may be the case in storage systems, for example) then a larger field can be used. However, the large number of elements makes using table-lookups for the arithmetic inefficient. A larger field can also be used if more than 2^8-1 packets needed to be erasure coded, or if more than 100% redundancy is needed.

3 Optimization and Simulation

Our goal in this chapter is to derive a method of optimizing the number of bits[3] (N_i) transmitted on each network path to maximize the number of received bits (M), given the probability of failure of each independent path (p_i). We simulate the system to verify the optimization strategies.

As discussed in the previous chapter, we model the system such that if a path fails, then all bits transmitted on that path are lost. We begin the analysis by calculating parameters for a Gaussian approximation. The average number of bits received on each path at the destination is $M_i = N_i(1-p_i)$, thus the average of the total number of received bits (M) is

$$\begin{aligned}\mu &= \mathrm{E}[M] \\ &= \sum_{i=1}^{L} N_i(1-p_i) \\ &= \sum_{i=1}^{L} N_i q_i\end{aligned}$$

where $q_i = (1-p_i)$ is the probability of a path not failing (or the probability of a working path). The variance of the number of bits received from each path is:

$$\begin{aligned}\mathrm{VAR}[M_i] &= (N_i q_i)^2 p_i + (N_i - N_i q_i)^2 q_i \\ &= N_i^2 q_i^2 (1-q_i) + N_i^2 (1-q_i)^2 q_i \\ &= N_i^2 (q_i^2 - q_i^3 + q_i - 2q_i^2 + q_i^3) \\ &= N_i^2 (q_i - q_i^2) \\ &= N_i^2 q_i (1-q_i) \\ &= N_i^2 p_i q\end{aligned}$$

Next, by applying the rule below for variance of a sum of independent random variables, we find the variance of the total number of bits received at the destination:

$$\mathrm{VAR}(\sum_{i=1}^{L} X_i) = \sum_{i=1}^{L} \mathrm{VAR}(X_i)$$

[3] For generality, we say "bits", but the analysis applies to symbols, frames, or packets.

$$\sigma^2 = \mathrm{VAR}[M] = \sum_{i=1}^{L} N_i^2 p_i q_i \ .$$

Because we model the paths as independent, the number of bits received from each path is independent of that received from every other path. Thus, if the number of paths is large, we can consider the total number of received bits (M) to be the sum of many independent random variables. By application of the Central Limit Theorem (CLT), we can approximate M by a Gaussian random variable, and the probability of receiving fewer than m bits is:

$$\Pr[M < m] = \int_{-\infty}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(m-\mu)^2}{2\sigma^2}} \, \mathrm{d}m \ .$$

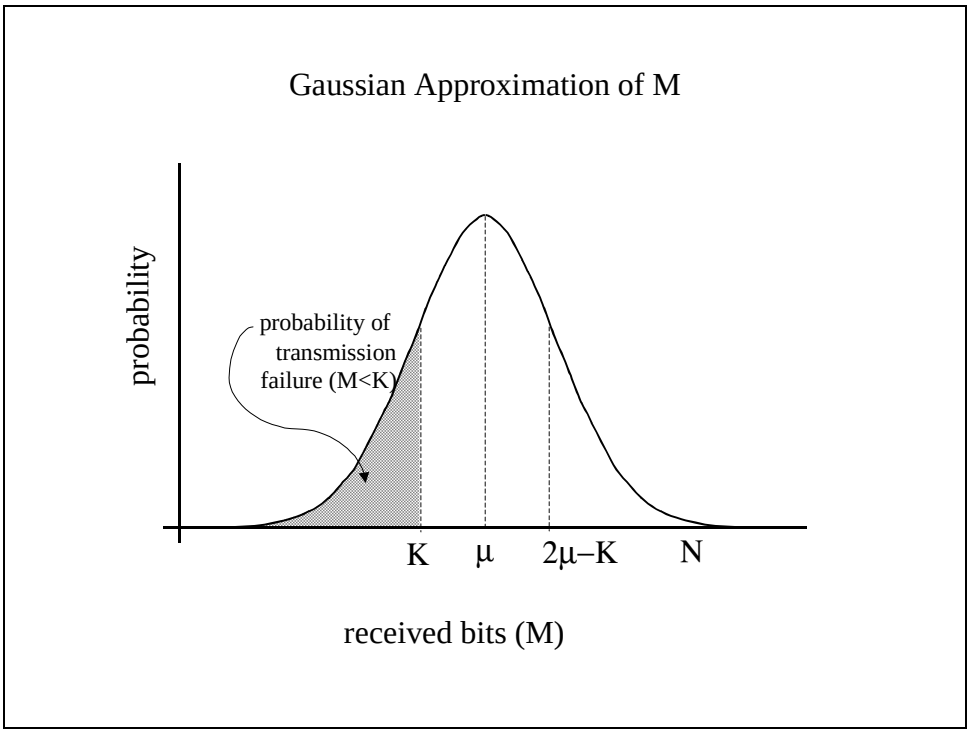

Figure 3-1

Since we need at least K bits to reconstruct the message, the probability of not receiving a sufficient number of bits for reconstruction is $\Pr[M < K]$. Since the Gaussian distribution is symmetric (see Figure 3-1), observe that $\Pr[M < K] = \Pr[M > 2\mu - K]$. Define $Z = (M - \mu)/\sigma$, which is a zero-mean unit-variance Gaussian distributed random variable. We have,

$$\begin{aligned}\Pr[M < K] &= \Pr[M > 2\mu - K] \\ &= \Pr\left[\frac{M-\mu}{\sigma} > \frac{2\mu - K - \mu}{\sigma}\right] \\ &= \Pr\left[Z > \frac{\mu - K}{\sigma}\right] \\ &= Q\left[\frac{\mu - K}{\sigma}\right]\end{aligned}$$

where the Q-function is the standard function [Lee94, Star94] defined as

$$Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{\frac{-\alpha^2}{2}} \mathrm{d}\alpha .$$

The Q-function is bounded by $(1/2)e^{-x^2/2}$ [Lee94] (see Figure 3-2[4]). Thus,

$$\Pr[M < K] = Q\left[\frac{\mu - K}{\sigma}\right] < \frac{1}{2} e^{\frac{-(\mu-K)^2}{2\sigma^2}} .$$

[4] Generated by the following MATLAB script. Note that we need to convert the MATLAB erfc function into the Q-function:

```
% Plotting the Q-function and a bound to it.
x = linspace(0,4,100);
y1 = erfc(x/sqrt(2))/2;
y2 = exp( - x.^2 /2 )/2;
y3 = (y2-y1);

semilogy(x,y1,'k-',x,y2,'k:',x,y3,'k--')
legend('Q','bound ','diff')
text(1.0,0.45,'bound=(1/2)exp(-x^2/2)')
text(2.8,0.001,'Q(x)')
text(0.25,0.05,'diff=bound-Q')
title('Q-function and bound')
```

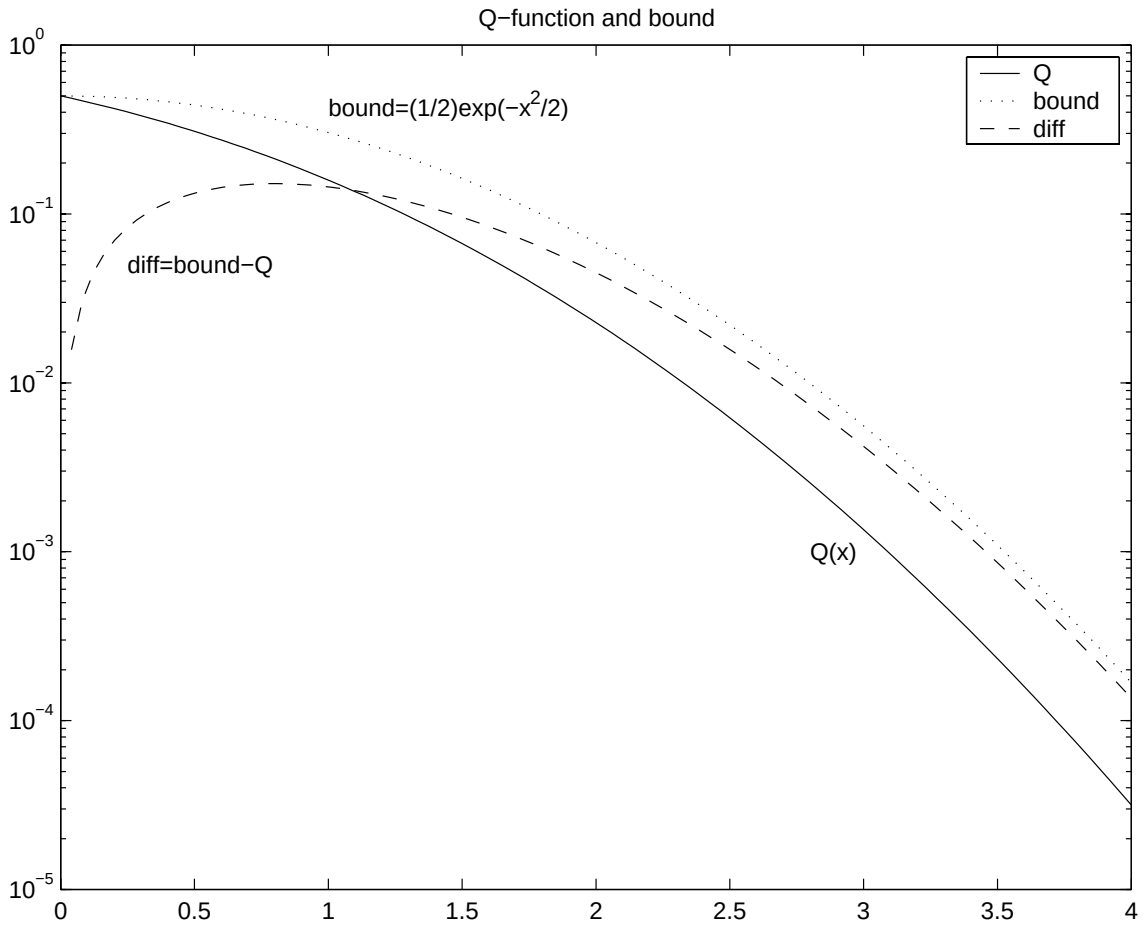

Figure 3-2

To minimize the probability of not receiving enough bits, we can minimize the bound by choosing a set of N_i that maximizes the exponent $(\mu - K)^2 / 2\sigma^2$ subject to the constraint $N = \sum_{i=1}^{L} N_i$. To perform the constrained optimization we can form the cost function

$$f(\{N_i\}) = \frac{(\mu - K)^2}{2\sigma^2} + \lambda\left(\sum_{i=1}^{L} N_i - N\right)$$

and the problem can be solved by finding

$$\max_{\{N_i\}} f(\{N_i\})$$

subject to the constraint $N = \sum_{i=1}^{L} N_i$.

To simplify the algebra, let $v = \sigma^2$. Differentiating with respect to N_i and setting equal to zero, we find:

$$\frac{\partial}{\partial N_i}\left[\frac{(\mu - K)^2}{2v} + \lambda\left(\sum_{i=1}^{L} N_i - N\right)\right] = 0$$

$$\frac{\partial}{\partial N_i}\left[\frac{(\mu - K)^2}{2v}\right] + \lambda = 0$$

$$\frac{\partial}{\partial N_i}\left[\frac{(\sum_{i=1}^{L} N_i q_i - K)^2}{2\sum_{i=1}^{L} N_i^2 p_i q_i}\right] + \lambda = 0$$

$$\frac{2(\mu - K)q_i}{2v} - \frac{(2N_i p_i q_i)(\mu - K)^2}{2v^2} + \lambda = 0$$

$$\frac{(\mu - K)}{v}\left(\frac{(\mu - K)N_i p_i q_i}{v} - q_i\right) = \lambda$$

$$\frac{(\mu - K)}{\sigma^2}\left(\frac{(\mu - K)N_i p_i q_i}{\sigma^2} - q_i\right) = \lambda$$

Differentiating with respect to λ recovers the constraint $N = \sum_{i=1}^{L} N_i$. We can use this result to find the optimal number of bits (N_i) to send over each path to maximize the probability of receiving a sufficient number of bits at the destination to reconstruct the message. For any given $\{p_i\}$, these equations can be used to numerically find N_i and the resulting performance. To gain some analytic insight of the problem, we next examine the following three cases:

1) when the paths are very reliable ($p_i \approx 0$),

2) when the paths are very unreliable ($p_i \approx 1$), and

3) when the paths are approximately 50% reliable ($p_i \approx 1/2$).

3.1 Case of Reliable Paths

For the case where the paths are very reliable, $p_i = \varepsilon_i$, where ε_i is small.

$$\lambda = \frac{(\mu - K)}{\sigma^2}\left(\frac{(\mu - K)N_i p_i q_i}{\sigma^2} - q_i\right)$$

$$\lambda = \frac{(\mu - K)}{\sigma^2}\left(\frac{(\mu - K)N_i p_i (1 - p_i)}{\sigma^2} - (1 - p_i)\right)$$

Taking the approximation where p_i is small,

$$\lambda \approx \frac{(\mu - K)}{\sigma^2}\left(\frac{(\mu - K)N_i p_i}{\sigma^2} - 1\right)$$

Eliminating the terms that are constant and independent of i (c denotes a constant),

$$c_1 \approx \left(\frac{(\mu - K)N_i p_i}{\sigma^2} - 1\right)$$

$$c_2 \approx N_i p_i$$

Thus, we see that the selection of N_i is optimized when $N_i p_i \approx \text{constant}$. This agrees with our intuition: it means that if one of the paths is more unreliable than another, then we should send fewer bits over the less reliable path. To find an explicit expression for N_i, we solve for c_2 by using the constraint $N = \sum_{i=1}^{L} N_i$:

$$N = \sum_{i=1}^{L} \frac{c_2}{p_i}$$

$$c_2 = \frac{N}{\sum_{i=1}^{L} \frac{1}{p_i}}$$

$$c_2 = \frac{N}{\beta}$$

where $\beta = \sum_{i=1}^{L} \frac{1}{p_i}$. Thus,

$$N_i = \left(\frac{N}{\beta}\right)\frac{1}{p_i}$$

Substituting this into the expressions for μ and σ^2, we arrive at an analytical expression for the bound for $\Pr[M < K]$:

$$\begin{aligned}
\mu &= \sum_{i=1}^{L} N_i q_i \\
&= \frac{N}{\beta}\sum_{i=1}^{L}\frac{q_i}{p_i} \\
&= \frac{N}{\beta}(\beta - L) \\
&= N\left(1 - \frac{L}{\beta}\right)
\end{aligned}$$

$$\begin{aligned}
\sigma^2 &= \sum_{i=1}^{L} N_i^2 p_i q_i \\
&= \sum_{i=1}^{L}\left(\frac{N}{\beta}\right)^2 \frac{q_i}{p_i} \\
&= \left(\frac{N}{\beta}\right)^2 (\beta - L) \\
&= \frac{N^2}{\beta}\left(1 - \frac{L}{\beta}\right)
\end{aligned}$$

Upon further substitution, we obtain the bound:

$$\Pr[M < K] < \frac{1}{2} e^{\frac{-(\mu - K)^2}{2\sigma^2}}$$

$$= \frac{1}{2} \exp\left(\frac{-\left(N\left(1 - \frac{L}{\beta}\right) - K \right)^2}{2\left(\frac{N^2}{\beta}\left(1 - \frac{L}{\beta}\right)\right)} \right)$$

$$= \frac{1}{2} \exp\left(\frac{-\left(1 - \frac{L}{\beta} - \frac{K}{N}\right)^2}{\frac{2}{\beta}\left(1 - \frac{L}{\beta}\right)} \right)$$

Note that the N_i we have found above are not integers; we need to round each N_i for a feasible solution for a real system. The results of the simulation for this case are highlighted below. The same set of plots is generated for the other two cases. Notes regarding the mechanics of the simulation follow.

3.1.1 Simulation Method

We simulate the scenarios using MATLAB (source code is listed in Appendix B). Random probabilities of failure, uniformly distributed in the range $[p_{\min}, p_{\max}]$, for each link are generated. The range of probabilities is indicated on the plots. Another set of random numbers, uniformly distributed in the range [0,1] are used to determine if a path has succeeded; if this number is greater than the failure probability assigned to the path, then the path is considered to have succeeded, and the bits assigned to that path are counted in the total M. This whole process is considered one "trial". In the simulations, we run 10^6 trials, and determine the probability distribution of M by taking normalized histograms of all the trial results. All relevant parameters of the simulation are indicated on the plots.

We use an iterative procedure to assign (integer) bits to each path. For example, in the case of highly reliable paths, we search for a constant c such that $N_i p_i = c$ for all i that allows the resulting N_i to also satisfy $N = \sum_{i=1}^{L} N_i$. (We check that the constant arrived at using this method and using the analytic expression for N_i are substantially the same, to eliminate anomalous results.)

3.1.2 Simulation Results

- Figure 3-3 shows the Gaussian approximation that results when the N_i are selected using two methods: An "equal" selection where the N_i are the same (N_i=N/L), and an "optimized" selection where the N_i are chosen according to the optimization procedure discussed above. We see that the mean of the "optimized" case is greater than that of the "equal" case, indicating that we expect to receive more bits using that selection process. For all the cases presented, the "equal" selection is the reference to which we compare the optimization results.

- Figure 3-4 is a normalized histogram of the total received number of bits (M) from all the trials. This shows that our use of the Central Limit Theorem is justified, and that we observe the same increase in the mean M in the optimized case.

- Figure 3-5 plots the bound of $\Pr[M < K]$ derived above compared to the experimentally determined cumulative distribution of M. There are five curves on this plot:

    - An analytic bound calculated using the non-integer optimized values of N_i;

    - A bound calculated where the N_i are the same ("equal"), and a bound calculated using the optimized and rounded values of N_i ("optimized");

    - The numerically determined cumulative probabilities of M for the "equal" and "optimized" case. Observe that in the case of highly reliable paths, the bound calculated using non-rounded N_i is almost exactly the same as the bound calculated using the rounded N_i, which is why they appear to overlap each other. For this particular case, the bound for the "equal" case holds quite well, whereas the bound for the "optimized" case departs from the experimental data at lower probabilities. This is because the Gaussian approximation using the CLT is not as good an approximation at small probabilities (i.e., at the tails of the distribution). For example, a large N_i is invested in reliable links, causing a significant fraction of the total number of bits to be lost if those links fail (albeit it with low probability), thus diverging from the CLT approximation. Keep in

mind that we connect the points that are numerically determined to clearly show the grouping, not to indicate that intermediate points can be interpolated.

- Figure 3-6 is plot of the probabilities randomly assigned to each path. (We have sorted the probabilities to ease visualization.) Observe that the probabilities are indeed uniformly distributed within the interval.

- Figure 3-7 is a plot of the optimized and rounded N_i for each path. We see that if some paths have a very high reliability, then those paths are assigned many more bits than other paths. This can lead to singular N_i selections unless some minimum probability of failure is assigned to each path. (This is why we set a $p_{\min}$ for the paths, for if a path was truly 100% reliable, then we would of course place all the bits on that path.)

- The last plot for this case, Figure 3-8, shows the variation in the value of $N_i p_i$ for each path i. The jaggedness is due to the rounded values of the N_i. (If the N_i were not rounded, we would see a flat line, since $N_i p_i$ is a constant for all i.)

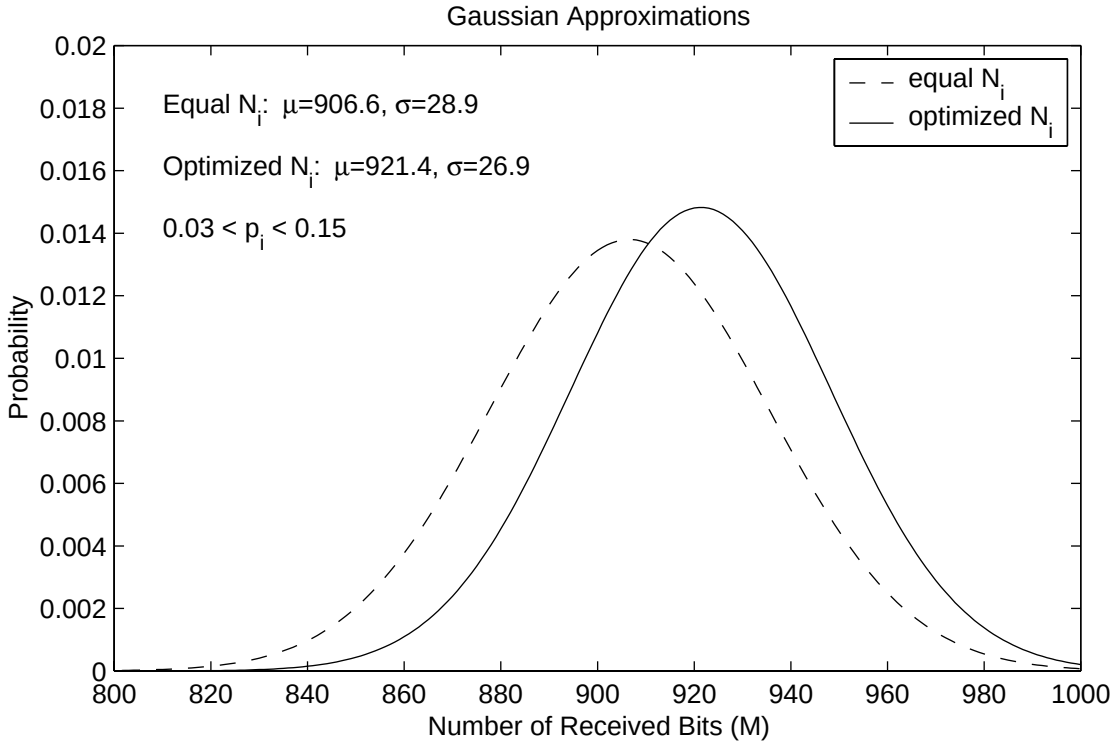

Figure 3-3

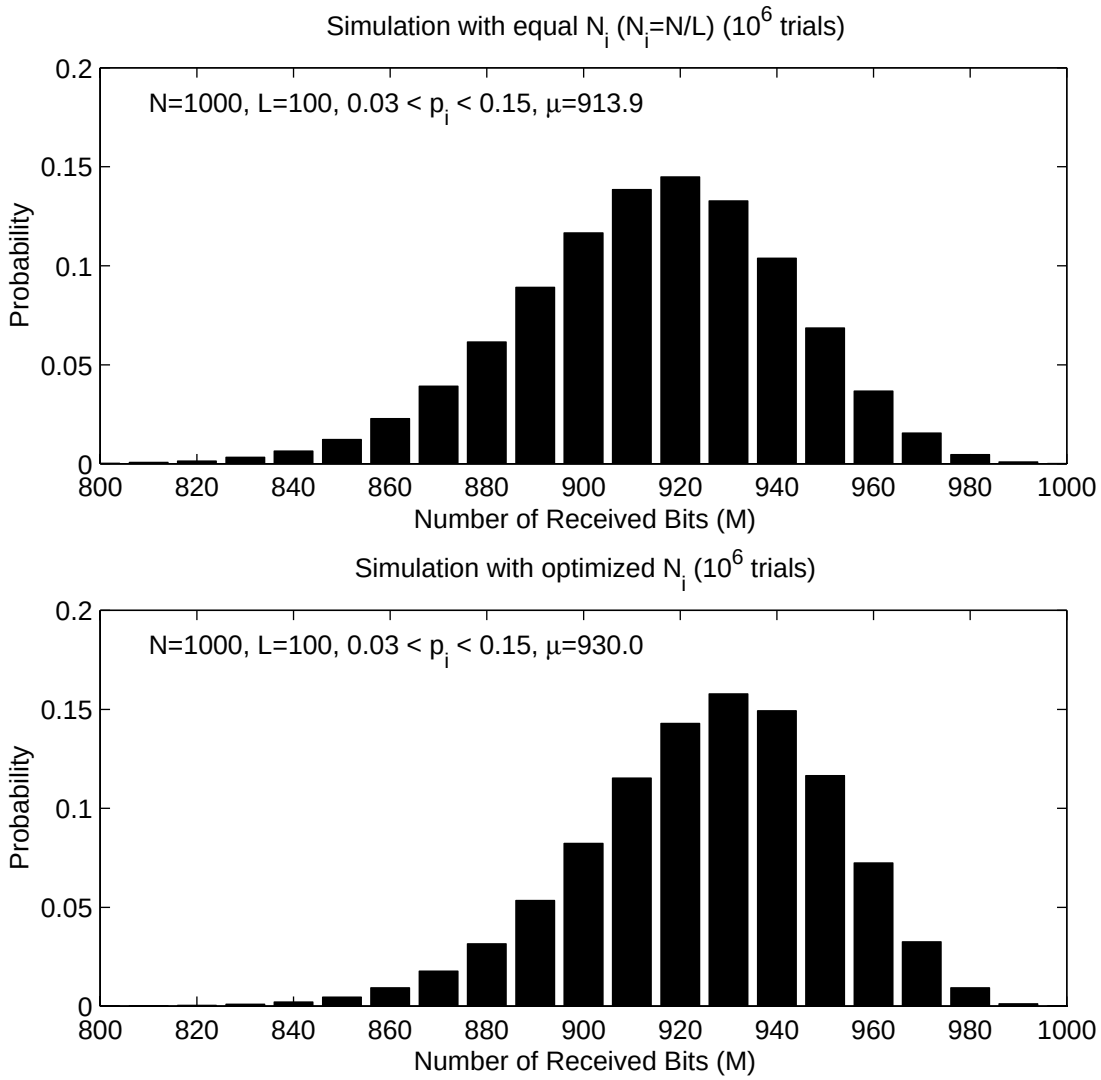
Simulation with equal N_i (N_i=N/L) (10^6 trials)
N=1000, L=100, 0.03 < p_i < 0.15, μ=913.9
Probability
Number of Received Bits (M)
Simulation with optimized N_i (10^6 trials)
N=1000, L=100, 0.03 < p_i < 0.15, μ=930.0
Probability
Number of Received Bits (M)

Figure 3-4

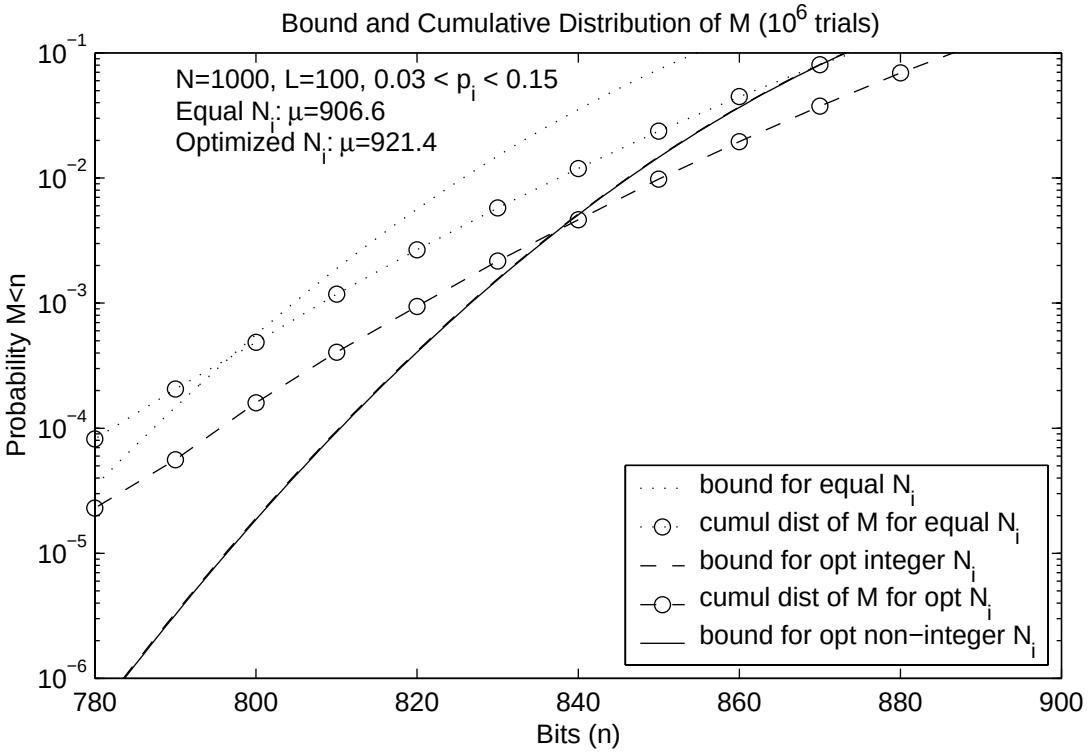

Figure 3-5

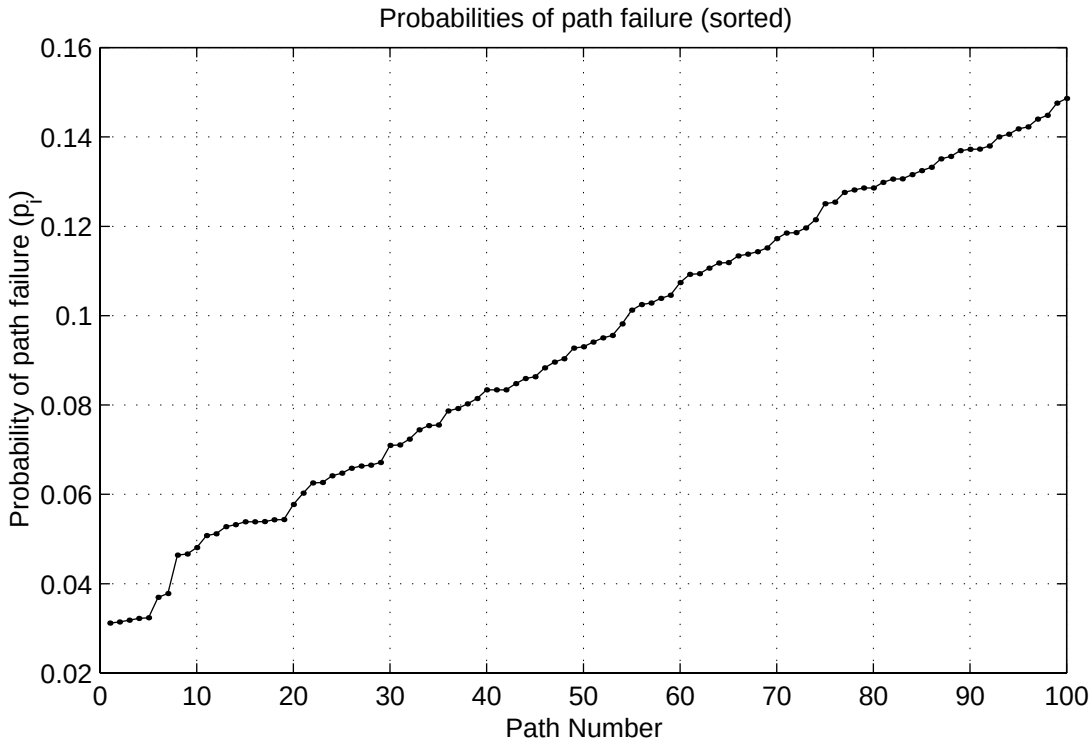

Figure 3-6

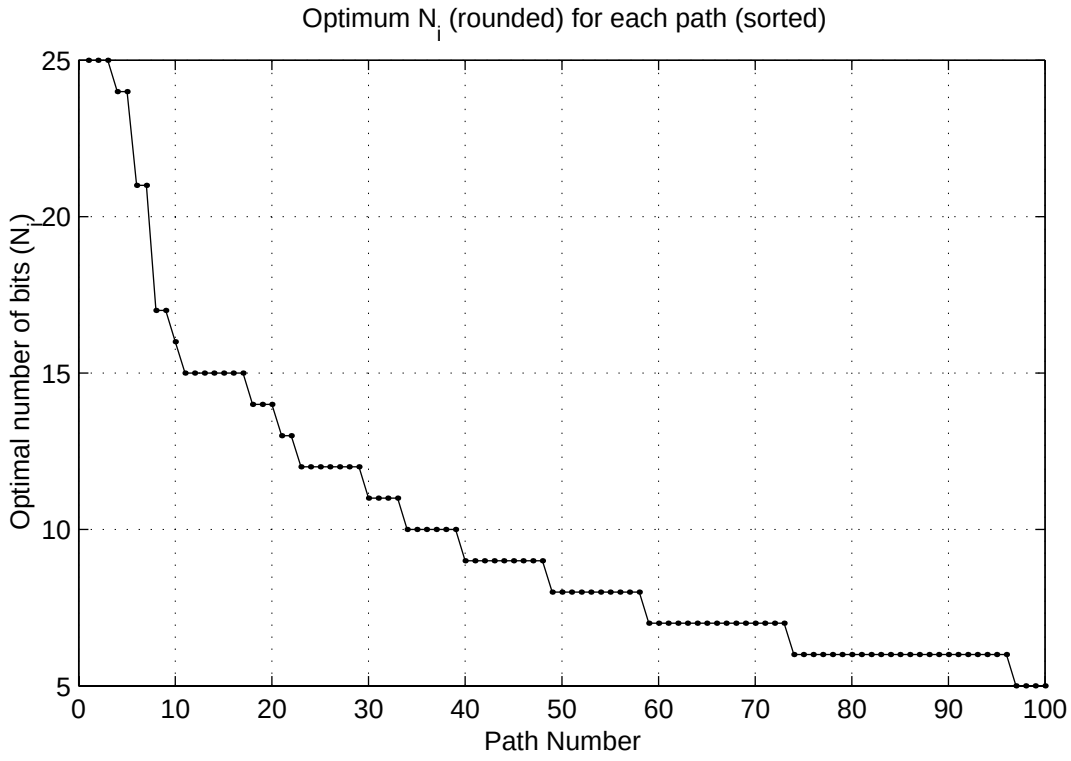

Figure 3-7

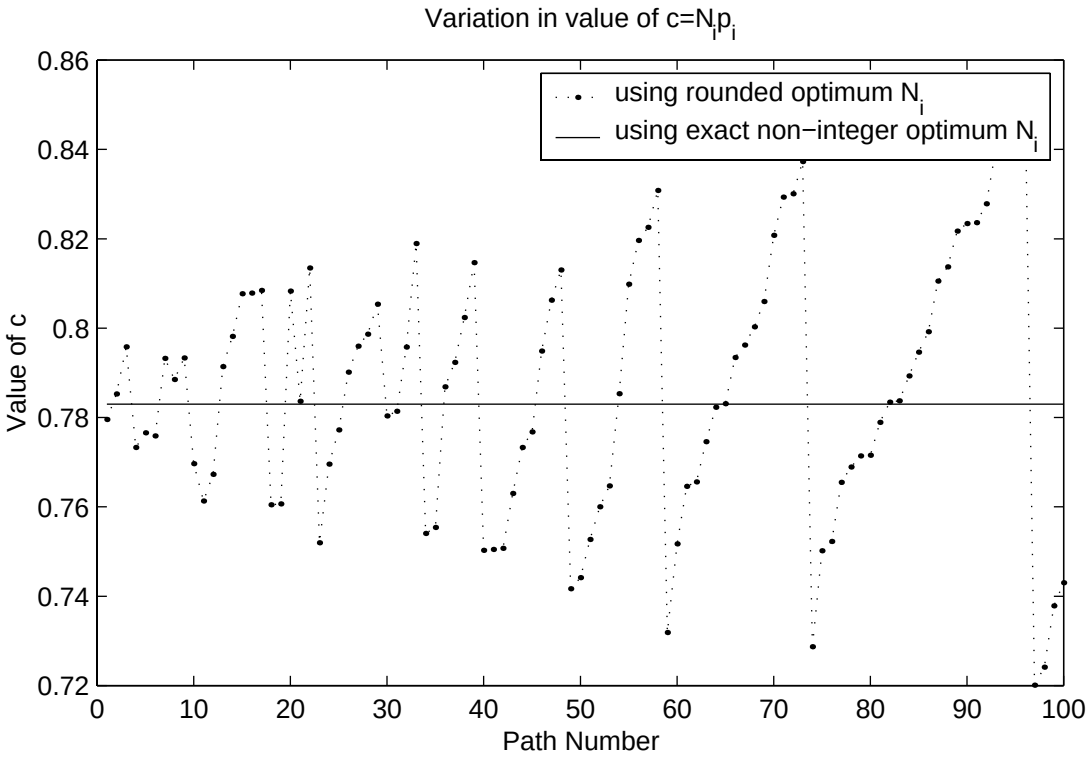

Figure 3-8

3.2 Case of Unreliable Paths

For the case where the paths are very unreliable, $p_i = (1-\varepsilon_i)$, where ε_i is small, we arrive at an interesting conclusion. For convenience, let $\theta = (\mu - K)/\sigma^2$. Starting with the optimization equation, and then making appropriate approximations,

$$\lambda = \theta(\theta N_i p_i q_i - q_i)$$
$$c_1 = \theta N_i p_i q_i - q_i$$

Since p_i is very close to one,

$$c_2 \approx \theta N_i q_i - q_i$$
$$c_2 \approx (\theta N_i - 1) q_i$$

If we continue to solve the system for c_2 just as we did in the previous case, we would eventually arrive at

$$N_i = \frac{1}{\theta}\left(\frac{c_2}{\theta q_i} + 1\right)$$
$$c_2 = \frac{N\theta - L}{\sum \frac{1}{\theta q_i}}$$
$$N_i = \frac{N\theta - L}{q_i \sum \frac{1}{q_i}} + \frac{1}{\theta}$$

which does not provide us with much insight. However, if $\theta N_i >> 1$, then the optimal choice of N_i would be such that $N_i q_i \approx \text{constant}$. We now find the conditions under which this holds. Let $\hat{N} = \max(N_i)$, $q_{\min} = \min(q_i)$, and $q_{\max} = \max(q_i)$.

$$\begin{aligned}\sigma^2 &= \mathrm{VAR}[M] \\ &= \sum_{i=1}^{L} N_i^2 p_i q_i \\ &\approx \sum_{i=1}^{L} N_i^2 q_i \\ &\le \hat{N}^2 \sum_{i=1}^{L} q_i \\ &\le \hat{N}^2 L q_{\max}\end{aligned}$$

Since we are trying to operate in a regime with a high probability of receiving sufficient bits for reconstruction, we can say that

$$(\mu - K) = \tau\sigma$$

where τ is around 3 or 4 for the system to ensure high reliability, and as high as 6 for "six-sigma" operations. Substituting this into the definition for θ,

$$\theta = \frac{\tau\sigma}{\sigma^2} = \frac{\tau}{\sigma}$$

Substituting this into $\theta N_i >> 1$ yields the equivalent conditions:

$$\begin{aligned}\frac{\tau}{\sigma} N_i &>> 1 \\ \left(\frac{\tau}{\sigma} N_i\right)^2 &>> 1 \\ \left(\frac{\tau}{\sigma} N_i\right)^2 &> \frac{(\tau N_i)^2}{\hat{N}^2 L q_{\max}} >> 1\end{aligned}$$

Thus, if $\frac{(\tau N_i)^2}{\hat{N}^2 L q_{\max}} >> 1$, then $N_i q_i \approx \text{constant}$. If the probabilities of path failure are clustered and bounded close to each other, then by symmetry the individual N_i should be similar to each other and to $\hat{N}$, thus we can make the following further simplification:

$$\frac{(\tau N_i)^2}{\hat{N}^2 L q_{\max}} \approx \frac{\tau^2}{L q_{\max}}$$

And thus,

$$L << \frac{\tau^2}{q_{\max}}$$

Thus, when $3 \le \tau \le 6$, and the path success probabilities are less than 0.1, then L needs to be around 100, which is reasonable from the discussion in Chapter 1. If $\tau = 6$, then $L \sim \tau^2 / q_{\max} \sim 360$. Continuing as we did in the previous case, we find an explicit expression for N_i by solving for c_2 by satisfying the constraint $N = \sum_{i=1}^{L} N_i$:

$$N = \sum_{i=1}^{L} \frac{c_2}{q_i}$$

$$c_2 = \frac{N}{\sum_{i=1}^{L} \frac{1}{q_i}}$$

$$c_2 = \frac{N}{\alpha}$$

where $\alpha = \sum_{i=1}^{L} \frac{1}{q_i}$. Thus,

$$N_i = \left(\frac{N}{\alpha}\right)\frac{1}{q_i}$$

We substitute this into the expressions for μ and σ^2, and find an expression for the bound for $\Pr[M < K]$.

$$\begin{aligned}\mu &= \sum_{i=1}^{L} N_i q_i \\ &= \frac{NL}{\alpha} \\ \sigma^2 &= \sum_{i=1}^{L} N_i^2 p_i q_i \\ &= \sum_{i=1}^{L} \left(\frac{N}{\alpha}\right)^2 \frac{p_i}{q_i} \\ &= \left(\frac{N}{\alpha}\right)^2 (\alpha - L) \\ &= \frac{N^2}{\alpha}\left(1 - \frac{L}{\alpha}\right)\end{aligned}$$

And then substituting into the expression for $\Pr[M < K]$:

$$\begin{aligned}\Pr[M < K] &< \frac{1}{2} e^{\frac{-(\mu - K)^2}{2\sigma^2}} \\ &= \frac{1}{2}\exp\left(\frac{-\left(\frac{NL}{\alpha} - K\right)^2}{2\left(\frac{N^2}{\alpha}\left(1 - \frac{L}{\alpha}\right)\right)}\right) \\ &= \frac{1}{2}\exp\left(\frac{-\left(\frac{L}{\alpha} - \frac{K}{N}\right)^2}{\frac{2}{\alpha}\left(1 - \frac{L}{\alpha}\right)}\right)\end{aligned}$$

The simulation results show that the optimization does improve the situation, and that the bounds hold reasonably well. See Figures 3-9 through 3-14. (This is the same series of plots described for the previous case.) These simulations look at a scenario where the network has 300 paths, and $\tau = 6$.

One should realize this analysis holds only if the Gaussian approximation is valid. This means that the left-side tail of the distribution must not be truncated at the y-axis; therefore, we choose a scenario in which the mean number of received bits is shifted far enough from the y-axis so as to preserve the tail.

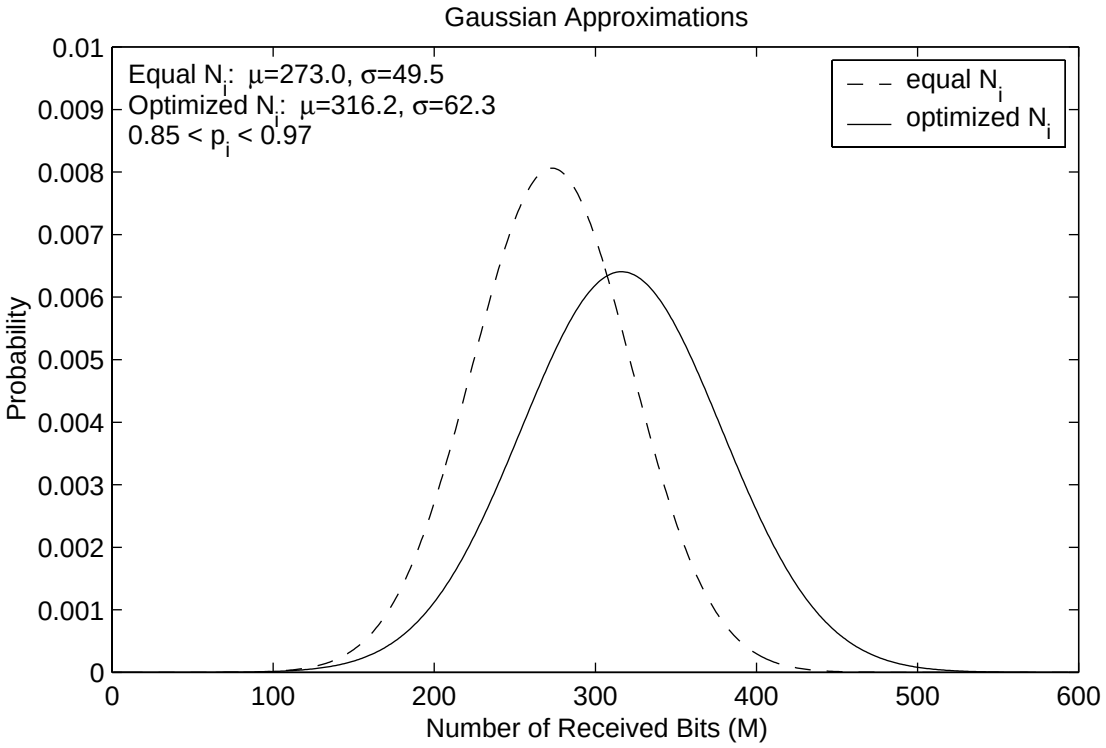
Gaussian Approximations
Equal N_i: μ=273.0, σ=49.5
Optimized N_i: μ=316.2, σ=62.3
0.85 < p_i < 0.97
equal N_i
optimized N_i
Probability
Number of Received Bits (M)
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01
0
100
200
300
400
500
600

Figure 3-9

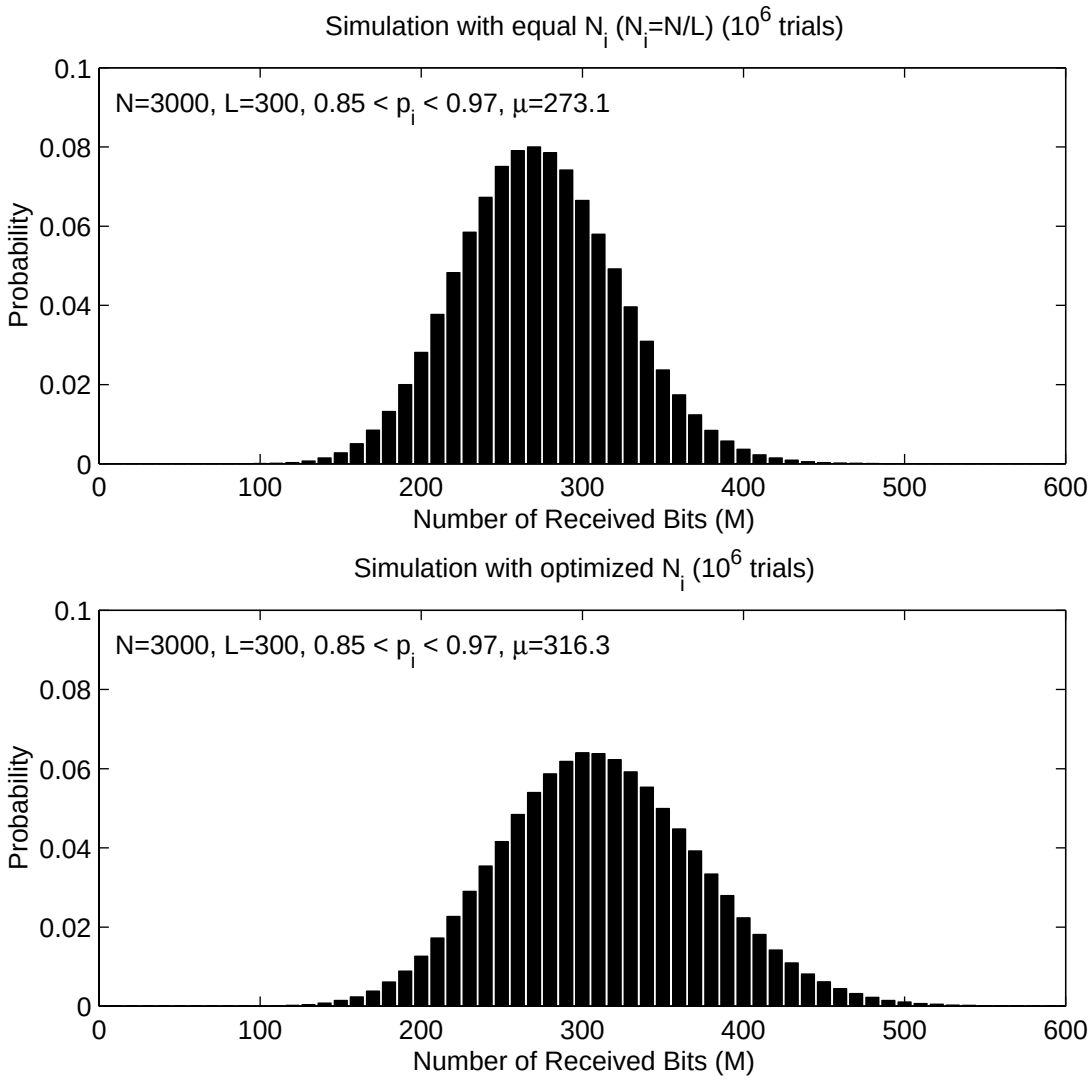
Simulation with equal N_i (N_i=N/L) (10^6 trials)
N=3000, L=300, 0.85 < p_i < 0.97, μ=273.1
Probability
Number of Received Bits (M)
Simulation with optimized N_i (10^6 trials)
N=3000, L=300, 0.85 < p_i < 0.97, μ=316.3
Probability
Number of Received Bits (M)

Figure 3-10

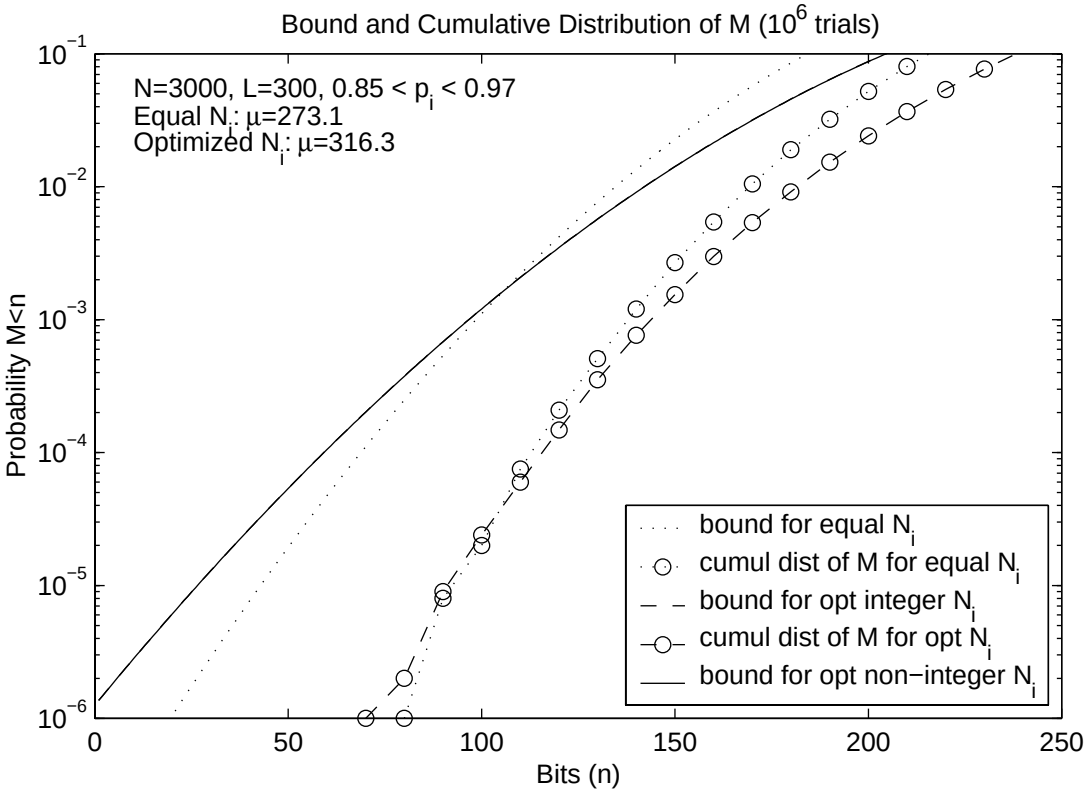

Figure 3-11

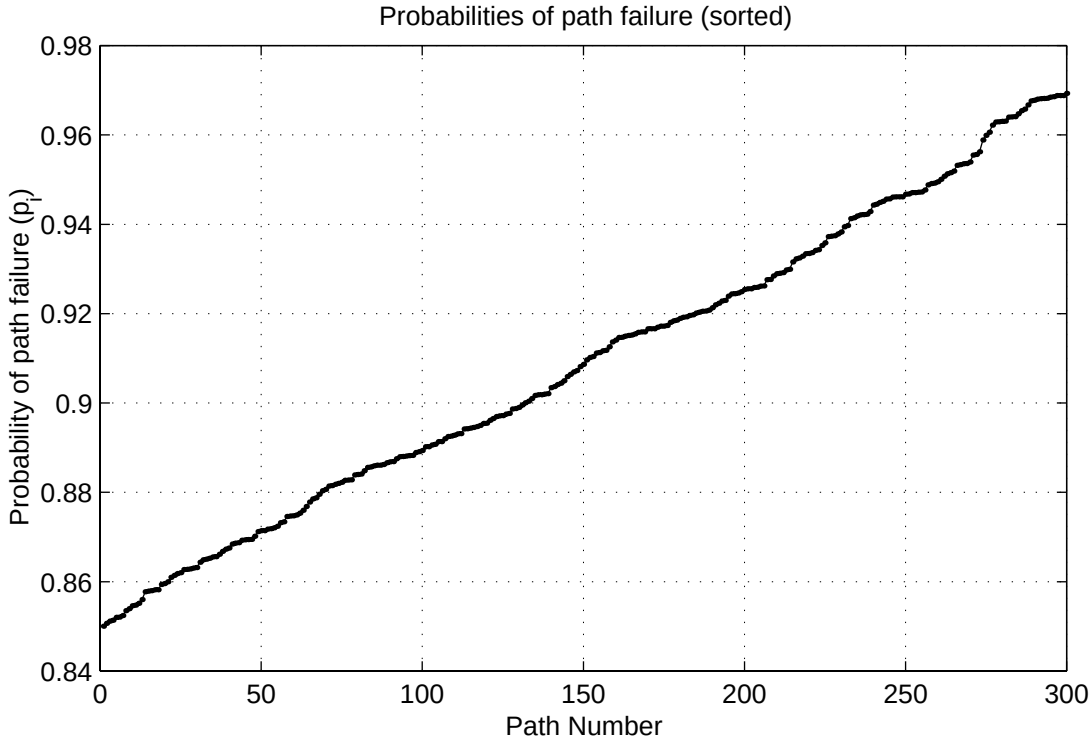

Figure 3-12

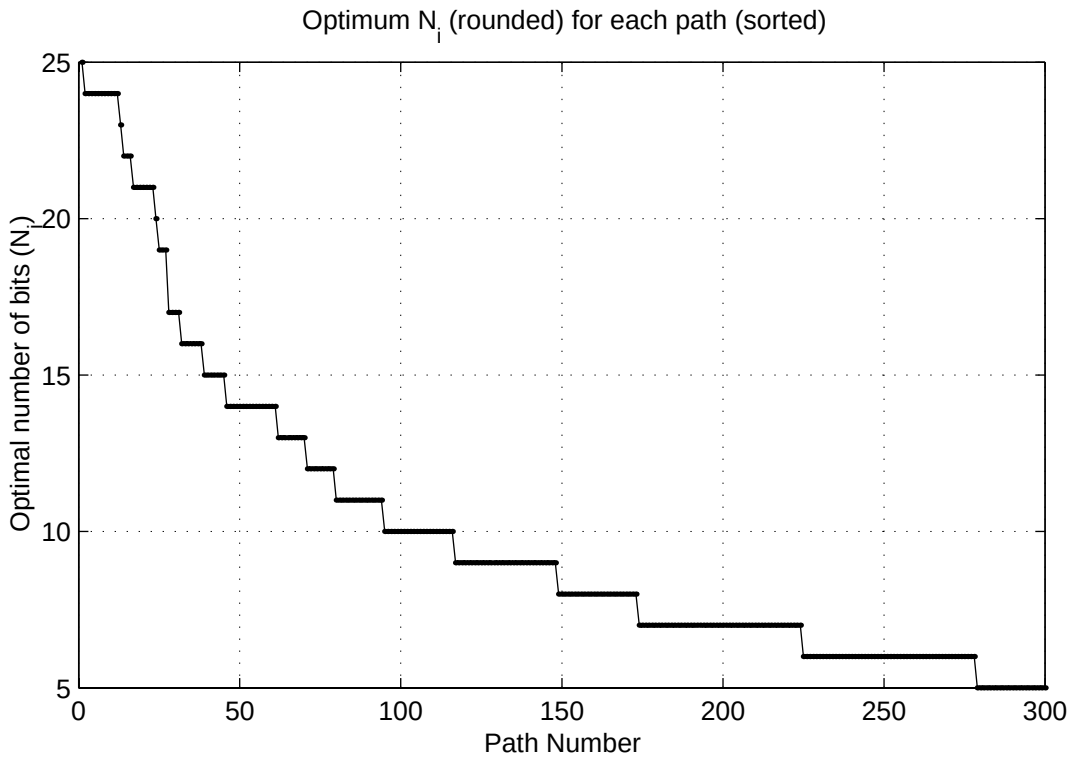

Figure 3-13

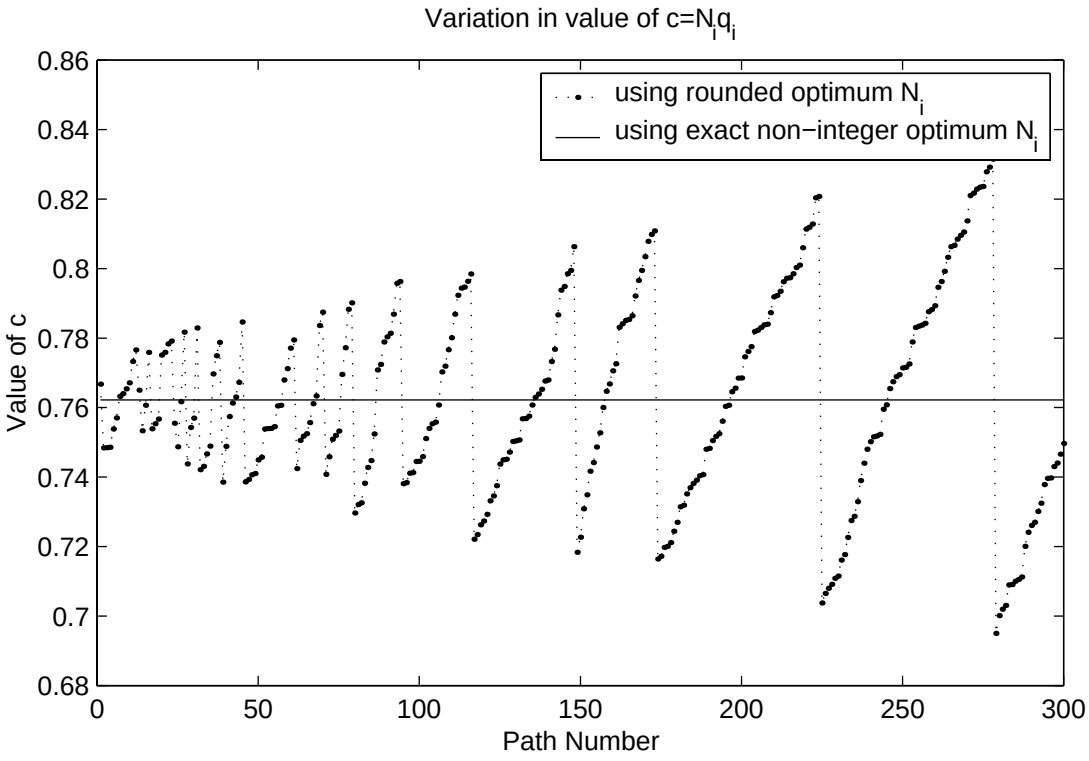

Figure 3-14

3.3 Case of paths with 50% reliability

We finally examine the case where the probabilities of path failure are close to 0.5: $p_i = \left(\frac{1}{2} + \varepsilon_i\right)$, with ε_i in a small interval around zero. We proceed just as we did in the first case.

$$\lambda = \frac{(\mu - K)}{\sigma^2}\left(\frac{(\mu - K)N_i p_i q_i}{\sigma^2} - q_i\right)$$

$$\lambda = \frac{(\mu - K)}{\sigma^2}\left(\frac{(\mu - K)N_i (1/2 + \varepsilon_i)(1/2 - \varepsilon_i)}{\sigma^2} - (1/2 - \varepsilon_i)\right)$$

$$\lambda = \frac{(\mu - K)}{\sigma^2}\left(\frac{(\mu - K)N_i (1/4 - \varepsilon_i^2)}{\sigma^2} - (1/2 - \varepsilon_i)\right)$$

$$\lambda \approx \frac{(\mu - K)}{\sigma^2}\left(\frac{(\mu - K)N_i (1/4)}{\sigma^2} - (1/2 - \varepsilon_i)\right)$$

$$c_2 \approx \left(\frac{(\mu - K)N_i (1/4)}{\sigma^2} - (1/2 - \varepsilon_i)\right)$$

$$c_3 - \varepsilon_i \approx \left(\frac{(\mu - K)N_i (1/4)}{\sigma^2}\right)$$

$$4(c_3 - \varepsilon_i) \approx \theta N_i$$

where $\theta = \frac{(\mu - K)}{\sigma^2}$. To find an explicit expression for N_i, we proceed to satisfy the constraint $N = \sum_{i=1}^{L} N_i$:

$$N = \sum_{i=1}^{L} \frac{4(c_3 - \varepsilon_i)}{\theta}$$

$$N = \frac{4}{\theta}\sum_{i=1}^{L} (c_3 - \varepsilon_i)$$

$$N = \frac{4}{\theta}\left(Lc_3 - \sum_{i=1}^{L} \varepsilon_i\right)$$

$$c_3 = \frac{1}{L}\left(\frac{N\theta}{4} + \sum_{i=1}^{L} \varepsilon_i\right)$$

Substituting into $4(c_3 - \varepsilon_i) \approx \theta N_i$,

$$N_i = \frac{4}{\theta}\left(\frac{1}{L}\left(\frac{N\theta}{4} + \sum_{i=1}^{L} \varepsilon_i\right) - \varepsilon_i\right)$$

$$N_i = \frac{4}{\theta}\left(\frac{N\theta}{4L} + \frac{\sum_{i=1}^{L} \varepsilon_i}{L} - \varepsilon_i\right)$$

$$N_i = \frac{N}{L} + \frac{4}{\theta}\left[\frac{\sum_{i=1}^{L} \varepsilon_i}{L} - \varepsilon_i\right]$$

Suppose that ε_i are uniformly distributed in a small interval around 0. In this case, the first term within the brackets becomes negligible as L increases. Thus, when $\sum_{i=1}^{L} \varepsilon_i \approx 0$,

$$N_i = \frac{N}{L} - \frac{4\varepsilon_i}{\theta},$$

which implies that we should place more bits on the more reliable paths. (Note that when the ε_i are the same, then all the p_i are equal, and this expression reduces to $N_i = N/L$ as we would expect.) After we find the closest set of integers for N_i, simulation shows that the optimization improves the situation, as shown in Figures 3-15 through 3-20. The last plot in the series is slightly different than in the other cases. It shows the variation in the rounded N_i compared to the analytic expression above. Also, note that the x-axis is ε_i.

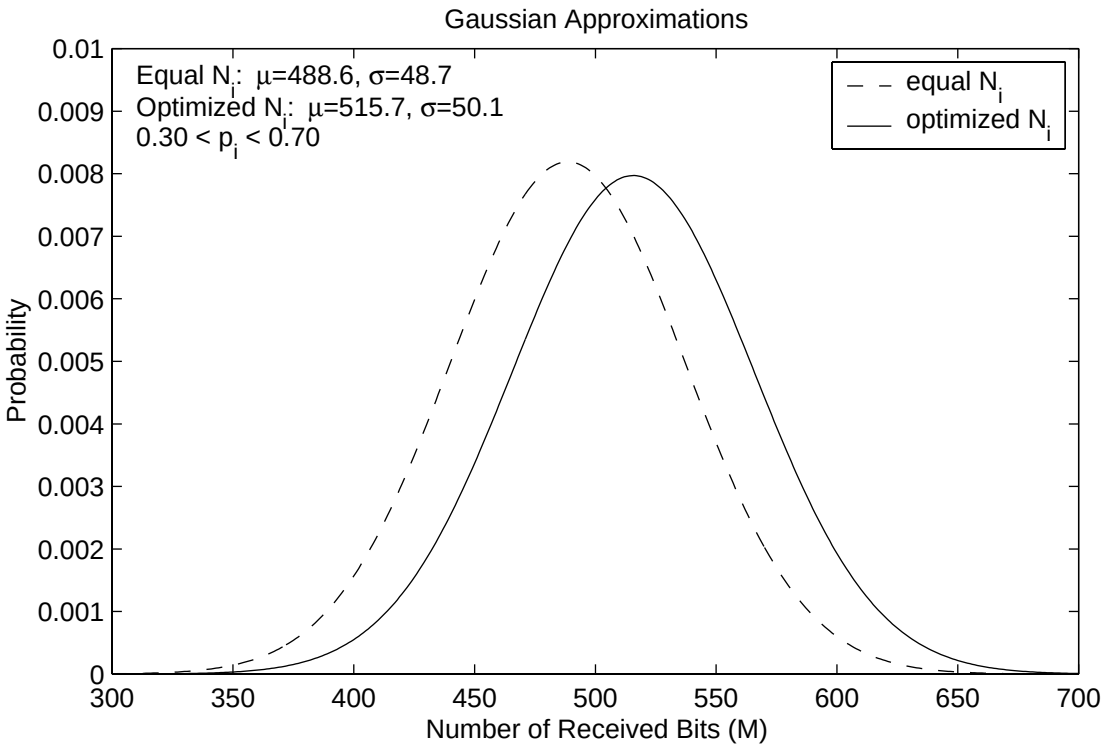
Gaussian Approximations
Equal N_i: μ=488.6, σ=48.7
Optimized N_i: μ=515.7, σ=50.1
0.30 < p_i < 0.70
equal N_i
optimized N_i
Probability
Number of Received Bits (M)
0.01
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
300
350
400
450
500
550
600
650
700

Figure 3-15

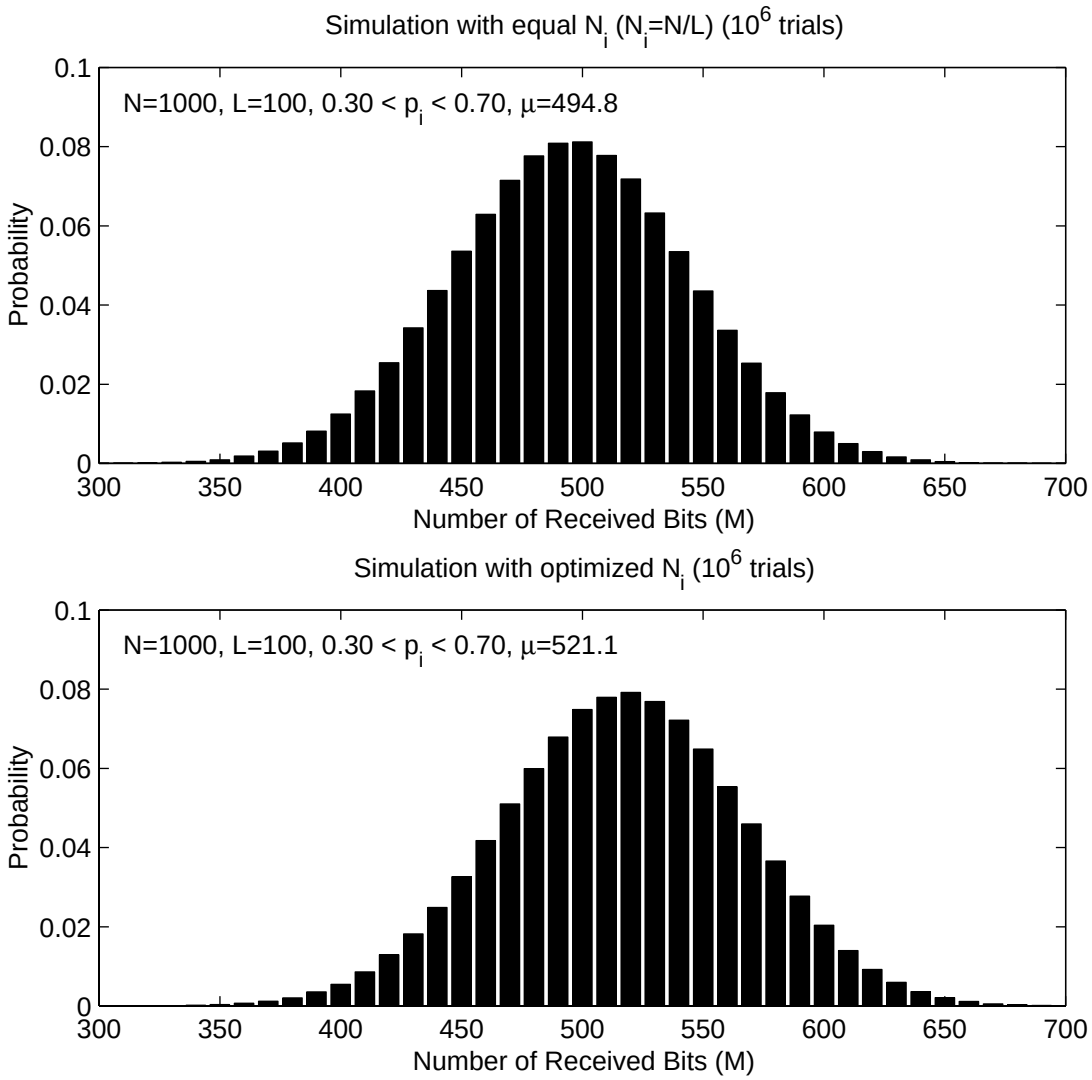
Simulation with equal N_i (N_i=N/L) (10^6 trials)
N=1000, L=100, 0.30 < p_i < 0.70, μ=494.8
Probability
Number of Received Bits (M)
Simulation with optimized N_i (10^6 trials)
N=1000, L=100, 0.30 < p_i < 0.70, μ=521.1
Probability
Number of Received Bits (M)

Figure 3-16

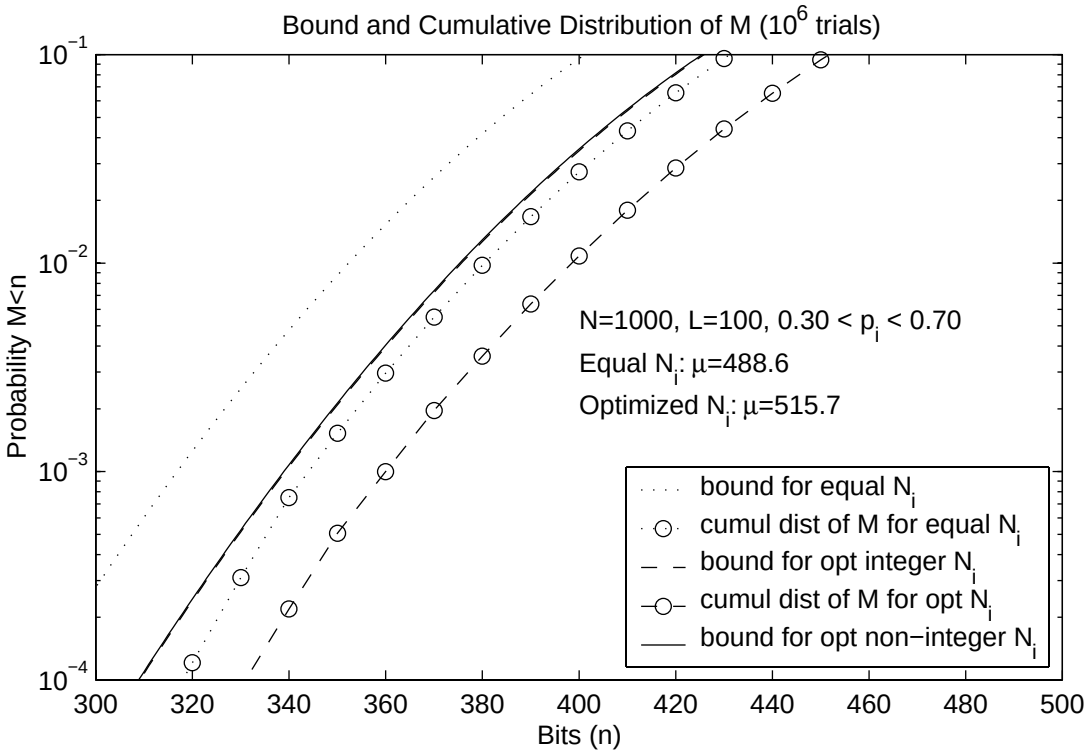

Figure 3-17

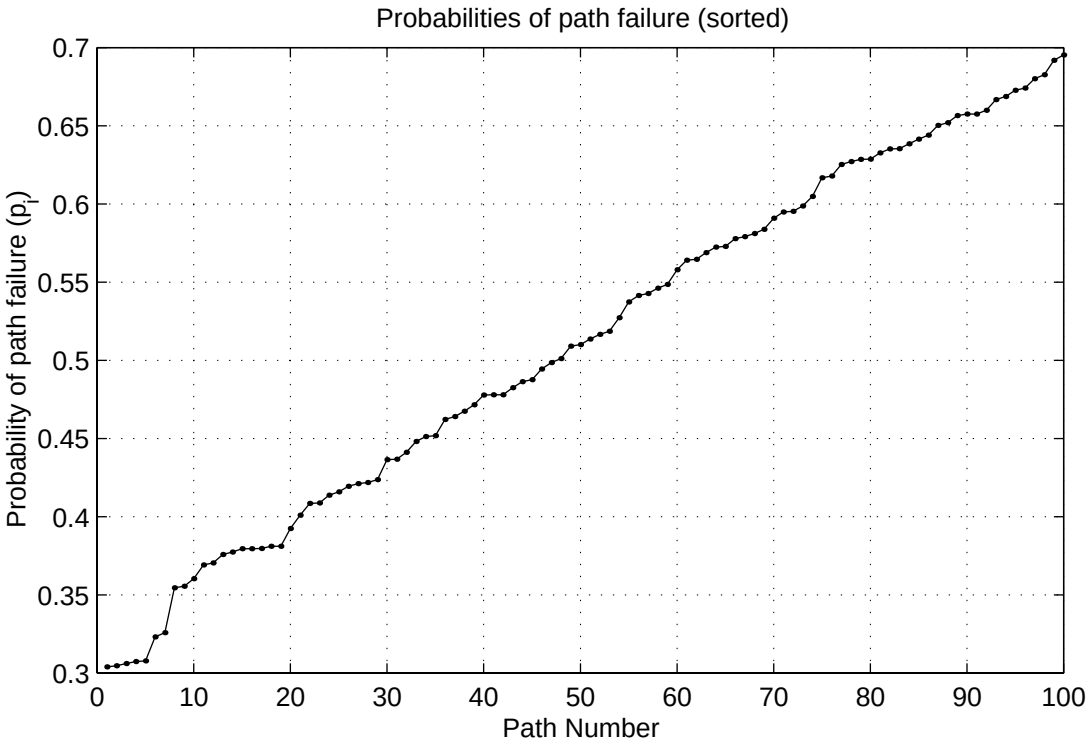

Figure 3-18

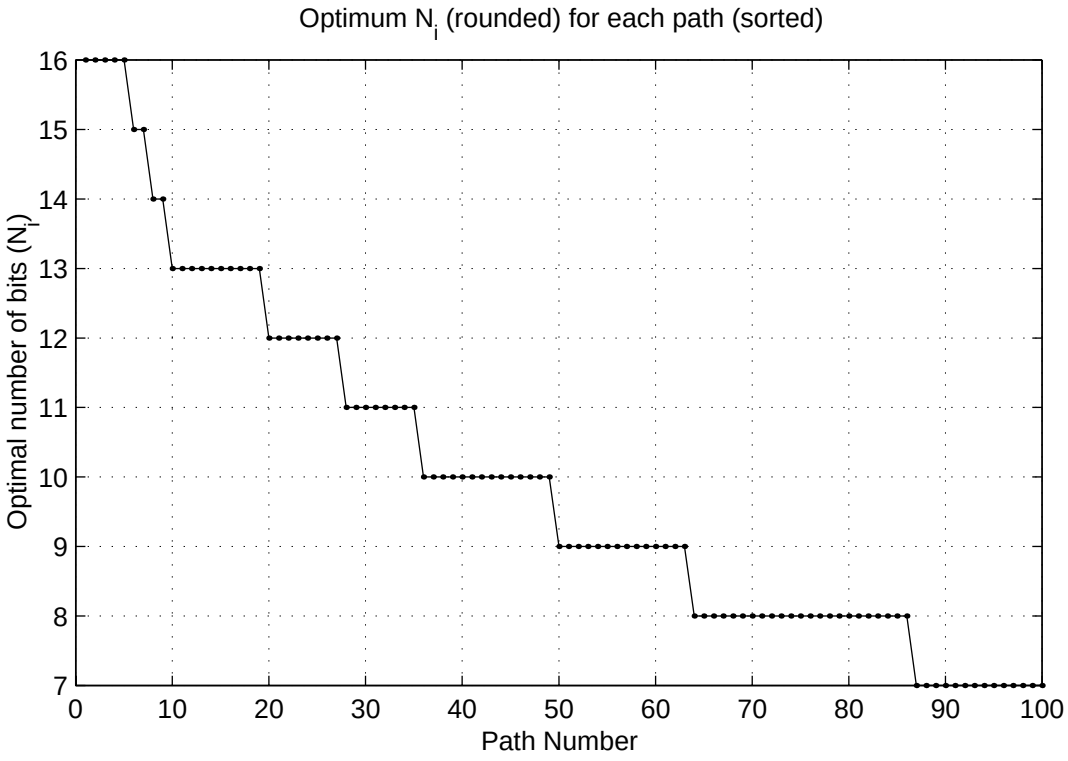

Figure 3-19

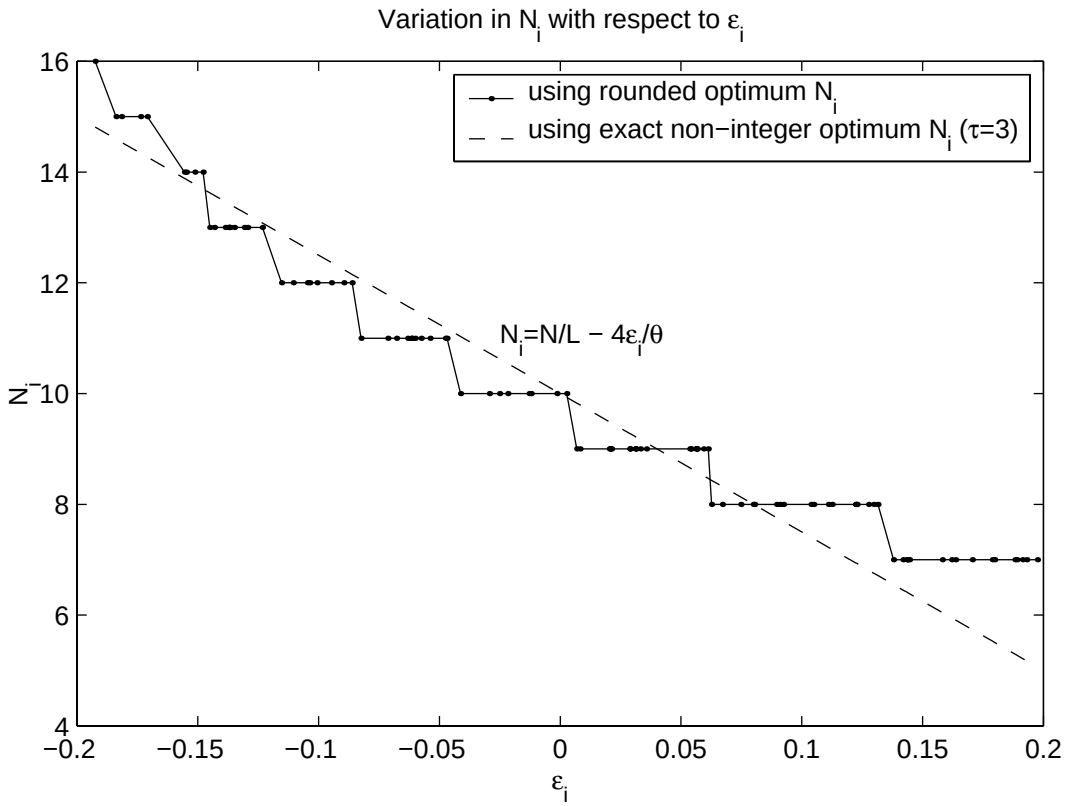

Figure 3-20

4 Applications and Future Research

In this chapter, we propose and discuss additional applications of using erasure codes in data networks. Previous chapters studied the case where the network is densely connected, with multiple paths available from source to destination – a case very relevant to defense networks. In this section, we instead consider potential benefits of using erasure coding even when only a single path is available. We discuss several applications and environments that we have not yet witnessed in the literature or industry that could be interesting avenues for further development and study. We see that the general technique is widely applicable, amenable to new applications as networks become faster and more pervasive.

4.1 Latency reduction in interactive web sessions

The Hypertext Transfer Protocol (HTTP) is the Internet standard mechanism by which a web page is transferred from a web site (server) to user (client). It has been pointed out by many studies that TCP, the underlying transport protocol used by HTTP, is not the most efficient choice given the nature and characteristics of the material being transferred [Pad95]. Indeed, in response to these findings, protocols that provide the reliability of TCP without the overhead have been developed, such as T/TCP [Stev96]. Applying erasure control coding principles to the transfer of web pages could improve the user experience by reducing download latency, especially in situations where the likelihood of a packet being dropped increases as it proceeds closer to the user, such as in modem and mobile/wireless networks [Bala95]. Even when the connection speed is not a low speed service, a few packet drops can trigger a rapid decrease in link utilization due to congestion avoidance strategies employed by TCP.

Adding redundancy to packet data by using an erasure code is currently used in [Byer98, Byer99] to speed-up download times. The idea is for a client to listen to multiple sources concurrently, gathering enough packets to reconstitute the source data. This yields faster downloads because the clients do not need to receive packets in the correct order, and lost packets from one server can be acquired by a redundant coded packet from another server. Also, because a point-to-point connection-oriented data path need not be established between source and destination, servers need not maintain per-connection state information, significantly reducing the amount of server-side processing [Stev94], resulting in an increase in the number of clients each server can accommodate. When combined with multicasting, this can produce an efficient mass distribution system.

An important statistic in web page requests that is often cited is that the average length of a TCP session used for an HTTP page request is about 10 packets [Bak2K]. Clearly, TCP is not used in its preferred and most efficient regime here. Indeed, protocols like T/TCP have been proposed as an alternative to TCP for certain applications, but it doesn't appear to have enjoyed much practical success. Caching is a popular and successful approach to reducing the latency of HTTP page requests, by locating web content closer to the edge of the network. We suggest that by integrating an erasure coding technique with web caching systems, even higher response rates can be achieved.

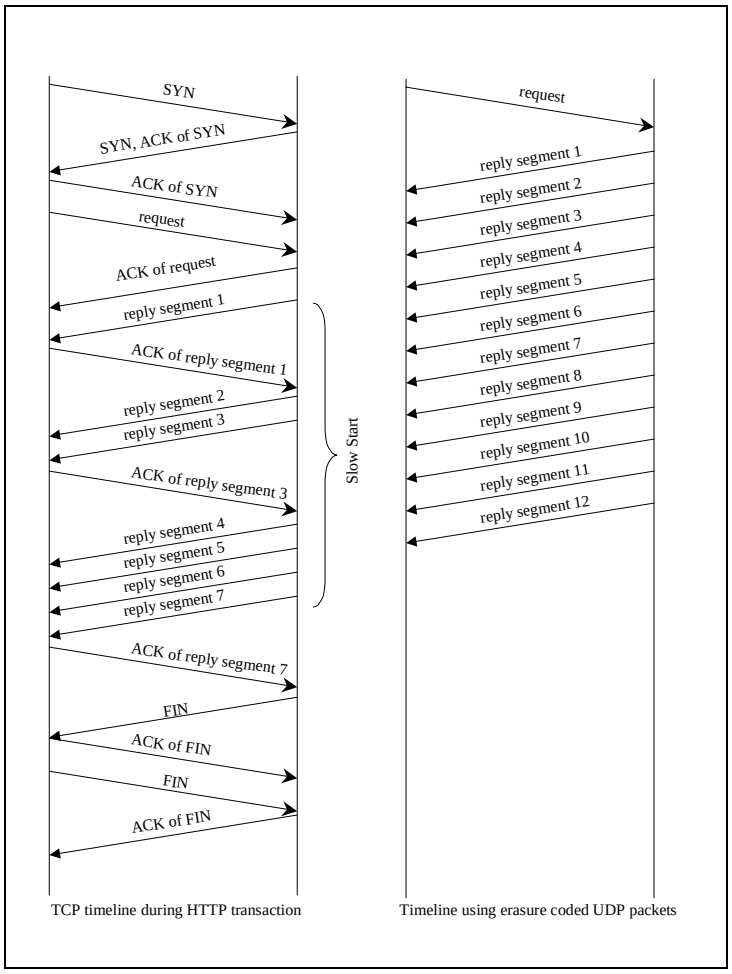

Figure 4-1

Figure 4-1 shows a timeline for an ideal HTTP/TCP transaction: negligible server processing time, no lost packets, and no packet reordering. As discussed by Stevens in detail [Stev94 and Stev96], lost, delayed or reordered packets changes the picture considerably, making the duration of the timeline much longer. For short sessions on the order of the typical HTTP transaction, it can be argued that all the overhead and protocol complexity is overkill. Instead, if we simplify

the situation by sending a burst of erasure-coded UDP packets (as depicted on the right side of Figure 4-1), we eliminate the handshakes, overcome lost packets, and make packet reordering insignificant. We also dramatically reduce the amount of protocol state that needs to be maintained by the server (again, see [Stev94] and [Stev96]).

There are, however, some serious consequences of this approach that need to be discussed. Most notably, we completely bypass TCP's slow-start procedure and can cause congestion in network routers. In response, we argue that modern routers and traffic-shaping appliances (now commonly used throughout the Internet) employ sophisticated rate-shaping and prioritization methods (such as Differentiated Services – "DiffServ") that can compensate for packet bursts, and that the servers need not be troubled with these functions. (This argument violates TCP's end-to-end philosophy, but the applicability of this philosophy is being questioned and rethought by its own developers [Clar2K].) Note well, that while this approach may be potentially suitable for the typical small HTTP session, TCP (along with the associated congestion avoidance and congestion control strategies) is indeed the more appropriate choice for large objects or files. An interesting project would be to study the interplay of DiffServ, commercial traffic-shaping devices, and congestion control methods (like RED), to determine the range of file sizes that can effectively use our approach while still being network-friendly.

An erasure-coding scheme may also help improve performance of mobile users accessing web content. For this to be valid, we need to make an assumption regarding the usage model. Users accessing the web over mobile/wireless links are primarily interested in obtaining relatively small amounts of information quickly. For example, a traveler with a laptop computer and a cellular phone may want to access small amounts of information like news briefs (weather, headlines, financial market numbers). For interactive sessions and applications like electronic mail or transactions, it is preferable to use ordinary HTTP/TCP methods (with the system modifications discussed by [Bala95]). The bottleneck of the path is the cellular modem link, which is far more error-prone than wired links; and as we have discussed previously, bit errors are observed as packet losses by the end-stations. Both the server and the client side have excess computational power to quickly run an erasure-code decoder. The idea is for a web server to provide a UDP service that encodes outgoing content, and intentionally packs it into small UDP packets (no larger than the smallest MTU in any link of the path, and perhaps smaller than the smallest MTU). Thus, if a packet is lost due to bit errors or cellular handoffs, other packets can recover from it more quickly than TCP level retransmissions. On the client side, a specialized web browser (with the decoder integrated into it) that can access these services needs to be used.

Unlike schemes that need to introduce software and protocol processing at the transition between the wired and mobile network (such as Balakrishnan's "snoop" module), this scheme only needs to make some straightforward additions to the web server, and provide a browser client with the ability to make the proper socket calls to the UDP server and decode incoming data. Note that this scheme is possible because the critical element in the transmission, the slow and low-reliability link, has independent errors, effectively resulting in independently dropped packets.

Much of the literature that discuses the use of FEC techniques in packet networks argue that the decoding is too computationally expensive [Byer99]. We argue the opposite: that computers today have a excess of processing power and memory, thus making the use of error coding techniques more attractive then originally assumed, especially for small exchanges with a modest number of packets per object. The argument that FEC increases overhead is still true, however, in this situation, the overhead can be justified. Note that encoding processing overhead need not be suffered for every request, as content can be stored at the server in encoded form, thereby eliminating the encoding step at the server for every page request. In most cases, the small increase in data size is more then justified considering the poor response of a protocol like TCP in a bad transmission environment.

4.2 Transport layer for critical messaging

The need for the reliable transmission of critical messages was the primary motivation behind this work. Critical messaging can be required at the application level by users of the network, or it can be needed by the network itself, for the dissemination of critical network management and control information required for operations such as fault-recovery. The Simple Network Management Protocol is the de-facto standard for the organization and transport of network management information used to perform standard network management functions oftentimes called FCAPS (Fault Recovery, Configuration, Accounting, Performance, and Security) [Rose96]. Such tasks are needed to manage equipment ranging from local area network devices, such as servers, printers, and packet switches, to the management of core IP packet routers and core ATM cell switches in wide-area networks.

During the development and standardization of SNMP, the use of a connection-oriented transport service (such as TCP) versus a datagram service (UDP) to relay management messages was debated. In the end, UDP was selected for the reason that during a network failure event, one would want to use the simplest transport possible. A consequence of this, however, is that every SNMP message effectively needs to be limited in size to the maximum transmission unit (MTU)

of the network. Although UDP supports a larger packet size than the MTU, the fragmented packets could easily be lost or delivered out of order, thus rendering all the packets useless, because UDP does not provide for retransmissions (unless handled at the application layer), and IP does not retransmit fragments. Recent studies have shown that in modern networks, packet ordering is not preserved far more often than previously assumed due to parallelism within network packet routers [Ben99]. Thus it seems likely that coding network management messages with an erasure code can improve the situation of SNMP (or SNMP-like interactions). Since the received data packets need to be sorted for the decoding process, the need for ordered delivery requirements is eliminated, while still allowing for transactions consuming more than one packet. In the scenario where a network is experiencing rapid reconfigurations (either intentionally or due to network failure or attack), messages can be sent with the appropriate level of reliability by increasing the amount of redundancy.

Early in the development of SNMP, it was also argued that the SNMP protocol needed to be kept very simple to allow for integration in a variety of network elements (because embedding a TCP stack in network devices was too difficult). But today's model has changed in that it is cost-effective and easier with respect to development time to embed a full processor into most network devices. These processors, often times with the capacity of computers just one generation old, can easily handle the nominal load of the erasure coding process.

SNMP relies on a polling mechanism to distribute most network management information. It also provides for an asynchronous "trap" mechanism to alert a management station of an event that requires urgent attention, or of an event that occurs infrequently enough to not warrant regular polling. With the use of an erasure code, the polling mechanism can be dispensed with, and each device can simply transmit a steady stream of periodically requested management information to the SNMP manager without having to be polled specifically for it. Polling, instead, can be reserved for infrequently needed information. For example, much information carried by SNMP packets is in the form of data tables that need to be communicated to management stations, which require more than one IP packet to convey it, again suggesting the benefit of using an erasure code.

4.3 Application layer protocol for high-bandwidth networks

Optical network test-beds have demonstrated the feasibility of providing multi-gigabit circuits between users of the network that have direct access to optical circuit services over a wide-area optical network. In such networks, the latency of connection setup is often much longer than the

actual transfer of the materials due to the high bit-rates available. In response to the huge bandwidth that optical networks afford, recent studies have examined the limits of TCP on high-speed links. While there is a surplus of network bandwidth and client-side computational ability, these studies have found that the operating system's TCP stack is being stressed and that network interfaces cannot keep up with high-speed network connections, and often drop packets simply because of buffer space limitations and internal system bus limits, which then severely affect the performance of TCP and limit the effective throughput. Since an erasure code is resilient to packet drops, we can further forward the prospect of applying erasure codes to overcome these bottlenecks.

Proposals such as Virtual Interface[5] have been forwarded to allow applications to bypass the operating system to access network communication. By incorporating an erasure coding layer into the application that uses such a direct interface, a transport layer serving the needs of certain applications that need fast and bursty access to network interfaces can be provided that fits a niche between an unreliable datagram service and a reliable flow-controlled connection-oriented service. The application has direct control over the reliability of the messages that it sends by adjusting the amount of redundancy it sends with each message transmission.

Packet ordering, often referred to as flow corruption in the context of optical networks, is an issue present in optical networks that use electronic routers between multi-gigabit connections, as well as in all-optical networks because of wavelength conversion, multi-path routing, deflection routing, and protocols such as optical burst switching [He01]. Here again, erasure codes appear to be a plausible means to overcome this issue for small to moderate size messages, because the packets need to be sorted upon receipt anyway, rendering moot the issue of reordered packets.

[5] See http://www.viarch.org/

5 Summary

This work explored a data network scenario where multiple paths between source and destination are available. Such networks are emerging as interconnections of heterogeneous networks. By applying an erasure code to the message that we want to send, we can use the path diversity to achieve reliable transmission of messages without simply repeating the message across all the paths. The erasure code allows reconstruction of the message with only a subset of the original coded message.

We presented an example of an Erasure Correcting Code, known as a Reed-Solomon Erasure Code. This code has enjoyed success in research implementations, and is straightforward to implement and embed in network application software. The encoding and decoding process can be explained as a linear-algebraic procedure in which all arithmetic is carried out using finite-field operations. (Recently, [Byer99] proposed an alternative coding method for similar purposes.)

Since each of the diverse paths fails independently of the others, and since we have a large number of paths, we can invoke the Central Limit Theorem to analyze the system. We derived a procedure to optimize the number of bits (or frames or packets) to send over each path in order to maximize the probability of receiving enough bits at the destination to reconstruct the message using the erasure correcting code. We examined three cases: (1) where the paths are very reliable, (2) where the paths are very unreliable, and (3) where the paths have an approximately 0.5 probability of failure.

Finally, we discussed potential applications of erasure codes in networks where only a single path is used or available. We proposed the use of erasure coding for short Web sessions to reduce the overall latency experienced by the user, by eliminating protocol overheads and handshakes. We further discussed the applicability of erasure codes in Transport Layers for the delivery of critical messages like network management information. Lastly, we briefly discussed the application of erasure codes in high-bandwidth circuit-oriented networks to improve the efficiency of network resources.

6 Appendix A

```
#! /usr/bin/python

import random
import time
time1 = time.time()

GFPOWER = 8
GFNUMSYMBOLS = (1<<GFPOWER)

NUMPKTS = 16
PKTSIZE = 4

gflog = {}; gfexp = {}; gfinv = {}

def gfgen():
      #Octal representation of primitive polynomial
      prim_poly = {}
      prim_poly[4] = 023
      prim_poly[8] = 0435
      prim_poly[16] = 0210013

      #Generate the powers of the GF primitive element alpha,
      #and build a table for the field logarithm
      bits = 1
      for exp in range(GFNUMSYMBOLS):
            gfexp[exp] = bits
            gflog[bits] = exp
            bits = bits<<1
            if (bits & GFNUMSYMBOLS):
                  bits = bits ^ prim_poly[GFPOWER]

      #Generate table for GF muliplicative inverse
      for sym in range(1, GFNUMSYMBOLS):
            invlog = GFNUMSYMBOLS - 1 - gflog[sym]
            gfinv[sym] = gfexp[invlog]


def gftest():
      keys = gfexp.keys(); keys.sort()
      print "Powers of alpha"
      for key in keys: print key, gfexp[key]

      print
      keys = gflog.keys(); keys.sort()
      print "Element logarithms"
      for key in keys: print key, gflog[key]

      #Make sure that the exponents and logs were
      #generated correctly.
      for sym in range(1, GFNUMSYMBOLS):
            if (sym != gfexp[gflog[sym]]):
                  print "error with symbol", sym


def gfmult(a, b):
      if (a==0 or b==0): return 0
      logsum = gflog[a] + gflog[b]
      if (logsum >= (GFNUMSYMBOLS - 1)):
            logsum = logsum - (GFNUMSYMBOLS - 1)
      return gfexp[logsum]
```

```
def gfmod(a):
      #There are slicker ways to do this,
      #but we don't care about efficiency here,
      #so we'll leave it as a placeholder
      return a % (GFNUMSYMBOLS - 1)

def RseEncode(packets, index):

      codedpkt = []
      #initialize coded packet to zero
      for j in range(PKTSIZE): codedpkt.append(0)

      for i in range(NUMPKTS):
            vme = gfexp[gfmod(i*index)]
            for j in range(PKTSIZE):
                  codedpkt[j] = codedpkt[j] ^ gfmult(vme, packets[i][j])

      return codedpkt

#---Main---
gfgen()
#gftest()

#Create some test packets
sourcepkts = []
for i in range(NUMPKTS):
      packet = []
      for j in range(PKTSIZE):
            dummy = random.choice(range(GFNUMSYMBOLS))
            dummy = i * PKTSIZE + j
            packet.append(dummy)
      sourcepkts.append(packet)

for index in range(NUMPKTS):
      print index, sourcepkts[index]

print

#Encode the packets
codedpkts = []
codedpktindex = []
for i in range(NUMPKTS):
      codedpktindex.append(i+NUMPKTS)
      packet = RseEncode(sourcepkts, i)
      codedpkts.append(packet)

for i in range(NUMPKTS):
      print codedpktindex[i], codedpkts[i]
```

7 Appendix B

```
clear all

%rand('state',0);

nsimexp = 6;        %Log of number of simulation runs
nsim = 10^nsimexp;  %Number of simulation trials

scenario = 'unreliable';

%Triggers for second-level optimizations
unreliable_optimization_phase2 = 0;
half_optimization_phase2 = 0;

switch scenario
     case 'reliable'
          L = 100;            %Number of paths
          N = 10*L;           %Total number of packets
          step = N/L;         %Step size for integration
                              %(Be wary of quantization effects!)
                              %Make sure N/L is an integer!

          pmin = 0.08;        %Min probability of failure per path
          pmax = 0.18;        %Max probability of failure per path
          p = rand(L,1);
          p = p*(pmax-pmin)+pmin;  %Random probability of failure
                                   %from pmin to pmax
          mark1 = 0.0;
          mark2 = 1.5;

          fig1axis = [800 N 0 0.02];
          fig1_legend_position = 1;
          x1_text = 810;
          y1_diff = 0.002;
          y1_text = 0.02 - y1_diff;

          fig2axis = [800 N 0 0.20];
          x2_text = 810;
          y2_text = 0.20 - 0.02;

          fig3axis = [720 860 0.000001 0.1];
          x3_text = 790;

          y3_text(1) = 10^(-1.25);
          y3_text(2) = 10^(-1.50);
          y3_text(3) = 10^(-1.75);

          sv = p;
          fig6title = 'Variation in value of c=N_ip_i';

     case 'unreliable'
          L = 300;            %Number of paths
          N = 10*L;           %Total number of packets
          step = N/L;         %Step size for integration
                              %Make sure N/L is an integer!
```

```
        qmin = 0.03;        %Min probability of success per path
        qmax = 0.15;        %Max probability of success per path
        q = rand(L,1);      %Random (bounded) probability of
                            %success per path
        q = q*(qmax-qmin)+qmin;  %Random (bounded) probability of
                                 %success per path
        q = sort(q);
        p = 1-q;  %Random probability of failure (from 1 to 1-qmax)
        pmin = 1-qmax;
        pmax = 1-qmin;
        mark1 = 0.0;
        mark2 = 1.5;

        if (L==100)
             fig1axis = [0 200 0 0.02];
             fig1_legend_position = 1;
             x1_text = 10;
             y1_diff = 0.001;
             y1_text = 0.02 - y1_diff;

             fig2axis = [0 200 0 0.2];
             x2_text = 10;
             y2_text = 0.2 - 0.02;

             fig3axis = [0 70 0.0001 0.1];
             x3_text = 1;

             y3_text(1) = 10^(-0.30);
             y3_text(2) = 10^(-0.50);
             y3_text(3) = 10^(-0.70);
        elseif (L==300)
             fig1axis = [0 600 0 0.01];
             fig1_legend_position = 1;
             x1_text = 10;
             y1_diff = 0.0005;
             y1_text = 0.01 - y1_diff;

             fig2axis = [0 600 0 0.10];
             x2_text = 10;
             y2_text = 0.10 - 0.01;

             fig3axis = [0 250 0.000001 0.1];
             x3_text = 10;

             y3_text(1) = 10^(-1.30);
             y3_text(2) = 10^(-1.50);
             y3_text(3) = 10^(-1.70);
        end

        sv = q;
        fig6title = 'Variation in value of c=N_iq_i';

    case 'half'
        L = 100;
        N = 1000;
        step = N/L;
```

```
            nudgemax = 0.2;          %Max probability of failure per path
            nudge = (rand(L,1)*nudgemax*2)-(nudgemax);
            p = 0.5 + nudge;              %Random probability of failure
            pmin = 0.5-nudgemax;     %Min probability of failure per path
            pmax = 0.5+nudgemax;     %Max probability of failure per path


            mark1 = 0.0;
            mark2 = 15.0;

            fig1axis = [300 700 0 0.01];
            fig1_legend_position = 1;
            x1_text = 310;
            y1_diff = 0.0005;
            y1_text = 0.01 - y1_diff;

            fig2axis = [300 700 0 0.10];
            x2_text = 310;
            y2_text = 0.10 - 0.01;

            fig3axis = [300 500 0.0001 0.1];
            x3_text = 400;

            y3_text(1) = 10^(-2.30);
            y3_text(2) = 10^(-2.50);
            y3_text(3) = 10^(-2.70);


            sv = p;
            fig6title = 'Variation in value of c=N_ip_i';

end

p = sort(p);   %Sort to help visualization
sv = sort(sv);

%plotting parameters (convenient to put them here for tweaking)

for i=1:6
     figure(i)
     clf
end

%Assign packets to each path

%Baseline case: Same number of packets per path
Ni_eq = ones(L,1)*N/L;

%Optimum Ni (non-integral) "raw" for reference
c_exact = N/sum(1./sv);
Ni_raw = c_exact./sv;

%Now choose Ni (integral) so that (N_i)(p_i) is constant
flag = 0;
for i = 1:10
     if (flag==1)
```

```
          break
     end
     incr = (mark2-mark1)/10.0;
     for c = mark1:incr:mark2
          Ni_opt = c./sv;
          Ni_opt = round(Ni_opt);
          checksum = sum(Ni_opt);

          s = sprintf('(%d,%d)  c=%f:  %d', i, flag, c, checksum);
          disp(s)

          if (checksum == N)
               flag = 1; %Found a match
               break
          end

          %Otherwise, keep subdividing
          if (checksum < N)
               mark1 = c;
          else
               mark2 = c;
               break
          end
     end
end
c_approx = c;

if (0)
     Ni_opt2 = Ni_raw;
     for i=1:20
          mu = sum((1-p).*Ni_opt2);
          sig = sum(p.*(1-p).*(Ni_opt2).^2);
          K = 250;
          theta = (mu-K)/sig;
          Ni_opt2 = N/L - 4*nudge./theta;
          val(i) = sum(Ni_opt2);
     end
     val'
end


disp('Sanity Checks:');
s=sprintf('constants: c_exact=%f  c_approx=%f', c_exact, c_approx);
disp(s)

%-------------------------------
%Plot the Gaussian approximation
%M is the number of packets received at destination
figure(1)

g_mean_eq = sum((1-p).*Ni_eq);
g_var_eq = sum(p.*(1-p).*(Ni_eq).^2);
mytext = sprintf('g_mean_eq: %f  g_var_eq: %f', g_mean_eq, g_var_eq);
disp(mytext)

g_mean_raw = sum((1-p).*Ni_raw);
g_var_raw = sum(p.*(1-p).*(Ni_raw).^2);
```

```
mytext = sprintf('g_mean_raw: %f  g_var_raw: %f', g_mean_raw,
g_var_raw);
disp(mytext)

g_mean_opt = sum((1-p).*Ni_opt);
g_var_opt = sum(p.*(1-p).*(Ni_opt).^2);
mytext = sprintf('g_mean_opt: %f  g_var_opt: %f', g_mean_opt,
g_var_opt);
disp(mytext)

%possible refinement of the optimization for unreliable case
if (unreliable_optimization_phase2)
     disp('refining Ni_opt, phase2');
     sum(Ni_opt)
     theta = 6/sqrt(g_var_opt);
     s1 = sum(1./sv);
     s2 = sv.*s1;
     Ni_optfix = (N*theta - L)./(s2) + 1/theta;
     optfixsum = sum(Ni_optfix)
     Ni_optfix = ((Ni_optfix/optfixsum)*N);
     optsum = sum(Ni_optfix)
     Ni_opt = Ni_optfix;

     g_mean_opt = sum((1-p).*Ni_opt);
     g_var_opt = sum(p.*(1-p).*(Ni_opt).^2);
     mytext = sprintf('g_mean_opt: %f  g_var_opt: %f', g_mean_opt,
g_var_opt);
     disp(mytext)
end

%possible refinement of the optimization for 50% reliability case
if (half_optimization_phase2)
     disp('refining Ni_opt');
     sum(Ni_opt)
     qqq_theta = 6/sqrt(g_var_opt)
     Ni_optfix = ( (N/L)-4*nudge/qqq_theta );
     qqq_optfixsum = sum(Ni_optfix)
     Ni_optfix = round((Ni_optfix/qqq_optfixsum)*N);
     qqq_optsum = sum(Ni_optfix)
     Ni_opt = Ni_optfix;

     g_mean_opt = sum((1-p).*Ni_opt);
     g_var_opt = sum(p.*(1-p).*(Ni_opt).^2);
     mytext = sprintf('g_mean_opt: %f  g_var_opt: %f', g_mean_opt,
g_var_opt);
     disp(mytext)
end



M = 1:N;
gauss_eq = exp( -(M-g_mean_eq).^2 /(2*g_var_eq)) / sqrt(2*pi*g_var_eq);
gauss_raw = exp( -(M-g_mean_raw).^2 /(2*g_var_raw)) /
sqrt(2*pi*g_var_raw);
gauss_opt = exp( -(M-g_mean_opt).^2 /(2*g_var_opt)) /
sqrt(2*pi*g_var_opt);
```

```
plot(M,gauss_eq,'k--', M,gauss_opt,'k-')
title('Gaussian Approximations');
xlabel('Number of Received Bits (M)')
ylabel('Probability')
axis(fig1axis);

mytext = sprintf('Equal N_i:  \\mu=%.1f, \\sigma=%.1f', g_mean_eq,
sqrt(g_var_eq));
text(x1_text, y1_text, mytext)
mytext = sprintf('Optimized N_i:  \\mu=%.1f, \\sigma=%.1f', g_mean_opt,
sqrt(g_var_opt));
text(x1_text, y1_text-y1_diff, mytext)
mytext = sprintf('%4.2f < p_i < %4.2f', pmin, pmax);
text(x1_text, y1_text-2*y1_diff, mytext)
legend('equal N_i','optimized N_i',fig1_legend_position);


%-----------------
%Random Simulation
M_eq = zeros(nsim,1);
M_opt = zeros(nsim,1);
for n = 1:nsim
     linkfailures = rand(L,1);
     sim = linkfailures > p;

     M_eq(n) = sum(sim .* Ni_eq);
     M_opt(n) = sum(sim .* Ni_opt);
end

binedges = 0:step:N;

sim_mean_eq = mean(M_eq);
hights_eq = histc(M_eq, binedges);
normalized_eq = hights_eq / nsim;

sim_mean_opt = mean(M_opt);
hights_opt = histc(M_opt, binedges);
normalized_opt = hights_opt / nsim;

figure(2)

subplot(2,1,1)
bar(binedges, normalized_eq, 'k')
axis(fig2axis);
mytitle = sprintf('Simulation with equal N_i (N_i=N/L) (10^%d
trials)',nsimexp);
title(mytitle);
xlabel('Number of Received Bits (M)')
ylabel('Probability')
fmtstring = 'N=%d, L=%d, %.2f < p_i < %.2f, \\mu=%4.1f';
simstats_eq = sprintf(fmtstring, N, L, pmin, pmax, sim_mean_eq);
text(x2_text, y2_text, simstats_eq);

subplot(2,1,2)
bar(binedges, normalized_opt, 'k')
axis(fig2axis);
```

```
mytitle = sprintf('Simulation with optimized N_i (10^%d
trials)',nsimexp);
title(mytitle);
xlabel('Number of Received Bits (M)')
ylabel('Probability')
%fmtstring = 'N=%d, L=%d, min p_i=%.2f, max p_i=%.2f, mean(M)=%4.1f';
simstats_opt = sprintf(fmtstring, N, L, pmin, pmax, sim_mean_opt);
text(x1_text, y2_text, simstats_opt);

%---------------------
%Plots of error bounds
nbins = length(binedges);
cumsum_eq = zeros(nbins,1);
cumsum_opt = zeros(nbins,1);
cumsum_x = 1:nbins;
for j = 1:nbins
     cumsum_eq(j) = sum(normalized_eq(1:j));
     cumsum_opt(j) = sum(normalized_opt(1:j));
end

k = 1:N;
bound_eq = (1/2)*exp(-(g_mean_eq - k).^2/(2*g_var_eq));
bound_raw = (1/2)*exp(-(g_mean_raw - k).^2/(2*g_var_raw));
bound_opt = (1/2)*exp(-(g_mean_opt - k).^2/(2*g_var_opt));

t = binedges;
t = t(2:end);
t = [t t(end)];
t = t';

figure(3)
semilogy(k, bound_eq,'k:', t, cumsum_eq, 'ko:', k, bound_opt, 'k--', t,
cumsum_opt, 'ko--', k,bound_raw,'k-' );
axis(fig3axis);

if 0
p = polyfit(t,cumsum_opt,2)
f = polyval(p,k');
hold on
semilogy(k',f)
end

mytitle = sprintf('Bound and Cumulative Distribution of M (10^%d
trials)',nsimexp);
title(mytitle);
xlabel('Bits (n)');
ylabel('Probability M<n');

fmtstring = 'N=%d, L=%d, %4.2f < p_i < %4.2f';
commontext = sprintf(fmtstring, N, L, pmin, pmax);
simstats_eq = sprintf('Equal N_i: \\mu=%4.1f', sim_mean_eq);
simstats_opt = sprintf('Optimized N_i: \\mu=%4.1f', sim_mean_opt);
text(x3_text, y3_text(1), commontext);
text(x3_text, y3_text(2), simstats_eq);
text(x3_text, y3_text(3), simstats_opt);

stuff(1)={'bound for equal N_i'};
```

```
stuff(2)={'cumul dist of M for equal N_i'};
stuff(3)={'bound for opt integer N_i'};
stuff(4)={'cumul dist of M for opt N_i'};
stuff(5)={'bound for opt non-integer N_i'};
legend(stuff, 4);

%--------------------------------------
%Various auxiliary plots

y = Ni_eq .* sv;
y_mean = mean(y);
y_std = std(y);
checksum = sum(Ni_eq);
fmt = 'Stats on (Ni_eq)(p_i): mean=%.2f  std=%.3f  checksum=%.2f';
s = sprintf(fmt, y_mean, y_std, checksum);
disp(s)

y = Ni_opt .* sv;
y_mean = mean(y);
y_std = std(y);
checksum = sum(Ni_opt);
fmt = 'Stats on (Ni_opt)(p_i): mean=%.2f  std=%.3f  checksum=%.2f';
s = sprintf(fmt, y_mean, y_std, checksum);
disp(s)


figure(4)
plot(p,'k.-')
grid on
title('Probabilities of path failure (sorted)')
xlabel('Path Number')
ylabel('Probability of path failure (p_i)')

figure(5)
plot(Ni_opt,'k.-')
grid on
title('Optimum N_i (rounded) for each path (sorted)')
xlabel('Path Number')
ylabel('Optimal number of bits (N_i)')

if (strcmp(scenario,'half'))
     figure(6)
     x = sv-0.5;
     y = Ni_opt;
     theta = 6/sqrt(g_var_opt);
     tau = 3;
     ni = N/L - tau.*x./theta;
     plot(x,y,'k.-',x,ni,'k--');
     grid off
     fig6title = sprintf('Variation in N_i with respect to
\\epsilon_i');
     foostring = sprintf('using exact non-integer optimum N_i
(\\tau=%d)', tau);
     legend('using rounded optimum N_i', foostring);
     title(fig6title);
     xlabel('\epsilon_i');
     ylabel('N_i');
```

```
     text(-0.025,11,'N_i=N/L - 4\epsilon_i/\theta');
else
     figure(6)
     xpoints = [1:L];
     konst1 = Ni_opt .* sv;
     konst2 = Ni_eq .* sv;
     cvector = ones(L,1) * c_approx;
     plot(xpoints, konst1,'k.:', xpoints,cvector,'k-');
     legend('using rounded optimum N_i', 'using exact non-integer
optimum N_i');
     grid off
     title(fig6title)
     xlabel('Path Number')
     ylabel('Value of c')
end

%Dump all the figures to EPS files, sized at 2 per page
if (1)
cd('C:\Documents and Settings\salil\Desktop\matlab_plots');
for i=1:6
     figure(i)
     plotname = sprintf('%s_fig%d_%d.eps',scenario,i,L);

     set(gcf,'PaperPosition',[0,0,6.5,4.25]);
     if (i==200)
          set(gcf,'PaperPosition',[0,0,6.5,6.0]);
     end


     print('-deps', plotname)
end
end


disp('Done!')
disp(' ')

return
```

8 References

[Aya93] E. Ayanoglu, R.D. Gitlin, and N.C. Oguz. "Performance improvement in broadband networks using forward error correction for lost packet recovery." J. High Speed Networks, vol 2, 1993.

[Bak2K] Fred Baker. "Interactions between Quality of Service and Routing" A talk given at MIT on May 11, 2000.

[Bala95] Hari Balakrishnan, Srinivasan Seshan, and Randy Katz. "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks." ACM Wireless Networks, December 1995.

[Ben99] J.C.R. Bennett, C. Partridge, and N. Shectman. "Packet reordering is not pathological network behavior." IEEE/ACM Transactions on Networking, December 1999.

[BG87] Dimitri Bertsekas and Robert Gallager. Data Networks. Prentice Hall, Inc, 1987.

[Byer99] Byers, J.W.; Luby, M.; Mitzenmacher, M. "Accessing multiple mirror sites in parallel: using Tornado codes to speed up downloads" INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , Volume: 1 , 1999 Page(s): 275 -283 vol.1

[Byer98] Byers, J.W.; Luby, M.; Mitzenmacher, M.; Rege, Ashutosh. "A Digital Fountain Approach to Reliable Distribution of Bulk Data." Proceedings of ACM Sigcomm '98, Vancouver, Canada, September 1998.

[Chan97] Agnes Hui Chan and Vincent W.S. Chan. "Reliable Message Delivery Via Unreliable Networks." IEEE International Symposium on Information Theory, 1997. Ulm, Germany.

[Chan01] Professor Vincent W. S. Chan. Private communications and discussions during thesis research. MIT, Feb-May 2001.

[Clar2K] David Clark and Marjory Blumenthal. "Rethinking the design of the Internet: The end-to-end argument vs. the brave new world." August 2000. Available from the authors.

[Gibb2K] Terry Gibbons, USC Information Sciences Institute. Private communication. November 18, 2000.

[He01] Jenny He and Dimitra Simeonidou. "Flow routing and its performace analysis in optical IP networks." (To appear in Photonic Network Communications, Vol.3, No.1/2, January~June 2001, pp. 49-62.)

[Huit96] C. Huitema. "The case for packet level FEC." Proc IFIP 5th Int. Workshop on Protocols for High Speed Networks. Sophia Antipolis, France, Oct. 1996.

[Huit95] C. Huitema. Routing in the Internet. Prentice Hall, 1995.

[Kesh97] Keshav, S. An Engineering Approach to Computer Networking. Addison-Wesley, 1997.

[Lee94] Edward A. A. Lee, David G. Messerschmitt. Digital Communication. Kluwer Academic Publishers, January 1994

[Lin83] S. Lin and D.J. Costello. Error Correcting Coding: Fundamentals and Applications. Englewood Cliffs, NJ. Prentice Hall, 1983.

[Luby01] Luby, M.G.; Mitzenmacher, M.; Shokrollahi, M.A.; Spielman, D.A. Efficient erasure correcting codes. Information Theory, IEEE Transactions on , Volume: 47 Issue: 2 , February 2001. Page(s): 569 –584

[Lutz99] Mark Lutz and David Ascher. Learning Python. O'Reilly & Associates, 1999.

[Mac90] Anthony MacAuley. "Reliable broadband communication using a burst erasure correction code." Proc ACM SIGCOMM'90, September 1990.

[Max93] Maxemchuk, N.F. "Dispersity routing on ATM networks." INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE , 1993 Page(s): 347 -357 vol.1

[Niel97] Henrik Frystyk Nielsen, Jim Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Håkon Wium Lie, and Chris Lilley, "Network Performance Effects of HTTP/1.1, CSS1, and PNG", W3C, SIGCOMM'97.

[Non98] Jorg Nonnenmacher, Ernst Biersack, and Don Towsley. "Parity Based Loss Recovery for Reliable Multicast Transmission." IEEE/ACM Transactions on Networking, August 1998.

[Pad95] Venkata N. Padmanabhan, Jeffrey C. Mogul. "Improving HTTP Latency." Computer Networks and ISDN Systems, December 1995.

[Pla99] James Plank. "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems." University of Tennessee, Department of Computer Science. February 1999.

[NumRec92] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. Numerical Recipes in C. Cambridge University Press, 1992.

[Pur95] Purser, Michael. Introduction to Error-Correcting Codes. Artech House, 1995.

[Rab89] M.O. Rabin. "Efficient dispersal of information for security, load balancing, and fault-tolerance." Journal of the Association for Computing Machinery, April 1989.

[Riz97] Luigi Rizzo. "Effective erasure codes for reliable computer communication protocols." Comput Commun Rev. April 1997.

[Rose96] Rose, Marshall T. The Simple Book: An Introduction to Internet Management. Prentice Hall, 1996.

[Sch97] Schein, B.E., Bernstein, S.L. A Forward Error Control Scheme for GBS and BADD. MIT Lincoln Laboratory, Technical Report 1039. 22 July 1997.

[Serv01] Servetto, S.D. "Coding Problems in Communication Networks." Department of Communication Systems, Swiss Federal Institute of Technology. A talk given at MIT, 23 April 2001.

[Stall93] William Stallings. SNMP, SNMPv2, and CMIP. Addison-Wesley, 1993.

[Star94] Henry Stark, John Woods. Probability, Random Processes, and Estimation Theory for Engineers. (2ed.) Prentice Hall, 1994.

[Stev94] W. Richard Stevens. TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley, 1994.

[Stev96] W. Richard Stevens. TCP/IP Illustrated, Volume 3. Addison-Wesley, 1996.

[Tan96] Andrew S. Tanenbaum. Computer Networks. Prentice Hall, New Jersey. 1996.

[Wel99] Wells, Richard B. Applied Coding and Information Theory for Engineers. Prentice Hall, 1999.