On Accessing Multiple Mirror Servers in Parallel

By

Yousry Salaheldin Abdel-Hamid
B.Sc, Ain Shams University, Cairo, 1987

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

University of Victoria

Supervisor: Professor T.A. Gulliver

ABSTRACT

In this thesis, an extensive simulation study is done to gauge the performance of parallel access to multiple mirror sites on the Internet. The study is based on the Digital Fountain approach designed by J. Byers *et al.* in which Tornado codes are used to minimize decoding time at the expense of injecting extra packets into the system. In this study, both Reed-Solomon and Tornado codes are considered. The results indicate that randomly permuting the packets at every mirror site is not the optimal solution. A new technique, which staggers the packets, is shown to be superior. This study employs **OPNET** Modeler, which is a powerful event driven simulation tool. Simulations results show that using Reed-Solomon codes with staggered packet transmission provides dramatically improved system performance.

Table of Contents

List of Figures

List of Tables

Dedication

To my beloved mother,

whose prayers and supplications always reach me, wherever I am.

Acknowledgements

First, and most importantly, I prostrate to thank God the almighty for his blessings and knowledge.

"They said, Exalted are You; we have no knowledge except what you have taught us. Indeed, it is You who is the Knowing, the Wise" Quran 1:32.

This is the best opportunity to express my deepest gratitude to all who have helped me, contributed to this thesis and wished me success.

Next, I would like to thank Professor Aaron Gulliver for his trust in my potential and abilities to perform this work in its best form. Without his immense knowledge and incomparable friendliness, I would have been crushed under the load of graduate studies and would have never achieved this success. He created an amazing atmosphere of pleasurable work and exceptional innovation while demonstrating an exceptional skill in supervising students distantly.

I would like to express my recognition to Mr. Mohamed Kamel at the American University in Cairo, who spent many hours solving problems related to my work. I must also express my sincere appreciation to the technical support staff at OPNET Technologies Inc. Their support was invaluable to understand the sophisticated tool, develop smooth simulations and produce accurate results.

I would like to express my deepest thanks to Dr. Watheq El-Kharashi for his great support. I would also like to thank Professor Fayez Gebali, Dr. Wayne Beckham and Mr. Erik Laxdal, for working me through many difficulties.

I would like to mention many close friends. These friends include Mr. K. Sami, Dr. M.H. Bakr, Mr. H. Elakany, Mr. J. Foxgord, Dr. Z. Blazek, Mr. M. Yassein, Mr. M. Fayed, Mr. Y. Hamad, and many others that I am sure I will regret not having them included here.

Finally, I would like to acknowledge my mother, brothers Amir and Ossama, my sister-in-law Seham, my nephew Omar and my niece Mariam for their enormous moral support, which traveled halfway around the globe to reach me. Thanks to all.

Yousry Abdel-Hamid, October 2003

1. Introduction

1.1 Packet Level Forward Error Control

1.1.1 Overview

Forward Error Correction (FEC) is achieved by adding a number of redundant packets R to a K-packet message such that the original message can be retrieved from a subset of the N packets. Researchers often focus on the effectiveness of different FEC techniques with regards to the speed at which data is encoded and decoded (recovered), and the amount of redundancy required.

The complexity of different layer protocols, network synchronization, round trip times, characteristics of packet loss, connection setup times and delays are very important issues to be considered. Adding redundancy to a bandwidth-starved network can heavily impact the quality of service and network performance in general, thus appropriate FEC must be used.

FEC mainly anticipates some amount of losses, and obviates this by sending redundant data as a compensation that allows the receiver to reconstruct up to a certain number of missing packets. Inevitably, the process includes an *encoding* phase at the transmitter, where redundant packets are constructed from the source data and a *decoding* phase at the receiver, where source data is extracted and the message is retrieved intact, if possible, from the subset of packets received.

FEC cannot guarantee an absolutely reliable and flawless delivery unless the number of redundant packets is infinity. However, with proper FEC the packet loss ratio at the receiver can be made arbitrarily small. In contrast, ARQ is based on the retransmission of missing packets, and is triggered by the ACK/NACK (positive/negative acknowledgements) or timeout techniques. This requires a highly precise, error free feedback channel accompanied by large buffer capacities to maintain the robustness of the protocol. Thus, FEC is considered here to reduce communication overhead between the sender and receiver without the need of a feedback channel.

In computer networks, data is sent in the form of packets. The best example to demonstrate this is a JPEG file. The file is divided into packets of fixed size. Since

packets are indexed, erasures (lost packets) can be easily detected in the packet sequence. Consider a large file is transmitted over a narrowband link on the Internet. Due to congestion, buffers fill up and packets are dropped until eventually the congestion is cleared and the flow resumes. Hence, losses on the Internet are mostly uncorrelated and unpredictable. However, even burst losses, which are considered the worst case on the Internet, are insensitive or nearly independent of the actual content of each packet. Moreover, even if packets are grouped in data blocks, they may also be corrupted. But, considering the recovery of data with no retransmissions using (FEC), any corrupted data block is considered dropped or missing during transmission and is deemed absent or totally invalid [1].

Bit level error correction operates on bit sequences. In the case of block codes, a set of k input bits is appended with r redundancy bits forming the codeword of length $n = k + r$. When a codeword is received, the redundancy is used to determine the most likely error pattern by the decoder.

In computer communications, error detection is a very dominant issue, it mostly takes place in the lower protocol layers (Data link Layer) which use checksums and syndrome detection using parity and generator polynomials e.g. CRC (Cycle Redundancy Checks), to discard corrupted packets. Corruption usually takes place in the physical layer, which can be a wired bottleneck such as a modem link or a congested buffer, or it can be a noisy wireless channel [2], [3].

After processing data in the Data Link Layer, the upper layers of the hierarchy have to deal with *erasures,* i.e. missing packets in the stream of data. Erasures originate from uncorrectable errors at the link layer, congestions in the channels, bottlenecks and more frequently from lack of bandwidth, which all cause otherwise valid packets to be dropped. This often happens due to a lack of buffer space and may be rarely due to collisions. Erasures are easier to deal with than errors since the exact position of the missing data is known. However, despite the robustness of erasure codes, the issue of choosing an error correcting code is still a function of the application, e.g. wireless mobile telecommunications may need error correction while erasure codes are of great

interest in computer networks which deal mainly with packet-based communications. Moreover, packet tagging and frame definitions make error detection an easy task.

The key idea behind erasure codes is the same as error correcting codes. At the transmitter, the *source data* consisting of K packets are encoded with R redundant packets to produce the encoded N-packet message. Ideally, once enough packets arrive at the receiver, losses in a group of N-encoded packets can be recovered or *decoded.* In computer communications, a packet may represent hundreds of bytes of data. In our study, we will consider a fixed size packet of 1024 bytes. The level of redundancy can be tuned as a function of the network characteristics.

1.1.2 Packet Encoding Methodology

Consider an unencoded 1Mbyte (1024x1024 = 2^{20} bytes) file. In Figure 1.1 the K- packet file is encoded with N - K redundant packets to form the N-packet file.

Stretching the k-symbol codeword performs the encoding process; every packet contains a unique symbol of each codeword block. If $n = k + r = ck$, hence N = K + R = c K. and K/k = N/n. where c is termed the *stretch factor.*

If K = 2^{20} symbols, each packet contains 2^{10} symbols, hence N = $2^{20}c$ / 2^{10} = 2^{10} c packets. Therefore, packet level FEC is a process of encoding the individual codeword blocks with parity (redundant) symbols by adding parity packets to the original file as Figure 1.1 shows. Note that 0 symbols are used to complete the last k-byte block if necessary.

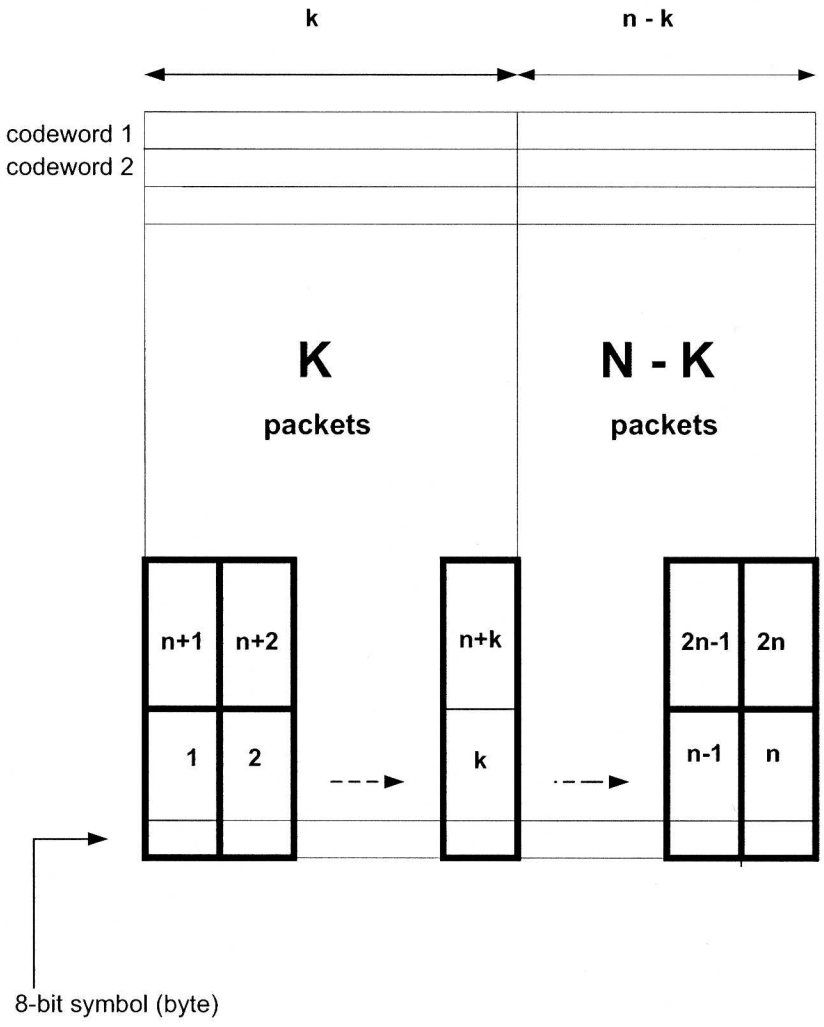


Figure 1.1 Encoded file

1.2 Reed-Solomon Codes

1.2.1 Overview

Reed-Solomon codes are block error correcting codes with a wide range of applications in digital communications and storage. The wide gap in physical properties and noise levels of the channels between wireless, satellite, digital television and computer communications has rendered Reed-Solomon codes to be the most widely employed codes in communication systems. They are efficiently used in high-speed modems such as ADSL and xDSL.

Reed-Solomon (RS) codes are a subset of BCH codes and are linear block codes. Usually a Reed-Solomon code is specified as RS (n, k) but with m-bit symbols. This means that the encoder takes k data symbols, each of size m-bits, and encodes them by appending n - k parity symbols also of m-bits each, to produce an n-symbol codeword. In other words, the encoder transforms k data symbols to an n-symbol codeword [4], [5]. A Reed-Solomon decoder can correct up to t symbols that contain errors in a codeword, where $2t = n - k$. A t-error correcting RS code with symbols from GF (2^m) has a nominal block length n of 2^m-1, which is a factor of $X^n - 1$. A typical Reed-Solomon codeword is shown in Fig. 1.2.

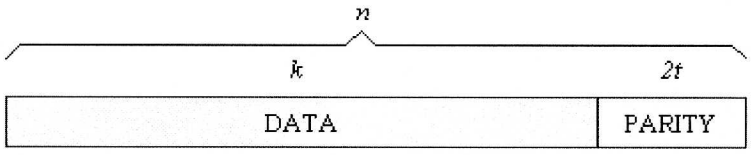


Figure 1.2 Systematic Reed-Solomon codeword

When the data is left intact and the parity symbols are appended to the data, the code is termed systematic.

Reed-Solomon codes are optimal (maximum distance separable (MDS) codes), so the minimum distance is

$$d_{\min} = n - k + 1 \qquad (1.1)$$

An RS code can correct up to $t = (d_{\min} - 1) / 2$ symbol errors. In addition, RS codes are capable of handling a mixture of errors and erasures, and up to e errors and s erasures can be corrected where

$$2e + s \leq n - k \tag{1.2}$$

This means that with only erasures, $n - k$ symbols can be recovered if corrupted locations are already known at the decoder, which is the case in this thesis.

A popular RS code is RS (255,223) with 8-bit symbols. Each codeword contains 255 symbols, i.e. bytes, of which 223 are data and 32 are parity. Thus, we have:

$$n = 255,\ k = 223,\ m = 8$$

$$2t = 32,\ t = 16$$

The decoder can correct up to 16 symbol errors in the received codeword (i.e. 16 bytes). Errors can be randomly located anywhere in the codeword. An error occurs when one or more bits arrive in error. One symbol error occurs if 1 bit in a symbol is wrong or when all the bits in the symbol are wrong. Obviously, the worst case happens when bit errors occur each in a separate symbol (or byte if $m = 8$). In a best-case scenario, the bit errors occur in a burst in which case the decoder can correct a maximum of 16 x 8 bit errors. Hence, Reed-Solomon codes are well suited for burst error channels.

The amount of processing "power" required to encode and decode Reed-Solomon codes is a function of the number of parity symbols per codeword. A large value of t means that the number of errors that can be corrected is large and consequently, the probability of recovering the original codeword is relatively higher. However, this requires more computational power than for smaller values of t. Encoding is considerably faster than decoding, since it requires less computation [6].

A Reed-Solomon codeword is generated with a generator polynomial, and all valid codewords are divisible by this polynomial.

Multiplying the generator polynomial with the information polynomial i(x) generates the codeword

$$c(x) = g(x)i(x) \tag{1.3}$$

The $2t$ parity symbols in a systematic Reed-Solomon codeword are given by

$$p(x) = i(x) \,.\, X^{n-k} \text{ modulo } g(x) \tag{1.4}$$

where p(x) is the parity polynomial representing the parity symbols to be transmitted over the channel.

At the receiver, the received word r(x) is the original (transmitted) codeword plus the error polynomial

$$r(x) = c(x) + e(x) \qquad (1.5)$$

Syndrome polynomial calculation is a process similar to parity calculation. A Reed-Solomon code has a degree *2t* syndrome polynomial that depends only on the errors (not on the transmitted codeword). The coefficients of the syndrome polynomial can be calculated by substituting the *2t* roots of the generator polynomial g(x) into r(x), the syndromes are used to determine the error locations and the values of the errors.

Finding the symbol error locations involves solving simultaneous equations with *t* unknowns. Several fast algorithms are available to perform this. Mainly, two major steps take place;

1. Finding the error locator polynomial

   This can be done using the Berlekamp-Massey algorithm or Euclid's algorithm. Euclid's algorithm tends to be more popular for its easier implementation; however, the Berlekamp-Massey algorithm leads to more hardware and software efficiency

2. Finding the roots of the locator polynomial

   This can be done using the Chien search algorithm (exhaustive search) afterwards the error value at each error location is recovered.

1.2.2 Reed-Solomon Decoders

Progress in VLSI hardware technology has dramatically improved the design of Reed-Solomon encoders and decoders. FPGA (Field Programmable Logic Arrays) and ASIC (Application Specific Integrated Circuits) devices can hold multiple modules on the same chip, allowing very efficient logic cores capable of handling the complex encoding and decoding calculations short time. With ever increasing processor speeds, a software implementation can operate at a data rate of 1 Mbps or more for low redundancy codes. We will give some examples of "off the shelf" integrated circuits that encode and decode

Reed-Solomon codes in which the number of correctable errors or erasures can be programmed to handle system requirements.

The nominal Reed-Solomon codeword length is $n = 2^m - 1$. A Reed-Solomon code can correct up to t symbol errors or $2t$ erasures, where $2t = r = n - k$. Of course, the higher the redundancy i.e., the value of t, the larger the number of errors or erasures that can be corrected. However, the required computational power and hence decoding time, increases as t get larger. Because encoding is considerably faster than decoding, our main concern in studying the performance of the system will be focused on the decoding speed. Until recently, software implementations of Reed-Solomon codes (even with small values of t) required too much computational power for "real time" decoding requirements where speed is an essential requirement.

In software, the major difficulty in implementing Reed-Solomon codes is that computer processors do not support Galois field arithmetic operations. For example, to implement a Galois field GF (2^m) multiplication in software can requires a test; two log table look-ups, modulo add and then an anti-log table look-up. However, with the dramatic progress of processor speed (computational power), several RS decoders have been developed that can operate at relatively high data rates. Some of the decoding benchmarks are shown in Table 1 for a 166MHz Pentium PC (considered a low speed processor), for various *stretch factors c,* where $c = n/k$.

Stretch factor c	Code	Data Rate
1.015	RS (255,251)	12 Mbps
1.066	RS (255,239)	2.7 Mbps
1.143	RS (255,223)	1.1 Mbps

Table 1.1 Software decoding benchmarks for RS codes using a 166 MHz Pentium processor. (Courtesy of 4i2i Communications Inc.)

In hardware, products from two manufacturers are presented as examples of off-the-shelf FPGAs [6], [7].

i. *4i2i* Communications Ltd. Soft IP® Reed-Solomon core decoder

This core can be designed using Verilog and VHDL and has been sold widely and is well tested in silicon. It is suitable for just about any application requiring block error correcting codes at data rates up to fiber optic level (Gb/s). The core can support full length and shortened RS codes, a shortened RS code can be done by putting data symbols zero at the encoder, not transmitting them, and then re-inserting them at the decoder with m-bit symbols, where ($m > 2$). A variety of customer options are provided which meet the requirements of most modern-day RS error correction/erasure applications. It is compatible with a wide range of International Standards including CCSDS (Consultative Committee for Space Data Systems), DECT (Digital Enhanced Cordless Telecommunications) and xDSL.

The number of required clock cycles depends on the particular code parameters (n, k, m) and primarily the number of errors/erasures being corrected. Two versions are available, A *sequential* decoder which processes one codeword at a time, and the extremely fast and flexible *pipelined* decoder which is capable in most cases of a rate of 1 symbol per clock pulse [6]. The decoder can also provide an erasure decoding capability that allows up to $2t$ erasures to be corrected. The timing numbers are derived from the formula: clocks = $2n + 2t^2 + 13t + 13$ (*sequential*) and max (n, $t^2 + 12t + 7$) (*pipelined*).

ii. Altera Reed-Solomon Megacore®

Another example of an RS decoder is that produced by the FPGA manufacturer Altera Corporation Inc. This core is available with the Altera® RS Compiler comprising the RS encoder and RS decoder Megacore functions, known as the Megacore Wizard-Plug [7].

The decoder core comes in three versions that are capable of erasure decoding. The first is the *discrete* decoder that can process one codeword at a time and must be reset between each codeword; the decoder receives and outputs the symbol on the rising and the falling edges of the synchronization clocks, respectively. The *Streaming* decoder has a pipeline depth of three codewords before it places the first corrected codeword at the output.

The *Continuous* decoder is similar to the streaming decoder having also a pipeline length of three codewords, however, the continuous decoder does not have idle gaps between codewords. The codewords enter and are placed at the output continuously.

The timing values are based on the performance formula of the erasure-enabled streaming decoder with half *keysize*. The *keysize* is the parameter that allows the trade-off of the amount of logic resources against the supported logic. The preference of using half keysize is mentioned in [7].

The formulae for the maximum number of clock cycles required to process each codeword is given in Table 2.

Keysize	Discrete	Streaming	Continuous
Half	$3n$ + 11check + 6	Max [(n + check), (10check + 6)]	n
Full	$3n$ + 9 check + 6	Max [(n + check), (8check + 6)]	n

Table 1.2 RS decoder performance formulae for RS Megacore® chip (Courtesy of Altera Corporation Inc.)

The Streaming decoder is chosen in our work, as it is the most suitable for typical file sizes and level of redundancy.

Table 3 gives some benchmark timing requirements calculated for a high performance (pipelined/streaming) RS decoder. Times are computed based on an unencoded 1MB (1024 packets or 4096 codewords) file with a symbol size m = 8 bits and a codeword length n = 255 symbols. The processor clock frequency; F_{MAX} = 100 MHz. The decoding times calculated are based on the performance formulae for the number of clock cycles to decode each codeword for an erasure-enabled hardware decoder. File total decoding times are computed accordingly.

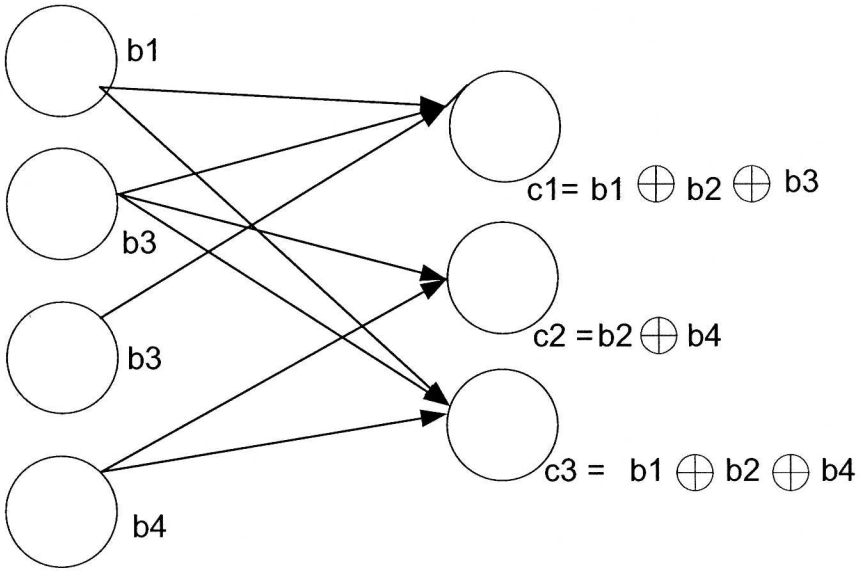
Stretch factor c	Codeword RS (n, k)	Decoding time (4i2i RS IP core) Pipelined ®	Decoding time (Altera Megacore) Erasure-enabled/Half keysize
1.00	RS (255,255)	10.49 ms	10.69 ms
1.02	RS (255,250)	10. 49 ms	10.90 ms
1.04	RS (255,245)	10.49 ms	10.98 ms
1.05	RS (255,243)	10.69 ms	11.39 ms
1.1	RS (255,233)	10.64 ms	11.34 ms
1.14	RS (255,223)	18.71 ms	13.41 ms
1.3	RS (255,196)	49.18 ms	24.51 ms
1.4	RS (255,182)	71.34 ms	30.26 ms
1.5	RS (255,170	93.55 ms	35.20 ms
2.0	RS (255,128)	194.55 ms	52.47 ms
3.0	RS (255,85)	339.31 ms	70.15 ms
4.0	RS (255,64)	418.31 ms	78.79 ms

Table 1.3 RS hardware decoding times of a 1MB file while varying the stretch factor

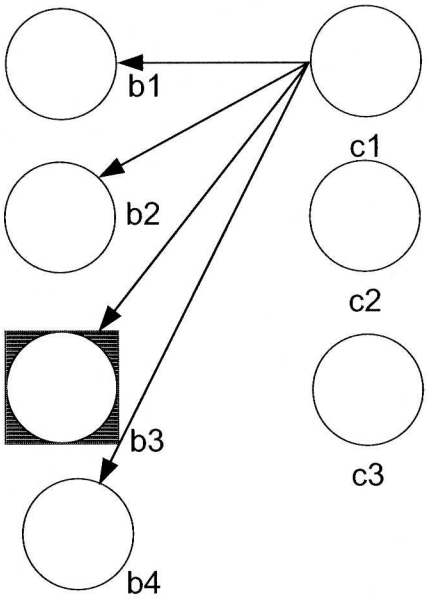
1.3 Tornado Codes

1.3.1 Overview

M. Luby *et al.* developed a new class of codes called Tornado Codes based on a simple XOR strategy [9]. The design of Tornado Codes requires the construction of a random irregular bipartite graph [8], [9]. The encoding and decoding algorithms of such codes are very similar, only requiring XOR operations, so that encoding and decoding are very fast. The encoding algorithm creates a set of redundant check bits. The XOR of a randomly chosen set of bipartite data bits produces the parity bits that are simply augmented to the original message data bits and sent over the channel [9].



Encoding



Decoding

Figure 1.3 Encoding and decoding of a 4-bit data with Tornado codes

Because each node represents a bit, a packet or a group of interleaved packets and is sequentially numbered, the position of the missing node is known. The decoding process is quite simple and straightforward. As Figure 1.4.a shows, the contents of each bit is XOR-ed with the contents of its preceding (left) neighbor in the graph generating the parity bits c1, c2 and c3. At the decoder having enough data and knowing the position of the lost bit, we should be able to reproduce the lost bits. Thus, these codes provide a reliable method for erasure decoding in data networks.

In perspective, the difference between RS and Tornado Codes lies in the number of bits or symbols the decoder has to acquire in order to be able to retrieve the original file. We consider in the next section a simple illustration where a block of data is sent over a typical network with a fraction p of the packets is lost [9].

1.3.2 Decoding of Reed-Solomon and Tornado Codes

Reed-Solomon (RS) codes provide ideal protection against data losses as pn of the transmitted symbols in each codeword can be lost, where $p = (n-k)/n$. The entire codeword can be recovered by receiving any $(1-p)n = k$ symbols from the codeword. In a systematic code, a message consisting of k symbols is appended with $pn = r$ parity symbols to form an n-symbol codeword. The RS code rate is defined by

$$R = \frac{\text{Number of original message symbols}}{\text{Total number of symbols sent}} \qquad (1.6)$$

$$R = (1-p)$$

Tornado codes are less efficient than RS codes and can be decoded only after receiving an extra ε packets, so that

$$(1-p)\varepsilon n, \quad \varepsilon > 1 \qquad (1.7)$$

packets are required for decoding a codeword. As a result, Tornado codes require the correct reception of more packets compared to Reed-Solomon codes, but their decoding

speed is faster. The price paid in network traffic will drastically lower computational requirements [10].

2. Internet Mirroring and Parallel Access

2.1 Motivation and Goals

The conventional downloading of any file where the receiver (client) retrieves data from a single source (server) is the typical scenario of accessing the Internet. From the time that the connection is established, the downloading process proceeds via a single client-server connection until the complete file is retrieved successfully. The efficiency of the download process is dependent on many factors. Most important of all is the Download Time. The time to download a file is influenced by the efficiency of the server to handle the load of the receivers, the link bandwidth, intermediate router's buffer sizes, fluctuations of traffic at different times of the day which may result in packet collisions at bottlenecks, and finally the receiver's physical link type e.g. Cable, DSL, Modems.

An interesting study by Schooler and Gemmel [11] in 1997 at Microsoft Research Division was the first to focus on download performance. The problem occurred when Microsoft Corp. released its first Internet Explorer v.3.0 browser. The response of the receivers was extraordinary; requests for download simply overwhelmed the servers in and around the University of Washington. This created such a highly congested area of network traffic that it was almost impossible to access any of the individual web servers at that time. Web site hits climbed to 15 times their normal statistics in addition to causing problems for overseas receivers in the WAN. This was characterized as "*The Midnight Madness Problem*" [11] due to the time when the files were made available for downloading. Later, when a security patch of 400 KB was released, 140,000 copies were demanded, resulting in 5.5 GB of data downloaded in a single day. Similarly, highly congested web traffic occurs regularly at popular events around the world. Two recent episodes were the NASA web site airing of the first landing on Mars in several forms (live video clips, text, huge picture files), and the Kasparov vs. Deep Blue chess rematch.

2.1.1 Multicasting

Multicasting is a widely used technique for reliable delivery of data on the Internet, especially streaming media and real time applications over lossy channels where packet retransmissions can cause unacceptable delays. The first approach was to use a technique that serves a large numbers of receivers. IP multicast is a powerful and efficient way to transmit data to multiple parties. A range of IP addresses is reserved for multicasting (e.g. 224.0.0.0 - 239.255.255.255) [2], [3]. The receiver interested in multicasting simply subscribes to the service to be able to join the session. In contrast to Unicast, in which the server-client negotiations may result in sending individual copies of the same packets of the required file to each receiver. IP multicast relieves the sender of taking care of the individual receiver's needs. Multicast packets are only duplicated at the network intermediate branching points (nodes) as necessary. This allows only a single copy of a packet to reside in a network node. Hence, duplicates are forwarded from these nodes only as needed to subnets that have expressed interest in the multicast address. Using the Internet Group Management Protocol (IGMP) is one of the solutions to minimize redundancy; a receiver sends an IGMP message to subscribe to the multicast group and receives all the required packets within the scope of deployment of the multicast data. However, this suffers from the problem of systematized scalability and becomes unreliable if the number of receivers reaches a million or more, and any further interaction between servers and receivers would be a high price to pay [11], [12].

Nonenmacher *et al.* [13] proposed a hybrid solution for reliable multicast involving retransmissions and the use of redundancy. Their approach was to break the source data into small blocks and encode over these blocks. The receivers that have not received a packet from a given block request a retransmission of an additional packet (or packets) from that particular block, thus reducing the number of retransmissions when the packet loss over the network is very low. However, this procedure cannot completely avoid the need for ARQ at higher loss rates and with a large number of receivers. Thus, it imposes

the need for feedback channels in some traffic circumstances and with asynchronous network accesses.

The simplest protocol to ensure reliable delivery is the use of a *Data Carousel* or *Broadcast Disks*, where the sender loops repeatedly through all the data. The receiver asynchronously accesses the stream until all required packets are obtained, and there should be no need for retransmission requests. The only cost involved with this approach is waiting until the sender retransmits a desired packet that was lost and which the receiver seeks after obtaining all other packets. Another drawback is that in high loss environments, the reception overhead may be high due to unnecessary receptions waiting for a particular packet to reconstruct the file.

Eliminating the principle of retransmissions and reducing waiting times is a strong motivation to employ Forward Error Correction (FEC). The use of redundant data and proper encoding allows receivers to efficiently collect the necessary packets. It is worth mentioning that this technique handles *erasures* (complete packet loss/drop) rather than errors. This makes the application to IP multicasting valid because multicast packets may be lost/dropped, and corrupt/erroneous packets are handled by the lower network layers (e.g. Data Link Layer) error correction and/or detection techniques (e.g. CRC).

2.1.2 The Digital Fountain Approach

Typically, a conventional web server sends its content to different TCP receivers or intermediate nodes in the network. Consequently, each of these modules has to contend for server resources such as traffic load buffers, disk space etc. Even in the absence of a feedback channel, a 10-fold increase in receiver numbers results in a 10-fold increase in servers and a 10-fold increase in bandwidth. Luby *et al.* [15], [16], introduced the concept of a Digital Fountain as a powerful solution to scalability for reliable distribution of bulk data. A fountain server sends a stream of distinct *encoded* packets and the fountain client just needs to dip into the flood of incoming data to obtain a sufficient number of distinct packets to reconstruct the original file. The metaphor of the fountain is used due to the

analogy of a water fountain; one can quench his thirst by filling his glass from the fountain irrespective of the drops that fill the glass.

This technique is an ideal implementation of erasure decoding since k packets can be encoded as a stream of n packets that constitute larger data blocks at the sender so that a subset of these packets suffices to reconstruct the file at the receiver [12].

A receiver simply listens until the desired packets are obtained from the flowing fountain.

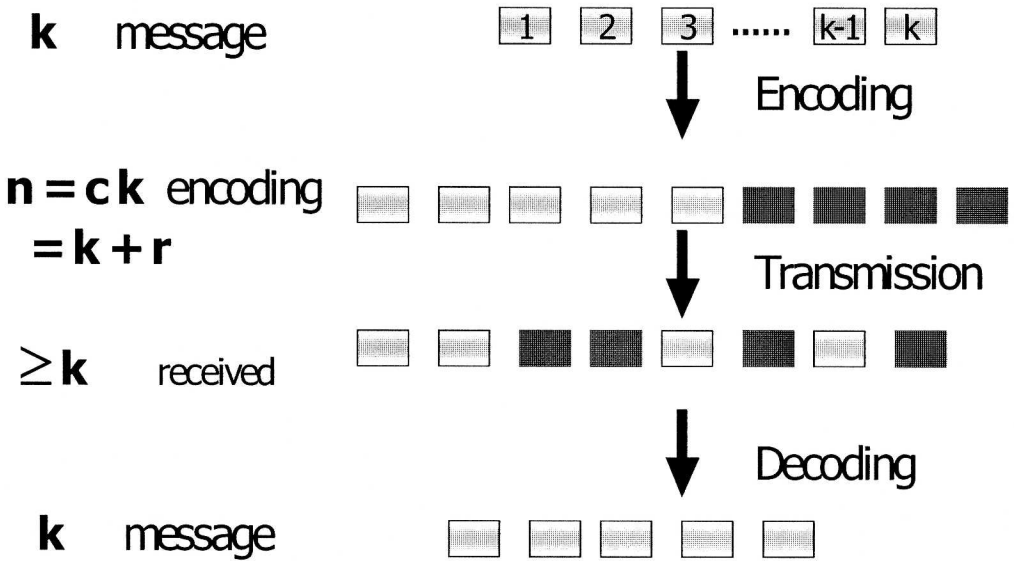


Figure 2.1Erasure encoding/decoding process

Note that a receiver only cares how many, not which distinct packets successfully arrived to perform the decoding. The key to the digital fountain approach is that the source data can be fully reconstructed from any subset of the encoded packets. Figure 2.1 illustrates the Digital Fountain using erasure decoding.

Ideally, Reed-Solomon codes are the best solution, since any k symbols or packets can be used to reconstruct the source file.

The digital fountain technique can be used a diverse number of applications, not only to multicast and broadcast over wired networks but also to satellite and wireless networks

where environments are quite different in terms of packet loss characteristics, congestion control mechanisms, end-to-end delays and burst packet losses.

2.2 Applications

A good way to test the efficiency and reliability of a certain protocol is to study how it performs under practical circumstances. Mirroring and Parallel Downloading from multiple mirror sites (*Paraloading*) is not only one of the first applications of the digital fountain technique, but is also one of the best in terms of performance measures, since the flood of packets from simultaneous multiple servers is heavily dependent on the individual load of every server and the traffic conditions involved in the system [17]. Before we proceed with the investigation of paraloading, we first discuss the types of mirroring deployed on the Internet. Data Mirroring is classified into two types:

1. Mirror Site Selection
2. Parallel Multiple Access

2.2.1 Mirror Selection Methodology

A growing number of websites introduce mirrors mainly to increase throughput and minimize download time. Mirror servers, which serve replicas of popular data items, have been employed for many years on the Internet as a way to increase reliability and performance in the presence of frequent accesses by many receivers. Mirroring, on the other hand, provides aggregation of throughput. The challenge for receivers comes mainly in determining which mirror site will offer the best performance when a file is to be downloaded. The choice of an individual mirror site is crucial in order to achieve reasonable performance. Unfortunately, *ad hoc* techniques are often employed as a common and easy solution as sufficient information about the server behavior is not always made available to the receiver in a simple form. Network designers, when characterizing the performance of an individual receiver when retrieving files from mirror servers should study many factors. These factors, such as file size or popularity,

can significantly affect the performance of a server. While considering that all files are replicated at each mirror, the size of the files can be assigned to the servers by the network designer. However, factors like geographical location cannot be dynamically adapted due to the growing number of receivers in a particular area. The corresponding distribution of servers will be controlled by many networking factors such as the proximity of the receivers or the number of hops in terms to network latency. Even if these factors have been taken into consideration and intelligently optimized by the network designer, the receiver should be well acquainted with the network performance metrics. For example, many web pages are available in both high-bandwidth and low-bandwidth versions; some provide the choice of an FTP or an HTML download application, while other pages present to users a long list of mirror sites to choose from based on the experience and information offered by the application. For web pages that have a choice of content type to present to a web receiver, the receivers have to choose the server based on the content *fidelity*, e.g. a movie clip or full graphics representation for high-bandwidth or text only for low-bandwidth. The so-called network-client feedback can be represented by an icon or a hyperlink that indicates the bandwidth to help with the appropriate decision. These applications require the creation of an independent system that predicts the expected network performance between a pair of Internet hosts. These systems provide active dynamic probing from peer hosts [17]. The probed information from these systems will introduce additional traffic, which will quickly grow to be non-negligible traffic reaching the busy server. Moreover, probing from a single host prevents the receiver from using past information from nearby receivers to predict future performance, reliability or traffic intensity [18].

2.2.2 Simultaneous Parallel Access of Mirror Sites

Accessing multiple mirror sites in parallel (*paraloading*) is a novel approach for retrieving files from the Internet by establishing parallel connections with multiple mirrored servers.

The method contrasts with the conventional point-to-point download method from a single server. Increased efficiency and improved performance is not only gained by aggregation of bandwidth but also by the load balancing possible among parallel connections, and the increased resilience against congestions and impending link failures of a single path. As the number of users increase, upgrades of data applications become more frequent, and paraloading becomes more widespread as result. It is clear that paraloading, if implemented properly, will offer significant performance gains over single mirror server selections by enabling intelligent and dynamic automatic load balancing and well distributed multicast sessions offering an organized mechanism to speed up downloads.

The basic scheme of paraloading is conceptually simple. When a receiver wishes to download a particular file, a list of mirror servers is made available. We assume the information can be easily obtained by querying a directory service. The paraloader being offered this quality of service is a member of a subnet within the area of a group of mirror servers. Knowing enough information about the file length and type of media, the receiver sends the download request. The requests are then duplicated by means of intermediate nodes and sent to the group of servers. As the copies of requests reach the servers, each server's response process begins by partitioning the file into smaller blocks (packets), and then transmitting them over the link to the receiver. When enough packets arrive, the receiver is able to reconstitute the file irrespective of their origin. Of course, it may not be obvious how to take this conceptually simple approach and derive a workable implementation that delivers optimum performance. To maintain efficiency, erasure codes are utilized.

The use of a feedback-free protocol provides a simple method of multicast transmission. As explained earlier, a file consisting of k packets is used to generate n packets via encoding. For multicast, a server transmits a stream of distinct packets from these n packets. This ensures that receivers can recover from lost packets provided they receive a sufficiently large subset of the transmission. With paraloading, each server transmits packets simultaneously from the same encoding and transmits them over the channel to

the receiver. In order to fulfill the requirements of a protocol that ensures reliable multicast, the following factors should be carefully considered [15], [16].

- **Time-efficient:** The amount of processing required to generate packets at the servers and the decoding time to reconstruct the original data at the receiver should be minimal.
- **Reception-efficient:** The total number of packets each receiver needs to reconstruct the file should be minimal. Ideally, the aggregate length of the packets needed is equal to the length of the original file.
- **Server-independent**: Receivers should collect packets from the Digital Fountain created by one or more servers, i.e. they can collect the packets regardless of their originating server. No coordination between servers should be required for this.
- **Tolerant:** The Digital Fountain protocol should tolerate the heterogeneous population of receivers, especially the variety if random packet losses and end-to-end delay incurred by narrow band noisy links and limited buffer space.
- **Reliable:** Each receiver should be able to reconstruct an exact copy of the original file.

A protocol following the previous requirements not only is reliable for the Internet protocol (IP) on the wired internet, but also for group communications over satellite channels and wireless LANs where losses are likely to be unpredictable and more importantly, where feedback channels are of extremely limited capacity or nonexistent. The servers transmit the stream of distinct encoded packets whenever there are active receivers. It is worth mentioning that the protocol is also time-independent, i.e. receivers can join anytime at their discretion. During interruptions, the servers will continue sending the stream of encoding packets to other interested receivers or those subscribed in the service, an important aspect of the robustness with multiple mirror sites in parallel is the resilience against losses which parallelism provides. This compensates for the unavoidable packet losses due to bandwidth limitations allowing all ISDN and telephone modem users to enjoy reliable communications.

2.3 Related Work

Byers *et al.*. [19] were the first to study the scalable solution of allowing receivers to access data simultaneously from multiple mirror sites. Their solution was simply implemented by building a digital fountain with erasure codes. Their work focused on the first two requirements for system robustness mentioned earlier:

- **Time efficiency**

Erasure codes are used to build a digital fountain where users (receivers) can gather packets in parallel from an encoded file. The file can be reconstituted efficiently as soon as a sufficient number of packets arrive from any combination of the servers. The main hurdle in designing such a system is the encoding and decoding speeds. Achieving fast encoding and decoding times is difficult in general. Hence, Tornado codes were chosen because they have extremely fast encoding and decoding times [9], [15]. The price of such tremendous gain in speed is that slightly more than k distinct packets are required to reconstitute the original message.

With Tornado codes, decoding is guaranteed if a total of εk distinct packets of the total of n packets are recovered at the receiver, where $\varepsilon > 1$. We refer to ε as the *decoding inefficiency.*

The advantage of Tornado codes over Reed-Solomon codes (which are optimally efficient having $\varepsilon = 1$) is that Tornado codes tradeoff a slight increase in decoding inefficiency for a substantial decrease in decoding times. Thus, their performance is based upon the time for the arrival of enough packets to decode, and not on the actual decoding time of the file, which is negligible, indeed receivers can virtually perform decoding as soon as enough packets arrive.

In [9], [10] and [16] a detailed comparative analysis of Reed-Solomon and Tornado coded implementations of the digital fountain was given. In this analysis, Luby *et al.* claimed that Tornado codes are a better solution because they avoid the complex decoding times of Reed-Solomon codes.

The study was based on the following simulation parameters; Sun 167 MHz UltraSPARC 1, with 64 MB RAM. The packet length was P = 1KB and stretch factor was c = 2. Hence, $n = 2k$ and ½ the packets are redundant or parity and ½ the packets are original data. The variance of the decoding inefficiency ε was found to be generally very small, approximately 0.0073 at ε = 1.05536. The study was carried out with a slight overestimation using a decoding inefficiency ε = 1.0556.

It is worth mentioning again that the decoding times for tornado codes are considered minimal, i.e. the decoding procedure is deemed complete once εk distinct packets have been obtained at the receiver.

- **Reception efficiency**

As defined earlier, the second vital requirement for robustness of the system is *reception efficiency*, i.e. reconstruction of the original file should be accomplished with the minimum number of packets, ideally with k packets. Reconstruction at the receiver can take place in one of two ways; the first is an incremental approach where decoding is done in real time and the receiver performs decoding operations on the fly after packets begin to arrive. This may lead to redundant computation, as reconstructed source data may later arrive intact [16]. The second, a simpler approach is that the receiver waits until a given number of packets arrive prior to decoding, based on the statistical observations of the file. If decoding cannot be completed at this time, the receiver waits for more packets to arrive, after which it is likely that the file can be reconstructed. While the former approach appears to have the benefit of speed by enabling both decoding computations and reception of packets to be done in parallel, the latter approach is simpler to implement in practice and requires slightly more time to recover the file. In the case of Tornado codes, the receiver simply waits until $1.0556k$ packets arrive to guarantee successful decoding [15], [16].

Now, consider the measurement of reception efficiency. Two types of *inefficiency* can be defined:

1. *Decoding Inefficiency* ε is defined as the additional number of distinct packets that must be received over the minimum necessary to guarantee successful decoding (file reconstruction)

$$\varepsilon = \frac{Total\ numberof\ distinct\ packets\ received}{Total\ number\ of\ source\ data\ packets} \tag{2.1}$$

2. *Distinctness Inefficiency* η, is defined as the loss in efficiency due to the reception of duplicate packets. This can occur either by a server cycling through all the packets (which can occur due to high link loss rates), or due to the effect of multiple mirror sites as will be explained later.

$$\eta = \frac{Total\ number\ of\ packets\ received}{Total\ number\ of\ distinct\ packets\ received} \tag{2.2}$$

Combining these two parameters yields the *Reception Inefficiency*, ζ, given by

$$X = \frac{Total\ number\ of\ packets\ received}{Total\ number\ of\ source\ data\ packets} \tag{2.3}$$

It is clear that $X = \varepsilon\ \eta$. Thus, the receiver should receive a total of Xk packets before decoding to guarantee reconstruction of the original message [15], [19].

To minimize Reception Inefficiency, two approaches are considered in the following section.

2.4 Packet Transmission Techniques

2.4.1 Random Permutation

In order to minimize the inefficiency due to duplicate packets, Byers *et al.*. considered a very large stretch factor to decrease the likelihood of receiving duplicate packets, but can

increase the complexity of encoding and decoding would result in impractically (very long) decoding times.

Hence, their work adopted Tornado codes to improve speed at the price of extra bandwidth [19]. In addition, with multiple mirror sites, it is obvious that increasing the number of servers (mirrors) increases the likelihood of receiving duplicate packets. Note that the reception of duplicate packets is maximized when the receiver obtains the same packets from each server at the same rate.

To avoid such a situation, in their protocol, each server *randomly* permutes the packets before transmission. Each server uses a different permutation to determine the order in which packets are sent, but the set of Tornado coded packets is the same for all servers. In high loss environments, which would be rare in wired networks, a server may send out all the packets before the receivers gets enough to decode. In this case, the server continues to transmit packets in the same order until all receivers have the file. As a result, a receiver may receive more duplicate packets than those caused by the effect of multiple senders. In [19], a very high stretch factor was assumed so that duplication caused by high loss rates did not occur.

While decoding inefficiency ε is chosen to be a fixed quantity, the distinctness inefficiency varies significantly with the number of senders and the stretch factor.

2.4.2 Staggered

We have found that randomly permuting the packets at each server is not the best solution. As a result, a new technique is introduced here which 'staggers' the packets and has superior performance compared to the random technique above.

As the servers are staggered, overlap between packet copies is less likely to occur at the receivers, thus allowing the required number of distinct packets to reach each receiver more quickly at very low stretch factors (typical code rates), which is highly desirable. In addition, the technique has proven by simulation to perform very well at high loss rates, as shown in the next chapter.

The technique is described below:

Every server sends the packets from a file in numerical order starting at a point depending on the server's order in the network, which is arbitrary but fixed. In other words, every server is assigned a point in the packet sequence to start transmissions from, which is fixed throughout the transmission session. If the server exhausts the encoded packets to the end (N), it continues from packet 1 until the receivers have obtained a sufficient number of packets. The first of M servers begins transmission from packet 1 in the file. The second server in the network starts transmitting from packet number [(N/M) + 1], the third server from packet number [(2N/M) + 1], and so on. Consequently, the M^{th} server will start transmission from packet number [(M - 1) N/M + 1].

For example; If the total number of encoded packets is N = 1000, and the number of mirror servers is M = 4, the first server starts transmitting from packet 1, the second server starts from packet 251, the third server from packet 501 and the fourth server from packet 761. Ideally, a receiver will obtain a sufficient number of distinct packets before any server begins to repeat packets. This minimizes the *reception inefficiency* due to duplicate packets from multiple servers, and thus the download time for a file. Figure 2.2 shows the staggered technique transmission of an N-packet encoded file downloaded from M servers.

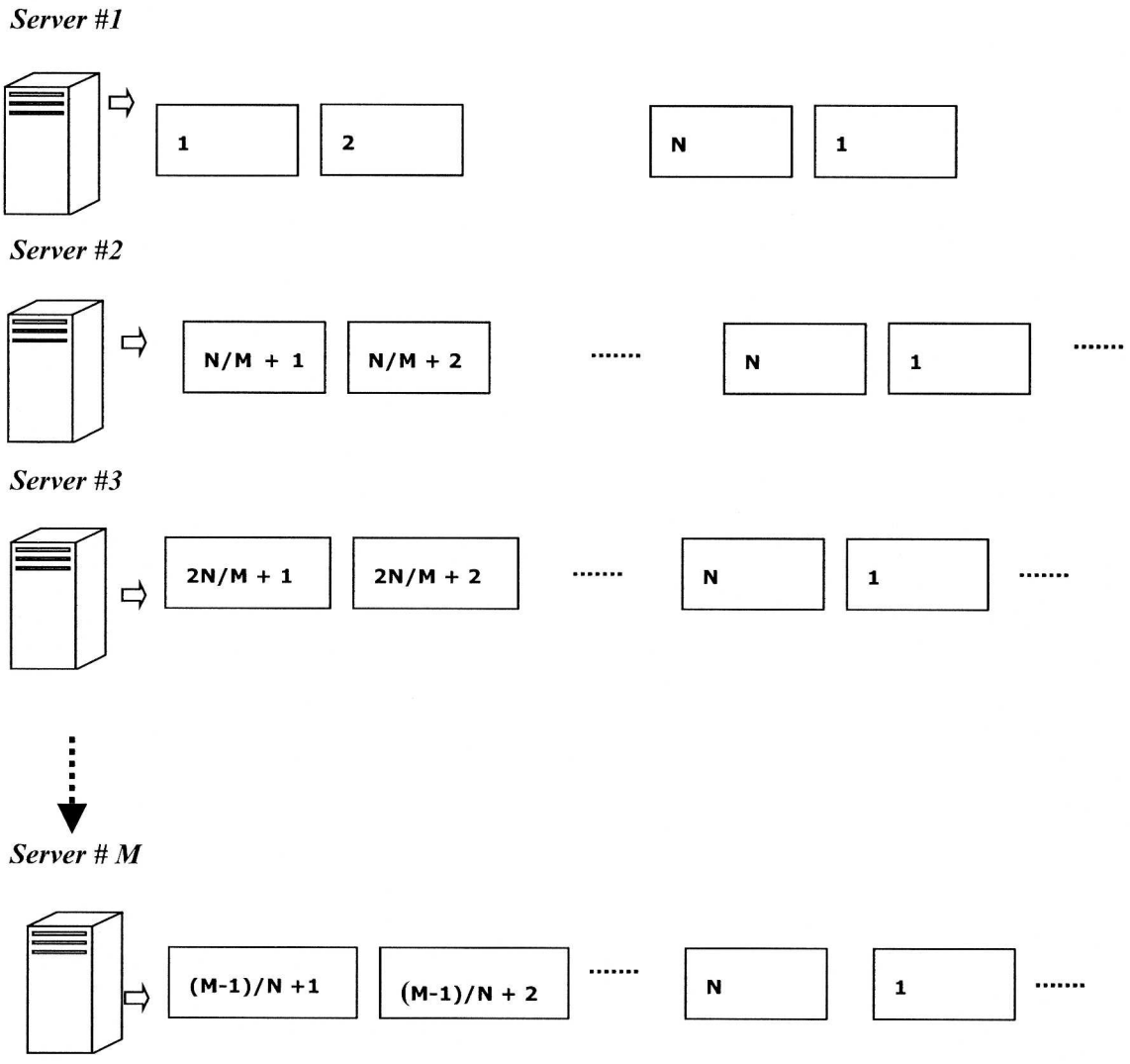


Figure 2.2 Staggered servers technique

2.4.3 Enhanced Staggered Technique

An attractive advantage of the staggered technique is its flexibility in compensating for slow or inefficient servers among the mirror sites. This technique can easily be enhanced to compensate for these servers for a given receiver or set of receivers by simply modifying the starting point of the servers. This can be done by dividing the packets amongst servers according to their individual data rates, and thus assigning a smaller number of distinct packets to be transmitted first by the slow servers to compensate for their lower transmission rates. This is termed the *Enhanced Staggered* technique. To demonstrate the technique, we use the notation in [19]. The individual data rates are always referenced to the fastest server; For example, 3 equal data rate servers are symbolized by (1:1:1), (2:2:1) indicates that the third server has half the data rate of the first two, (2:1:1) indicates that 2 servers have half the rate of the first server, and (4:2:1) indicates that the second and third servers transmit at half and 1/4 the rate of the first server, respectively.

The starting points for servers with data rates (2:1:1) are determined based on the three ratios (1/5,2/5,2/5), so that the three servers transmit the N packets for a file for the first time in the least time possible. Thus the starting points are (1:2N/5:4N/5), and similarly for the other two patterns, (2:1:1) and (4:2:1), the enhanced staggered technique will use (1:2N/4:3N/4) and (1:4N/7:6N/7), respectively.

The simulation results given in the next chapter will show that this technique can provide significant improvements to the system performance. The simulations are done using OPNET, and the performance of the system with multiple mirror sites is obtained with respect to the above two essential requirements for efficiency [16], namely:

- Completion Time.
- Reception Inefficiency.

The goal in this thesis is to provide a performance comparison of the following:

- ✓ Reed-Solomon vs. Tornado codes.
- ✓ Staggered vs. random permutation techniques.

3. System Performance and Experimental Design

3.1 The OPNET Model

OPNET is one of the most powerful Graphic User Interface (GUI) network simulation tools. It is based on the concept of event driven simulation, which allows a very close look at the performance of the system under investigation. The software embedded within OPNET provides precise probing of system parameters. OPNET provides a comprehensive development environment supporting the modeling of communication networks and distributed systems. Both behavior and performance of modeled systems can be analyzed by performing discrete event simulations.

The OPNET model used in this thesis is shown in Figure 3.1. It consists of the following objects:

1. Server:

The OPNET server is a simple source module transmitting fixed size packets of size P = 1KB with exponentially distributed inter-arrival times. The data rate of the server is simply controlled by controlling the mean inter-arrival time between packet transmissions.

2. Point-to-Point Links

All model links are chosen to be a standard simplex-wired DS-1 link with a maximum data rate (bandwidth) of 1.55 Mbps.

3. Bottleneck

A custom OPNET packet discarder module represents the link bottleneck. The number of dropped packets is uniformly distributed throughout each transmission session and controlled by the module simulation attributes.

4. Receiver

The receiver module is an OPNET packet sink. The number of packets in an unencoded file is controlled by the file size attribute, and is set to 1MB or 1024 packets. The decoding inefficiency ε attribute is toggled between 1.000 (Reed-Solomon codes) and 1.0556 (Tornado codes).

5. Application Control Module

The application control module is an independent node controlling two main parameters during the simulation, the stretch factor c and the number of servers on line N. The main function of this module is to assign values to the stretch factor and the number of servers dynamically during the simulation process.

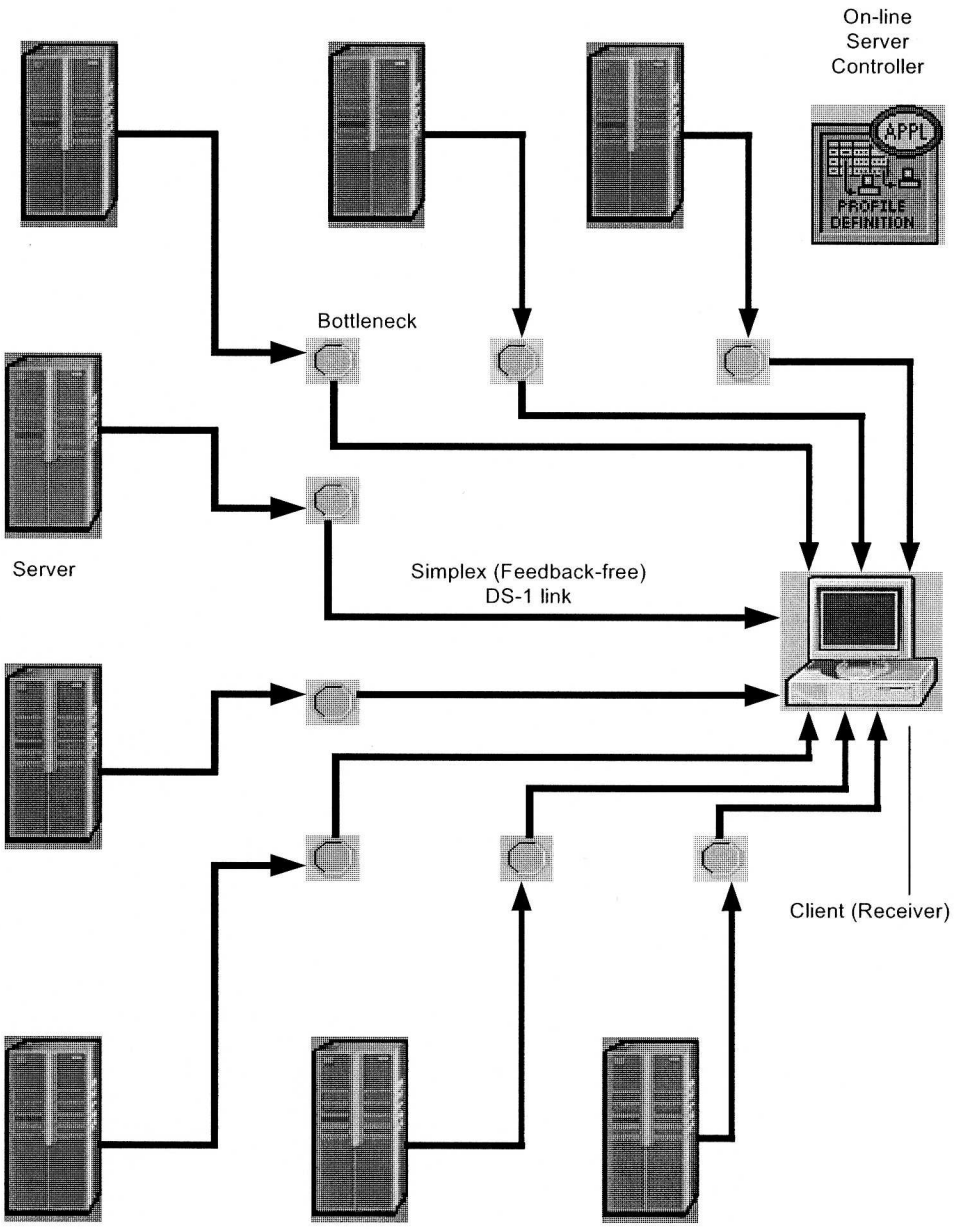


Figure 3.1The OPNET model

3.2 Performance Results

The assumption in [19] is also considered in the simulation model here. All packets are bottleneck disjoint, i.e. no packets are dropped due to other mirror servers accessing the same receiver.

A typical transmission rate of 50 KB/s. (nominal download data rate of the www.opnet.com ftp site) was chosen. The performance results presented here are for an average of 40 trials for both the Tornado and RS coded systems.

We considered high stretch factors in the range (c = 1 to 4) and a uniform 10% packet loss rate while varying the number of servers and their individual data rates; the rates of the servers are shown in brackets if needed. The figure suffixes (a) and (b) will note the simulation results for completion time and reception inefficiency respectively.

3.2.1 Reed-Solomon vs. Tornado codes.

In [19], simulation was used to gauge the performance of the system quantifying the cost versus the gain, i.e. the decoding inefficiency versus the system improvement using Tornado codes.

We will carefully follow the same route while comparing Reed-Solomon codes and Tornado codes using the decoding times given in Table 3.

In our OPNET simulation, we used an un-stretched 1 MB file (1,024 - 1 Kbytes packets), Using Tornado codes, a 1MB file comprising 1024 packets, can be decoded once 1024 x (ε = 1.055) = 1,081 distinct packets arrive at the receiver, while Reed-Solomon codes require only 1024 (ε = 1) packets, the further simplification is justified as well when ignoring the decoding time which is considered minimal. The scenario is performed with steady uniformly distributed losses assuming that one packet is dropped out of every ten packets [19].

In Figure 3.1.a. we find that by varying the stretch factor, Reed-Solomon codes have the same response as Tornado codes regarding downloading time improvements. With two servers transmitting at the same rate, the plot shows that the completion time difference

for stretch factors between 1.5 and 4.0 is approximately 1-1.2 sec.. According to Table 3, this interval is enough time to decode the file using a Reed-Solomon well before the additional packets required for Tornado code decoding arrive at the receiver. Using the 4i2i IP decoder, the RS decoding time is 16% of the additional time required to acquire enough packets to successfully decode the file using a Tornado code. If the ALTERA decoder is employed, decoding is done in only 4% of the additional time. Note that this time is necessary to acquire an additional 57 packets.

In Figure 3.1.b. using Reed-Solomon codes improves reception inefficiency by approximately 8% at c = 2 and approx 4.5% at c = 4.

In Figures 3.2.a, we verify the results depicted by Byers *et al.*. in regarding the advantage of using mirror sites to speedup download with Tornado codes. In [19], the main point in the plot is to show that beyond a moderate stretch factor of 3 adding additional servers dramatically reduces completion time with minimal additional reception inefficiency. With a further stretch of the original message at $c = 4$, doubling the number of senders reduces the completion time by almost 4 sec.. Using four senders at this stretch factor further improves the reduction the download time to approximately 30% with only a loss of 4% in efficiency.

Our results show that using Reed-Solomon, the improvement still holds, moreover the difference in completion time with the increase of the stretch factor to $c = 4$ still a solid interval of 1-1.25 sec., which is quite enough time to decode the file with the referred hardware decoders in Table 3 as we have previously shown.

In Figure 3.2.b, we show that Reed-Solomon codes ($\varepsilon = 1$) still outperforms Tornado codes ($\varepsilon = 1.055$) as we increase the stretch factor and the number of servers with an average of approximately 8% at $c = 2$ which is the typical value of the stretch factor used in [16]. Moreover, with Reed-Solomon codes the loss in efficiency did not exceed the 4% change result at $c = 4$ with Tornado codes deduced before. Recall that this case where duplicates are maximized due to equal rate servers is the worst-case for reception inefficiency.

In Figures 3.3.a, b and 3.4.a, b we verify the most common case where packets arrive at different rates from 2 and 3 senders respectively. This example is a good illustration of a practical Internet environment using mirror sites, where it happens for a variety of reasons. One server may be heavily loaded, or one link from server to receiver is heavily congested, triggering congestion control solutions forcing the corresponding servers to lower down the transmission rate along the route till the congestion is cleared and traffic is recovered. Moreover, the receiver may be accessing the server through a limited-bandwidth (narrow band) link e.g. ISDN line or a telephone modem. Alternatively, this is a good way to measure the performance of the system if we intentionally add more servers to speedup download.

Figure 3.3.a shows the speedup as we vary the stretch factor using two servers at different rates. Note that the label (2:1) indicates that the second server is half the rate of the first, similarly (4:1) the second server is 1/4 the transmission rate of the first. Of course, the speedup is much less significant if the other server transmits at a lower rate. One advantage in this case as Figure 3.3.b shows is that the receiver obtains less duplicate packets, so the reception inefficiency is reduced by virtue of a reduction in distinctness inefficiency especially at higher stretch factors while the minimal loss in efficiency is worth the gain in speedup by adding equal-rate servers.

The figure also depicts that this result holds true when using Reed-Solomon codes where the reception inefficiency is equal to the distinctness inefficiency ($\varepsilon = 1.00$). With regards to completion time (Figure 3.3.a) it is clear the Tornado codes still outperformed by more than 1.0–1.2 sec. at $c = 1.5$–4.0.

Moreover, the loss in efficiency using Reed-Solomon codes is still less significant than Tornado codes by almost 7% at $c = 2$ to 6% at $c = 4$ as Figure 3.3.b shows.

Figures 3.4 a, b provide similar results for the case of three servers. The completion time results are referred to the fastest server so naturally the speedup will be less when other servers share the transmission process at a lower rate. Of course, it is obvious that equal rate servers will yield to the optimum case of speedup, however, we can make a further interesting comparison study to gauge the system performance when more servers are

added, we look again to Figure 3.3.a, to quantify the benefit of a receiver accessing a third slower sender in addition to the two which it is already accessing. For example, in the (2:1:1) sending pattern, the gain in speedup is quite marginal than accessing two servers on a (1:1) pattern as we see e.g. at $c = 2$, the download time of the file is on a (1:1) pattern is approximately 23.6 sec. on a (2:1:1) pattern the download time is approximately 23.9 sec. On a (4:2:1) pattern, things are even worse here the download time is approximately 25.7 sec. Thus, we deduce that, maintaining two servers at a closer transmission rate has much impact on the download time improvement and cost reduction than trying to access slower servers. In the case of efficiency loss, Figure 3.4.b shows that accessing more servers still sacrifices minimal efficiency to gain tangible system speedup.

All figures, show that when using 3 servers at either equal or different rates Reed-Solomon codes still outperforms Tornado codes by at least 1 second, which is enough time to decode the file. The Tornado codes reception inefficiency is still outperformed by Reed-Solomon codes by 7.6% margin through the increasing course of the stretch factor.

3.2.2 Staggered vs. Random-Permutation Transmission.

A limitation of the Byers *et al.* protocol, is the lack of coordination between the servers, which results in high duplicate transmissions and hence an increase in the distinctness inefficiency. In [19] two suggested solutions provide the necessary offset for minimizing duplication; the first solution requires all the server transmit roughly at the same rate using the same encoding and the same random permutation, at connection set up, the receiver request to download source data at an offset which is a function of the number of servers that is already known by the receiver. The second solution requires explicit renegotiations in addition to that at the initial call setup. The server transmits packets from a specified range and the other sources from a different range requested by the receiver rather than transmitting them in random.

We have suggested a new staggered technique. An advantage of this technique on those suggested in [19], is that the receiver-server negotiations are minimal. Of course, as the

number of servers increase the individual starting point will vary accordingly, thus the only event where high coordination is required is only when servers join the session unintentionally, which is not the case here.

The OPNET simulation parameters e.g. number of servers, data rates and link losses, are the same as those used previously in the random technique to gauge the performance of the system in a similar manner.

We have noticed that using the staggered technique not only dramatically reduces the distinction inefficiency but also sharply reduces the download time.

Figure 3.5.a, shows packets arrive at equal rates from multiple servers; we notice that the staggered technique outperforms the random technique at low stretch factors. For example, at $c = 1.15$, staggered outperforms random by approximately 5.5 sec. or 28.5 % of the total download time. Any further increase in the stretch factor does not affect the improvement in download time, and any increase in the stretch factor is unnecessary. Moreover, the response is similar when mirror servers are added, thus; this technique is a highly desirable solution to avoid using large stretch factors.

In Figure 3.5.b., the staggered technique shows a significant improvement in the reception inefficiency, for example, at $c = 1.15$, the inefficiency is reduced by 50%, 70% and 90% in the case of 2, 4, and 8 servers respectively. At this value of c, the reception inefficiency is optimal (a value of 1). Second, the addition of more servers hardly affects the loss in efficiency for the staggered technique. If Tornado codes are used, the optimal reception inefficiency will occur at the value of the decoding inefficiency ε, which is 1.055 in our case.

In Figures: 3.6.a,b, 3.7.a,b and 3.8.a,b, packets arrive at the receiver from multiple servers at different rates using Reed-Solomon codes; with small stretch factors (high code rates), which are highly desirable.

In Figure, 3.6.a. with (2:2:1) servers pattern, better performance is obtained with the staggered technique at very low stretch factors, As an example, the common RS (255,223) code which has a stretch factor of approximately $c = 1.15$, and the staggered technique outperforms random by 3.25 sec., i.e. 12% of the total time.

When compared to Tornado codes, referring to Table 3, using the slower 4i2i IP core, decoding can be done in approximately 1.5% of this time, which is minimal. This indicates that the decoding process of Reed-Solomon codes using the staggered technique can be almost done in real time.

The performance is even better when we use the enhancement method by assigning the slow server less distinct packets to transmit first with respect to the two fast servers. The method outperforms the random permutation scenario by 4.0 sec. (24% of download time) at the same stretch factor (c = 1.15). However, this only applies to a receiver that begins reception at the start of the transmission sequence.

The results also significantly impact the reception inefficiency as Figure 3.6.b shows by lessening the likelihood of transmitting duplicates. The optimal reception inefficiency is achieved at a very low value of the stretch factor c = 1.2, with a 70% improvement in efficiency over random and approximately 38% over normal staggered.

Similarly, in Figure 3.7.a, the (2:1:1) servers where two slow servers at half the rate of the fastest server are accessed by the receiver, the staggered technique reduces download time by approximately 3.0 sec. or 11% of download time at c = 1.15. In addition, with staggered, an improvement of 6% in download time and approximately 60% in efficiency is achieved as shown in Figure 3.7.b.

In Figures 3.8.a, with (4:2:1) servers, the disadvantage with the staggered technique shows the need of further enhancement. At a stretch factor of c =1.05, the performance degrades sharply due to the number of duplicate packets in the system. The slowest server is sending new packets when its faster counterparts are repeating packets as the receiver waits until a sufficient number of distinct packets are present in the system. When the enhanced staggered technique is introduced, a very significant performance is achieved. The packets are divided among all servers with respect to their rates; hence, (4:2:1) servers will yield the starting point (1: 4/7: 6/7) and a reduction of approximately 9 sec. (28%) in download time.

In Figure3.8.b, an interesting result is achieved by using the enhanced staggered technique. The distribution of starting points according to individual data rates has a significant impact on efficiency results. At approximately value of $c = 1.15$ which is very desirable, the reception inefficiency is optimal. This is similar to the staggered result with 3 equal data rate (1:1:1) servers pattern and consequently, with 4 and 8 server pattern as Figure 3.5.b' shows. Therefore, the performance of the enhanced staggered technique with respect to efficiency tends to be identical to using equal rate servers with the normal staggered technique.

3.3 Putting it All Together

We finally complete our work with Figures 3.9, 3.10, 3.11 and 3,12, which show the performance of the staggered and random techniques using both Reed-Solomon and Tornado codes. With different loss rates, we find that a Reed-Solomon code using the staggered technique provides the best performance results. We induce a 1% packet loss ratio, a typical value for most wired networks. To make our comparison consistent, we choose a value of c = 1.144, the stretch factor for RS (255,223).

In Figure 3.9.a, with Reed-Solomon codes the staggered technique outperforms random by 5 sec., while using Tornado codes results in approximately 6 sec. difference. In Figure 3.9.b, a 50% reduction in reception inefficiency is achieved with RS codes to 38% using Tornado codes.

In Figure 3.10.a, we use the same 10% loss ratio as in [19], and notice that the loss increase has a less significant impact on the staggered technique by as the difference in completion time is approximately 4 sec.;
In Figure 3.10.b, the staggered technique outperforms the random by approximately 18% for both Reed-Solomon and Tornado codes.

In Figures 3.11 a, 3.11 b, we notice that the response of the system with both RS and Tornado codes using the staggered technique under different loss rates is similar.

The plots are merely shifted horizontally and vertically by a value equivalent to the change of the decoding inefficiency ε for each value of packet loss. At the chosen value of ε = 1.055, the staggered technique still outperforms the random at very low stretch factors. In Figure 3.11 a, at c = 1 - 1.05, Tornado codes cannot successfully be decoded while decoding of Reed-Solomon codes can be performed. In Figure 3.11 b, Reed-Solomon codes reception inefficiency outperforms that of Tornado codes, by at least 5.5% in all cases.

Thus, the staggered technique is the better solution for mirror site transmission for various packet loss rates at the desirable low stretch factors (high code rates)

Thus, this technique is a robust approach when accessing Internet multiple mirror sites in parallel.

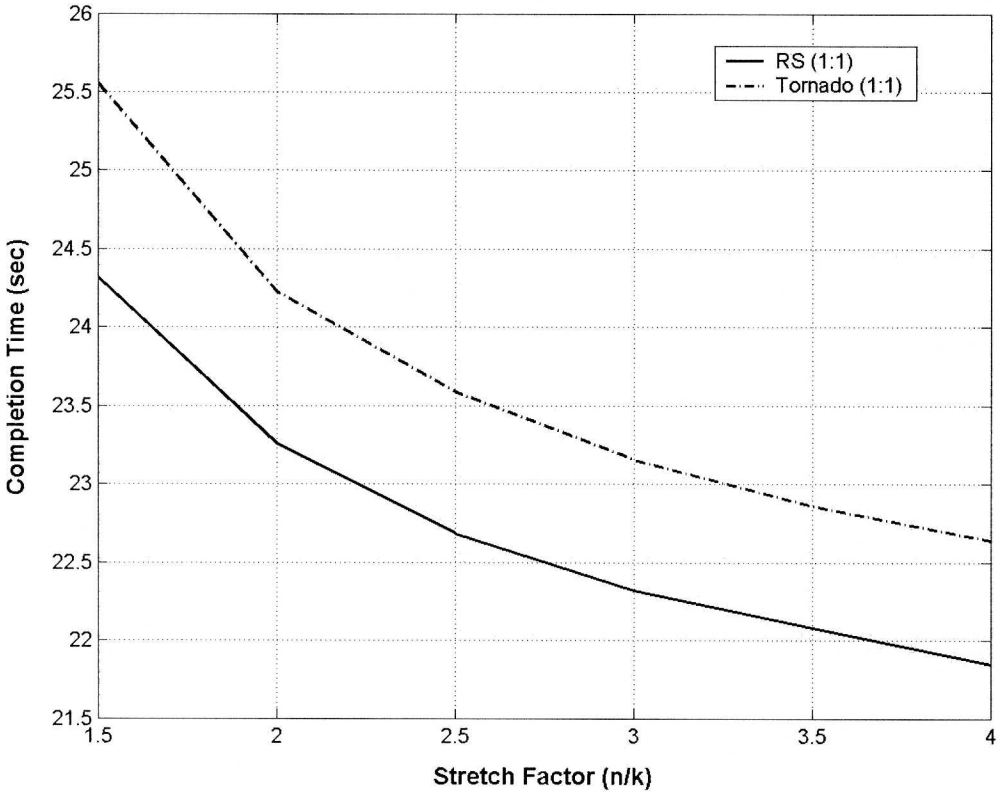


Figure 3.1a Completion time vs. stretch factor. Packets arrive at equal rates from 2 servers with 10% loss rate using random technique.

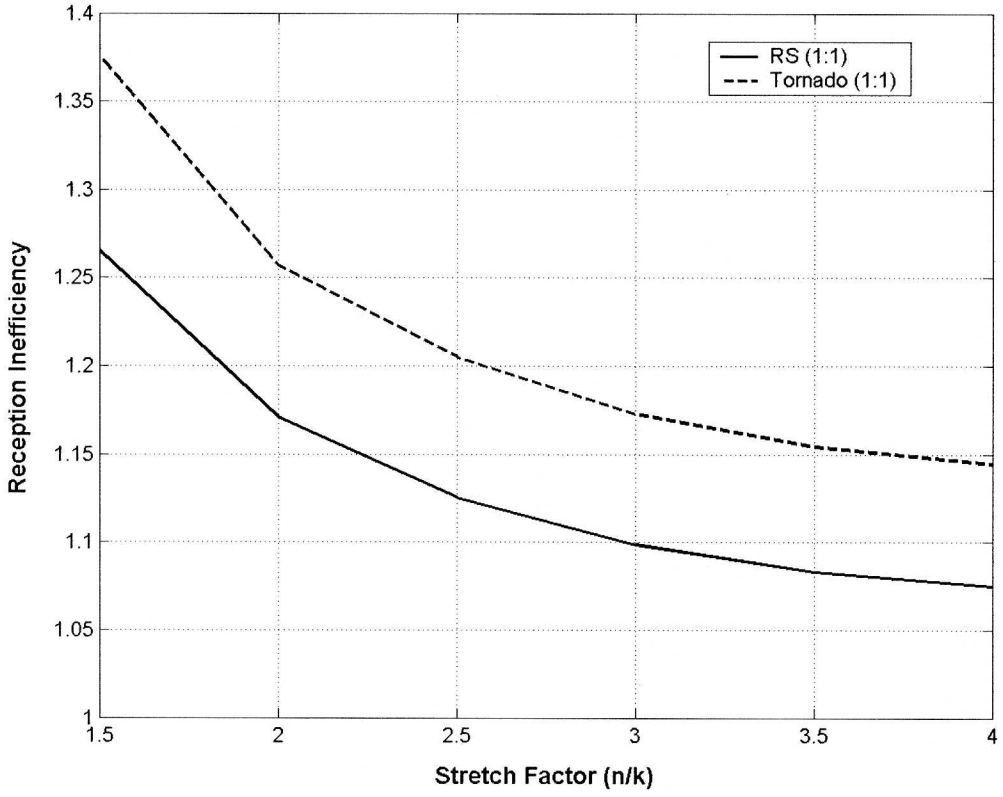


Figure 3.1.b Reception inefficiency vs. stretch factor. Packets arrive at equal rates from 2 servers with 10% loss rate.

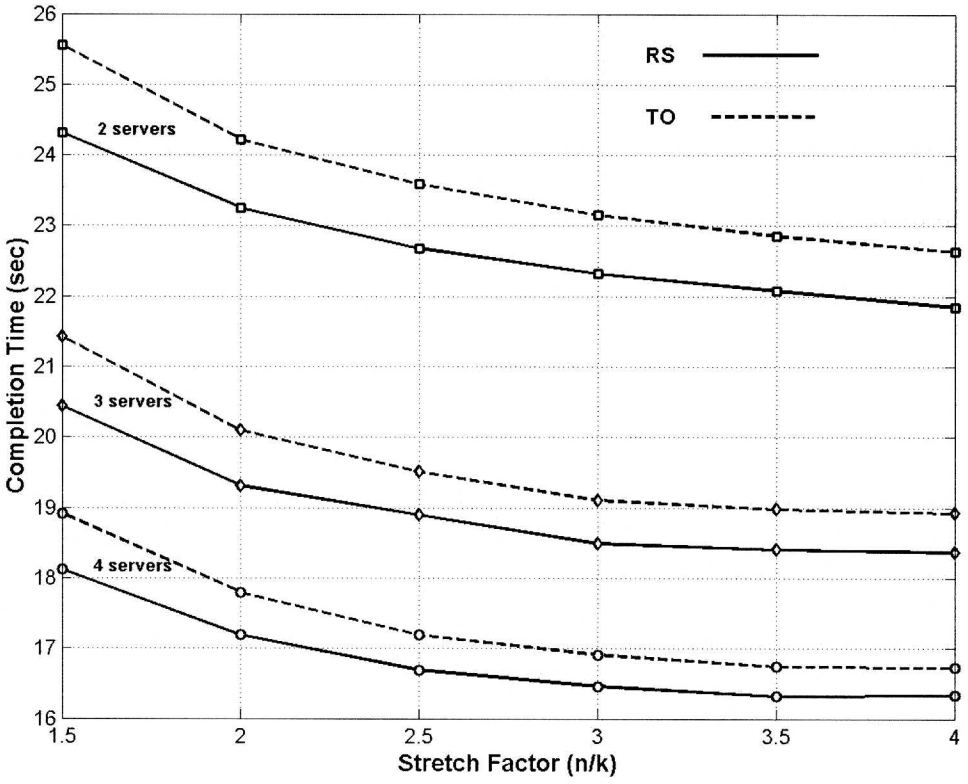


Figure 3.2.a Completion time vs. stretch factor. Packets arrive at equal rates from multiple servers with 10% loss rate using random technique.

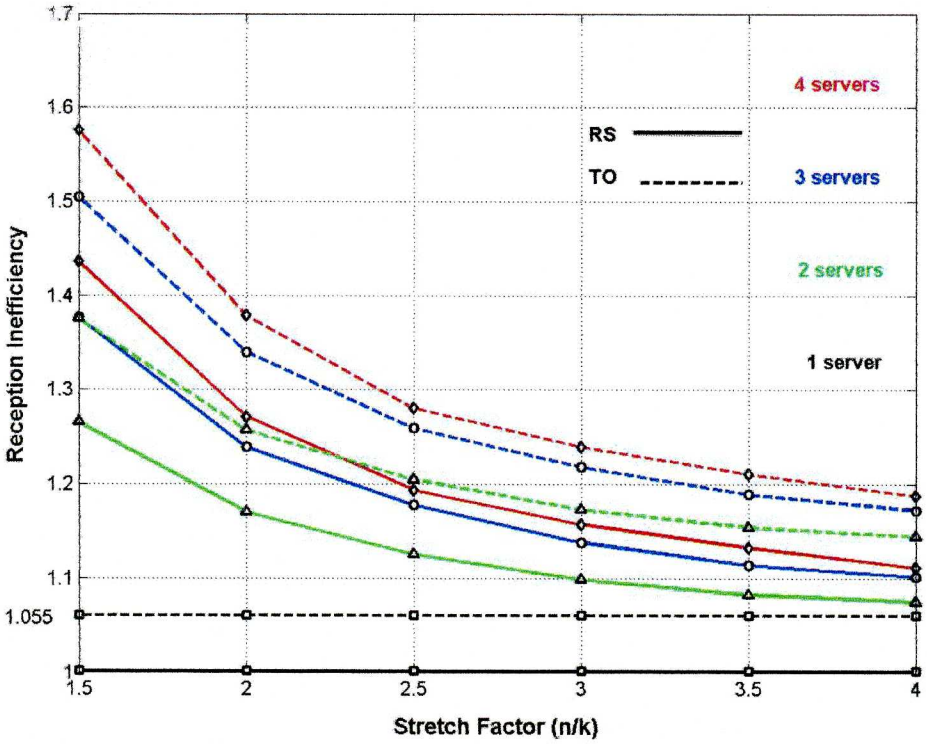


Figure 3.2.b Reception inefficiency vs. stretch factor. Packets arrive at equal rates from multiple servers with 10% loss rate using random technique.

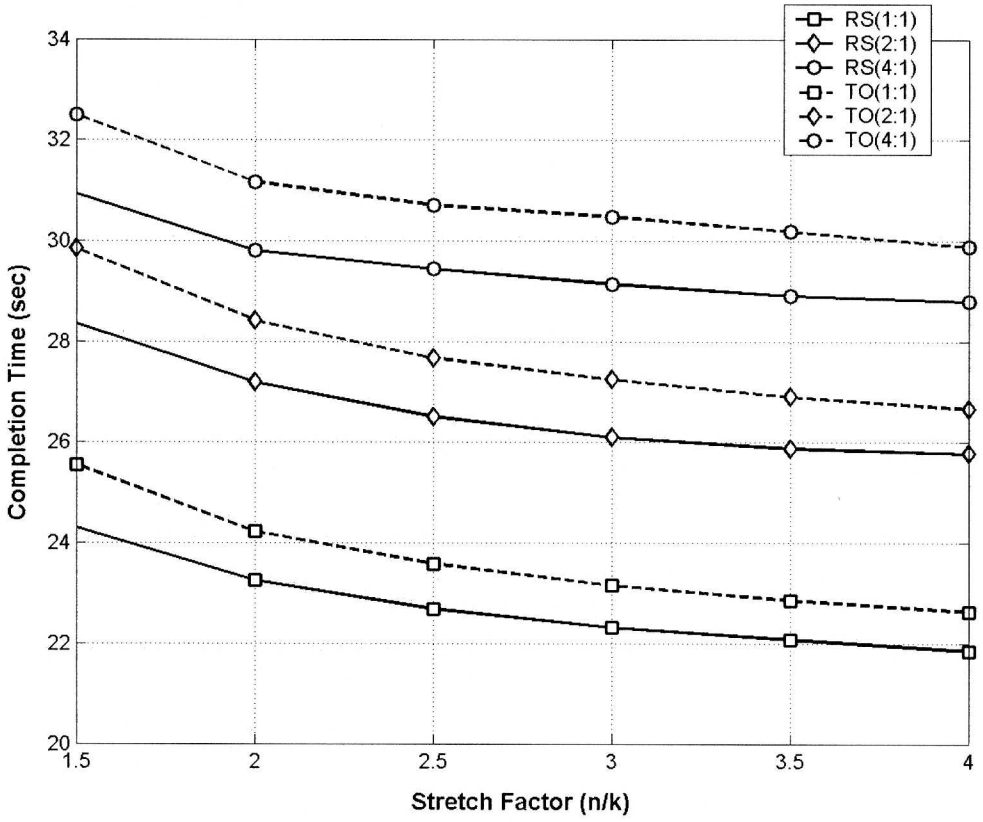


Figure 3.3.a Completion time vs. stretch factor. Packets arrive at different rates from 2 servers with 10% loss rate using random technique.

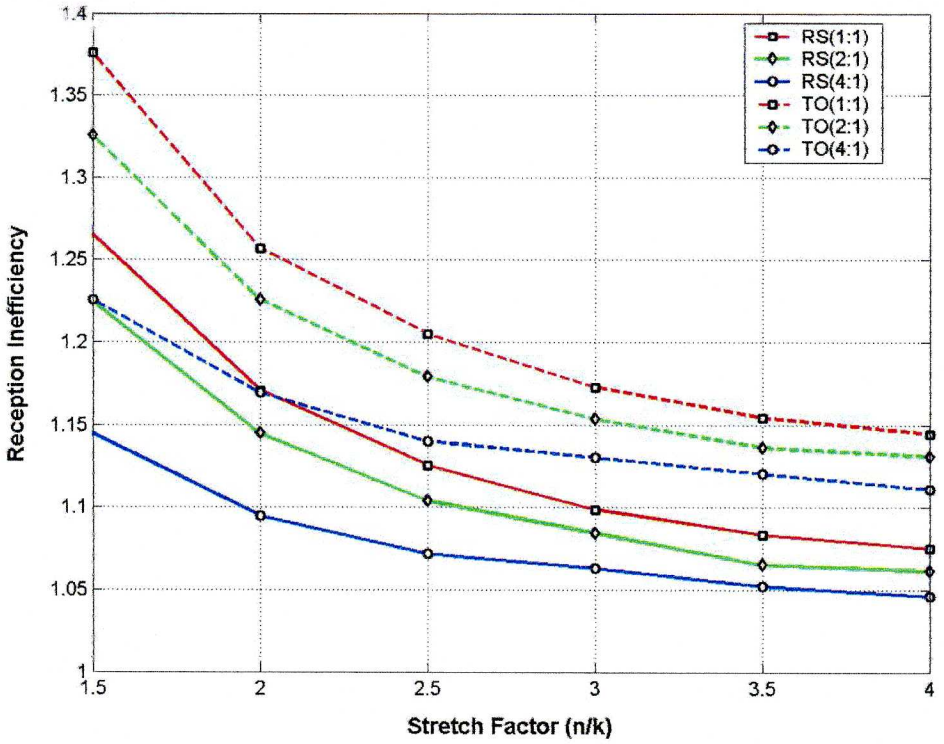


Figure 3.3.b Reception inefficiency vs. stretch factor. Packets arrive at different rates from 2 servers with 10% loss rate using random technique.

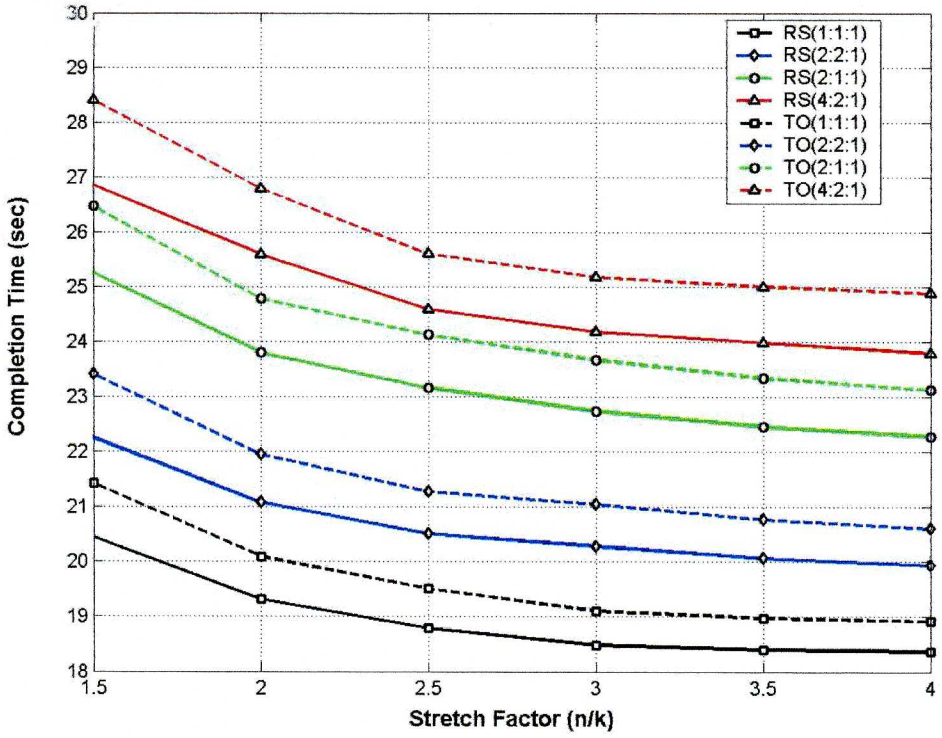


Figure 3.4.a Completion time vs. stretch factor. Packets arrive at different rates from 3 servers with 10% loss rate using random technique.

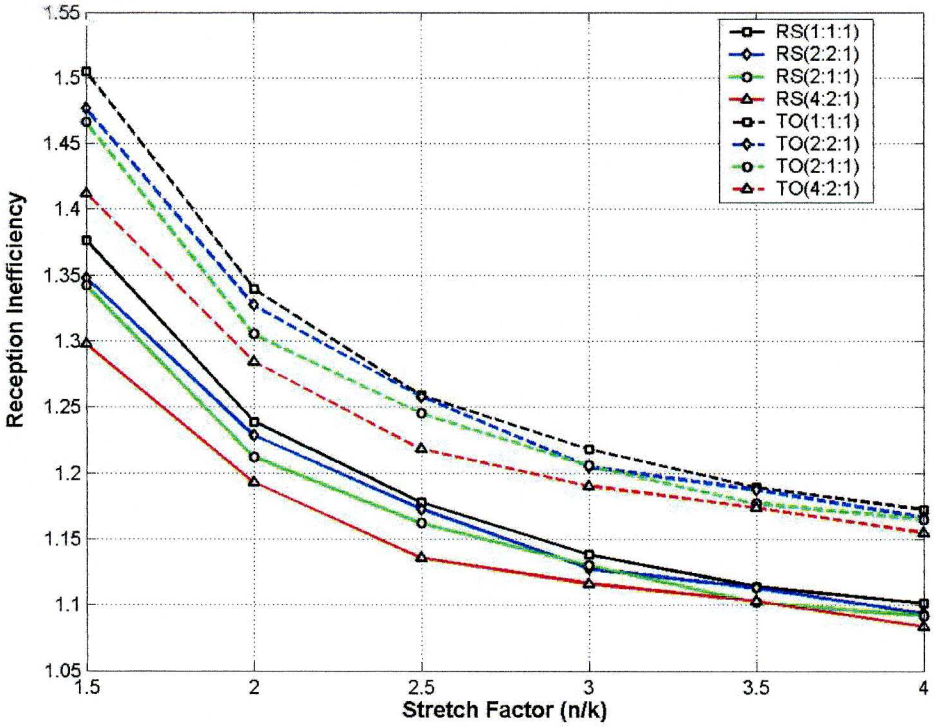


Figure 3.4.b Reception inefficiency vs. stretch factor. Packets arrive at different rates from 3 servers with 10% loss rate using random technique.

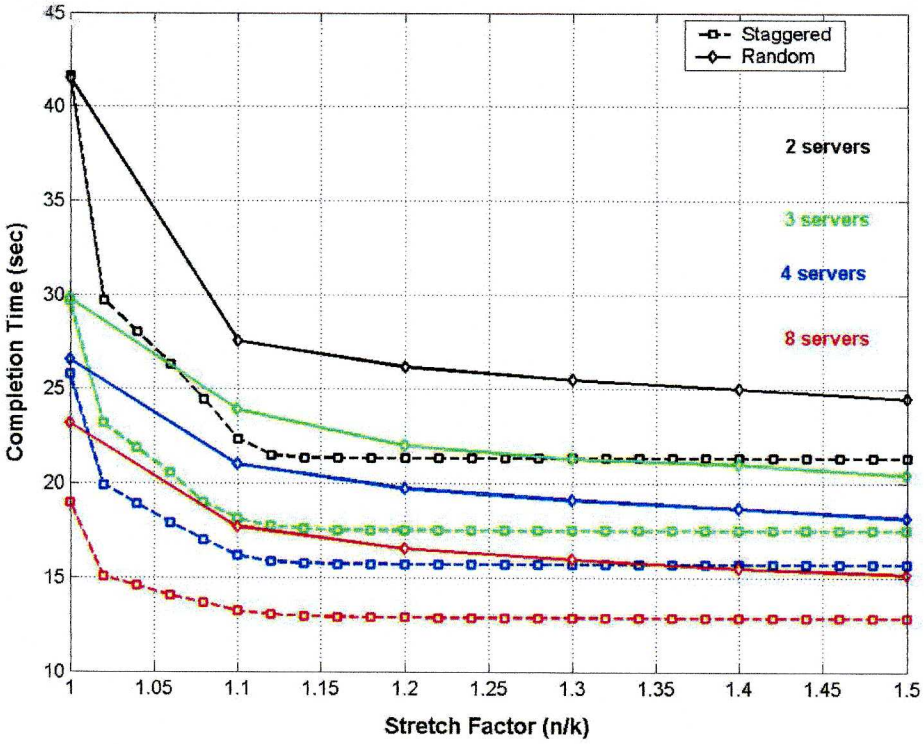


Figure 3.5.a Completion time vs. stretch factor. Packets arrive from multiple servers at equal rates using Reed-Solomon codes with 10% loss rate

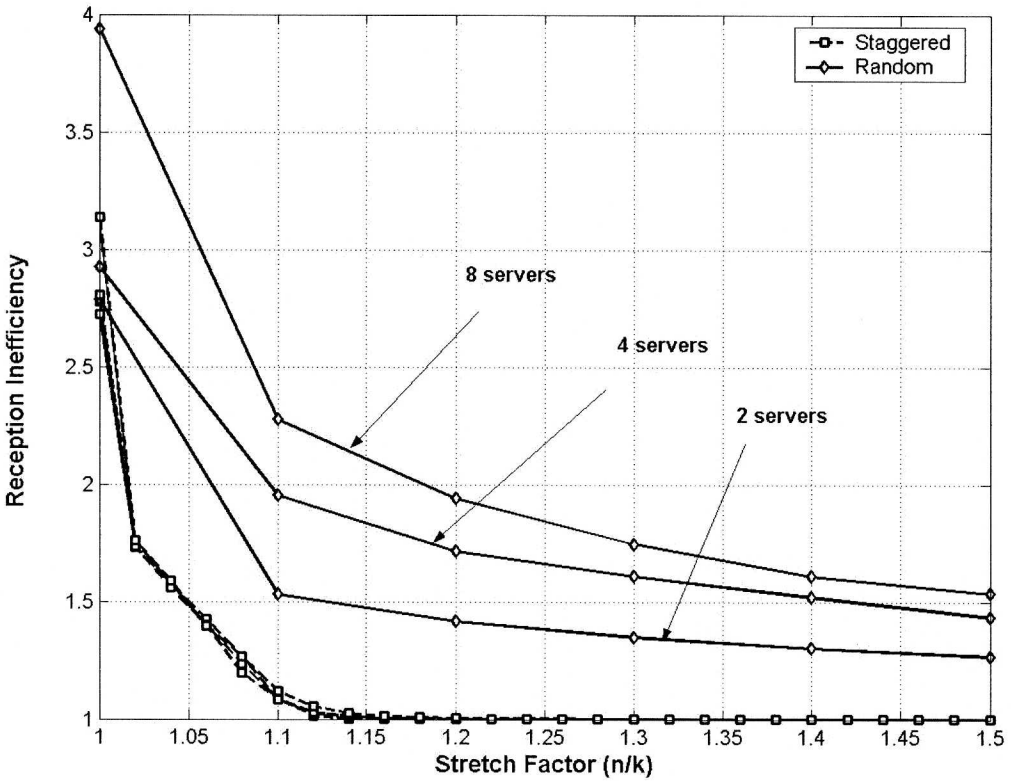


Figure 3.5.b Reception inefficiency vs. stretch factor. Packets arrive at equal rates from 3 servers using Reed-Solomon codes with 10% loss rate.

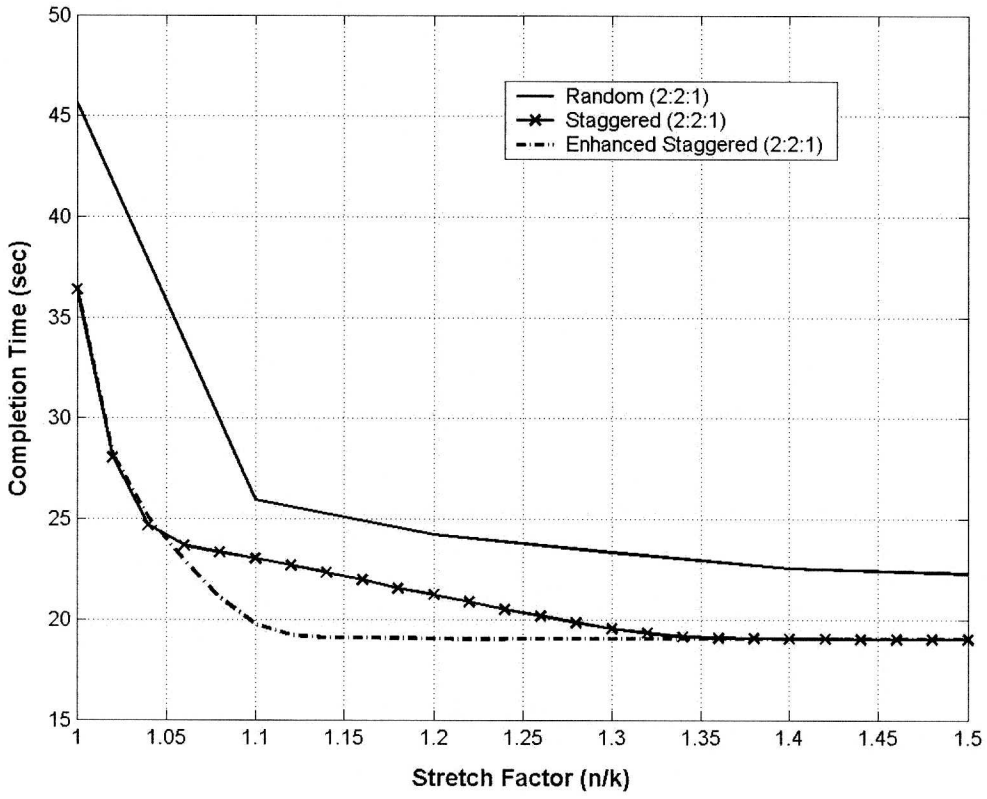


Figure 3.6.a Completion time vs. stretch factor. Packets arrive at different rates from 3 servers using Reed-Solomon codes with 10% loss rate.

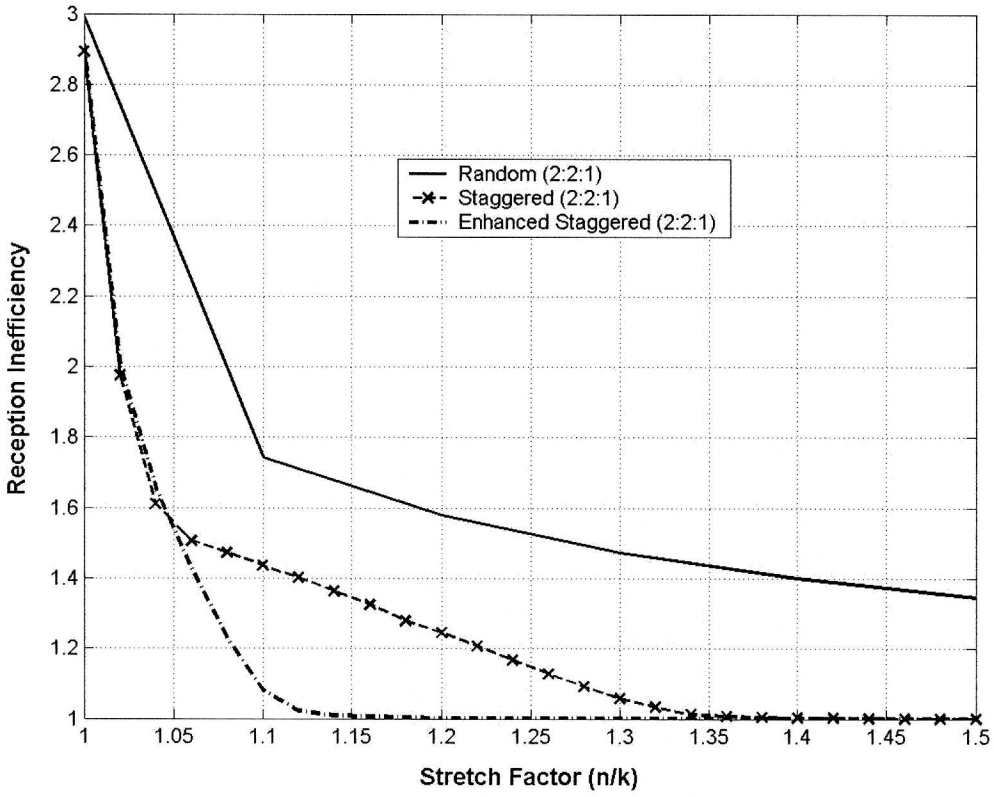


Figure 3.6.b Reception inefficiency vs. stretch factor. Packets arrive at different rates from 3 servers using Reed-Solomon codes with 10% loss rate.

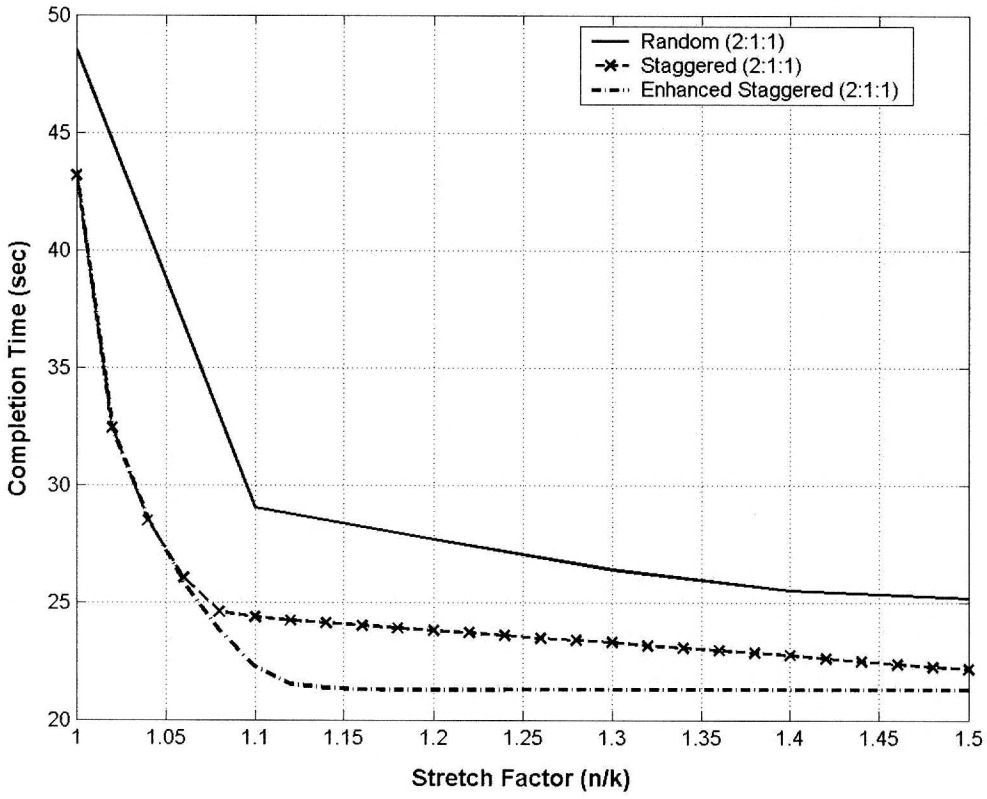


Figure 3.7.a Completion time vs. stretch factor. Packets arrive at different rates from 3 servers using Reed-Solomon codes with 10% loss rate.

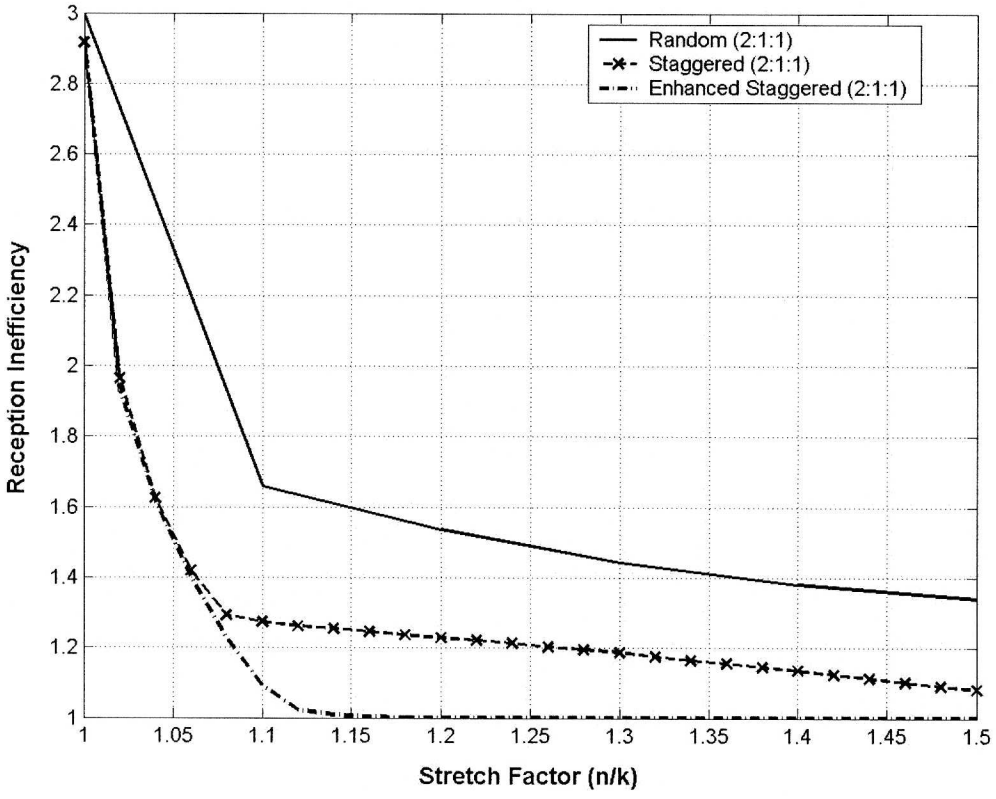


Figure 3.7.b Reception inefficiency vs. stretch factor. Packets arrive at different rates from 3 servers using Reed-Solomon codes with 10% loss rate.

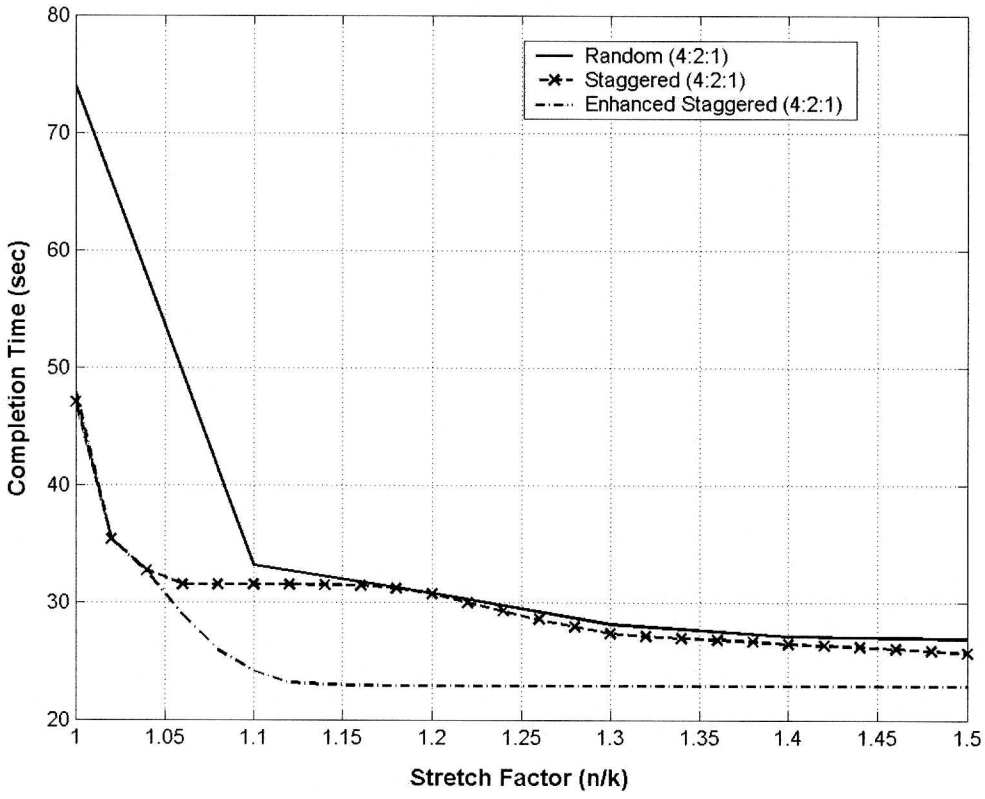


Figure 3.8.a Completion time vs. stretch factor. Packets arrive at different rates from 3 servers using Reed-Solomon codes with 10% loss rate.

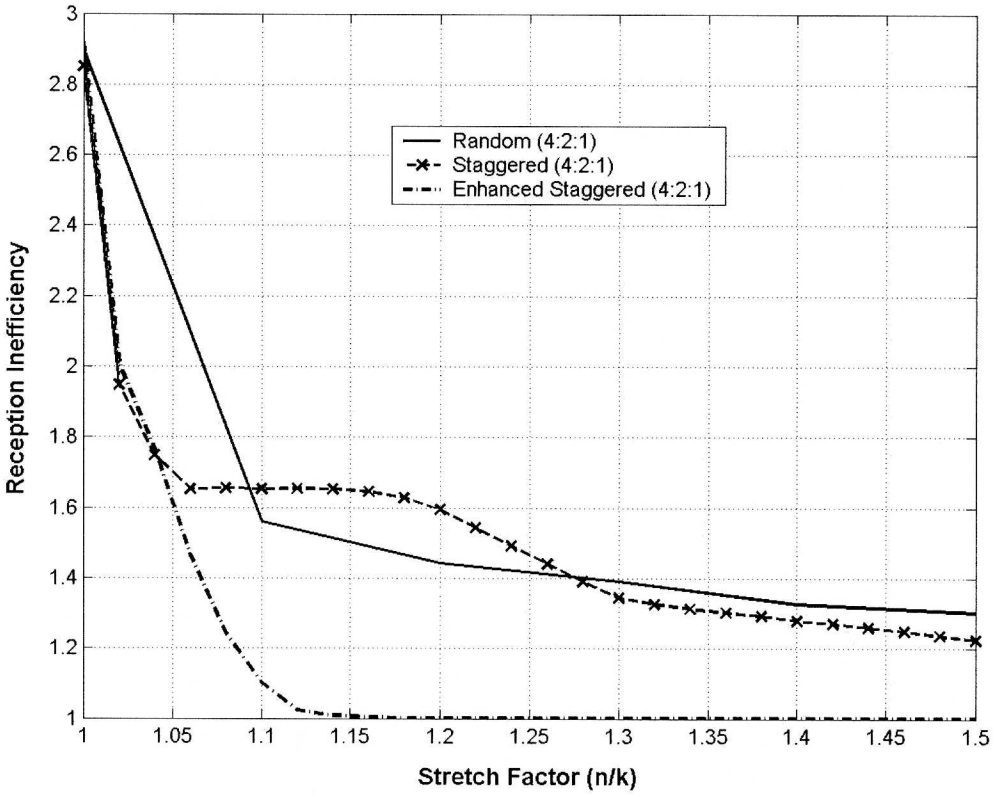


Figure 3.8.b Reception inefficiency vs. stretch factor. Packets arrive at different rates from 3 servers using Reed-Solomon codes with 10% loss rate.

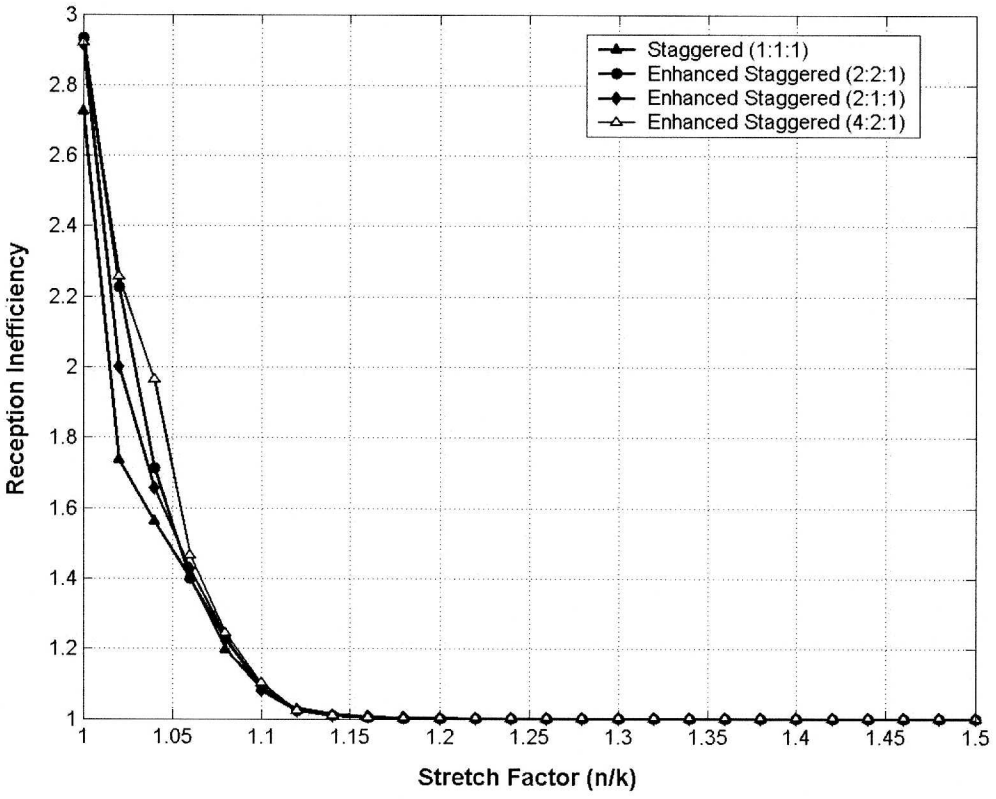


Figure 3.8.b' Reception inefficiency vs. stretch factor. Packets arrive at equal and different rates from 3 servers using Reed-Solomon codes with 10% loss rate.

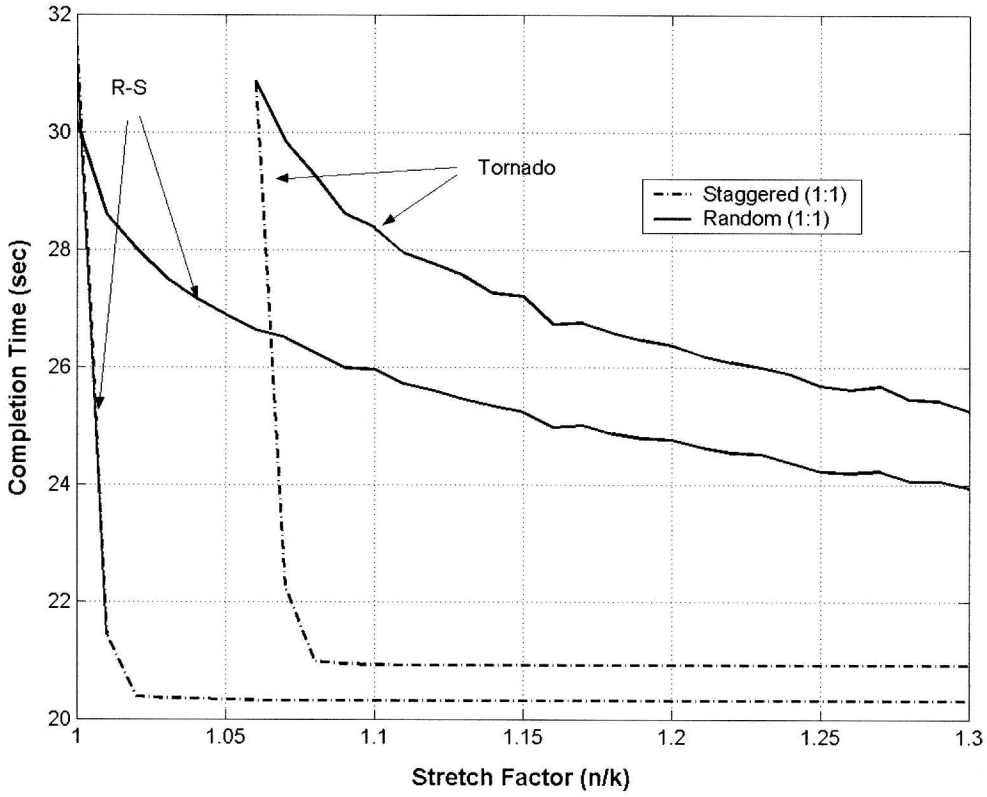


Figure 3.9.a Completion time vs. stretch factor. Packets arrive at equal rate from 2 servers with 1% loss rate.

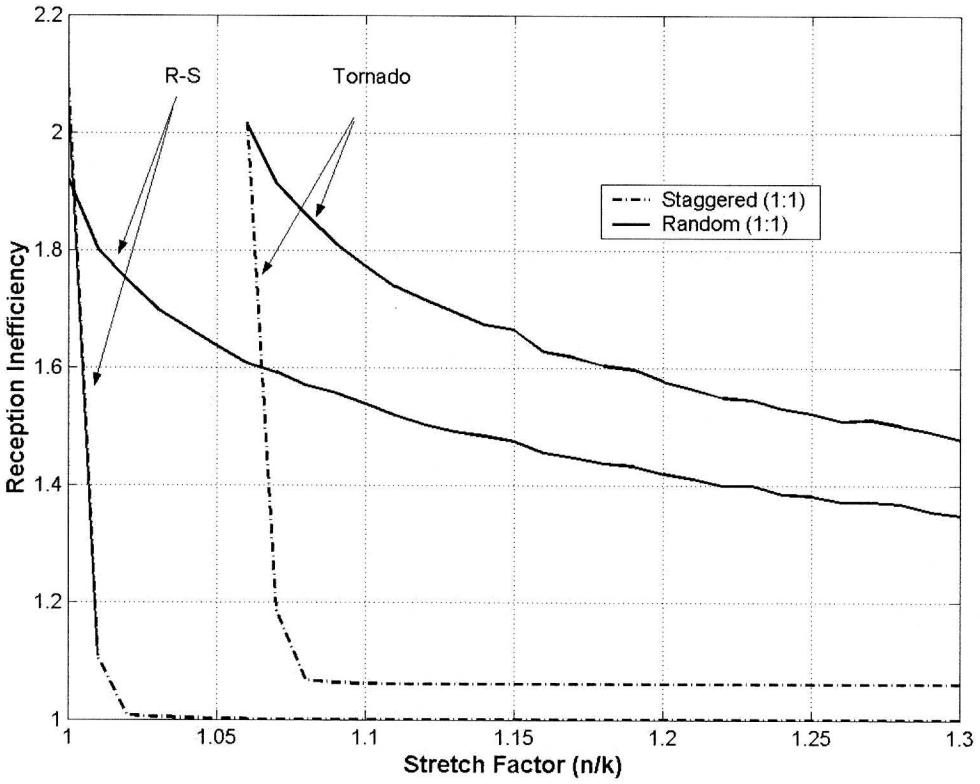


Figure 3.9.b Reception inefficiency vs. stretch factor Packets arrive at equal rates from 2 servers with 1% packet loss rate.

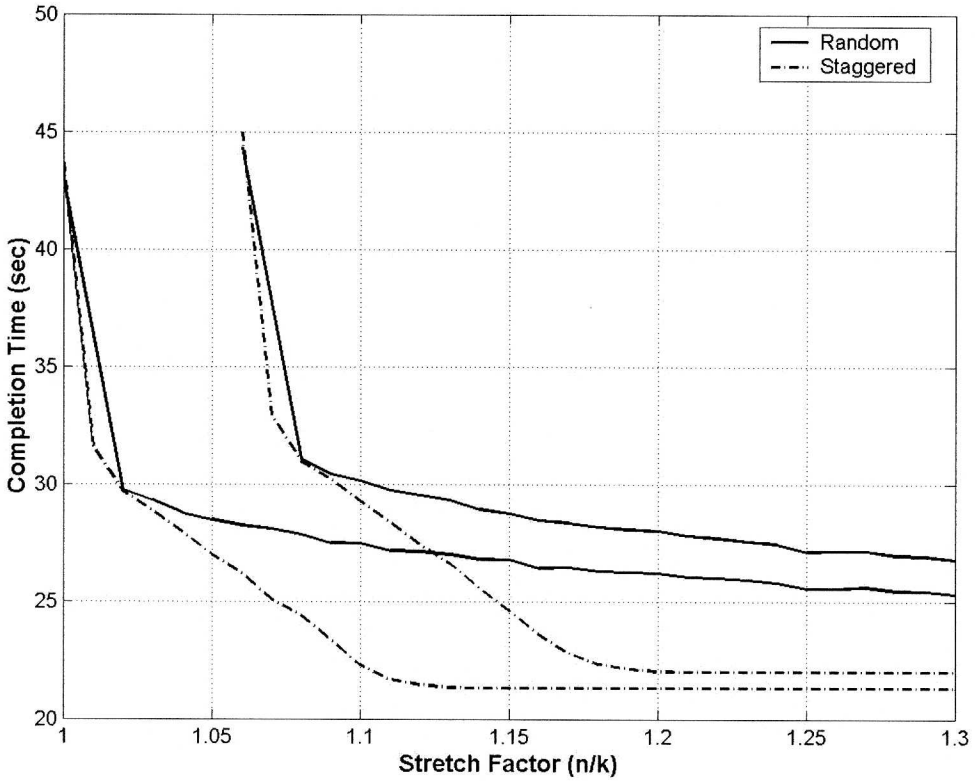


Figure 3.10.a Completion time vs. stretch factor. Packets arrive at equal rates from 2 servers with 10% loss rate

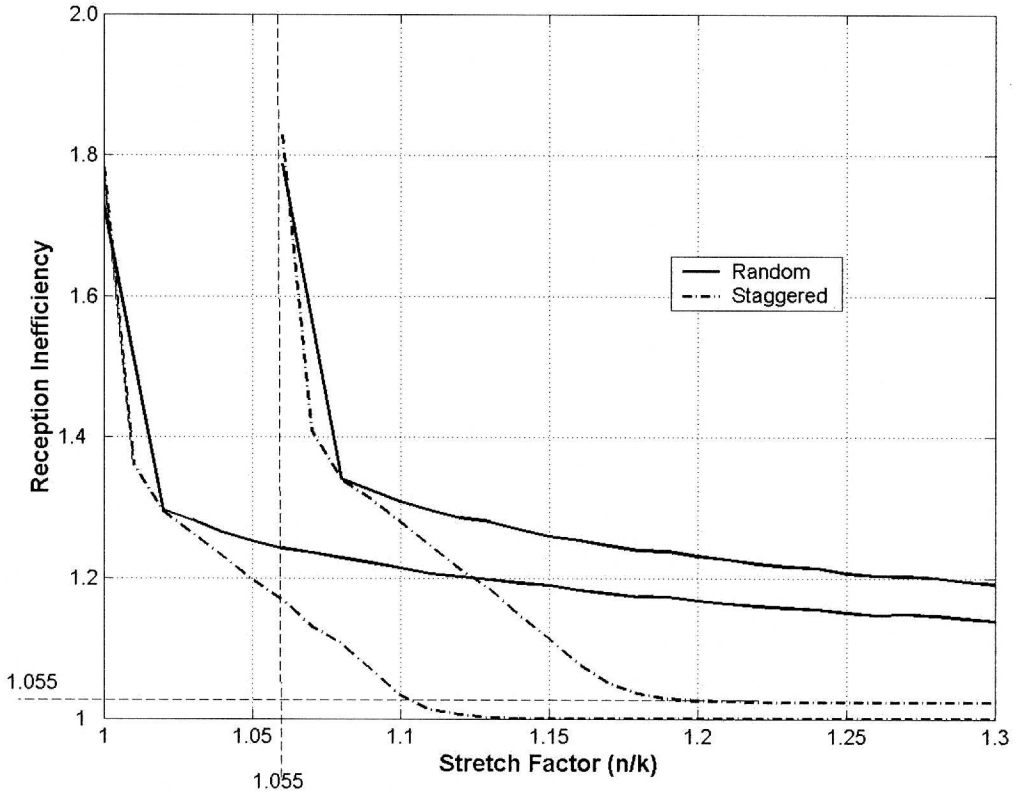


Figure 3.10.b Reception inefficiency vs. stretch factor. Packets arrive at equal rates from 2 servers with 10% packet loss rate.

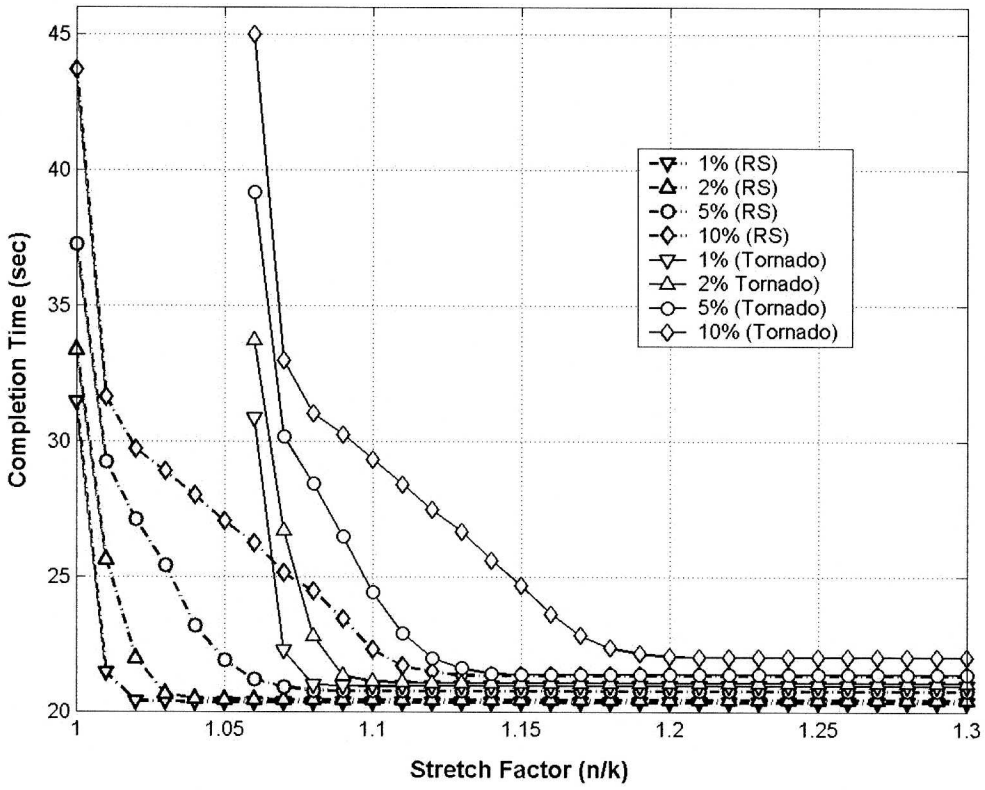


Figure 3.11.a Completion time vs. stretch factor. Packets arrive at equal rates from 2 servers with different loss rates using staggered technique.

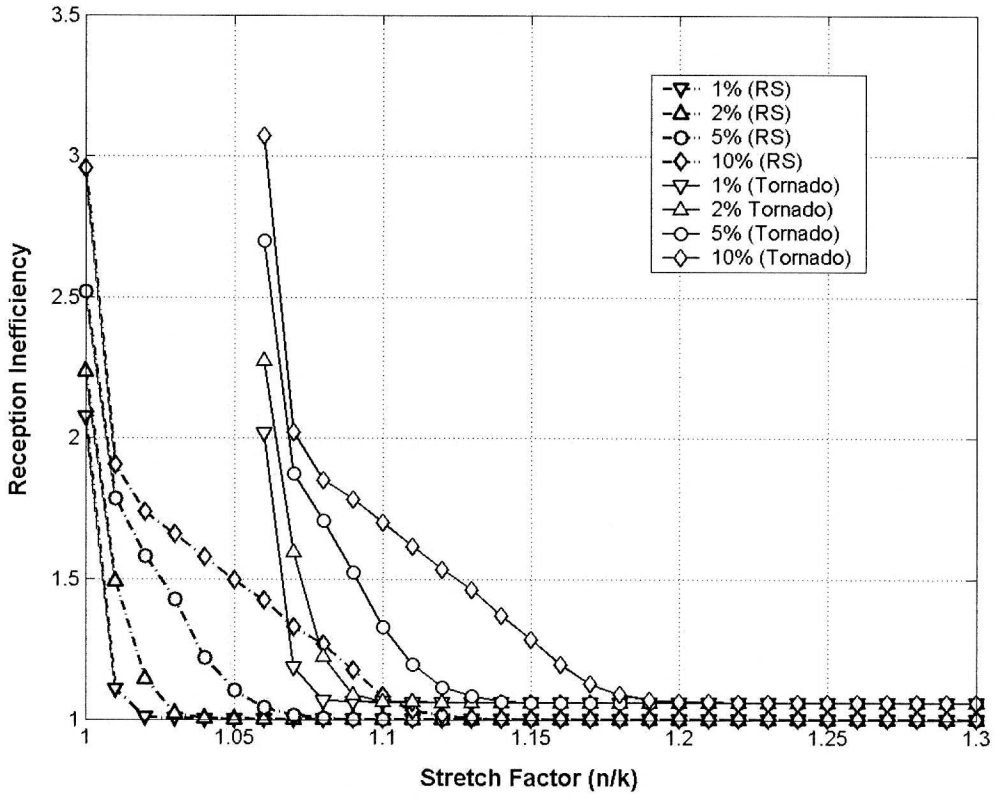


Figure 3.11.b Reception inefficiency vs. stretch factor. Packets arrive at equal rates from 2 servers with different loss rates using staggered technique.

4. Conclusions

In this thesis, a study of the performance of multiple mirror sites to speedup file downloads was presented. The high demand of speed for Internet file downloads and real time multimedia applications motivated the creation of this system. As the number of Internet receivers dramatically increases, and with the large number of multimedia and e-commerce applications, the deployment of multiple mirror sites proves to be a possible solution for efficient downloading. The tradeoff of bandwidth for speed is the reason Tornado codes have been proposed as they provide very fast decoding at the receiver due to their low decoding complexity. The only price to be paid is that Tornado codes require additional distinct packets at the receiver to accomplish successful decoding.

Specifically, the simulation results have shown that the time to acquire the extra distinct packets necessary to decode a Tornado code is quite sufficient to decode the message using standard Reed-Solomon codes. With the progress in VLSI technology and high processor speeds, most off-the-shelf RS decoders are capable of decoding a Reed-Solomon code in an extremely short time. This means that Reed-Solomon codes are the most suitable codes for erasure decoding in many Internet applications.

Moreover, randomly permuting the packets and using a very high stretch factor at the mirror servers is not the optimum solution to minimize packet duplication. A new technique that staggers the servers by assigning a starting point at every server has been shown to have a significant impact on the performance of the system. This technique can be enhanced to compensate for the inefficiency of slow servers by dividing the starting points in the packet sequence amongst the servers according to the individual data rates.

This technique significantly improves system speedup by dramatically reducing completion time. The inefficiency due to duplicate packets is improved at very high code rates; hence, the need to use a very high stretch factor is eliminated for most applications. In the presence of high losses, this technique was shown to be a robust solution to reliable delivery.

The significant performance of the staggered technique using standard Reed-Solomon erasure codes encourages future work. Our results are all based on a feedback free protocol; no attempt has been made to investigate the system with congestion control mechanisms. Hence, the implementation of our technique is worth studying on existing network infrastructure using TCP/IP.

Another interesting aspect is the deployment of the system in a wireless environment; the OPNET simulation can be extended to ad hoc networks models to determine the performance of the system with fading channels and limited bandwidth domains. OPNET now provides a wide variety of wireless scenarios with a wide range of applications.

Finally, an analytical study could be done to develop a mathematical model for the staggered technique with and without error control coding.

Bibliography

[1] C. Huitema, "The Case of Packet Level FEC," *Proc. IFIP Int. Workshop on Protocols for High Speed Networks,* pp. 109-120, Sophia Antipolis, France, Oct. 1996.

[2] W. Stallings, *Data & Computer Communications*, Prentice-Hall; 6th Ed., 1999.

[3] A.S. Tannenbaum, *Computer Networks*, Prentice-Hall; 4th Ed. 2002.

[4] S.B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice-Hall, 1995.

[5] S. Lin and D.J. Costello, Jr., *Error Control Coding*, Prentice Hall, 1982.

[6] Data Sheet for Reed-Solomon IP Core, 4i2i Communications Ltd., Aberdeen, UK, http://www.4i2i.com.

[7] Reed-Solomon Compiler Megacore Function User Guide, Version 1, ALTERA Corp., San Jose, CA, http:/www.altera.com, Jan. 2003.

[8] J.Blomer, M. Kalfane, M. Karpinski, M. Luby and D. Zuckerman," An XOR-Based Erasure-Resilient Coding Scheme," ICSI Technical Report No. TR-95-048, August 1995.

[9] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielmann and V. Stemmann, "Practical Loss-Resilient Codes." *In proceeding of the 29th ACM symposiumon Theory of Computing*, 1997

[10] S. Surana , "Tornado Codes", Scribe Notes based on lectures by Professor Bruce Magggs and on [9].

[11] E. Schooler and J. Gemmel, "Using Multicast FEC to Solve the Midnight Madness Problem," *Microsoft Research Technical Report MS-TR-97-25,* Sept. 1997.

[12] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," Computer Communications. Rev., vol. 27, pp. 24-36, Apr. 1997.

[13] J. Nonemacher and E.W. Biersack, 'Asynchronous Multicast Push AMP". *In Proc. of International Conference on Computer Communications*, Cannes, France, November 1997.

[14] L. Rizzo and L. Vicisano, "A Reliable Multicast Data Distribution Protocol Based on FEC Techniques," *Proc. IEEE HPCS,* Greece, June 1997.

[15] J. Byers, M. Luby, M. Mitzenmacher and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," *Proc. ACM SIGCOMM*, pp. 56-67, Vancouver, Canada, 1998.

[16] J. Byers, M. Luby and M. Mitzenmacher, "A Digital Fountain Approach to Reliable Asynchronous Multicast," *IEEE J. Selected Areas in Commun.*, vol. 20, pp. 1528-1540,Oct. 2002.

[17] S. Seshan, M. Stemm and R. H. Katz, "SPAND: Shared Passive Network Performance Discovery," Proc. *USENIX Symp. on Internet Technologies and Systems,* Monterey, CA, Dec. 1997.

[18] A. Myers, P. Dinda and H. Zhang, "Performance Characteristics of Mirror Servers on the Internet," *Proc. IEEE INFOCOM*, pp. 304-312, Mar. 1999.

[19] J. Byers, M. Luby and M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel; Using Tornado Codes to Speed up Downloads," *Proc. IEEE INFOCOM, pp. 275-283,* New York, NY, Mar. 1999.

[20] OPNET Technologies Inc. www.opnet.com , www.opnet.com/support/home1.html, http://enterprise16.opnet.com/support/cont_models.html

[21] Xinje Chang, "Network Simulations with OPNET", *Proc. Winter Simulation Conference*, Network Technology Research Center School of EEE, Nanyang Technological University, Singapore, 1999.

[22] OPNET Modeler, Product Documentation, OPNET Modeler versions, 8.0,8.1,9.0 and 9.1.A PL1, OPNET Technologies Inc. 2003.

[23] Advanced Modeler Training, Course Documentation, OPNET University Training Program, Santa Clara, CA. July, 2001