CHAPTER 1

INTRODUCTION

This chapter introduces to the problems regarding packet loss, congestion in the network followed by motivation and objectives of a new approach for forward error correction coding.

1.1 AN INTRODUCTION TO PROBLEM AREA

The packet transport service provided by representative packet-switched networks, including IP networks, is not reliable and the quality-of-service (QoS) cannot be guaranteed. Packets may be lost due to buffer overflow in switching nodes, be discarded due to excessive bit errors and failure to pass the cyclic redundancy check (CRC) at the link layer, or be discarded by network control mechanisms as a response to congestion somewhere in the network. Forward error correction (FEC) coding has often been proposed for end-to-end recovery from such packet losses. From an end user's perspective, FEC can help recover the lost packets in a timely fashion through the use of redundant packets, and generally adding more redundancy can be expected to improve performance provided this added redundancy does not adversely affect the network packet loss characteristics.

On the other hand, from the network's perspective, the widespread use of FEC schemes by end nodes will increase the raw packet-loss rate in a network because of the additional loads resulting from transmission of redundant packets. Therefore, in order to optimize the end-to-end performance, the appropriate tradeoff, in terms of the amount of redundancy added, and its effect on network packet-loss processes, needs to be investigated under specific and realistic modeling assumptions.

We provide a study of the overall effectiveness of packet-level FEC coding, employing interlaced Reed-Solomon codes, in combating network packet losses and provide an information-theoretic methodology for determining the optimum compromise between end-to-end

performance and the associated increase in raw packet-loss rates using a realistic model-based analytic approach. Intuitively, for a given choice of block length we expect that there is an optimum choice of redundancy, or channel coding rate, since a rate too high (low redundancy) is simply not powerful enough to effectively recover packet losses while a rate too low (high redundancy) results in excessive raw packet losses due to the increased overhead which overwhelms the packet recovery capabilities of the FEC code. The optimum channel coding rate results in an optimum compromise between the set woo effects.

In this work, we focus on evaluating the capability of FEC in recovering packet losses over IP networks using residual packet-loss rate as the performance measure. In terms of characterizing end-to-end performance, we assume that performance is directly proportional to the source coding rate, or network load, that can be supported for a fixed residual packet-loss rate. The analytic procedure developed is then used to determine the maximum load that can be supported as a function of coding parameters.

By modeling the fully interleaved network transport channel as a block interference channel (BIC), we provide information theoretic bound on the performance achievable with FEC. This bound provides a useful context for assessing the efficacy of FEC in this application as a function of coding parameters.

1.2 EXISTING SYSTEM

For analysis purposes the packet-loss process resulting from the single-multiplexer model was assumed to be independent and, consequently, the simulation results provided show that this

simplified analysis considerably overestimates the performance of FEC. In existing system a recursive algorithm to compute the packet-loss statistics (block error density), through which the 2exact residual packet-loss rate after decoding was computed. Surprisingly, all numerical results given indicates that the resulting residual packet-loss rates with coding are always greater than without coding, i.e., FEC is ineffective in this application. The increase in the redundant packets added to the data will increase the performance, but it will also make the data large and it will also lead to increase in data loss.

1.3 MOTIVATION

Until recently, most known decoding algorithms for convolutional codes were based on either algebraically calculating the error pattern or on trellis graphical representations such as in the MAP and Viterbi algorithms. With the advent of turbo coding , a third decoding principle has appeared: iterative decoding. Iterative decoding was also introduced in Tanner's pioneering work , which is a general framework based on bipartite graphs for the description of LDPC codes and their decoding via the belief propagation (BP) algorithm.

In many respects, convolutional codes are similar to block codes. For example, if we truncate the trellis by which a convolutional code is represented, a block code whose codewords correspond to all trellis paths to the truncation depth is created. However, this truncation causes a problem in error performance, since the last bits lack error protection. The conventional solution to this problem is to encode a fixed number of message blocks L followed by m additional all-zero blocks, where m is the constraint length of the convolutional code.

This method provides uniform error protection for all information digits, but causes a rate reduction for the block code as compared to the convolutional code by the multiplicative factor L/(L+m). In the tail-biting convolutional code, zero-tail bits are not needed and replaced by payload bits resulting in no rate loss due to the tails. Therefore, the spectral efficiency of the channel code is improved. Due to the advantages of the tail-biting method over the zero-tail, it has been adopted as the FEC in addition to the turbo code for data . The conventional solution to

this problem is to encode a fixed number of message blocks L followed by m additional all-zero blocks, where m is the constraint length of the convolutional code

1.4 TECHNIQUES USED

FEC is accomplished by adding redundancy to the transmitted information using a predetermined algorithm. Each redundant bit is invariably a complex function of many original information bits. An extremely simple example would be an analog to digital converter that samples three bits of signal strength data for every bit of transmitted data. If the three samples are mostly all zero, the transmitted bit was probably a zero, and if three samples are mostly all one, the transmitted bit was probably a one. The simplest example of error correction is for the receiver to assume the correct output is given by the most frequently occurring value in each group of three.

1.5 OBJECTIVE

In this project we are going to evaluate the efficacy of FEC coding. To evaluate it we are going to transfer data from Source to Destination. Before receiving the data in the Destination we create packet loss in Queue. Thus after creating packet loss, we receive the remaining packets in the Destination. Then we recover the lost Packets in the Destination and evaluate FEC's performance.

1.6 SUMMARY

In this chapter ,the concept of forward error correction coding are introduced. The existing system is introduced .The techniques used. The objectives and motivation behind the project are also mentioned.

CHAPTER 2

LITERATURE SURVEY

Our project makes use of socket programming, and requires configuration to run on JDK 6.0.10. The Java foundation classes is a group of packages used to enhance GUI design.

2.1 THE JAVA PROGRAMMING LANGUAGE

Java is two things: a programming language and a platform .The Java Programming Language. Java is a high-level programming language that is with the following features:

- Simple
- Object-oriented
- Distributed
- Interpreted
- Robust
- Secure
- Architecture-neutral

- Portable
- High-performance
- Multithreaded
- Dynamic

Java is also unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called Java byte codes--the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java

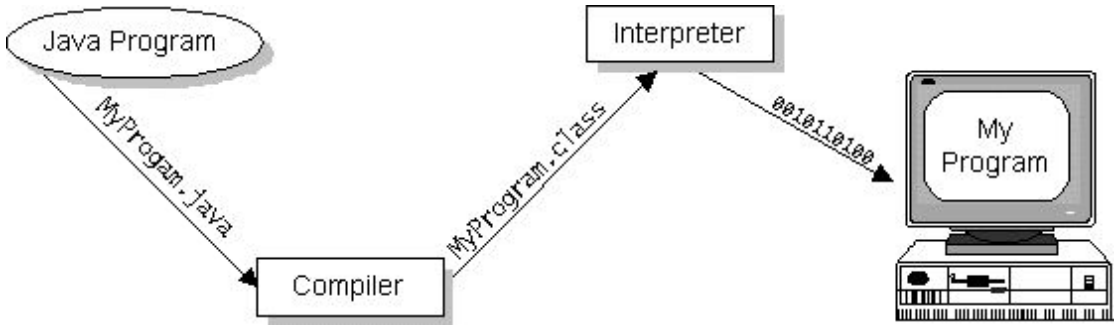


Fig 2.1 Working of a java program.

byte code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how this works. Java byte codes can be considered as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java byte codes help make "write once, run anywhere" possible. The Java program can be compiled into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh.

2.1.1 THE JAVA PLATFORM

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system. The Java platform has two components: The Java Virtual Machine (Java VM) and The Java Application Programming Interface (Java API)

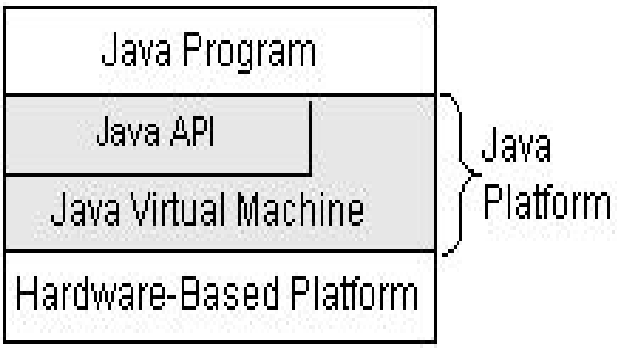


Fig 2.2 The java platform

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (packages) of related components. The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring Java's performance close to that of native code without threatening portability. Probably

the most well-known Java programs are Java applets. An applet is a Java program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, Java is not just for writing cute, entertaining applets for the World Wide Web ("Web"). Java is a general-purpose, high-level programming language and a powerful software platform. Using the generous Java API, we can write many types of programs. The most common types of programs are probably applets and applications, where a Java application is a standalone program that runs directly on the Java platform. With packages of software components that provide a wide range of functionality. The core API is the API included in every full implementation of the Java platform. The core API gives you the following features:

- The Essentials: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- Applets: The set of conventions used by Java applets.
- Networking: URLs, TCP and UDP sockets, and IP addresses.
- Internationalization: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- Security: Both low-level and high-level, including electronic signatures, public/private key management, access control, and certificates.
- Software components: Known as JavaBeans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture, OpenDoc, and Netscape's Live Connect.
- Object serialization: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- Java Database Connectivity (JDBC): Provides uniform access to a wide range of relational databases.

- Java not only has a core API, but also standard extensions. The standard extensions define APIs for 3D, servers, collaboration, telephony, speech, animation, and more.

Java is likely to make your programs better and requires less effort than other languages. We believe that Java will help you do the following:

- Get started quickly: Although Java is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- Write less code: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in Java can be four times smaller than the same program in C++.
- Write better code: The Java language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Java's object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.
- Develop programs faster: Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code with Java and Java is a simpler programming language than C++.
- Avoid platform dependencies with 100% Pure Java: You can keep your program portable by following the purity tips mentioned throughout this book and avoiding the use of libraries written in other languages.

Write once, run anywhere: Because 100% Pure Java programs are compiled into machine-independent byte codes, they run consistently on any Java platformDistribute software more easily: You can upgrade applets easily from a central server. Applets take advantage of the Java feature of allowing new classes to be loaded "on the fly," without recompiling the entire program. We explore the **java.net** package, which provides support for networking. Its creators have called Java “programming for the Internet.” These networking classes encapsulate the “socket” paradigm pioneered in the Berkeley Software Distribution (BSD) from the University of California at Berkeley.

2.2 NETWORKING BASICS

Ken Thompson and Dennis Ritchie developed UNIX in concert with the C language at Bell Telephone Laboratories, Murray Hill, New Jersey, in 1969. In 1978, Bill Joy was leading a project at Cal Berkeley to add many new features to UNIX, such as virtual memory and full-screen display capabilities. By early 1984, just as Bill was leaving to found Sun Microsystems, he shipped 4.2BSD, commonly known as Berkeley UNIX. 4.2BSD came with a fast file system, reliable signals, interprocess communication, and, most important, networking.

The networking support first found in 4.2 eventually became the de facto standard for the Internet. Berkeley's implementation of TCP/IP remains the primary standard for communications with the Internet. The socket paradigm for interprocess and network communication has also been widely adopted outside of Berkeley.

2.3 SOCKET OVERVIEW

A *network socket* is a lot like an electrical socket. Various plugs around the network have a standard way of delivering their payload. Anything that understands the standard protocol can "plug in" to the socket and communicate. *Internet protocol (IP)* is a low-level routing protocol that breaks data into small packets and sends them to an address across a network, which does not guarantee to deliver said packets to the destination.

Transmission Control Protocol (TCP) is a higher-level protocol that manages to reliably transmit data. A third protocol, *User Datagram Protocol (UDP)*, sits next to TCP and can be used directly to support fast, connectionless, unreliable transport of packets.

2.3.1 CLIENT/SERVER

A *server* is anything that has some resource that can be shared. There are *compute servers*, which provide computing power; *print servers*, which manage a collection of printers; *disk*

servers, which provide networked disk space; and *web servers*, which store web pages. A *client* is simply any other entity that wants to gain access to a particular server.

In Berkeley sockets, the notion of a socket allows as single computer to serve many different clients at once, as well as serving many different types of information. This feat is managed by the introduction of a *port*, which is a numbered socket on a particular machine. A server process is said to "listen" to a port until a client connects to it. A server is allowed to accept multiple clients connected to the same port number, although each session is unique. To mange multiple client connections, a server process must be multithreaded or have some other means of multiplexing the simultaneous I/O.

- **RESERVED SOCKETS**

Once connected, a higher-level protocol ensues, which is dependent on which port you are using. TCP/IP reserves the lower, 1,024 ports for specific protocols. Port number 21 is for FTP, 23 is for Telnet, 25 is for e-mail, 79 is for finger, 80 is for HTTP, 119 is for Netnews-and the list goes on. It is up to each protocol to determine how a client should interact with the port.

- **INETADDRESS**

The **InetAddress** class is used to encapsulate both the numerical IP address and the domain name for that address. We interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The **InetAddress** class hides the number inside. As of Java 2, version 1.4, **InetAddress** can handle both IPv4 and IPv6 addresses.

- **FACTORY METHODS**

The **InetAddress** class has no visible constructors. To create an **InetAddress** object, we use one of the available factory methods. *Factory methods* are merely a convention whereby static methods in a class return an instance of that class. This is done in lieu of overloading a constructor with various parameter lists when having unique method names makes the results much clearer. Three commonly used **InetAddress** factory methods are shown here.

static InetAddress getLocalHost( ) throws UnknownHostException

static InetAddress getByName(String hostName) throws UnknowsHostException

static InetAddress[ ] getAllByName(String hostName)

throws UnknownHostException

The **getLocalHost( )** method simply returns the **InetAddress** object that represents the local host. The **getByName( )** method returns an **InetAddress** for a host name passed to it. If these methods are unable to resolve the host name, they throw an **UnknownHostException**.

On the internet, it is common for a single name to be used to represent several machines. In the world of web servers, this is one way to provide some degree of scaling. The **getAllByName ( )** factory method returns an array of **InetAddress**es that represent all of the addresses that a particular name resolves to. It will also throw an **UnknownHostException** if it can't resolve the name to at least one address. Java 2, version 1.4 also includes the factory method **getByAddress( )**, which takes an IP address and returns an **InetAddress** object. Either an IPv4 or an IPv6 address can be used.\

- **INSTANCE METHODS**

The **InetAddress** class also has several other methods, which can be used on the objects returned by the methods just discussed. Here are some of the most commonly used. Boolean equals (Object other)-Returns **true** if this object has the same Internet address as other.

- ➢ byte[ ] getAddress( )- Returns a byte array that represents the object's Internet address in network byte order.
- ➢ String getHostAddress( )- Returns a string that represents the host address associated with the **InetAddress** object.

- ➢ String getHostName( )- Returns a string that represents the host name associated with the **InetAddress** object.
- ➢ boolean isMulticastAddress( )- Returns **true** if this Internet address is a multicast address. Otherwise, it returns **false**.
- ➢ String toString( )- Returns a string that lists the host name and the IP address for conveneince.

Internet addresses are looked up in a series of hierarchically cached servers. That means that your local computer might know a particular name-to-IP-address mapping autocratically, such as for itself and nearby servers. For other names, it may ask a local DNS server for IP address information. If that server doesn't have a particular address, it can go to a remote site and ask for it. This can continue all the way up to the root server, called InterNIC (internic.net).

- **TCP/IP CLIENT SOCKETS**

TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point, stream-based connections between hosts on the Internet. A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.

There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients. The **ServerSocket** class is designed to be a "listener," which waits for clients to connect before doing anything. The **Socket** class is designed to connect to server sockets and initiate protocol exchanges. The creation of a **Socket** object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection. Here are two constructors used to create client sockets:

Socket(String *hostName*, int *port*) Creates a socket connecting the local host to the named host and port; can throw an UnknownHostException or anIOException.

Socket(InetAddress *ipAddress*, int *port*) Creates a socket using a preexisting **InetAddress** object and a port; can throw an **IOException**.

A socket can be examined at any time for the address and port information associated with it, by use of the following methods:

- InetAddress getInetAddress( )- Returns the **InetAddress** associated with the Socket object.
- int getPort( )-Returns the remote port to which this **Socket** object is connected.
- int getLocalPort( ) Returns the local port to which this **Socket** object is connected.
- InputStream getInputStream( )Returns the **InputStream** associated with the invoking socket.
- OutputStream getOutputStream( ) Returns the **OutputStream** associated with the invoking socket.

Once the **Socket** object has been created, it can also be examined to gain access to the input and output streams associated with it. Each of these methods can throw an **IOException** if the sockets have been invalidated by a loss of connection on the Net.

- **TCP/IP SERVER SOCKETS**

Java has a different socket class that must be used for creating server applications. The **ServerSocket** class is used to create servers that listen for either local or remote client programs to connect to them on published ports. **ServerSockets** are quite different form normal **Sockets**.

When we create a **ServerSocket**, it will register itself with the system as having an interest in client connections. The constructors for **ServerSocket** reflect the port number that we wish to accept connection on and, optionally, how long we want the queue for said port to be. The queue length tells the system how many client connection it can leave pending before it should simply

refuse connections. The default is 50. The constructors might throw an IOException under adverse conditions. Here are the constructors:

ServerSocket(int port) Creates server socket on the specified port with a queue length of 50.

Serversocket(int *port*, int *maxQueue*)-Creates a server socket on the specified *port* with a maximum queue length of *maxQueue*.

ServerSocket(int *port*, int *maxQueue*, InetAddress *localAddress*)-Creates a server socket on the specified *port* with a maximum queue length of *maxQueue*. On a multihomed host, *localAddress* specifies the IP address to which this socket binds.

ServerSocket has a method called **accept( )**, which is a blocking call that will wait for a client to initiate communications, and then return with a normal **Socket** that is then used for communication with the client.

2.3.2 URL

The Web is a loose collection of higher-level protocols and file formats, all unified in a web browser. One of the most important aspects of the Web is that Tim Berners-Lee devised a scalable way to locate all of the resources of the Net. The Uniform Resource Locator (URL) is used to name anything and everything reliably.

The *URL* provides a reasonably intelligible form to uniquely identify or address information on the Internet. URLs are ubiquitous; every browser uses them to identify information on the Web. Within Java's network class library, the URL class provides a simple, concise API to access information across the Internet using URLs. Within Java's network class library, the URL class provides a simple, concise API to access information across the Internet using URLs.

- **FORMAT**

A URL specification is based on four components. The first is the protocol to use, separated from the rest of the locator by a colon (:). Common protocols are http, ftp, gopher, and file, although these days almost everything is being done via HTTP. The second component is the host name or IP address of the host to use; this is delimited on the left by double slashes (/ /) and on the right by a slash (/) or optionally a colon (:) and on the right by a slash (/). The fourth part is the actual file path. Most HTTP servers will append a file named index.html or index.htm to URLs that refer directly to a directory resource. **http;//www.osborne.com/** and **http:// www.osborne.com:80/index.htm** are two examples.

Java's **URL** class has several constructors, and each can throw a **MalformedURLException**. One commonly used form specifies the URL with a string that is identical to what is displayed in a browser: URL(String *urlSpecifier*)

The next two forms of the constructor breaks up the URL into its component parts:

- URL(String *protocolName*, String *hostName*, int port, String *path*)
- URL(String *protocolName*, String hostName, String *path*)

Another frequently used constructor uses an existing URL as a reference context and then create a new URL from that context.URL(URL *urlObj*, String *urlSpecifier*)

The following method returns a **URLConnection** object associated with the invoking URL object. it may throw an **IOException**.URLConnection openConnection( )-It returns a

URLConnection object associated with the invoking URL object. it may throw an **IOException**.

2.3 JDBC

In an effort to set an independent database standard API for Java, Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMS. This consistent interface

is achieved through the use of "plug-in" database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC's framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

- **JDBC GOAL**

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

➢ **SQLConformance**

SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.

➢ **JDBC must be implemental on top of common database interfaces**

The JDBC SQL API must "sit" on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.

➢ **Provide a Java interface that is consistent with the rest of the Java system**

Because of Java's acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.

➢ **Keep it simple**

This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.

- **Use strong, static typing wherever possible**

  Strong typing allows for more error checking to be done at compile time; also, less errors appear at runtime.

- **Keep the common cases simple**

  Because more often than not, the usual SQL calls used by the programmer are simple SELECT's, INSERT's, DELETE's and UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

2.4 SUMMARY

This chapter deals with the java programming language. Why java is necessary , what is socket programming and detailed description of those.

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATIONS

System analysis is a detailed study of various operations performed by the system and relations which exists inside and outside the system. The main objective of the feasibility study is to test operational ,technical feasibility to develop the project maintenance system.

3.1 PROJECT REQUIREMENTS

The requirements of the project are divided into various sections and are given below:

3.1.1 HARDWARE REQUIREMENTS:

Processor	:	Pentium Iv 2.6 Ghz
Ram	:	512 Mb
Monitor	:	15”
Hard Disk	:	20 Gb
Cddrive	:	52x
Keyboard	:	Standard 102 Keys
Mouse	:	3 Buttons

3.1.2 SOFTWARE REQUIREMENTS:

Front End	:	Java, Swing
Tools Used	:	Jframe Builder

Operating System : Windows Xp

3.1.3 FUNCTIONAL REQUIREMENTS SPECIFICATION

- **DESCRIPTION**

In this module we receive the data from the Source system. This data is the blocks after FEC Encoding and Interleaving processes are done. These blocks come from the Source system through Server Socket and Socket. Server socket and Socket are classes available inside Java. These two classes are used to create a connection between two systems inside a network for data transmission. After we receive the packets from Source, we create packet loss. Packet loss is a process of deleting the packets randomly. After creating loss we send the remaining blocks to the Destination through the socket connection.

- **INPUTS**

The input is a file containing text messages. The text available in the input text file is converted into binary. The binary conversion is done for each and every character in the input file.

- **SOURCE**

FEC is a system of error control for data transmission, where the sender adds redundant data to its messages. This allows the receiver to detect and correct errors (within some bounds) without the need to ask the sender for additional data. In this module we add redundant data to the given input data, known as FEC Encoding. The text available in the input text file is converted into binary. The binary conversion is done for each and every character in the input file.

- **OUTPUTS**

This module gets the input from the De-Interleaver. The received packets are processed to remove the original bits from it. Thus we recover the original bits of a character in this

module. After retrieving the original bits, we convert it to characters and write it inside a text file.

- **DESTINATION**

In the destination we receive the packets. The received packets are processed to remove the original bits from it. Thus we recover the original bits of a character in this module. After retrieving the original bits, we convert it to characters and write it inside a text file.

- **SYSTEM GOAL**

In this system we are going to evaluate the efficacy of FEC coding. To evaluate it we are going to transfer data from Source to Destination. Before receiving the data in the Destination we create packet loss in Queue. Thus after creating packet loss, we receive the remaining packets in the Destination. Then we recover the lost Packets in the Destination and evaluate FEC's performance.

3.2 SUMMARY

The System specification chapter specifies the various hardware and software requirements for our project .It also describes the various functional requirements for the same.

CHAPTER 4

SYSTEM DESIGN

In this chapter ,we illustrate the design considerations, the various dataflow diagrams and usecase diagrams.

4.1 DESIGN CONSIDERATIONS

The word system is possibly the most overused and abused term in the technical lexicon. System can be defined as the "a set of fact, principles, rules etc., classified and arranged in an orderly form so as to show a logical plan linking the various parts" here the system design defines the computer based information system. The primary objective is to identify user requirements and to build a system that satisfies these requirements.

Design is much more creative process than analysis. Design is the first step in the development of any system or product. Design can be defined as "the process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization".

It involves four major steps, they are

- Understanding how the system is working now;
- Finding out what the system does now;
- Understanding what the new system will do; and
- Understanding how the new system will work.

So as to avoid these difficulties, a new system was designed to keep these requirements in mind. Therefore the manual process operation has been changed into GUI based environment, such that the user can retrieve the records in a user-friendly manner and it is very easy to navigate to the corresponding information.

- **INPUT DESIGN**

Input design is the bridge between users and information system. It specifies the manner in which data enters the system for processing it can ensure the reliability of the system and produce reports from accurate data or it may results in output of error information.

- **OUTPUT DESIGN**

Outputs from the computer system are rewired primary to communicate the results of processing to the uses. They also used to provide a permanent copy of these results for later consultation / verification. The main points on designing an output are deciding the media, designing layout and report to be printed. The outputs are designed from the system, are simple to read and interpreted.

- **USER INTERFACE**
  - ➢ **Swing**

Swing is a set of classes that provides more powerful and flexible components that are possible with AWT. In addition to the familiar components, such as button checkboxes and labels, swing supplies several exciting additions, including tabbed panes, scroll panes, trees and tables.

  - ➢ **Applet**

Applet is a dynamic and interactive program that can run inside a web page displayed by a java capable browser such as hot java or Netscape.

4.2 DATA FLOW DIAGRAM

A DFD is a logical model of the system. The model does not depend on the hardware, software and data structures of file organization. It tends to be easy for even non-technical users to understand and thus serves as an excellent communication tool.

DFD can be used to suggest automatic boundaries for proposed system at pa very high level; the entire system is shown as a single logical process clearly identifying the sources and destination of data. This is often referred to as zero level DFD.

Then the processing is exploded into major processes and the same is depicted as level one DFD.

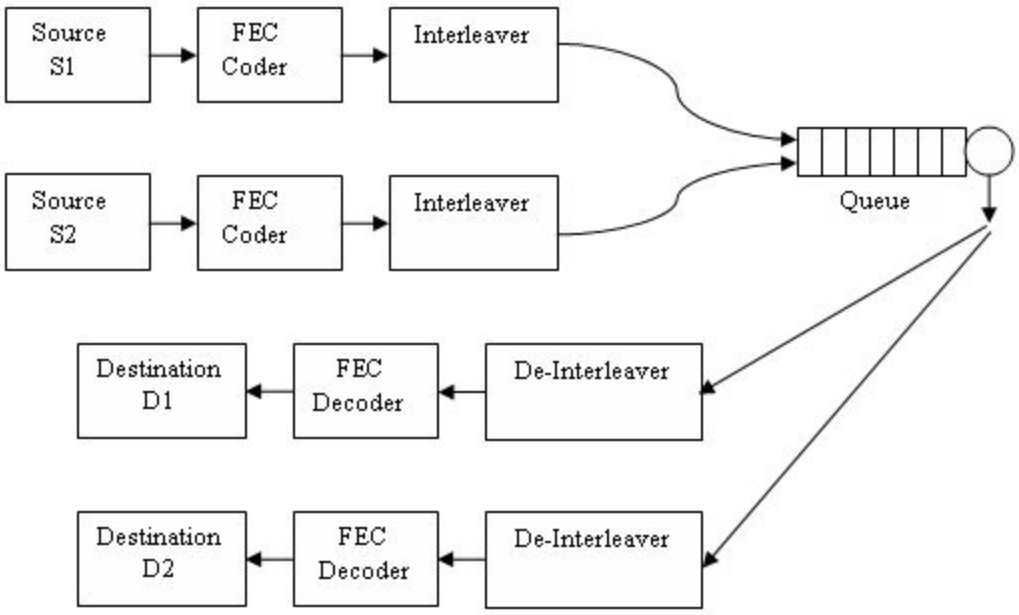


Fig 4.1 Data Flow Diagram

4.3 USE CASE DIAGRAM

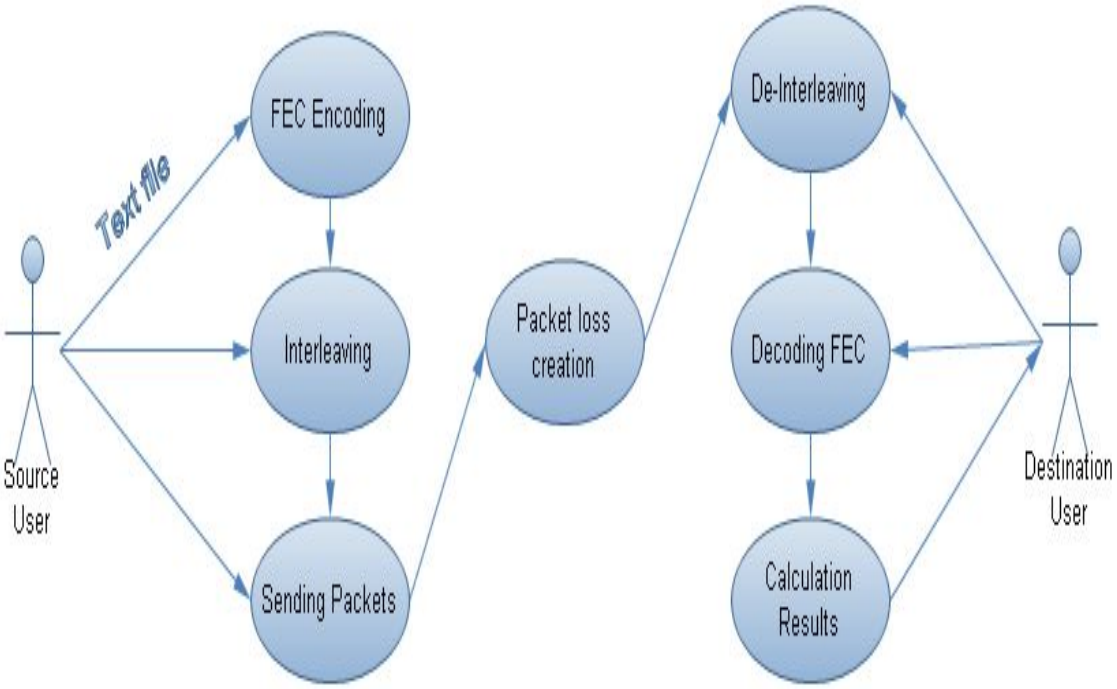


Fig 4.2 Use case diagram

Use Case (a case in the use of a system) is a list of steps, typically defining interactions between a role (known in UML as an "actor") and a system, to achieve a goal. The actor can be a human or an external system. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. In the Unified Modeling Language the relationships between all (or a set of) the use cases and actors are represented in a Use Case diagrams.

4.4 CLASS DIAGRAM

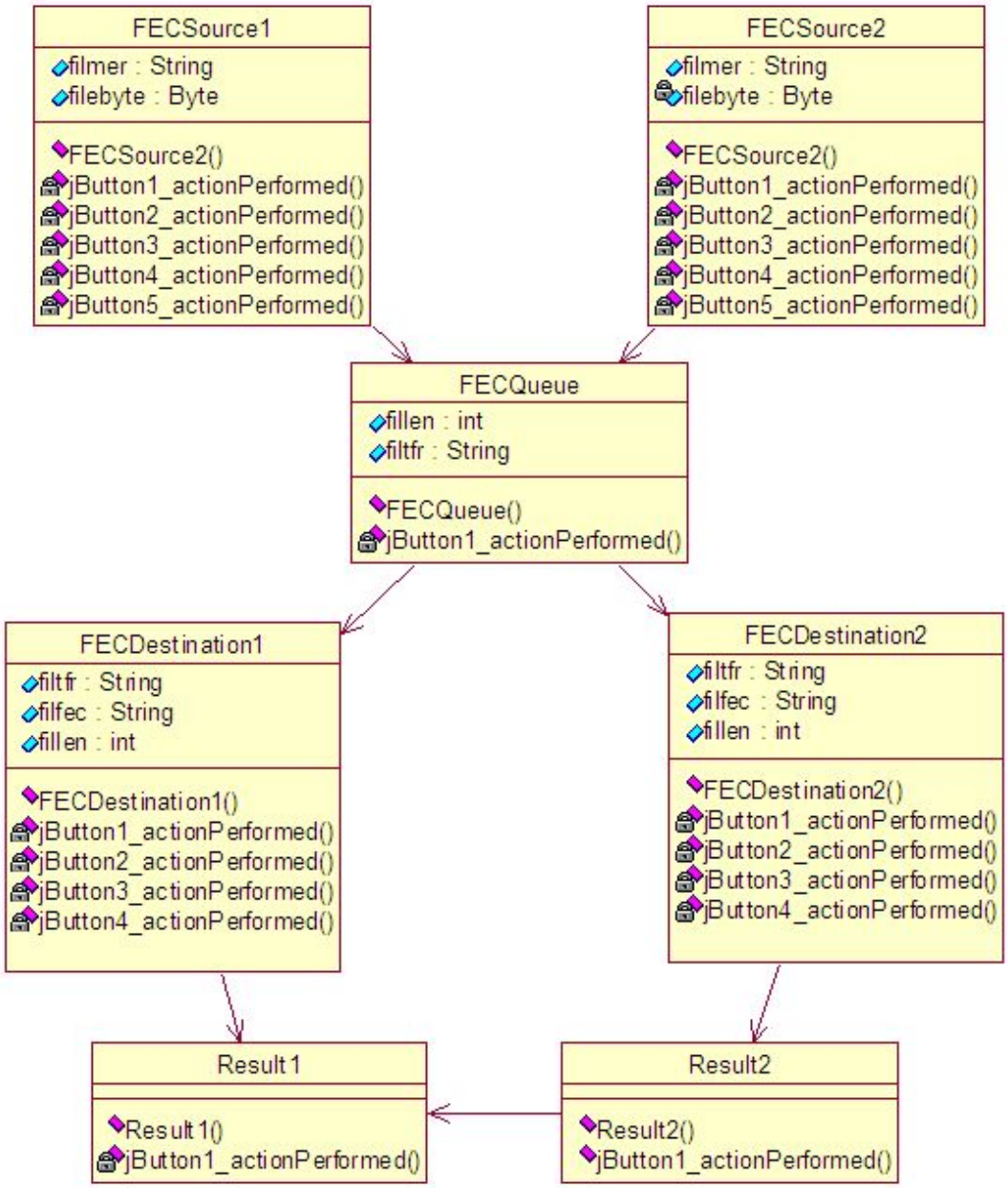


Fig 4.3 Class diagram

4.5 STATE DIAGRAM

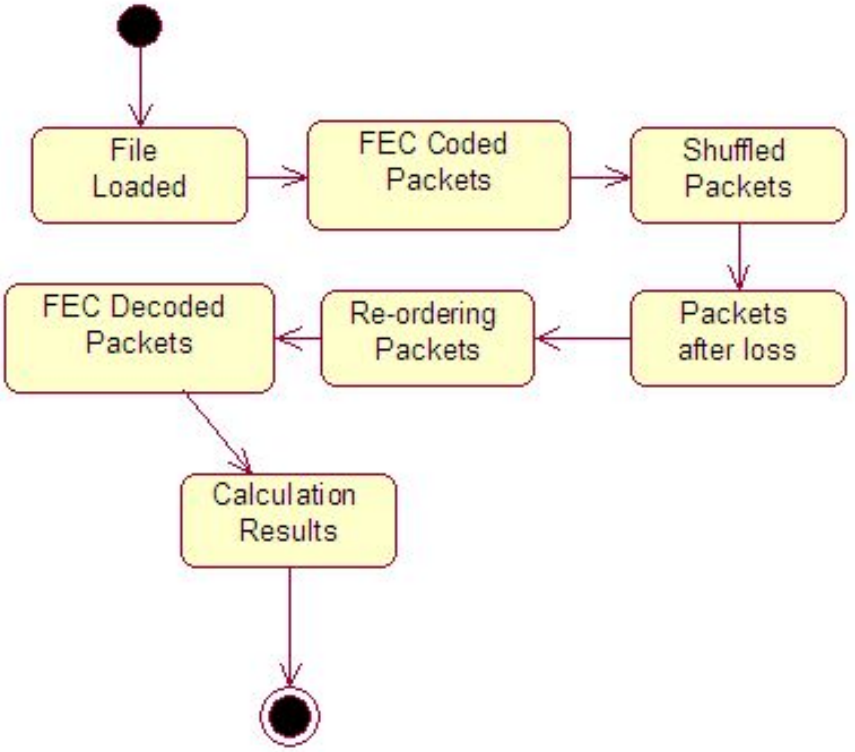
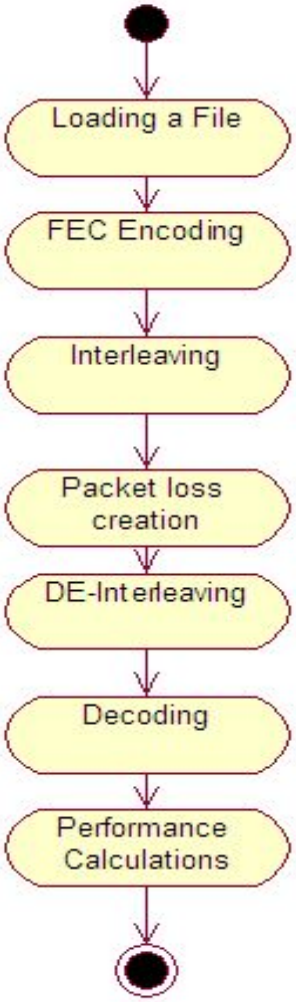


Fig 4.4 A State diagram

State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics**.** State diagrams are used to give an abstract description of the behavior of a system. This behavior is analyzed and represented in series of events that could occur in one or more possible states. Hereby "each diagram usually represents objects of a single class and track the different states of its objects through the system. State diagrams can be used to graphically represent finite state machines**.**

4.6 ACTIVITY DIAGRAM



.

Fig 4.5 Activity diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language,

activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

4.7 COLLABORATION DIAGRAM

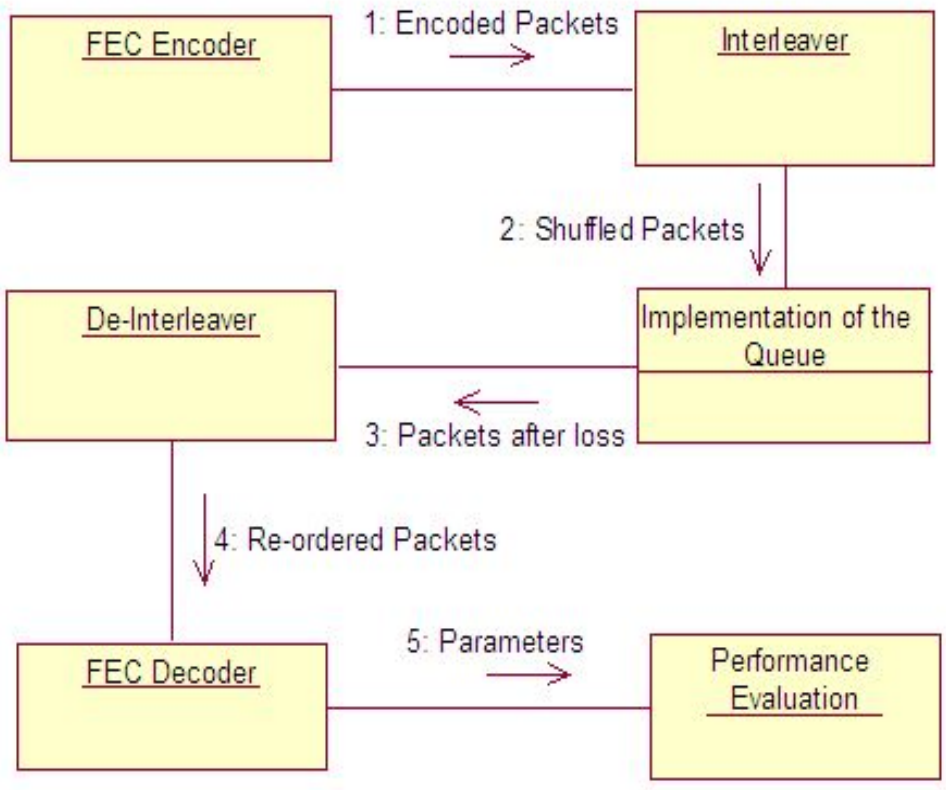


Fig 4.6 Collaboration diagram

A collaboration diagram models the interactions between objects or parts in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system. Collaboration diagrams show a lot of the same information as sequence

diagrams, but because of how the information is presented, some of it is easier to find in one diagram than the other. Collaboration diagrams show which elements each one interacts with better, but sequence diagrams show the order in which the interactions take place more clearly.

4.8 SEQUENCE DIAGRAM

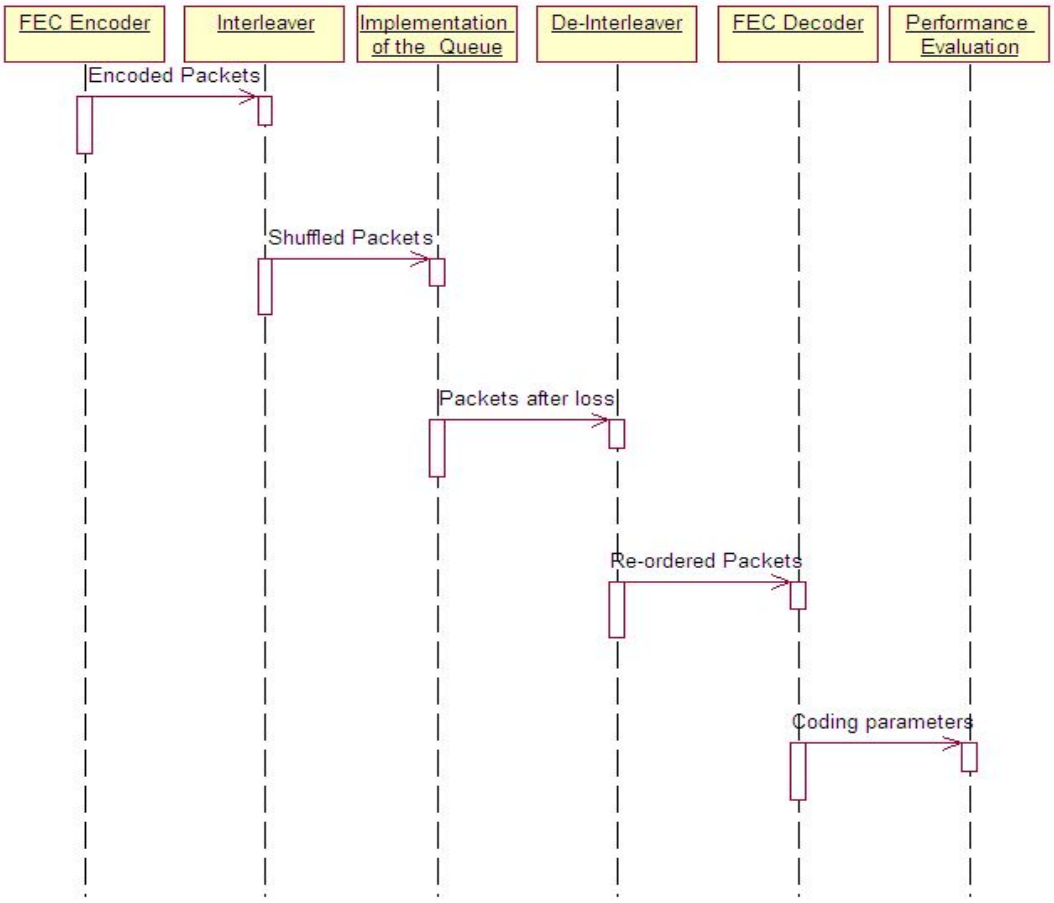


Fig 4.7 Sequence diagram

A **sequence diagram** in a UML is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the

objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams typically are associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

4.10 SYSTEM ARCHITECTURE DIAGRAM

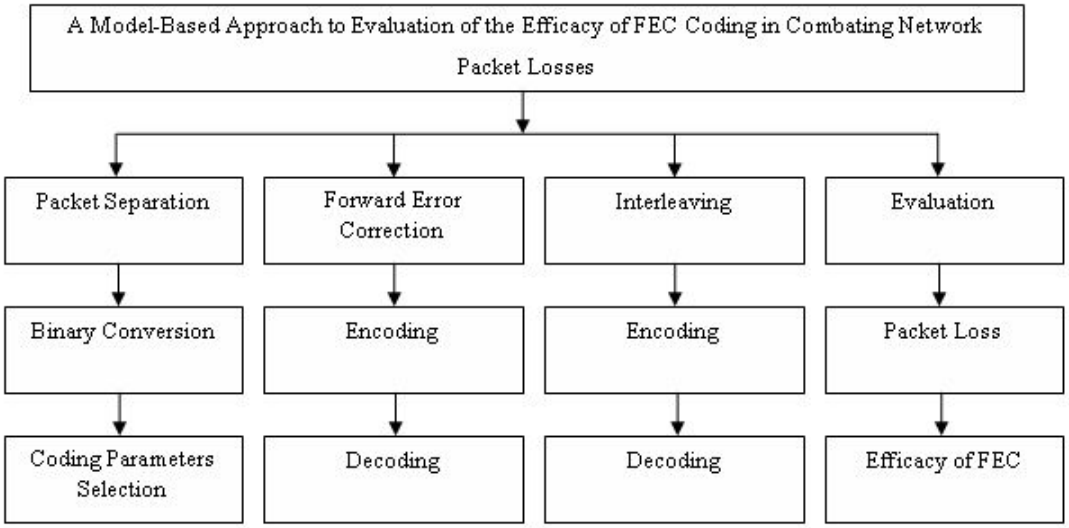


Fig 4.8 System architecture diagram

4.11 SUMMARY

The chapter gives detail of the design considerations. Data flow diagram modules are presented here.

CHAPTER 5

IMPLEMENTATION

This chapter provides implementation information of our project, it also deals with different modules of the system and their implementation.

5.1 GIVEN INPUT AND EXPECTED OUTPUT FOR EACH MODULE

- **ENCODING**

  Given Input: Text file.

  Expected Output: Packets coded with redundant data.

- **INTERLEAVING**

  Given Input: Packets coded with redundant data.

  Expected Output: Packets shuffled inside every block of data.

- **SENDING PACKETS**

  Given Input: Shuffled packets

  Expected Output: Packets passed successfully (other than lost packets)

- **DEINTERLEAVING**

  Given Input: Packets from the queue

  Expected Output: Packets re-ordered like it was before Interleaving

- **DECODING**

  Given Input: Packets from De-Interleaver

  Expected Output: Original packets

5.2 MODULE DIAGRAM

1. FEC ENCODER:

FEC is a system of error control for data transmission, where the sender adds redundant data to its messages. This allows the receiver to detect and correct errors (within some bounds) without the need to ask the sender for additional data. In this module we add redundant data to the given input data, known as FEC Encoding.

The text available in the input text file is converted into binary. The binary conversion is done for each and every character in the input file. Then we add the redundant data for each bit of the binary. After adding we have a block of packets for each character. The User Interface design is also done in this module. We use the Swing package available in Java to design the User Interface. Swing is a widget toolkit for Java.

Fig 5.1 FEC encoder diagram

2. **INTERLEAVER:**

Fig 5.2 Interleaver diagram

Interleaving is a way of arranging data in a non-contiguous way in order to increase performance. It is used in data transmission to protest against burst errors. In this module we arrange the data (shuffling) to avoid burst errors which is useful to increase the performance of FEC Encoding.

This module gets the input as blocks of bits from the FEC Encoder. In this module we shuffle the bits inside a single block in order to convert burst errors into random errors. This shuffling process is done for each and every block comes from the FEC Encoder. Then we create

a Socket connection to transfer the blocks from Source to the Queue. This connection is created by using the Server Socket and Socket class Available in Java.

3. IMPLEMENTATION OF THE QUEUE:

Fig 5.3 Implementation of the queue diagram

In this module we receive the data from the Source system. This data is the blocks after FEC Encoding and Interleaving processes are done. These blocks come from the Source system through Server Socket and Socket. Server socket and Socket are classes available inside Java. These two classes are used to create a connection between two systems inside a network for data transmission. After we receive the packets from Source, we create packet loss. Packet loss is a process of deleting the packets randomly. After creating loss we send the remaining blocks to the Destination through the socket connection.

4. DE-INTERLEAVER:

This module receives the blocks of data from the Queue through the socket connection. These blocks are the remaining packets after the loss in the Queue. In this module we re arrange the data packets inside a block in the order in which it is before Interleaving. This process of

Interleaving and De-Interleaving is done to convert burst errors into random errors. After De-Interleaving the blocks are arranged in the original order. Then the data blocks are sent to the FEC Decoder.

Fig 5.4 De-interleaver diagram

5. FEC DECODER:

Fig 5.5 FEC decoder diagram

This module gets the input from the De-Interleaver. The received packets are processed to remove the original bits from it. Thus we recover the original bits of a character in this module. After retrieving the original bits, we convert it to characters and write it inside a text file.

6. PERFORMANCE EVALUATION:

In this module we calculate the overall performance of FEC Coding in recovering the packet losses. After retrieving the original bits, we convert it to characters and write it inside a text file. This performance is calculated by using the coding parameters like Coding rate, Interleaving depth, Block length and several other parameters. First we calculate the amount of packet loss and with it we use various formulas to calculate the overall performance of Forward Error Correction in recovering the network packet losses.

Fig 5.6 Performance evaluation diagram

5.3 CODING

```
/*****************************************************************/
/*                  FECDestination                               */
/*                                                               */
/*****************************************************************/
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.*;
import java.io.*;
import java.net.*;
/**
 * Summary description for FECClient
 *
 */
public class FECDestination1 extends JFrame
{
        // Variables declaration
        private JLabel jLabel1;
        private JLabel jLabel2;
        private JLabel jLabel3;
        private JLabel jLabel4;
        private JTextField jTextField1;
        private JTextArea jTextArea1;
        private JScrollPane jScrollPane1;
        private JProgressBar jProgressBar1;
        //private JButton jButton1;
        private JButton jButton2;
```

```
private JButton jButton3;

private JButton jButton4;

private JButton jButton5;

private JPanel contentPane;

public String Fien="";

public String Dest1="";

public int i,j;

public int ch;

public static int fillen;

public int intfec[];

public int intval[];

public char chfec[];


public char pakch1[][];

public char filreord[][];

public String[] filtfr;

public String filfec[];

ServerSocket ss;

Socket so;

//ServerSocket ss1;
```

```
//Socket so1;
public float codrt;
public float bklen;
public float intdth,orgblk,Per,Effy;
public float k,Probjn,n,ENI,PLReff;
// End of variables declaration

//String rec="";
public FECDestination1()
{
        super();
        initializeComponent();
        //
        // TODO: Add any constructor code after initializeComponent call
        //

        this.setVisible(true);
        try
        {
                ss=new ServerSocket(4501);
```

```
while(true)
{
        so=ss.accept();
        jTextArea1.setText("\n   Packets Recieving Started");
        DataInputStream dis=new DataInputStream(so.getInputStream());
        fillen=dis.readInt();
        filtfr=new String[fillen];
        intval=new int[fillen];
        intfec=new int[fillen];
        chfec=new char[fillen+25];
        pakch1=new char[fillen+25][100];
        filreord=new char[fillen+25][100];
        filfec=new String[fillen];
        jTextField1.setText("//Destination 1/Result1.txt");

        System.out.println("File Length : "+fillen);
         for(i=0;i<fillen;i++)
         {
                filtfr[i]=dis.readUTF();
                System.out.println("Packet : ["+i+"] = "+filtfr[i]);
```

```
                                //Thread.sleep(25);
                            }
                            System.out.println("**********Packets Recieved From The
Source**********");
                            jTextArea1.append("\n\n   Packets Received");Per=100;
                            System.out.println("File Length : "+fillen);
                        }


                }
                catch (Exception er)
                {
                    System.out.println(er);
                }


        }


        /**
```

```
        * This method is called from within the constructor to initialize the form.
        * WARNING: Do NOT modify this code. The content of this method is always
regenerated
        * by the Windows Form Designer. Otherwise, retrieving design might not work properly.
        * Tip: If you must revise this method, please backup this GUI file for JFrameBuilder
        * to retrieve your design properly in future, before revising this method.
        */
        private void initializeComponent()
        {
                jLabel1 = new JLabel();
                jLabel1.setFont(new Font("Arial",Font.BOLD,14));
                jLabel2 = new JLabel();
                jLabel2.setFont(new Font("Arial",Font.BOLD,12));
                jLabel3 = new JLabel();
                jLabel3.setFont(new Font("Arial",Font.BOLD,12));
                jLabel4 = new JLabel();
                jLabel4.setFont(new Font("Arial",Font.BOLD,12));
                jTextField1 = new JTextField();
                jTextField1.setFont(new Font("Arial",Font.BOLD,12));
                jTextArea1 = new JTextArea();
```

```
jTextArea1.setFont(new Font("Arial",Font.BOLD,12));
jScrollPane1 = new JScrollPane();
jProgressBar1 = new JProgressBar();
//jButton1 = new JButton();
jButton2 = new JButton();//((new ImageIcon("bu1.gif")));
jButton3 = new JButton();//((new ImageIcon("bu1.gif")));
jButton4 = new JButton();//((new ImageIcon("bu1.gif")));
jButton5 = new JButton();//((new ImageIcon("bu1.gif")));
contentPane = (JPanel)this.getContentPane();

//
// jLabel1
//
jLabel1.setForeground(new Color(0, 0, 102));
jLabel1.setText("DESTINATION 1");
//
// jLabel2
//
jLabel2.setForeground(new Color(0, 0, 102));
jLabel2.setText("Status Information");
```

```
//
// jLabel3
//
jLabel3.setForeground(new Color(0, 0, 102));
jLabel3.setText("Recieved File Location : ");
//
// jLabel4
//
jLabel4.setForeground(new Color(0, 0, 102));
jLabel4.setText("");
//
// jTextField1
//
jTextField1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
                jTextField1_actionPerformed(e);
        }

});
```

```
//
// jTextArea1
//
//
// jScrollPane1
//
jScrollPane1.setViewportView(jTextArea1);
//
// jProgressBar1
//
//
// jButton1
//
//jButton1.setBackground(new Color(102, 102, 255));
/*jButton1.setForeground(new Color(0, 0, 102));
jButton1.setText("Browse");
jButton1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
                jButton1_actionPerformed(e);
```

```
            }


});*/
//
// jButton2
//
//jButton2.setBackground(new Color(102, 102, 255));
jButton2.setForeground(new Color(0, 0, 102));
jButton2.setText("De-Interleaving");
jButton2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                        jButton2_actionPerformed(e);
            }


});
//
// jButton3
//
//jButton3.setBackground(new Color(102, 102, 255));
```

```
jButton3.setForeground(new Color(0, 0, 102));
jButton3.setText("FEC Decoding");
jButton3.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
                jButton3_actionPerformed(e);
        }

});
//
// jButton4
//
//jButton4.setBackground(new Color(102, 102, 255));
jButton4.setForeground(new Color(0, 0, 102));
jButton4.setText("Result");
jButton4.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
                jButton4_actionPerformed(e);
        }
```

```
});
//
// jButton5
//
//jButton5.setBackground(new Color(102, 102, 255));
jButton5.setForeground(new Color(0, 0, 102));
jButton5.setText("Exit");
jButton5.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
                jButton5_actionPerformed(e);
        }

});
//
// contentPane
//
contentPane.setLayout(null);
contentPane.setBackground(new Color(204, 204, 255));
```

```
contentPane.setForeground(new Color(51, 51, 51));
addComponent(contentPane, jLabel1, 280,10,245,18);
addComponent(contentPane, jLabel2, 445,38,184,18);
addComponent(contentPane, jLabel3, 27,38,240,30);
addComponent(contentPane, jLabel4, 27,410,200,20);
addComponent(contentPane, jTextField1, 27,71,300,30);
addComponent(contentPane, jScrollPane1, 367,67,256,333);
addComponent(contentPane, jProgressBar1, 27,430,600,20);
//addComponent(contentPane, jButton1, 276,70,83,32);
addComponent(contentPane, jButton2, 90,130,120,30);
addComponent(contentPane, jButton3, 90,200,120,30);
addComponent(contentPane, jButton4, 90,270,120,30);
addComponent(contentPane, jButton5, 90,340,120,30);
//
// FECClient
//
this.setTitle("FEC Destination 1");
this.setLocation(new Point(630, 450));
this.setSize(new Dimension(649, 490));
this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
```

```
        //      this.setVisible(true);


        }
        /** Add Component Without a Layout Manager (Absolute Positioning) */
        private void addComponent(Container container,Component c,int x,int y,int width,int
height)
        {
                c.setBounds(x,y,width,height);
                container.add(c);
        }


        //
        // TODO: Add any appropriate code in the following Event Handling Methods
        //
        private void jTextField1_actionPerformed(ActionEvent e)
        {
                System.out.println("\njTextField1_actionPerformed(ActionEvent e) called.");
                // TODO: Add any handling code here
```

```
        }

        /*private void jButton1_actionPerformed(ActionEvent e)
        {
                System.out.println("\njButton1_actionPerformed(ActionEvent e) called.");
                // TODO: Add any handling code here

        }*/

        private void jButton2_actionPerformed(ActionEvent e)
        {
                System.out.println("\n***************De-Interleaving
Started***************");
                // TODO: Add any handling code here

                //System.out.println("fillen = "+fillen);
                jTextArea1.append("\n\n   De-Interleaving Process Started");
                for(i=0;i<fillen;i++)
                {
                        for(j=0;j<filtfr[i].length();j++)
```

```
            {
                pakch1[i][j]=filtfr[i].charAt(j);
            }
        }
        //Printing the values
        k=(float)(Math.random()*2);
        for(i=0;i<fillen;i++)
        {
            for(j=0;j<filtfr[i].length();j++)
            {
                System.out.print(pakch1[i][j]+" ");
            }
            System.out.print("\n");
            try
                {
                    Thread.sleep(2);
                }
                catch (Exception er)
                {
                    System.out.println("Sleep Disturbed : "+er);
```

```
                    }
}

//De-Interleaving
for(i=0;i<fillen;i++)
{
        for(j=0;j<1;j++)
        {
                if((filtfr[i].length())==21)
                {
                bklen=filtfr[i].length();
                filreord[i][0]=pakch1[i][8];
                filreord[i][1]=pakch1[i][10];
                filreord[i][2]=pakch1[i][18];
                filreord[i][3]=pakch1[i][15];
                filreord[i][4]=pakch1[i][9];
                filreord[i][5]=pakch1[i][0];
                filreord[i][6]=pakch1[i][4];
                filreord[i][7]=pakch1[i][13];
                filreord[i][8]=pakch1[i][6];
```

```
filreord[i][9]=pakch1[i][3];
filreord[i][10]=pakch1[i][5];
filreord[i][11]=pakch1[i][2];
filreord[i][12]=pakch1[i][1];
filreord[i][13]=pakch1[i][12];
filreord[i][14]=pakch1[i][20];
filreord[i][15]=pakch1[i][17];
filreord[i][16]=pakch1[i][14];
filreord[i][17]=pakch1[i][16];
filreord[i][18]=pakch1[i][19];
filreord[i][19]=pakch1[i][11];
filreord[i][20]=pakch1[i][7];
}
else if((filtfr[i].length())==18)
{
filreord[i][0]=pakch1[i][7];
filreord[i][1]=pakch1[i][9];
filreord[i][2]=pakch1[i][16];
filreord[i][3]=pakch1[i][13];
filreord[i][4]=pakch1[i][8];
```

```
filreord[i][5]=pakch1[i][0];

filreord[i][6]=pakch1[i][4];

filreord[i][7]=pakch1[i][11];

filreord[i][8]=pakch1[i][6];

filreord[i][9]=pakch1[i][3];

filreord[i][10]=pakch1[i][5];

filreord[i][11]=pakch1[i][2];

filreord[i][12]=pakch1[i][1];

filreord[i][13]=pakch1[i][15];

filreord[i][14]=pakch1[i][17];

filreord[i][15]=pakch1[i][10];

filreord[i][16]=pakch1[i][12];

filreord[i][17]=pakch1[i][14];

}

else

{

filreord[i][0]=pakch1[i][7];

filreord[i][1]=pakch1[i][9];

filreord[i][2]=pakch1[i][5];

filreord[i][3]=pakch1[i][10];
```

```
                    filreord[i][4]=pakch1[i][8];
                    filreord[i][5]=pakch1[i][0];
                    filreord[i][6]=pakch1[i][4];
                    filreord[i][7]=pakch1[i][11];
                    filreord[i][8]=pakch1[i][6];
                    filreord[i][9]=pakch1[i][3];
                    filreord[i][10]=pakch1[i][2];
                    filreord[i][11]=pakch1[i][1];
                    //System.out.println("Packets Lost");
                    }

            }
        }
        //Printing the values
        for(i=0;i<fillen;i++)
        {
            for(j=0;j<filtfr[i].length();j++)
            {
                System.out.print(filreord[i][j]+" ");
            }
```

```
                System.out.print("\n");
                try
                        {
                                Thread.sleep(2);
                        }
                        catch (Exception er)
                        {
                                System.out.println("Sleep Disturbed : "+er);
                        }
        }
        jTextArea1.append("\n\n   De-Interleaving Process Completed");
        System.out.println("\n***************De-Interleaving
Completed***************");
    }

    private void jButton3_actionPerformed(ActionEvent e)
    {
        System.out.println("\n**************FEC Decoding
Stsrted********************");
        // TODO: Add any handling code here
```

```
            orgblk=7;
        //Decoding FEC
        jTextArea1.append("\n\n   FEC Decoding Process Started");
        for(i=0;i<fillen;i++)
        {
            filfec[i]="";
            for(j=0;j<filtfr[i].length();j++)
            {

                if(filreord[i][j+2]!='\0')
                {
                    filfec[i]+=""+filreord[i][j+2];
                    j+=2;
                }
                else if(filreord[i][j+1]!='\0')
                {
                    filfec[i]+=""+filreord[i][j+1];
                    j+=2;
                }
                else if(filreord[i][j]!='\0')
```

```
                {
                        filfec[i]+=""+filreord[i][j];
                        j+=2;
                }
        }
}

//Printing after FEC Decoding
for(i=0;i<fillen;i++)
{

        System.out.println(filfec[i]);
        try
                {
                        Thread.sleep(2);
                }
                catch (Exception er)
                {
                        System.out.println("Sleep Disturbed : "+er);
                }
```

```
}

//Conersion of Binary Values to String
jTextArea1.append("\n\n   Conerting Binary to String");
for(i=0;i<fillen;i++)
{
        intval[i]= Integer.parseInt(filfec[i],2);
        chfec[i]=(char)intval[i];
}
try
{
        Fien="";
        FileOutputStream fw=new FileOutputStream("Output File/Output1.txt");
        for(i=0;i<fillen;i++)
        {
        Fien+=Character.toString(chfec[i]);
        fw.write(chfec[i]);
        }
}
catch (Exception er)
```

```
                {
                        er.printStackTrace();
                }

                jTextArea1.append("\n\n   Recieved Packets Processing Completed");
                System.out.println("\n\n******************FEC Decoding Process
Completed**********************");

                jTextArea1.append("\n\n   FEC Decoding  Process Completed");
                //System.out.println("\nOutput : "+Fien);
                //FileOutputStream fwtr=new
FileOutputStream("F:/Hector/FEC/Destination/Result.txt");
                //fwtr.writeUTF(Fien);

        }

        private void jButton4_actionPerformed(ActionEvent e)
        {
```

```
System.out.println("\nOpening Recieved File");
// TODO: Add any handling code here
Result1 re=new Result1();
re.show();
re.jTextArea1.setText(" ");

re.jTextArea1.setText(Fien);
intdth=bklen-orgblk;
System.out.println("InterLeaving Depth : "+intdth);
codrt=intdth/bklen;
System.out.println("Coding Rate : "+codrt);
n=bklen;
System.out.println("k = "+k);
System.out.println("n = "+n);
//System.out.println("Probjn = "+Probjn);
for(j=((int)(n-7)+1);j<n+1;j++)
{
        //System.out.println("j = "+j);
        ENI=ENI+(j*(k/n));
}
```

```
            System.out.println("ENI = "+ENI);


            PLReff=ENI/n;
            Effy=Per-PLReff;
            System.out.println("PLReff = "+PLReff);
            System.out.println("Effy = "+Effy);
            re.jTextField1.setText(""+Effy);
            re.jTextField2.setText(""+bklen);
            re.jTextField3.setText(""+codrt);
            re.jTextField4.setText(""+intdth);
            Effy=0;
            ENI=0;
            PLReff=0;


    }


    private void jButton5_actionPerformed(ActionEvent e)
    {
            System.out.println("\nExit");
            // TODO: Add any handling code here
```

```
                System.exit(0);


        }


        //
        // TODO: Add any method code to meet your needs in the following area
        //


**********************************************************************/
/*                  FECQueue                          */
/*                                                    */
/**********************************************************************/
//package Queue;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;
/**
 * Summary description for FECQueue *
```

```
*/
public class FECQueue extends JFrame
{
        // Variables declaration
        private JLabel jLabel1;
        private JLabel jLabel2;
        private JTextArea jTextArea1;
        private JScrollPane jScrollPane1;
        private JPanel contentPane;
        public int i,j;
        private JButton jButton1;
        public int fillen;
        public int fillen1;
        public int intfec[];
        public int intval[];
        public char chfec[];
        public char pakch1[][];
        public char filreord[][];
        public String filtfr;
        public String filfec[];
```

```
String Dest1,Dest2;
public int ch;
ServerSocket ss;

Socket so;

// End of variables declaration


public FECQueue()
{
        super();
        initializeComponent();
        //
        // TODO: Add any constructor code after initializeComponent call
        //

        this.setVisible(true);
        try
        {
```

```
ss=new ServerSocket(4500);
while(true)
{
    so=ss.accept();

    System.out.println("**************Packets Are Arriving From
The Source*********");
    jTextArea1.setText("\n   Packets Recieving Started");
    DataInputStream dis1=new DataInputStream(so.getInputStream());
    fillen=dis1.readInt();
    jTextArea1.append("\n   Recieved File Length = "+fillen);
    String Ack=dis1.readUTF();
    jTextArea1.append("\n   Address of the Target = "+Ack);
    //filtfr=new String[fillen];
    System.out.println("File Length : "+fillen);
    Dest1="";
    FileInputStream fis1=new FileInputStream("Dest1Address.txt");
    while((ch=fis1.read())!=-1)
    Dest1+=(char)ch;
```

```
Dest1.trim();
Dest2="";
FileInputStream fis2=new FileInputStream("Dest2Address.txt");
while((ch=fis2.read())!=-1)
Dest2+=(char)ch;
Dest2.trim();


if (Ack.equalsIgnoreCase("Dest1"))
{
i=0;
Socket De1=new Socket(Dest1,4501);
DataOutputStream dis2=new
DataOutputStream(De1.getOutputStream());
dis2.writeInt(fillen);
while(fillen>0)
{
filtfr=dis1.readUTF();
jTextArea1.append("\n   Packet["+i+++"] =
"+filtfr);
```

```
                                        System.out.println("Packet["+i+++"] = "+filtfr);
                                        dis2.writeUTF(filtfr);
                                        jTextArea1.append("\n   Packets Sending =
"+filtfr);
                                        fillen--;
                                        }

                                }
                                else if (Ack.equalsIgnoreCase("Dest2"))
                                {
                                        i=0;
                                        Socket De2=new Socket(Dest2,4502);
                                        DataOutputStream dis3=new
DataOutputStream(De2.getOutputStream());
                                        dis3.writeInt(fillen);
                                        while(fillen>0)
                                        {
                                        filtfr=dis1.readUTF();
                                        jTextArea1.append("\n   Packet["+i+++"] =
"+filtfr);
                                        System.out.println("Packet["+i+++"] = "+filtfr);
```

```
                                        dis3.writeUTF(filtfr);
                                        jTextArea1.append("\n   Packets Sending =
"+filtfr);
                                        fillen--;
                                        }
                                }

                                //System.out.println("Packet : ["+i+"] = "+filtfr);
                                //fillen--;
                                //Thread.sleep(25);



                        }

                }
                catch (Exception er)
                {
                        System.out.println("Socket : "+er);
```

```
            }

        }

        /**
         * This method is called from within the constructor to initialize the form.
         * WARNING: Do NOT modify this code. The content of this method is always
regenerated
         * by the Windows Form Designer. Otherwise, retrieving design might not work properly.
         * Tip: If you must revise this method, please backup this GUI file for JFrameBuilder
         * to retrieve your design properly in future, before revising this method.
         */
        private void initializeComponent()
        {
            jLabel1 = new JLabel();
            jLabel1.setFont(new Font("Arial",Font.BOLD,14));
            jLabel2 = new JLabel();
            jLabel2.setFont(new Font("Arial",Font.BOLD,12));
            jButton1 = new JButton();
            jTextArea1 = new JTextArea();
```

```
jTextArea1.setFont(new Font("Arial",Font.BOLD,12));
jScrollPane1 = new JScrollPane();
contentPane = (JPanel)this.getContentPane();


//
// jLabel1
//
jLabel1.setText("Queue");
//
// jButton1
//
//jButton1.setBackground(new Color(102, 102, 255));
jButton1.setForeground(new Color(0, 0, 102));
jButton1.setText("Exit");
jButton1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)
        {
                jButton1_actionPerformed(e);
        }
```

```
});
//
// jLabel2
//
jLabel2.setText("Status Information");
//
// jTextArea1
//
//
// jScrollPane1
//
jScrollPane1.setViewportView(jTextArea1);
//
// contentPane
//
contentPane.setLayout(null);
contentPane.setBackground(new Color(204, 204, 255));
addComponent(contentPane, jLabel1, 160,14,112,36);
addComponent(contentPane, jButton1, 243,330,83,32);
addComponent(contentPane, jLabel2, 130,65,197,30);
```

```
            addComponent(contentPane, jScrollPane1, 49,103,279,214);
            //
            // FECQueue
            //
            this.setTitle("FECQueue");
            this.setLocation(new Point(0, 0));
            this.setSize(new Dimension(387, 400));
        }

        /** Add Component Without a Layout Manager (Absolute Positioning) */
        private void addComponent(Container container,Component c,int x,int y,int width,int
height)
        {
            c.setBounds(x,y,width,height);
            container.add(c);
        }

        //
        // TODO: Add any method code to meet your needs in the following area
        //
```

```
    private void jButton1_actionPerformed(ActionEvent e)
    {
        System.exit(0);
    }

    /*public void Rec()
    {

    }*/

/*  public void Sen()
    {
        try
        {
            st=new ServerSocket(4502);
            while(true)
            {
            st1=st.accept();
```

```
                    DataOutputStream dos=new
DataOutputStream(st1.getOutputStream());
                    dos.writeInt(fillen);
                    for(i=0;i<fillen;i++)
                    {
                        dos.writeUTF(filtfr[i]);
                    }
                    }
                }
                catch(Exception er)
                {
                    System.out.println("Sending Failed : "+er);
                }

    }
```

5.4 SUMMARY

The implementation chapter describes the various modules of our project. The codes snippets for significant functions in all modules are included.

CHAPTER 6

TESTING AND VALIDATION

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its

requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.1.1 FUNCTIONAL TEST

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

6.1.2 SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

6.1.3 PERFORMANCE TEST

The Performance test ensures that the output be produced within the time limits,

and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

6.2 INTEGRATION TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Integration testing for Server Synchronization:

- ➢ Testing the IP Address for to communicate with the other Nodes
- ➢ Check the Route status in the Cache Table after the status information is received by the Node
- ➢ The Messages are displayed throughout the end of the application

The set of possible test cases for encoding module

Test case : select a text file from file system

Expected results : text data should be encoded

Actual results : text data is encoded

Passed : yes

Remarks : encoded data can be used for further process

Test case : select a file containing video data

Expected results : Data should not be encoded

Actual results : Data is not encoded

Passed : No

Remarks : Data cannot be used for further process

The set of possible test cases for interleaving module

Test case : Send encoded data for interleaving

Expected results : Data should be interleaved

Actual results : Data is interleaved

Passed : Yes

Remarks : Interleaved Data can be used for further process

The set of possible test cases for de-interleaving module

Test case : Received packets is sent for de-interleaving

Expected results : Data should be de-interleaved

Actual results : Data is de-interleaved

Passed : Yes

Remarks : De-interleaved Data can be used for further process

The set of possible test cases for decoding module

Test case : De-interleaved Data is sent for decoding

Expected results : De-interleaved Data should be decoded

Actual results : De-interleaved Data is decoded

Passed : Yes

Remarks : Decoded data can be used for further process

The set of possible test cases for result module

Test case : The file contents are viewed in destination

Expected results : File contents should be displayed

Actual results : File contents is displayed

Passed : Yes

Remarks : The file is successfully received and can be used

6.3 SUMMARY

This includes test cases to uncover errors and validates the requirements specifications.

CHAPTER 7

SNAPSHOTS

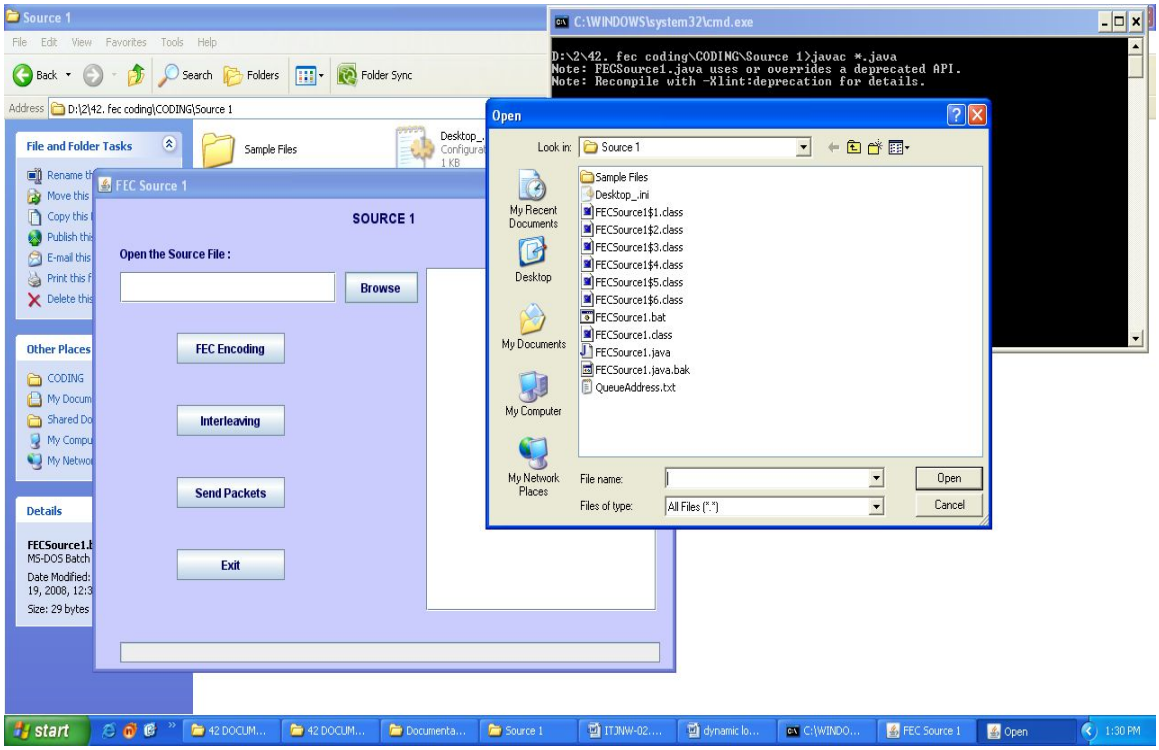


Fig 7.1 File browsed and loaded in source

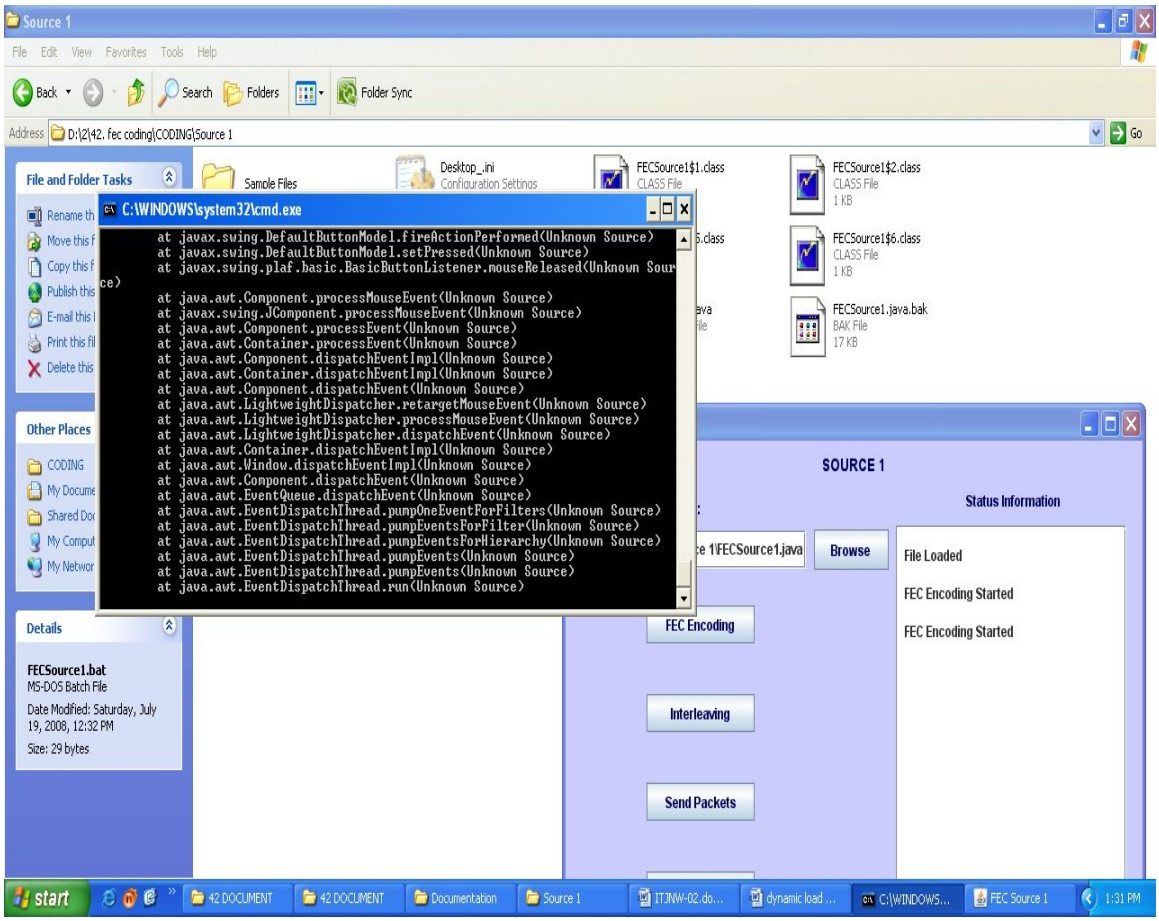


Fig 7.2 File loaded is encoded and interleaved

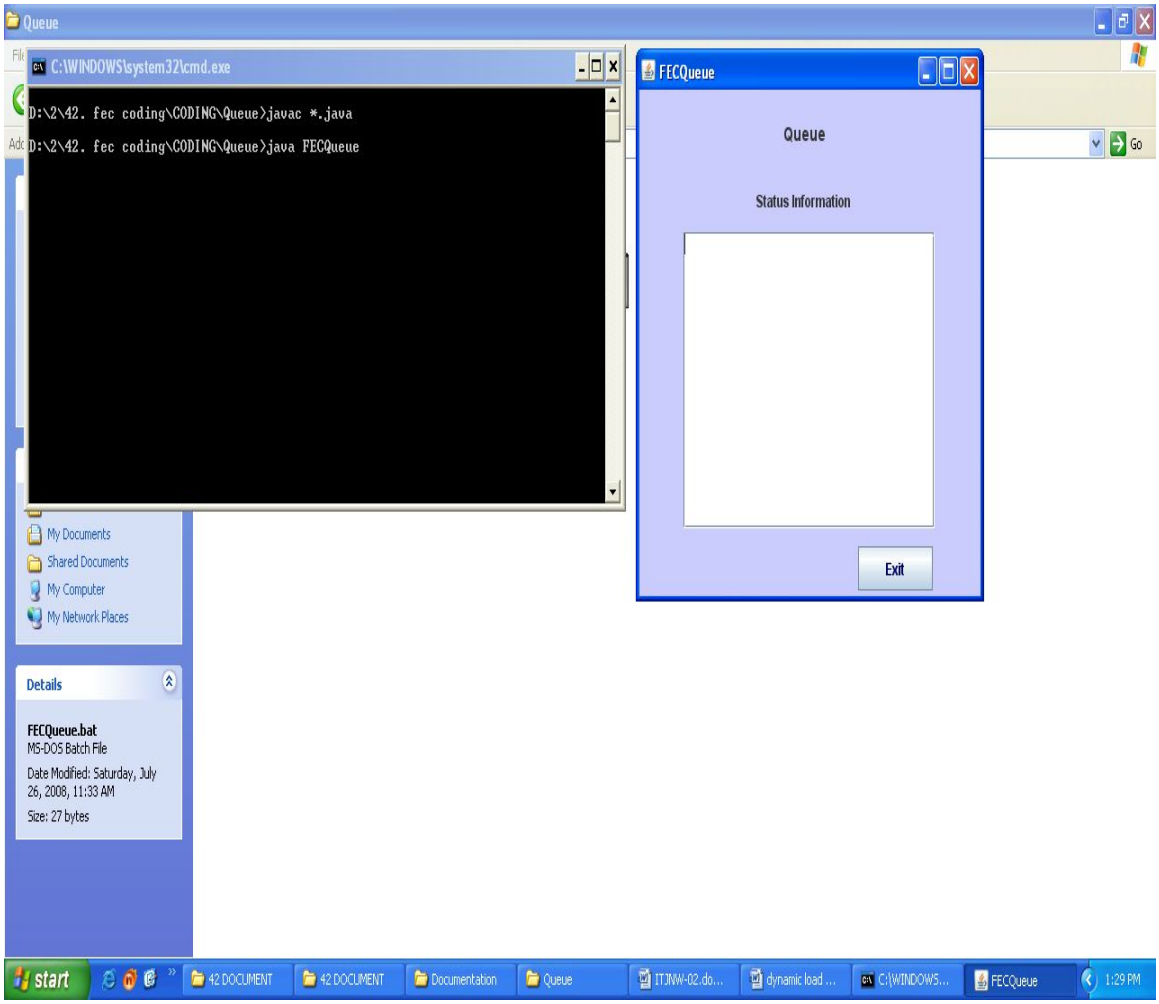


Fig 7.3 Queue to receive and send packets to destination

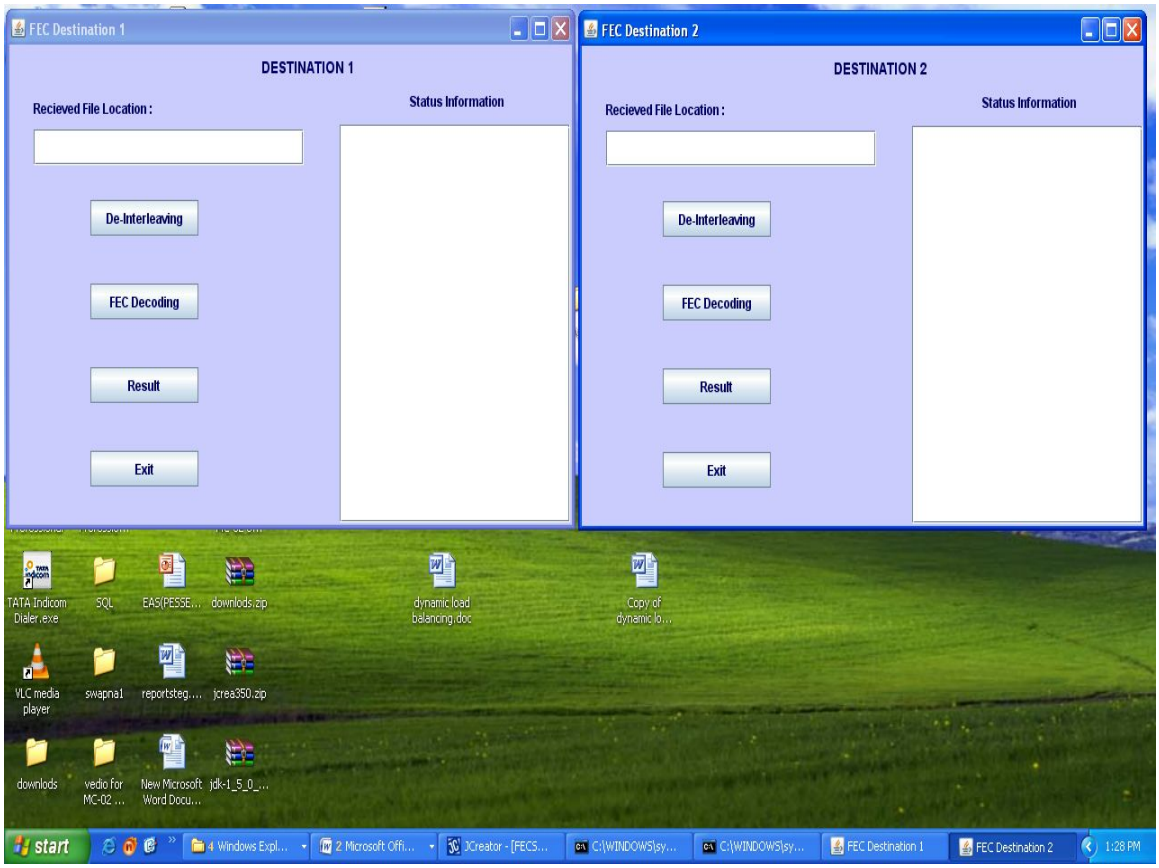


Fig 7.4 Destination to receive the packets

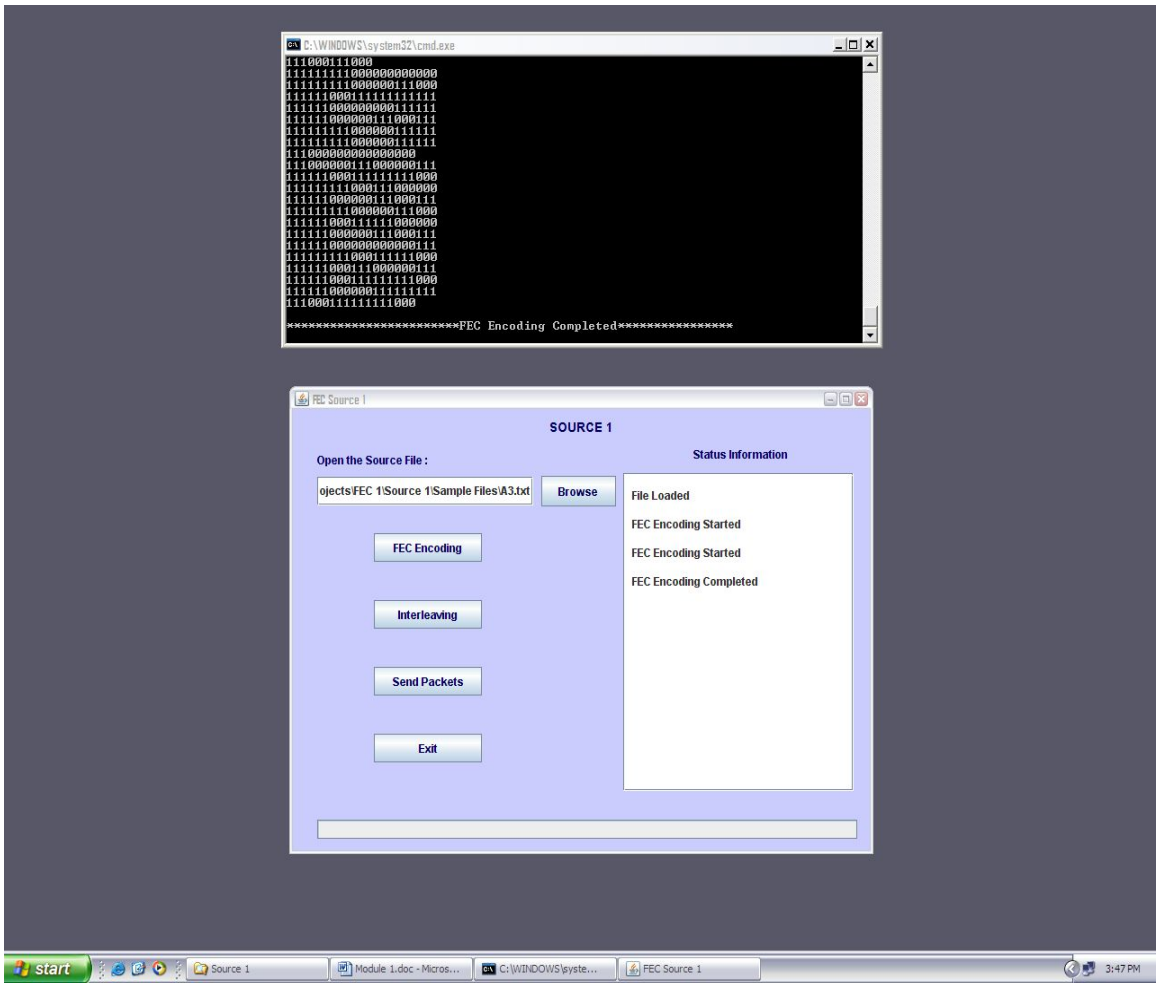

C:\WINDOWS\system32\cmd.exe
111000111000
1111111110000000000000
1111111110000000111000
1111111000111111111111
1111110000000000111111
1111111000000111000111
1111111110000000111111
1111111110000000111111
1111000000000000000
1110000000111000000111
1111110001111111111000
1111111110001110000000
1111111000000111000111
1111111110000000111000
1111111000111111000000
1111110000001110000111
1111111000000000000111
1111111110001111111000
1111111000111000000111
1111111000111111111000
1111111000000011111111
1110001111111111000
**********PEC Encoding Completed**********
FEC Source 1
SOURCE 1
Open the Source File :
Status Information
ojects\FEC 1\Source 1\Sample Files\A3.txt
Browse
File Loaded
FEC Encoding Started
FEC Encoding Started
FEC Encoding Completed
FEC Encoding
Interleaving
Send Packets
Exit
start
Source 1
Module 1.doc - Micros...
C:\WINDOWS\syste...
FEC Source 1
3:47 PM

Fig 7.5 Encoding completed in the source

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

We have analyzed the efficacy of FEC in combating network packet losses based on a single-multiplexer network model and demonstrated that FEC has great potential in recovering the packet losses caused by congestion at a bottleneck node of a packet-switched network, provided that the coding rate and other coding parameters are appropriately chosen. We developed a model to implement interleaving to improve the FEC performance and determined how much interleaving depth is required for FEC to approach the optimum performance. We derived an upper bound on the end-to-end performance using FEC based on an information-theoretic methodology, which is useful in predicting source rates that can be supported with arbitrarily high reliability.

8.1 APPLICATIONS

FEC can be used in ATM networks for data transmission. Thus by using FEC for data transmission in ATM networks we can recover the lost packets. Forward Error Correction (FEC) allows recovery from loss without retransmission. In Digital communication systems,

particularly for military use, need to perform accurately and reliably in the presence of noise and interference. Among many possible ways to achieve this goal, forward error-correction coding is the most effective and economical. Forward error-correction coding represents the most efficient, economical, and predictable way of improving the reliability of transmitted or stored data. In particular it offers designers a powerful tool for ensuring robust communications despite adverse conditions. Forward error-correcting codes play a larger role in helping military communication satellites to keep up with increasing system demands.

8.2 FUTURE ENHANCEMENTS

Despite the great potential of FEC coding in recovering network packet losses, the implementation complexity of FEC coding and the corresponding coding/decoding delay also need to be considered, which is a issue particularly important for real-time applications. One objective for future work is the analysis of the additional delay caused by the FEC coding, perhaps combined with interleaving/de-interleaving. Likewise, the application of FEC for network transport is limited by the time-varying and often uncertain error characteristics of the channel, which makes the appropriate choice of FEC coding rate difficult to determine. In real-world applications, FEC coders are required which can adapt the channel code rate to the time-varying channel conditions. This issue is also a topic for future work.

BIBLIOGRAPHY

[1] D. Y. Eun and N. B. Shroff, "Network decomposition: Theory and practice,"*IEEE/ACM Trans. Networking*, vol. 13, no. 3, pp. 526–539, Jun.2005.

[2] J. Bolot, "End-to-end delay and loss behavior in the Internet," in *Proc. ACM SIGCOMM 1993*, San Francisco, CA, Sep. 1993, pp. 289–298.

[3] N. Shacham and P. Mckenney, "Packet recovery in high-speed networks using coding and buffer management," in *Proc. IEEE INFOCOM 1990*, San Francisco, CA, Jun. 1990, vol. 1, pp. 124–131.

[4] I. Cidon, A. Khamisy, and M. Sidi, "Analysis of packet loss processes in high-speed networks," *IEEE Trans. Inf. Theory*, vol. 39, no. 1, pp. 98–108, Jan. 1993.

[5] R. Kurceren, “Joint source-channel coding approach to transport of digital video on lossy networks,” Ph.D. dissertation, Rensselaer Polytechnic Inst., Troy, NY, May 2001.

[6] R. J. McEliece, D. MacKay, and J.-Fu Cheng, “Turbo Decoding as an Instance of Pearl’s "Belief Propagation" Algorithm,” IEEE Trans. On Commun., vol. 16, no. 2, pp. 140-152, 1998.

[7] IEEE Std P802.16e/D10, “IEEE Standard for Local and Metropolitan Area Networks – Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems,” August, 2005.