**A Geometric Approach to Improving Active**

**Packet Loss Measurement**

**Abstract**

Measurement and estimation of packet loss characteristics are challenging due to the relatively rare occurrence and typically short duration of packet loss episodes. While active probe tools are commonly used to measure packet loss on end-to-end paths, there has been little analysis of the accuracy of these tools or their impact on the network. The objective of our study is to understand how to measure packet loss episodes accurately with end-to-end probes. We begin by testing the capability of standard Poisson- modulated end-to-end measurements of loss in a controlled laboratory environment using IP routers and commodity end hosts. Our tests show that loss characteristics reported from such Poisson-modulated probe tools can be quite inaccurate over a range of traffic conditions. Motivated by these observations, we introduce a new algorithm for packet loss measurement that is designed to overcome the deficiencies in standard Poisson-based tools. Specifically, our method entails probe experiments that follow a geometric distribution to 1) enable an explicit trade-off between accuracy and impact on the network, and 2) enable more accurate measurements than standard Poisson probing at the same rate. We evaluate the capabilities of our methodology experimentally by developing and implementing a prototype tool, called BADABING. The experiments demonstrate the trade-offs between impact on the network and measurement accuracy. We show that BADABING reports loss characteristics far more accurately than traditional loss measurement tools.TABLE OF CONTENTS

**TITLE PAGE NO**

ABSTRACT

INTRODUCTION

SYSTEM STUDY

FEASABILITY STUDY

EXISTING SYSTEM

PROPOSED SYSTEM

SYSTEM SPECIFICATION

SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

LANGUAGE SPECIFICATION

Java Technology

SYSTEM DESIGN

DFD DIAGRAMS

UML DIAGRAMS

SYSTEM TESTING AND MAINTENANCE

SYSTEM IMPLEMENTATION

SCOPE FOR FUTURE DEVELOPMENT

CONCLUSION

BIBLIOGRAPHY

APPENDIX

SCREEN SHOT

SAMPLE CODING

**Introduction**

**Scope of the Project**

The objective of our study is to understand how to measure packet loss episodes accurately with end-to-end probes. We begin by testing the capability of standard Poisson-modulated end-to-end measurements. Motivated by these observations, we introduce a new algorithm for packet loss measurement that is designed to overcome the deficiencies in standard Poisson-based tools.

**Introduction**

Measuring and analyzing network traffic dynamics between end hosts has provided the foundation for the development of many different network protocols and systems. Of particular importance is under-standing packet loss behavior since loss can have a significant impact on the performance of both TCP- and UDP-based applications. Despite efforts of network engineers and operators to limit loss, it will probably never be eliminated due to the intrinsic dynamics and scaling properties of traffic in packet switched network. Network operators have the ability to passively monitor nodes within their network for packet loss on routers using SNMP. End-to-end active measurements using probes provide an equally valuable perspective since they indicate the conditions that application traffic is experiencing on those paths.

Our study involves the empirical evaluation of our new loss measurement methodology. To this end, we developed a one-way active measurement tool called BADABING. BADABING sends fixed-size probes at specified intervals from one measurement host to a collaborating target host. The target system collects the probe packets and reports the loss characteristics after a specified period of time. We also compare BADABING with a standard tool for loss measurement that emits probe packets at Poisson intervals. The results show that our tool reports loss episode estimates much more accurately for the same number of probes. We also show that BADABING estimates converge to the underlying loss episode frequency and duration characteristics. Our observations about the weaknesses in standard Poisson probing motivate the second part of our study: the development of a new approach for end-to-end loss measurement that includes four key elements. First, we design a probe process that is geometrically distributed and that assesses the likelihood of loss experienced by other flows that use the same path, rather than merely reporting its own packet losses. The probe process assumes FIFO queues along the path with a drop-tail policy. Second, we design a new experimental framework with estimation techniques that directly estimate the mean duration of the loss episodes without estimating the duration of any individual loss episode. Our estimators are proved to be consistent, under mild assumptions of the probing process.

**SYSTEM STUDY**

**FEASIBLITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* ECONOMICAL FEASIBILITY
* TECHNICAL FEASIBILITY
* SOCIAL FEASIBILITY

**ECONOMICAL FEASIBILITY:**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

**TECHNICAL FEASIBILITY:**

### This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**SOCIAL FEASIBILITY:**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**Existing System:**

* In an Existing System, theyanalyze the usefulness of Poisson Arrivals See Time Averages in the networking context. Of particular relevance to our work is Paxson’s recommendation and use of Poisson- modulated active probe streams to reduce bias in delay and loss measurements.
* Several studies include the use of loss measurements to estimate network properties such as bottleneck buffer size and cross traffic intensity, which is not accurate.
* Network tomography based on using both multicast and unicast probes has also been demonstrated to be in-effective (in some cases) for inferring loss rates on internal links on end-to-end paths.

**Proposed System:**

* The purpose of our study was to understand how to measure end-to-end packet loss characteristics accurately with probes and in a way that enables us to specify the impact on the bottleneck queue.
* The goal of our study is to understand how to accurately measure loss characteristics on end-to-end paths with probes.
* Specifically, our method entails probe experiments that follow a geometric distribution to 1) enable an explicit trade-off between accuracy and impact on the network, and 2) enable more accurate measurements than standard Poisson probing at the same rate.
* Our study consists of three parts: (*i*) empirical evaluation of the currently prevailing approach, (*ii*) development of estimation techniques that are based on novel experimental design, novel probing techniques, and simple validation tests, and (*iii*) empirical evaluation of this new methodology.

**SYSTEM CONFIGURATION**

**Hardware & Software Requirements**

**Hardware specification:**

* **Monitor :** 800\*600 minimum resolution of 256 colors
* **Processor:** At least 166 MHz processor
* **Input :** Two or Three button mouse and standard 104 keyboards.

**Software specification:**

* **Front End:** Java 1.4
* **Back End:** Oracle 8i

**LANGUAGE SPECIFICATION**

**Java Technology**

Initially the language was called as “oak” but it was renamed as “Java” in 1995. The primary motivation of this language was the need for a platform-independent (i.e., architecture neutral) language that could be used to create software to be embedded in various consumer electronic devices.

* Java is a programmer’s language.
* Java is cohesive and consistent.
* Except for those constraints imposed by the Internet environment, Java gives the programmer, full control.
* Finally, Java is to Internet programming where C was to system programming.

#### Importance of Java to the Internet

Java has had a profound effect on the Internet. This is because; Java expands the Universe of objects that can move about freely in Cyberspace. In a network, two categories of objects are transmitted between the Server and the Personal computer. They are: Passive information and Dynamic active programs. The Dynamic, Self-executing programs cause serious problems in the areas of Security and probability. But, Java addresses those concerns and by doing so, has opened the door to an exciting new form of program called the Applet.

#### Java can be used to create two types of programs

Applications and Applets: An application is a program that runs on our Computer under the operating system of that computer. It is more or less like one creating using C or C++. Java’s ability to create Applets makes it important. An Applet is an application designed to be transmitted over the Internet and executed by a Java –compatible web browser. An applet is actually a tiny Java program, dynamically downloaded across the network, just like an image. But the difference is, it is an intelligent program, not just a media file. It can react to the user input and dynamically change.

#### Features of Java Security

Every time you that you download a “normal” program, you are risking a viral infection. Prior to Java, most users did not download executable programs frequently, and those who did scan them for viruses prior to execution. Most users still worried about the possibility of infecting their systems with a virus. In addition, another type of malicious program exists that must be guarded against. This type of program can gather private information, such as credit card numbers, bank account balances, and passwords. Java answers both these concerns by providing a “firewall” between a network application and your computer.

When you use a Java-compatible Web browser, you can safely download Java applets without fear of virus infection or malicious intent.

#### Portability

For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed .As you will see, the same mechanism that helps ensure security also helps create portability. Indeed, Java’s solution to these two problems is both elegant and efficient.

#### The Byte code

The key that allows the Java to solve the security and portability problems is that the output of Java compiler is Byte code. Byte code is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreter for byte code.

Translating a Java program into byte code helps makes it much easier to run a program in a wide variety of environments. The reason is, once the run-time package exists for a given system, any Java program can run on it.

Although Java was designed for interpretation, there is technically nothing about Java that prevents on-the-fly compilation of byte code into native code. Sun has just completed its Just In Time (JIT) compiler for byte code. When the JIT compiler is a part of JVM, it compiles byte code into executable code in real time, on a piece-by-piece, demand basis. It is not possible to compile an entire Java program into executable code all at once, because Java performs various run-time checks that can be done only at run time. The JIT compiles code, as it is needed, during execution.

#### Java Virtual Machine (JVM)

Beyond the language, there is the Java virtual machine. The Java virtual machine is an important element of the Java technology. The virtual machine can be embedded within a web browser or an operating system. Once a piece of Java code is loaded onto a machine, it is verified. As part of the loading process, a class loader is invoked and does byte code verification makes sure that the code that’s has been generated by the compiler will not corrupt the machine that it’s loaded on. Byte code verification takes place at the end of the compilation process to make sure that is all accurate and correct. So byte code verification is integral to the compiling and executing of Java code.

**Overall Description:**

# Java Source

## *Java byte code*

# JavaVM

**Java**

**.Class**

#### Picture showing the development process of JAVA Program

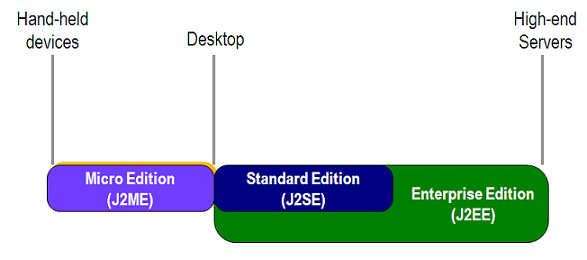
Java programming uses to produce byte codes and executes them. The first box indicates that the Java source code is located in a. Java file that is processed with a Java compiler called javac. The Java compiler produces a file called a. class file, which contains the byte code. The .Class file is then loaded across the network or loaded locally on your machine into the execution environment is the Java virtual machine, which interprets and executes the byte code.

#### Java Architecture

Java architecture provides a portable, robust, high performing environment for development. Java provides portability by compiling the byte codes for the Java Virtual Machine, which is then interpreted on each platform by the run-time environment. Java is a dynamic system, able to load code when needed from a machine in the same room or across the planet.

#### Compilation of code

When you compile the code, the Java compiler creates machine code (called byte code) for a hypothetical machine called Java Virtual Machine (JVM). The JVM is supposed to execute the byte code. The JVM is created for overcoming the issue of portability. The code is written and compiled for one machine and interpreted on all machines. This machine is called Java Virtual Machine.



**Compiling and interpreting Java Source Code**

During run-time the Java interpreter tricks the byte code file into thinking that it is running on a Java Virtual Machine. In reality this could be a Intel Pentium Windows 95 or SunSARC station running Solaris or Apple Macintosh running system and all could receive code from any computer through Internet and run the Applets.

### Simple

Java was designed to be easy for the Professional programmer to learn and to use effectively. If you are an experienced C++ programmer, learning Java will be even easier. Because Java inherits the C/C++ syntax and many of the object oriented features of C++. Most of the confusing concepts from C++ are either left out of Java or implemented in a cleaner, more approachable manner. In Java there are a small number of clearly defined ways to accomplish a given task.

**Object-Oriented**

Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean usable, pragmatic approach to objects. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non-objects.

**Robust**

The multi-platform environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. The ability to create robust programs was given a high priority in the design of Java. Java is strictly typed language; it checks your code at compile time and run time.

Java virtually eliminates the problems of memory management and de-allocation, which is completely automatic. In a well-written Java program, all run time errors can –and should –be managed by your program.

**TECHNOLOGY INFRASTRUCTURE:**

**CORE JAVA:**

Java can be used to create two types of programs: application and applet. An application is a program that runs on your computer, under the operating system of that computer. That is, an application created by java is more or less like one created using C or C++. When used to create application, java is not much different from any other computer language. Rather, it is java’s ability to create applets that makes it important. An applet is an application designed to be transmitted over the internet and executed by a java-compatible Web Browser. An applet is actually a tiny java program, dynamically downloaded across the network, just like an image, sound file, or video clip. The important difference is that an applet is an intelligent program, not just an animation or media file. In other words, an applet is a program that can react to user input and dynamically change-not just run the same animation or sound over and over.

Java having a major roll in internet and the intranet application. The reason for this is quite simple: Java expends the universe pf objects that can move about freely in cyberspace. In a network, two very broad categories ob objects are transmitted between the server and your personal computer: passive information and dynamic, active programs.

**SECURITY**

As you are likely aware, every time that you download a “normal” program , you are risking viral infection. Prior to java, most users did not download executable programs frequently, and those who did scanned them for viruses prior to execution. Even so, most users still worried about the possibility of infecting their system with a virus. When you use a java-compatible web browser, you can safely download java applets without fear of viral infection or malicious intent. Java achieves this protection by confining a java program to the java execution environment and not allowing it access to other parts of computer.

**PORTABILITY**

Many types of computers and operating systems are in use throughout the world-and many are connected to the internet. For program to be dynamically downloaded to all the various type of platforms connected to the Internet, some means of generating portable executable code is needed.

**BYTECODE**

The key that allows java to solve both the security and the portability problems just described is that output of a java compiler is not executable code. Rather, it is BYTECODE. Byte code is a highly optimized set of instruction designed to be executed by the java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreted code.

**SIMPLE:**

Java was designed to be easy for the professional programmer to learn and use effectively. Assuming that you have some programming experience, you will not find java hard to master. If we know the basic concept of object-oriented programming, learning java will be even easier.

**OBJECT-ORIENTED:**

Object-Oriented programming is the core of java. In fact, all java programs are object-oriented-this isn’t an option the way that it is in C++, for example. OOP is so integral to java that you must understand its basic principles before you can write even simple java programs.

**ABSTRACT:**

The essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of ten of individual parts. They think of it as a well-defined object with its own unique behavior. So this ignore the details of how the engine, transmission, and braking systems work.

**THE THREE-OOP PRINCIPLE:**

1. **ENCAPSULATION:**

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. One way to think about encapsulation is as a protective wrapper that prevents the code and data from begin arbitrarily accessed by other code defined outside of the wrapper. Access to he code and data inside the wrappers is tightly controlled through a well-defined interface. To relate this to the real world, consider the automatic transmission on an automobile. It encapsulates hundreds of bits of information about our engine, such as how much you are accelerating, the pitch of the surface you are on, and the position of the shift lever.

1. **INHERITANCE:**

Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. As mentioned earlier, most knowledge is made manageable by hierarchical classification. Inheritance interacts with encapsulation as well. If a given class encapsulates some attributes, then subclass will have the same attributes plus any that it adds as part of its specialization. Java supports two type of inheritance. They are,

* 1. **SINGLE INHERITANCE:**

The derived class is inherited from one super class.

|  |
| --- |
| Super-class |

|  |
| --- |
| Sub-class |

1. **MULTILEVEL INHERITANCE:**

This contains the hierarchical of classes.

|  |
| --- |
| Super-class |

|  |
| --- |
| Sub-class |

|  |
| --- |
| Sub-class |

**POLYMORPHISM:**

Polymorphism (from the Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. More generally, the concept of polymorphism is often expressed by the phrase “one interface, multiple methods”. This means that is possible to design a generic interface to a group or related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler’s job to select the specific action as it applies to each situation.

**SWING:**

The Swing toolkit includes a rich set of components for building GUIs and adding interactivity to Java applications. Swing includes all the components you would expect from a modern toolkit: table controls, list controls, tree controls, buttons, and labels. Swing is far from a simple component toolkit, however. It includes rich undo support, a highly customizable text package, integrated internationalization and accessibility support. To truly leverage the cross-platform capabilities of the Java platform, Swing supports numerous look and feels, including the ability to create your own look and feel. The ability to create a custom look and feel is made easier with Synth, a look and feel specifically designed to be customized. Swing wouldn't be a component toolkit without the basic user interface primitives such as drag and drop, event handling, customizable painting, and window management.

Swing is part of the Java Foundation Classes (JFC). The JFC also include other features important to a GUI program, such as the ability to add rich graphics functionality and the ability to create a program that can work in different languages and by users with different input devices.

The Swing toolkit includes a rich array of components: from basic components, such as buttons and check boxes, to rich and complex components, such as tables and text. Even deceptively simple components, such as text fields, offer sophisticated functionality, such as formatted text input or password field behavior. There are file browsers and dialogs to suit most needs, and if not, customization is possible. If none of Swing's provided components are exactly what you need, you can leverage the basic Swing component functionality to create your own.

Swing components facilitate efficient graphical user interface (GUI) development. These components are a collection of light weight visual components. Swing components contain a replacement for the heavyweight AWT components as well as complex user-interface components such as trees and tables. Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT. In addition to that the familiar components such as buttons, check box and labels swings supplies several exciting additions including tabbed panes, scroll panes, trees and tables. Even familiar components such as buttons have more capabilities in swing. For example a button may have both an image and text string associated with it. Also the image can be changed as the state of button changes. Unlike AWT components swing components are not implemented by platform specific code instead they are return entirely in JAVA and, therefore, are platform-independent. The term lightweight is used to describe such elements. The number of classes and interfaces in the swing packages is substantial. The Swing architecture is shown in the figure given blow:

Application Code

JFC

Swing

Java 2D

AWT

Drag & Drop

Accessibility

**THE SWING COMPONENT CLASSES:**

|  |  |
| --- | --- |
| Class | Description |
| Abstract Button | Abstract super class for Swing Buttons |
| Button Group | Encapsulates a mutually exclusive set of Buttons |
| ImageIcon | Encapsulates an Icon |
| JApplet | The Swing version of Applet |
| JButton | The Swing Push Button Class |
| JCheckBox | The Swing CheckBox class |
| JComboBOx | Encapsulates a combobox |
| JLabel | The swing version of a Label |
| JRadioButton | The Swing version of a RadioButton |
| JScrollPane | Encapsulates a scrollabel window |
| JTabbedPane | Encapsulates a Tabbed window |
| JTable | Encapsulates a Table-based control |
| JTextField | The swing version of a text-field |
| JTree | Encapsulates a Tree-based control |

**ADVANTAGES OF SWINGS:**

* Wide variety of Components
* Pluggable Look and Feel
* MVC Architecture
* Keystroke Handling
* Action Objects
* Nested containers
* Customized Dialogs
* Compound Borders
* Standard Dialog Classes
* Structured Table and Tree Components
* Powerful Text Manipulation
* Generic Undo Capabilities
* Accessibility Support

**JDBC connectivity**

The JDBC provides database-independent connectivity between the J2EE platform and a wide range of tabular data sources. JDBC technology allows an Application Component Provider to:

* Perform connection and authentication to a database server
* Manager transactions
* Move SQL statements to a database engine for preprocessing and execution
* Execute stored procedures
* Inspect and modify the results from Select statements

**Database:**

A database management system (DBMS) is computer software designed for the purpose of managing databases, a large set of structured data, and run operations on the data requested by numerous users. Typical examples of DBMSs include Oracle, DB2, Microsoft Access, Microsoft SQL Server, Firebird, PostgreSQL, MySQL, SQLite, FileMaker and Sybase Adaptive Server Enterprise. DBMSs are typically used by Database administrators in the creation of Database systems. Typical examples of DBMS use include accounting, human resources and customer support systems.

Originally found only in large companies with the computer hardware needed to support large data sets, DBMSs have more recently emerged as a fairly standard part of any company back office.

**Description**

A DBMS is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. A DBMS includes:

* A modeling language to define the schema of each database hosted in the DBMS, according to the DBMS data model.
* The four most common types of organizations are the hierarchical, network, relational and object models. Inverted lists and other methods are also used. A given database management system may provide one or more of the four models. The optimal structure depends on the natural organization of the application's data, and on the application's requirements (which include transaction rate (speed), reliability, maintainability, scalability, and cost).
* The dominant model in use today is the ad hoc one embedded in SQL, despite the objections of purists who believe this model is a corruption of the relational model, since it violates several of its fundamental principles for the sake of practicality and performance. Many DBMSs also support the Open Database Connectivity API that supports a standard way for programmers to access the DBMS.
* Data structures (fields, records, files and objects) optimized to deal with very large amounts of data stored on a permanent data storage device (which implies relatively slow access compared to volatile main memory).
* A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.
* It also controls the security of the database.
* Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data.
* If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.
* A transaction mechanism, that ideally would guarantee the ACID properties, in order to ensure data integrity, despite concurrent user accesses (concurrency control), and faults (fault tolerance).
* It also maintains the integrity of the data in the database.
* The DBMS can maintain the integrity of the database by not allowing more than one user to update the same record at the same time. The DBMS can help prevent duplicate records via unique index constraints; for example, no two customers with the same customer numbers (key fields) can be entered into the database. See ACID properties for more information (Redundancy avoidance).

The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.

Organizations may use one kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by database administrators.

Database servers are specially designed computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with RAID disk arrays used for stable storage. Connected to one or more servers via a high-speed channel, hardware database accelerators are also used in large volume transaction processing environments.

DBMSs are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system.

**OVERVIEW OF JAVA RMI**

**DISTRIBUTED COMPUTING**

In the present modern Internet World, Distributed Computing is one of the key areas that play an important role. **Distributed systems** require that computations running in different address spaces, potentially on different hosts, be able to communicate with each other. .

An alternative to sockets used in java is **Remote Procedure Call** (RPC), which abstracts the communication interface to the level of a procedure call. Instead of working directly with sockets, the programmer has the illusion of calling a local procedure, when in fact the arguments of the call are packaged up and shipped off to the remote target of the call. RPC systems encode arguments and return values using an external data representation, such as XDR. In order to match the semantics of object invocation, distributed object systems require **remote method invocation or RMI.** .

RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Distributed object systems finds its applications to locate remote objects, Communicate with remote objects and Load class byte codes for objects that are passed as parameters or return values. In such systems, a local surrogate (stub) object manages the invocation on a remote object.

The Java programming language's RMI system assumes the homogeneous environment of the Java virtual machine (JVM), and the system can therefore take advantage of the Java platform's object model whenever possible.

**REMOTE METHOD INVOCATION**

RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Server creates a number of remote objects, makes references to those remote objects. The client gets a remote reference to one or more remote objects in the server and then invokes methods on them.

Java provides a program called RMI Registry which executes on the server machine. The Registry maps names to object references and listens for client request on a designated port. The client looks up the remote object by its name in the server’s registry and then invokes the method of server object.

**ARCHITECTURE OF RMI**

JAVA VIRTUAL MACHINE JAVA VIRTUAL MACHINE

Application Layer

Application Layer

Transport Layer

Remote Reference Layer

Proxy Layer

Proxy Layer

Remote Reference Layer

**TCP**

Transport Layer

Figure 2.1 Architecture of RMI

**APPLICATION LAYER**

The application layer is the actual implementation of the client and the server application. It contains the actual object definition. The client can access the remote method through the interface that extends java.rmi.Remote. Once the methods described in the remote interfaces have been implemented, the object must be exported. This can be done implicitly if the object extends the UnicastRemoteObject class of the java.rmi.Server package. Then, the application will register itself with a naming server or registry. A client simply requests a remote object from either a registry or remote object whose reference has been already been obtained.

**PROXY LAYER**

The proxy layer consists of two parts namely Stub and Skeleton. They are created using the RMI compiler (RMIC). These are class files that represent the client and server side of a remote object. The Stub and Skeleton are used for marshaling and unmarshaling the data that is transferred through the network. Marshaling is the process of converting the Java byte codes into a stream of bytes, and unmarshaling is the reverse process of it. Stub is a proxy for the server. It is placed on the client side of the application whereas the skeleton is placed on the server side.

**REMOTE REFERENCE**

The remote reference layer is effective between the stub and skeleton classes and the transport layer thereby handling the actual communication protocols. It gets the stream of bytes from the transport layer and sends it to the proxy layer and vice versa.

**TRANSPORT LAYER**

The transport layer is responsible for handling the actual machine-to-machine communication; the default communication will take place through a standard TCP/IP (Transfer Control Protocol/Internet Protocol). It creates a stream that is accessed by the remote reference layer to send and receive data to and from other machines. The transport layer sets up the connections to remote machines, manages it, listens for connections from other machines and monitors the connection to make sure that the remote machines are alive.

**RMI REGISTRY**

RMI registry is a simple server that enables an application to lookup objects that are exported for remote method invocation. It is also called as bootstrap registry. The registry keeps track of the addresses of remote objects that are being exported by their application. All the objects are assigned unique names that are used to identify the object. The methods can be called from the rmi.registry.Registry interface or from the rmi.Naming class. It allows the application to add, remove and access remote objects in the registry’s table of objects and associated names. RMI registry can run either in a separate window or as a background job or process. It runs independently from the particular server and client but is required by them.

**STUB AND SKELETON**

RMI uses a standard mechanism employed in RPC systems for communicating with remote objects: stubs and skeletons. A stub for a remote object acts as a client's local representative or proxy for the remote object. The caller invokes a method on the local stub, which is responsible for carrying out the method call on the remote object. When a stub's method is invoked, it initiates a connection with the remote JVM containing the remote object, marshals (writes and transmits) the parameters to the remote JVM, waits for the result of the method invocation, unmarshals (reads) the return value or exception returned, and returns the value to the caller.

In the remote JVM, each remote object may have a corresponding skeleton. The skeleton is responsible for dispatching the call to the actual remote object implementation. When a skeleton receives an incoming method invocation it unmarshals (reads) the parameters for the remote method, invokes the method on the actual remote object implementation, and marshals (writes and transmits) the result to the caller.

**RMI INTERFACES**

A **remote object** is an instance of a class that implements a Remote interface. The remote interface will declare each of the methods that can be called from other Java virtual machines. Remote interfaces have the following characteristics:

* The remote interface must be declared public.
* The remote interface extends the java.rmi.Remote interface.
* The data type of any **remote** object that is passed, as an argument or return value must be declared as the **remote interface type** not the implementation class.

**REMOTE EXCEPTION CLASS**

The java.rmi.RemoteException class is the super class of exceptions thrown by the RMI runtime during a remote method invocation. To ensure the robustness of applications using the RMI system, each remote method declared in a remote interface must specify java.rmi.RemoteException in its throws clause. The exception java.rmi.RemoteException is thrown when a remote method invocation fails for some reason.

**GARBAGE COLLECTION**

In a distributed system, it is desirable to automatically delete those remote objects that are no longer referenced by any client. RMI uses a reference-counting garbage collection algorithm to accomplish reference-counting garbage collection; the RMI runtime keeps track of all live references within each Java virtual machine. When a live reference enters a Java virtual machine, its reference count is incremented. When any client does not reference a remote object, the RMI runtime refers to it using a weak reference. The weak reference allows the Java virtual machine's garbage collector to discard the object if no other local references to the object exist.

**PACKAGES**

Package objects contain version information about the implementation and specification of a Java package. This versioning information is retrieved and made available by the **Class Loader** instance that loaded the classes. Typically, it is stored in the manifest that is distributed with the classes.

**java.lang :** Provides classes that are fundamental to the design of the Java programming language.

## *java.io : Provides for system input and output through data streams, serialization and the file system.*

## *java.net: Provides the classes for implementing networking applications.*

## *java.rmi: Provides the classes and interfaces required for RMI application.*

## *javax.swing: Provides a set of "lightweight” components that, to the maximum degree possible, work the same on all platforms.*

**SOCKETS**

**Sockets In Java:**

You use URLs and URL Connections to communicate over the network at relatively highlevel and for a specific purpose accessing resources on the Internet. Sometimes your programs require lower level network communications, For example, when you want to write a client-server application.

In client-server applications, the server provides some service, such as processing database requires or sending out current stock prices. The client uses the service provided by the server to some end displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and server must be reliable-no data can be dropped and it must arrive on the client side in the same manner order that it was sent by the server.

TCP provides a reliable, point-to-point communication channel, which client –server applications on the Internet use to communicate. The Socket and Server Socket classes in java.net provides a system-independent communication channel using TCP.

A server application normally listens to a specific Port waiting for connection requests from a client. When a connection request arrive, the client and server establish a dedicated connection over which they can communicate. During the connection process, the client is assigned a local port number, and binds a socket to it. The client talks to the server by writing to the socket and gets information from the server by reading from it. Similarly, the server gets a new-local port number (it needs a new port number so that it can continue to listen for connection requests on the original port). The server also binds a socket to its local port and communicates with the client by reading from and writing to it.

The client and the server must agree on protocol-that is, they must agree on the language of the information transferred back and forth though the socket.

**Definition**: A Socket is one end-points of a two-way communicate link between two programs running on the network.

The java.net package in the java development environment provides a class-socket-that represents one end of a two-way connection between your java program and another program on the network. The Socket class implements the client side of two-way link. If you are writing server software, you will also be interested in the Server Socket class which implementing the server side of the two-way link.

**Server Socket:**

Java.lang.object

+----------------------java.net. Server Socket

Public class ServerSocket Extends Object

This class implements serversockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

The actual work of the server is performed by an instance of the Socket Implement class. An application can change the socket factory that implementation to configure itself to create appropriate to the local firewall.

**Socket:**

Java.lang.Object

+-------java.net.Socket

Public class Socket extends Object.

This class implements client sockets (also called just “sockets”). A socket is an endpoint for communication between two machines.

The actual work of the socket is performed by an instance of the Socket Implement class. An application, by changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall.

**SECURITY CONSIDERATIONS**

**THE RSA ALGORITHM**  INTRODUCTION: The RSA scheme is a block cipher in which the plaintext and cipher text are integers between 0 and n-1 for some n.A typical size for n is 1024 bits or 309 decimal digits. The RSA scheme has since that time reigned supreme as the most widely accepted and implemented general purpose to public key encryption. DESCRIPTION: The scheme developed by Rivest, Shamir and Adleman makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n.That is, the block size must be less than or equal to log2(n) ; in practice, the block size is k bits, where 2k < n < 2k+1 .Encryption and decryption are of the following form, for some plaintext block M and cipher text block C: C = Me mod n  M = Cd mod n = (Me) d mod n = Med mod nBoth sender and receiver must know the value of n.The sender knows the value of e and only the receiver knows the value of d.Thus, this is a public key encryption algorithm with a public key of KU = {d,n}.For this algorithm to be satisfactory for public key encryption, the following requirements to be met :1. It is possible to find the values of e, d, n such that Med = M mod n for all M < n.2. It is relatively easy to calculate Me and Cd for all values of M < n.3. It is infeasible to determine d given e and n. For now,we focus on the first requirement and consider the other questions later.We need to find a relationship of the form Med = M mod n Given two prime numbers and q, and two integers and m, such that n=pq and 0<m<n, and arbitrary integer k, the following relationships holds:

Mkφ (n)+1 = mk(p-1)(q-1)+1 = m mod n

Where φ(n) is the euler’s totient function, which is the number of positive integers less than n and relatively prime to n. it is shown that for p,q prime,

φ(p,q)=(p-1)(q-1). Thus we can achieve the desired relationship if

ed = kφ(n)+1

This is equivalent to saying:

ed = 1 mod φ(n)

d = e-1 mod φ(n)

That is, e and d are multiplicative inverses mod φ (n). Note that, according to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to φ (n). Equivalently, gcd (φ(n),d) = 1.

We are now ready to state the rsa scheme.the ingredients are the following:

P, q, two prime numbers (private, chosen)

n = pq (public, calculated)

e, with gcd(φ(n),e) = 1; 1< e < φ(n) (public, chosen)

d = e-1mod φ(n) (private, calculated)

The private key consist of {d,n} and the public key consists of {e,n}.suppose that user a has published its public key and that user b wishes to send the message M to A. then b calculates C = Me (mod n) and transmits C. on receipt of this cipher text, user a decrypts by calculating M = Cd (mod n).

It is worthwhile to summarize the justification for this algorithm. We have chosen e and d such that

d = e-1 mod ø (n)

Therefore,

Ed = 1 mod ø (n)

Therefore,ed is of the form kø(n)+1

So M ed = mod n .Now

C = Me mod n

M = Cd mod n = (Me)d mod n = M mod n

The keys were generated as follows:

1. Select two prime numbers p and q

2. Calculate n = pq

3. Calculate ø (n) = (p-1) (q-1)

4. Select e such that e is relatively prime to ø (n) and less than ø(n)

5. Determine d such that de = 1 mod ø (n)

KEY GENERATION

Before the application of the public key cryptosystem, each participant must generate a pair of keys. This involves the following tasks:

1. Determining two prime numbers, p and q

2. Selecting either e or d and calculating the other

The procedure for picking a prime number is as follows:

1. Pick an odd integer n at random

2. Pick an integer a < n at random

3. Perform the probabilistic primarily test, such as Miller Rabin. If n fails the test, reject the value n and go to step 1.

4. If n has passed a sufficient number of tests, accept n; otherwise, go to step

THE SECURITY OF RSA

Three possible approaches to attacking the RSA algorithm are as follows :

1. Brute force: This involves trying all possible private keys.

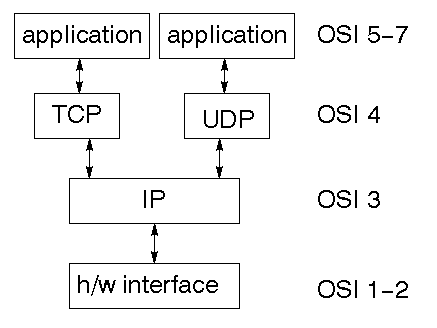
2. Mathematical attacks: There are several approaches, all equivalent in effect to factoring the product of two primes

3. Timing attacks: These depend on the running time of the decryption algorithm.

**Analysis of Network Client Server**

### TCP/IP stack

The TCP/IP stack is shorter than the OSI one:



TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

### IP datagram’s

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

### UDP

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

### TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

### Internet address

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

### Network address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

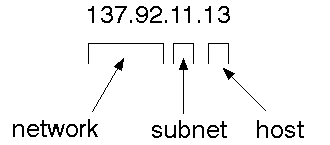
### Subnet address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

### Host address

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

### Total address



The 32 bit address is usually written as 4 integers separated by dots.

### Port address

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

**Eclipse IDE**

Eclipse is an open-source software framework written primarily in Java. In its default form it is an Integrated Development Environment (IDE) for Java developers, consisting of the Java Development Tools (JDT) and the Eclipse Compiler for Java (ECJ). Users can extend its capabilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. Language packs are available for over a dozen languages.

**Architecture**

The basis for Eclipse is the Rich Client Platform (RCP). The following components constitute the rich client platform:

* OSGi - a standard bundling framework
* Core platform - boot Eclipse, run plug-ins
* the Standard Widget Toolkit (SWT) - a portable widget toolkit
* JFace - viewer classes to bring model view controller programming to SWT, file buffers, text handling, text editors
* the Eclipse Workbench - views, editors, perspectives, wizards

Eclipse's widgets are implemented by a widget toolkit for Java called SWT, unlike most Java applications, which use the Java standard Abstract Window Toolkit (AWT) or Swing. Eclipse's user interface also leverages an intermediate GUI layer called JFace, which simplifies the construction of applications based on SWT.

Eclipse employs plug-ins in order to provide all of its functionality on top of (and including) the rich client platform, in contrast to some other applications where functionality is typically hard coded. This plug-in mechanism is a lightweight software componentry framework. In addition to allowing Eclipse to be extended using other programming languages such as C and Python, the plug-in framework allows Eclipse to work with typesetting languages like LaTeX, networking applications such as telnet, and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. Java and CVS support is provided in the Eclipse SDK.

The key to the seamless integration of tools with Eclipse is the plugin. With the exception of a small run-time kernel, everything in Eclipse is a plug-in. This means that a plug-in you develop integrates with Eclipse in exactly the same way as other plug-ins; in this respect, all features are created equal.

The Eclipse SDK includes the Eclipse Java Development Tools, offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. The IDE also makes use of a workspace, in this case a set of metadata over a flat filespace allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards. The Visual Editor project allows interfaces to be created interactively, hence allowing Eclipse to be used as a RAD tool.

The following is a list of notable projects and plugins for the Eclipse IDE.

These projects are maintained by the Eclipse community and hosted by the Eclipse Foundation.

1. **Core projects**

Rich Client Platform (Platform) is the core framework that all other Eclipse projects are built on.

Java Development Tools (JDT) provides support for core Java SE. This includes a standalone fast incremental compiler.

**Tools projects**

C/C++ Development Tools (CDT) adds support for C/C++ syntax highlighting, code formatting, debugger integration and project structures. Unlike the JDT project, the CDT project does not add a compiler and relies on an external tool chain.

Graphical Editing Framework (GEF) allows developers to build standalone graphical tools. Example use include circuit diagram design tools, activity diagram editors and WYSIWYG document editors.

**Web projects**

J2EE Standard Tools (JST) extends the core JDT to include support for Java EE projects. This includes EJBs, JSPs and Servlets.

PHP Development Tools (PDT)

Web Standard Tools (WST) adds standards compliant web development tools. These tools include editors for XML, HTML and CSS.

**Other projects**

Test and Performance Tools Platform (TPTP) which provides a platform that allows software developers to build test and performance tools, such as debuggers, profilers and benchmarking applications.

Business Intelligence and Reporting Tools Project (BIRT), an Eclipse-based open source reporting system for web applications, especially those based on Java

**SYSTEM DESIGN**

Design is multi-step process that focuses on data structure software architecture, procedural details, (algorithms etc.) and interface between modules. The design process also translates the requirements into the presentation of software that can be accessed for quality before coding begins.

Computer software design changes continuously as new methods; better analysis and broader understanding evolved. Software Design is at relatively early stage in its revolution.

Therefore, Software Design methodology lacks the depth, flexibility and quantitative nature that are normally associated with more classical engineering disciplines. However techniques for software designs do exist, criteria for design qualities are available and design notation can be applied.

**Module Description**

**User Interface Design:**

In this module we design the user interface window. The window is designed in order to display all the processes in this project. We use the Swing package available in Java to design the User Interface. Swing is a widget toolkit for Java. It is part of Sun Microsystems' Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs.We design the user interface Window by using Swing package available in Java.

**Packet Separation:**

In this module we use the browse button to load an input text file. This process is done by using the File Dialog class available in Java. After loading the file we read all the characters inside the input file. After that we separate the total characters available into blocks of equal numbers. This process is known as packet separation.

**Designing the Queue:**

The Queue is designed in order to create the packet loss. We create packet loss in this module voluntarily in order to measure it. The packets from the sender are received here and loss is created. Then the remaining packet which passes the Queue is sent to the Receiver.

**Packet Receiver:**

In this module we design a Receiver, which is used to receive the packets. The packets which are remaining after the loss in the Queue are received here. These packets are displayed in this window. Thus we can know the packet loss in the Receiver window. After that we can use the parameters, Badabing and Poisson modulated process to calculate the packet loss in next module.

**Packet Loss Calculations:**

In this module we calculate the packet loss. We calculate the packet loss accurately by using the Badabing. We also calculate the packet loss using the traditional technique known as Poisson modulated process. Finally we show the results in a window in order to compare the measurement of packet loss calculations to prove our accuracy in Badabing over the Poisson modulated process.

**UML DIAGRAMS**

**Unified Modeling Language**:

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

* + User Model View
    1. This view represents the system from the users perspective.
    2. The analysis representation describes a usage scenario from the end-users perspective.
  + Structural model view
    1. In this model the data and functionality are arrived from inside the system.
    2. This model view models the static structures.
* Behavioral Model View

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

* Implementation Model View

In this the structural and behavioral as parts of the system are represented as they are to be built.

* Environmental Model View

In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

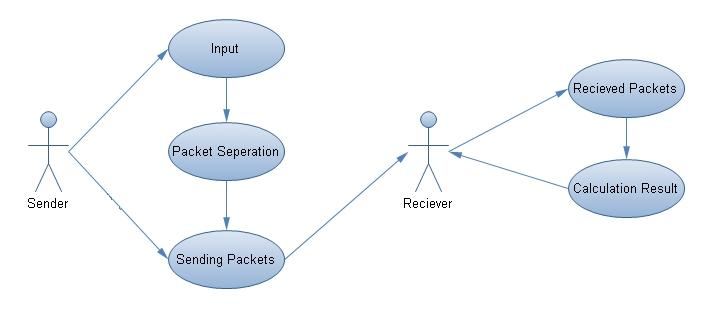
UML is specifically constructed through two different domains they are:

* UML Analysis modeling, this focuses on the user model and structural model views of the system.
* UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

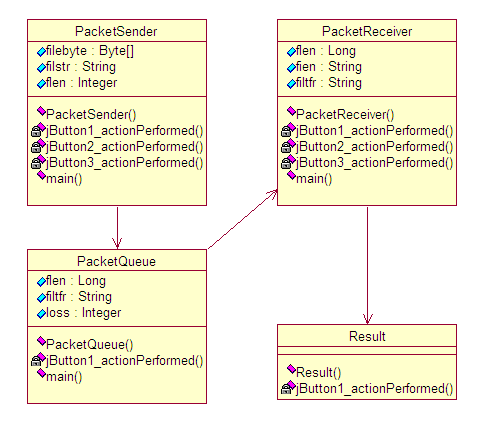
Use case Diagrams represent the functionality of the system from a user’s point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

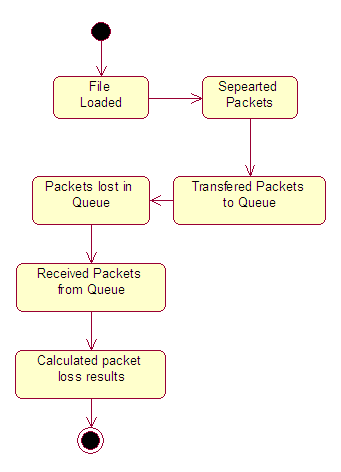
**Use Case Diagram**

****

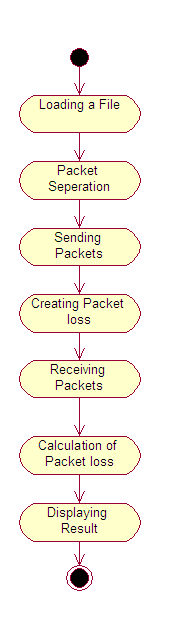
**Class Diagram**

****

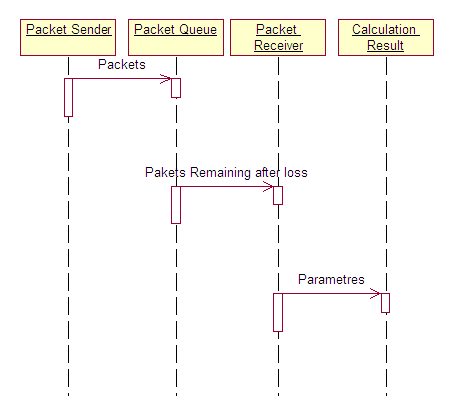
**State Diagram**

****

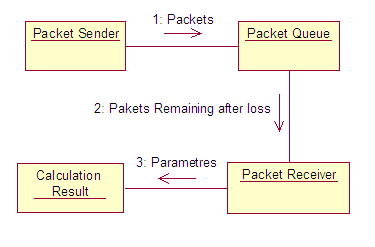
**Activity Diagram**

****

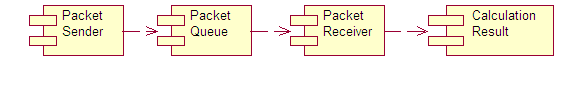
**Sequence Diagram**

****

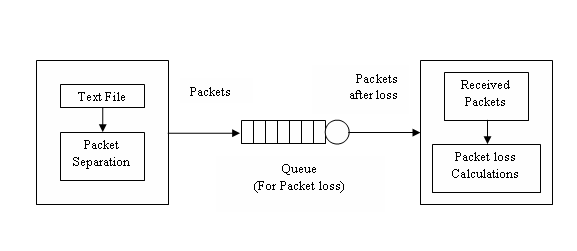
**Collaboration Diagram**

****

**Component Diagram**

****

**Data Flow Diagram**

****

**System Architecture**

A Geometric Approach to Improving Active Packet Loss Measurement

File Loading

Receiving Packets from Sender

Receiving Packets from Queue

Packet Separation

Sending Packets to Queue

Creating Packet loss

Sending Packets to Receiver

Packet loss Measurement

Calculations Result

**Technique used**

**Poisson modulated process**

We begin by using our capabilities to evaluate the simple Poisson-modulated loss probe measurements using the ZING tool. ZING measures packet loss in one direction on an end-to-end path. The ZING sender emits packets at Poisson-modulated intervals with timestamps and unique sequence numbers and the receiver logs the probe packet arrivals. We specify the mean probe rate, the probe packet size, and the number of packets in a “flight.”

**BADABING**

Moreover, simple techniques allows us to validate the measurement output are introduced. We implemented this method in a new tool, BADABING, which we tested in our laboratory. Our tests demonstrate that BADABING, in most cases, accurately estimates loss frequencies and durations over a range of cross traffic conditions. For the same overall packet rate, our results show that BADABING is significantly more accurate than Poisson probing for measuring loss episode characteristics.

Our methodology involves dispatching a sequence of probes, each consisting of one or more very closely spaced packets. The aim of a probe is to obtain a snapshot of the state of the network at the instant of probing. As such, the record for each probe indicates whether or not it encountered a loss episode, as evidenced by either the loss or sufficient delay of any of the packets within a probe.

The probes themselves are organized into what we term *basic experiments*, each of which comprises a number of packets sent in rapid succession. The aim of the basic experiment is to determine the dynamics of transitions between the congested and un-congested state of the network, i.e., beginnings and endings of loss episodes. Below we show how this enables us to estimate the duration of loss episodes.

A *full experiment* comprises a sequence of basic experiments generated according to some rule. The sequence may be terminated after some specified number of basic experiments, or after a given duration, or in an open-ended adaptive fashion, e.g., until estimates of desired accuracy for a loss characteristic have been obtained, or until such accuracy is determined impossible.

**Advantages**

The advantage of our study is to understand how to accurately measure loss characteristics on end-to-end paths with probes. We are interested in two specific characteristics of packet loss: loss episode frequency, and loss episode duration*.* Thus we improve the accuracy in measuring the packet loss. This is the major advantage of our work.

**Applications**

We can measure the packet loss more accurately than before. Thus we can find the networks performance by calculating the packet loss. By finding the networks performance we can use the network according to our needs. Thus we can evaluate the network performance by calculating the packet loss. We can measure packet loss in military networks, hospital networks, etc. These kind of networks transfers more important data, thus its performance should be high. So the measurement of packet loss should be negligible or nothing.

SYSTEM TESTING AND MAINTENANCE

**Testing**

Software Testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software. Testing is a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors. Quality is not an absolute; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behavior of the product against a specification. An important point is that software testing should be distinguished from the separate discipline of Software Quality Assurance (SQA), which encompasses all business process areas, not just testing.

There are many approaches to software testing, but effective testing of complex products is essentially a process of investigation, not merely a matter of creating and following routine procedure. One definition of testing is "the process of questioning a product in order to evaluate it", where the "questions" are operations the tester attempts to execute with the product, and the product answers with its behavior in reaction to the probing of the tester[citation needed]. Although most of the intellectual processes of testing are nearly identical to that of review or inspection, the word testing is connoted to mean the dynamic analysis of the product—putting the product through its paces. Some of the common quality attributes include capability, reliability, efficiency, portability, maintainability, compatibility and usability. A good test is sometimes described as one which reveals an error; however, more recent thinking suggests that a good test is one which reveals information of interest to someone who matters within the project community.

**Introduction**

In general, software engineers distinguish software faults from software failures. In case of a failure, the software does not do what the user expects. A fault is a programming error that may or may not actually manifest as a failure. A fault can also be described as an error in the correctness of the semantic of a computer program. A fault will become a failure if the exact computation conditions are met, one of them being that the faulty portion of computer software executes on the CPU. A fault can also turn into a failure when the software is ported to a different hardware platform or a different compiler, or when the software gets extended. Software testing is the technical investigation of the product under test to provide stakeholders with quality related information.

Software testing may be viewed as a sub-field of Software Quality Assurance but typically exists independently (and there may be no SQA areas in some companies). In SQA, software process specialists and auditors take a broader view on software and its development. They examine and change the software engineering process itself to reduce the amount of faults that end up in the code or deliver faster.

Regardless of the methods used or level of formality involved the desired result of testing is a level of confidence in the software so that the organization is confident that the software has an acceptable defect rate. What constitutes an acceptable defect rate depends on the nature of the software. An arcade video game designed to simulate flying an airplane would presumably have a much higher tolerance for defects than software used to control an actual airliner.

A problem with software testing is that the number of defects in a software product can be very large, and the number of configurations of the product larger still. Bugs that occur infrequently are difficult to find in testing. A rule of thumb is that a system that is expected to function without faults for a certain length of time must have already been tested for at least that length of time. This has severe consequences for projects to write long-lived reliable software.

A common practice of software testing is that it is performed by an independent group of testers after the functionality is developed but before it is shipped to the customer. This practice often results in the testing phase being used as project buffer to compensate for project delays. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes.

Another common practice is for test suites to be developed during technical support escalation procedures. Such tests are then maintained in regression testing suites to ensure that future updates to the software don't repeat any of the known mistakes.

It is commonly believed that the earlier a defect is found the cheaper it is to fix it.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Time Detected | | | | |
| Time Introduced | Requirements | Architecture | Construction | System Test | Post-Release |
| Requirements | 1 | 3 | 5-10 | 10 | 10-100 |
| Architecture | - | 1 | 10 | 15 | 25-100 |
| Construction | - | - | 1 | 10 | 10-25 |

In counterpoint, some emerging software disciplines such as extreme programming and the agile software development movement, adhere to a "test-driven software development" model. In this process unit tests are written first, by the programmers (often with pair programming in the extreme programming methodology). Of course these tests fail initially; as they are expected to. Then as code is written it passes incrementally larger portions of the test suites. The test suites are continuously updated as new failure conditions and corner cases are discovered, and they are integrated with any regression tests that are developed.

Unit tests are maintained along with the rest of the software source code and generally integrated into the build process (with inherently interactive tests being relegated to a partially manual build acceptance process).

The software, tools, samples of data input and output, and configurations are all referred to collectively as a test harness.

## History

The separation of debugging from testing was initially introduced by Glenford J. Myers in his 1978 book the "Art of Software Testing". Although his attention was on breakage testing it illustrated the desire of the software engineering community to separate fundamental development activities, such as debugging, from that of verification. Drs. Dave Gelperin and William C. Hetzel classified in 1988 the phases and goals in software testing as follows: until 1956 it was the debugging oriented period, where testing was often associated to debugging: there was no clear difference between testing and debugging. From 1957-1978 there was the demonstration oriented period where debugging and testing was distinguished now - in this period it was shown, that software satisfies the requirements. The time between 1979-1982 is announced as the destruction oriented period, where the goal was to find errors. 1983-1987 is classified as the evaluation oriented period: intention here is that during the software lifecycle a product evaluation is provided and measuring quality. From 1988 on it was seen as prevention oriented period where tests were to demonstrate that software satisfies its specification, to detect faults and to prevent faults. Dr. Gelperin chaired the IEEE 829-1988 (Test Documentation Standard) with Dr. Hetzel writing the book "The Complete Guide of Software Testing". Both works were pivotal in to today's testing culture and remain a consistent source of reference. Dr. Gelperin and Jerry E. Durant also went on to develop High Impact Inspection Technology that builds upon traditional Inspections but utilizes a test driven additive.

## White-box and black-box testing

To meet Wikipedia's quality standards, this section may require cleanup.  
Please discuss this issue on the talk page, and/or replace this tag with a more specific message.

White box and black box testing are terms used to describe the point of view a test engineer takes when designing test cases. Black box being an external view of the test object and white box being an internal view. Software testing is partly intuitive, but largely systematic. Good testing involves much more than just running the program a few times to see whether it works. Thorough analysis of the program under test, backed by a broad knowledge of testing techniques and tools are prerequisites to systematic testing. Software Testing is the process of executing software in a controlled manner; in order to answer the question “Does this software behave as specified?” Software testing is used in association with Verification and Validation. Verification is the checking of or testing of items, including software, for conformance and consistency with an associated specification. Software testing is just one kind of verification, which also uses techniques as reviews, inspections, walk-through. Validation is the process of checking what has been specified is what the user actually wanted.

* Validation: Are we doing the right job?
* Verification: Are we doing the job right?

In order to achieve consistency in the Testing style, it is imperative to have and follow a set of testing principles. This enhances the efficiency of testing within SQA team members and thus contributes to increased productivity. The purpose of this document is to provide overview of the testing, plus the techniques.

At SDEI, 3 levels of software testing is done at various SDLC phases

* Unit Testing: in which each unit (basic component) of the software is tested to verify that the detailed design for the unit has been correctly implemented
* Integration testing: in which progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a whole.
* System testing: in which the software is integrated to the overall product and tested to show that all requirements are met

A further level of testing is also done, in accordance with requirements:

* Acceptance testing: upon which the acceptance of the complete software is based. The clients often do this.
* Regression testing: is used to refer the repetition of the earlier successful tests to ensure that changes made in the software have not introduced new bugs/side effects.

In recent years the term grey box testing has come into common usage. The typical grey box tester is permitted to set up or manipulate the testing environment, like seeding a database, and can view the state of the product after his actions, like performing a SQL query on the database to be certain of the values of columns. It is used almost exclusively of client-server testers or others who use a database as a repository of information, but can also apply to a tester who has to manipulate XML files (DTD or an actual XML file) or configuration files directly. It can also be used of testers who know the internal workings or algorithm of the software under test and can write tests specifically for the anticipated results. For example, testing a data warehouse implementation involves loading the target database with information, and verifying the correctness of data population and loading of data into the correct tables.

## Test levels

* Unit testing tests the minimal software component and sub-component or modules by the programmers.
* Integration testing exposes defects in the interfaces and interaction between integrated components (modules).
* Functional testing tests the product according to programmable work.
* System testing tests an integrated system to verify/validate that it meets its requirements.
* Acceptance testing testing can be conducted by the client. It allows the end-user or customer or client to decide whether or not to accept the product. Acceptance testing may be performed after the testing and before the implementation phase. See also Development stage
  + Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.
  + Beta testing comes after alpha testing. Versions of the software, known as beta versions, are released to a limited audience outside of the company. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

It should be noted that although both Alpha and Beta are referred to as testing it is in fact use emersion. The rigors that are applied are often unsystematic and many of the basic tenets of testing process are not used. The Alpha and Beta period provides insight into environmental and utilization conditions that can impact the software.

After modifying software, either for a change in functionality or to fix defects, a regression test re-runs previously passing tests on the modified software to ensure that the modifications haven't unintentionally caused a regression of previous functionality. Regression testing can be performed at any or all of the above test levels. These regression tests are often automated.

## Test cases, suites, scripts and scenarios

A test case is a software testing document, which consists of event, action, input, output, expected result and actual result. Clinically defined (IEEE 829-1998) a test case is an input and an expected result. This can be as pragmatic as 'for condition x your derived result is y', whereas other test cases described in more detail the input scenario and what results might be expected. It can occasionally be a series of steps (but often steps are contained in a separate test procedure that can be exercised against multiple test cases, as a matter of economy) but with one expected result or expected outcome. The optional fields are a test case ID, test step or order of execution number, related requirement(s), depth, test category, author, and check boxes for whether the test is automatable and has been automated. Larger test cases may also contain prerequisite states or steps, and descriptions. A test case should also contain a place for the actual result. These steps can be stored in a word processor document, spreadsheet, database or other common repository. In a database system, you may also be able to see past test results and who generated the results and the system configuration used to generate those results. These past results would usually be stored in a separate table.

The term test script is the combination of a test case, test procedure and test data. Initially the term was derived from the byproduct of work created by automated regression test tools. Today, test scripts can be manual, automated or a combination of both.

The most common term for a collection of test cases is a test suite. The test suite often also contains more detailed instructions or goals for each collection of test cases. It definitely contains a section where the tester identifies the system configuration used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

Collections of test cases are sometimes incorrectly termed a test plan. They might correctly be called a test specification. If sequence is specified, it can be called a test script, scenario or procedure.

## A sample testing cycle

Although testing varies between organizations, there is a cycle to testing:

1. Requirements Analysis: Testing should begin in the requirements phase of the software development life cycle.

During the design phase, testers work with developers in determining what aspects of a design are testable and under what parameter those tests work.

1. Test Planning: Test Strategy, Test Plan(s), Test Bed creation.
2. Test Development: Test Procedures, Test Scenarios, Test Cases, Test Scripts to use in testing software.
3. Test Execution: Testers execute the software based on the plans and tests and report any errors found to the development team.
4. Test Reporting: Once testing is completed, testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.
5. Retesting the Defects

Not all errors or defects reported must be fixed by a software development team. Some may be caused by errors in configuring the test software to match the development or production environment. Some defects can be handled by a workaround in the production environment. Others might be deferred to future releases of the software, or the deficiency might be accepted by the business user. There are yet other defects that may be rejected by the development team (of course, with due reason) if they deem it inappropriate to be called a defect.

SYSTEM IMPLEMENTATION

Implementation is the most crucial stage in achieving a successful system and giving the user’s confidence that the new system is workable and effective. Implementation of a modified application to replace an existing one. This type of conversation is relatively easy to handle, provide there are no major changes in the system.

Each program is tested individually at the time of development using the data and has verified that this program linked together in the way specified in the programs specification, the computer system and its environment is tested to the satisfaction of the user. The system that has been developed is accepted and proved to be satisfactory for the user. And so the system is going to be implemented very soon. A simple operating procedure is included so that the user can understand the different functions clearly and quickly.

Initially as a first step the executable form of the application is to be created and loaded in the common server machine which is accessible to all the user and the server is to be connected to a network. The final stage is to document the entire system which provides components and the operating procedures of the system.

SCOPE FOR FUTURE DEVELOPMENT

Every application has its own merits and demerits. The project has covered almost all the requirements. Further requirements and improvements can easily be done since the coding is mainly structured or modular in nature. Changing the existing modules or adding new modules can append improvements. Further enhancements can be made to the application, so that the web site functions very attractive and useful manner than the present one.

CONCLUSION

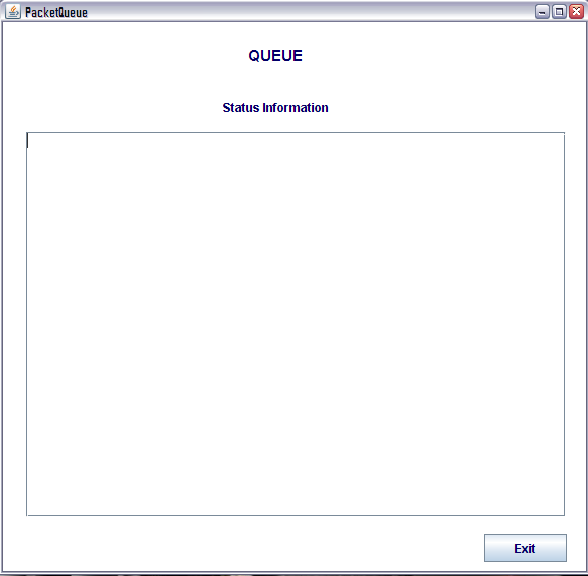
In this paper, we have studied the problem of combating Internet worms. To that end, we have developed a branching process model to characterize the propagation of Internet worms. Unlike deterministic epidemic models studied in the literature, this model allows us to characterize the early phase of worm propagation. Using the branching process model, we are able to provide a precise bound M on the total number of scans that ensure that the worm will eventually die out. Further, from our model, we also obtain the probability that the total number of hosts that the worm infects is below a certain level, as a function of the scan limit. The insights gained from analyzing this model also allow us to develop an effective and automatic worm containment strategy that does not let the worm propagate beyond the early stages of infection. Our strategy can effectively contain both fast scan worms and slow scan worms without knowing the worm signature in advance or needing to explicitly detect the worm. We show via simulations and real trace data that the containment strategy is both effective and non-intrusive.

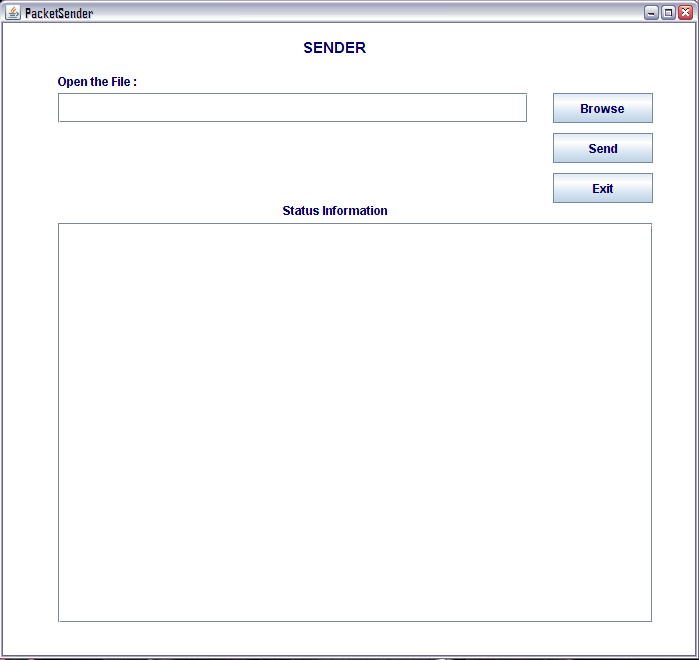
BIBLIOGRAPHY

|  |  |
| --- | --- |
| Core Java™ 2 Volume I – Fundamentals 7th Edition | * Cay S. Hortsman |
| Pearson Education – Sun Microsystems | Gary Cornell |
|  |  |
| Core Java™ 2 Volume II – Advanced | * Cay S. Hortsman |
| Pearson Education – Sun Microsystems | Gary Cornell |
|  |  |
| Head First Servlets & JSP | * Eric Freeman |
| O’Reilly – SPD | Elisabeth Freeman |
|  |  |
| JAVA 2: The Complete Reference | * Hert Schildt, |
| SPD | Herbert Schildt |
|  |  |
| Effective Java – Programming Language Guide | * Joshua Bloch |
| Pearson Education – Sun Microsystems |  |
|  |  |
| Java Networking Best Practices | * George Reese |
| O’Reilly – SPD |  |
|  |  |
| Pure JFC Swing | * Sathyaraj Pantham |
| O’Reilly – SPD | Sam Griffith |

**APPENDIX**

**SCREEN SHOTS**



****

**Sample Code:**

**Coding:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* PacketSender \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.net.\*;

import java.io.\*;

/\*\*

\* Summary description for PacketSender

\*

\*/

public class PacketSender extends JFrame

{

// Variables declaration

private JLabel jLabel1;

private JLabel jLabel2;

private JLabel jLabel3;

private JTextField jTextField1;

private JTextArea jTextArea1;

private JScrollPane jScrollPane1;

private JButton jButton1;

private JButton jButton2;

private JButton jButton3;

private JPanel contentPane;

public float filelength,r;

public byte filebyte[]=new byte[100000];

public String filstr[];

public String filpkt[];

public String Dest1="";

public int filint[];

public char filchar[];

public int i,j,k,l,m;

public int ch,packmest=0;

long stt,flen;

Socket st;

String s="";

String ss[];

// End of variables declaration

public PacketSender()

{

super();

initializeComponent();

//

// TODO: Add any constructor code after initializeComponent call

//

this.setVisible(true);

}

/\*\*

\* 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,15));

jLabel2 = new JLabel();

jLabel2.setFont(new Font("Arial",Font.BOLD,12));

jLabel3 = new JLabel();

jLabel3.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));

jTextArea1.setForeground(new Color(0, 0, 102));

jScrollPane1 = new JScrollPane();

jButton1 = new JButton();

jButton2 = new JButton();

jButton3 = new JButton();

contentPane = (JPanel)this.getContentPane();

//

// jLabel1

//

jLabel1.setForeground(new Color(0, 0, 102));

jLabel1.setText("SENDER");

//

// jLabel2

//

jLabel2.setForeground(new Color(0, 0, 102));

jLabel2.setText("Open the File : ");

//

// jLabel3

//

jLabel3.setForeground(new Color(0, 0, 102));

jLabel3.setText("Status Information");

//

// jTextField1

//

jTextField1.setForeground(new Color(0, 0, 102));

jTextField1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jTextField1\_actionPerformed(e);

}

});

//

// jTextArea1

//

//

// jScrollPane1

//

jScrollPane1.setViewportView(jTextArea1);

//

// jButton1

//

jButton1.setForeground(new Color(0, 0, 102));

jButton1.setText("Browse");

jButton1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jButton1\_actionPerformed(e);

}

});

//

// jButton2

//

jButton2.setForeground(new Color(0, 0, 102));

jButton2.setText("Send");

jButton2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jButton2\_actionPerformed(e);

}

});

//

// jButton3

//

jButton3.setForeground(new Color(0, 0, 102));

jButton3.setText("Exit");

jButton3.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jButton3\_actionPerformed(e);

}

});

//

// contentPane

//

contentPane.setLayout(null);

contentPane.setBackground(new Color(255, 255, 255));

addComponent(contentPane, jLabel1, 300,10,132,30);

addComponent(contentPane, jLabel2, 55,46,192,24);

addComponent(contentPane, jLabel3, 280,175,187,24);

addComponent(contentPane, jTextField1, 55,70,470,30);

addComponent(contentPane, jScrollPane1, 55,200,595,400);

addComponent(contentPane, jButton1, 550,70,100,30);

addComponent(contentPane, jButton2, 550,110,100,30);

addComponent(contentPane, jButton3, 550,150,100,30);

//

// PacketSender

//

this.setTitle("PacketSender");

this.setLocation(new Point(135, 133));

this.setSize(new Dimension(700, 660));

this.setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);

}

/\*\* 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)

{

}

private void jButton2\_actionPerformed(ActionEvent e)

{

}

private void jButton3\_actionPerformed(ActionEvent e)

{

}

//

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

//

//============================= Testing ================================//

//= =//

//= The following main method is just for testing this class you built.=//

//= After testing,you may simply delete it. =//

//======================================================================//

public static void main(String[] args)

{

/\*JFrame.setDefaultLookAndFeelDecorated(true);

JDialog.setDefaultLookAndFeelDecorated(true);

try

{

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

}

catch (Exception ex)

{

System.out.println("Failed loading L&F: ");

System.out.println(ex);

}\*/

new PacketSender();

}

//= End of Testing =

}