International Journal of Research (IJR) Vol-1, Issue-9, October 2014 **ISSN 2348-6848**

Multiple Routing Configuration for Fast IP Network Recovery

Ashok Koujalagi

Under the Guidance of

Dr. JAYAPRAKASH
(Assistant professor)
Department of Computer Science and Applications, The Oxford College of science, Bangalore-102

ABSTRACT

As the Internet takes an increasingly central role in our communications infrastructure, the slow convergence of routing protocols after a network failure becomes a growing problem. To assure fast recovery from link and node failures in IP networks, we present a new recovery scheme called Multiple Routing Configurations (MRC). Our proposed scheme guarantees recovery in all single failure scenarios, using a single mechanism to handle both link and node failures, and without knowing the root cause of the failure. MRC is strictly connectionless, and assumes only destination based hop-by-hop forwarding. MRC is based on keeping additional routing information in the routers, and allows packet forwarding to continue on an alternative output link immediately after the detection of a failure. It can be implemented with only minor changes to existing solutions. In this paper we present MRC, and analyze its performance with respect to scalability, backup path lengths, and load distribution after a failure. We also show how an estimate of the traffic demands in the network can be used to improve the distribution of the recovered traffic, and thus reduce the chances of congestion when MRC is used.

INTRODUCTION

In recent years the Internet has been transformed from a special purpose network to an ubiquitous platform for a wide range of everyday communication services. The demands on Internet reliability and availability have increased accordingly. A disruption of a link in central parts of a network has the potential to affect hundreds of thousands of phone conversations or TCP connections, with obvious adverse effects.

The ability to recover from failures has always been a central design goal in the Internet. IP networks are intrinsically robust, since IGP routing protocols like OSPF are designed to update the forwarding information based on the changed topology after a failure. This re-convergence assumes full distribution of the new link state to all routers in the network domain. When the new

state information is distributed, each router individually calculates new valid routing tables.

This network-wide IP re-convergence is a time consuming process, and a link or node failure is typically followed by a period of routing instability. During this period, packets may be dropped due to invalid routes. This phenomenon has been studied in both IGP and BGP context, and has an adverse effect on real-time applications. Events leading to a re-convergence have been shown to occur frequently.

Much effort has been devoted to optimizing the different steps of the convergence of IP routing, i.e., detection, dissemination of information and shortest path calculation, but the Manuscript received December 21, 2006, revised July 21 2007 All authors are with Simula Research Laboratory, Oslo, Norway convergence time is still too large for applications with real time demands . A key problem is that since most network failures are short lived, too rapid triggering of the reconvergence process can cause route flapping and increased network instability.

The IGP convergence process is slow because it is reactive and global. It reacts to a failure after it has happened, and it involves all the routers in the domain. In this paper we present a new scheme for handling link and node failures in IP networks. Multiple Routing Configurations (MRC) is a proactive and local protection mechanism that allows recovery in the range of milliseconds. MRC allows packet forwarding to continue over pre-configured alternative next-hops immediately after the detection of the failure. Using MRC as a first line of defense against network failures, the normal IP convergence process can be put on hold. This process is then initiated only as a consequence of non-transient failures. Since no global re-routing is performed, fast failure detection mechanisms like fast hellos or hardware alerts can be used to trigger MRC without compromising network stability. MRC guarantees recovery from any single link or node failure, which constitutes a large majority of the failures experienced in a network. MRC makes no assumptions with respect to the root cause of failure, e.g., whether the packet forwarding is disrupted due to a failed link or a failed router.

The main idea of MRC is to use the network graph and the associated link weights to produce a small set of backup network configurations. The link weights in these backup configurations are manipulated so that for each link and node failure, and regardless of whether it is a link or node failure, the node that detects the failure can safely forward the incoming packets towards the destination on an alternate link. MRC assumes that the network uses shortest path routing and destination based hop-by-hop forwarding.

The shifting of traffic to links bypassing the failure can lead to congestion and packet loss in parts of the network. This limits the time that the proactive recovery scheme can be used to forward traffic before the global routing protocol is informed about the failure, and hence reduces the chance that a transient failure can be handled without a full global routing re-convergence. Ideally, a proactive recovery scheme should not only guarantee connectivity after a failure, but also do so in a manner that does not cause an unacceptable load distribution. This requirement has been noted as being one of the principal challenges for precalculated IP recovery schemes. With MRC, the link weights are set individually in each backup configuration. This gives great flexibility with respect to how the recovered traffic is routed. The backup configuration used after a failure is selected based on the failure instance, and thus we can choose link

weights in the backup configurations that are well suited for only a subset of failure instances.

OBJECTIVE

The main objective is to assure fast recovery from link and node failures in IP networks, we present a new recovery scheme called Multiple Routing Configurations (MRC). This scheme guarantees recovery in all single failure scenarios, using a single mechanism to handle both link and node failures, and without knowing the root cause of the failure. MRC is based on keeping additional routing information in the routers, and allows packet forwarding to continue on an alternative output link immediately after the detection of a failure. In this paper we present MRC, and analyze its performance with respect to scalability, backup path lengths, and load distribution after a failure. We also show how an estimate of the traffic demands in the network can be used to improve the distribution of the recovered traffic, and thus reduce the chances of congestion when MRC is used.

MRC OVERVIEW

MRC is based on building a small set of backup routing configurations that are used to route recovered traffic on alternate paths after a failure. The backup configurations differ from the normal routing configuration in that link weights are set so as to avoid routing traffic in certain parts of the network. We observe that if all links attached to a node are given sufficiently high link weights, traffic will never be routed through that node. The failure of that node will then only affect traffic that is sourced at or destined for the node itself. Similarly, to exclude a link (or a group of links) from taking part in the routing, we give it infinite weight. The link can then fail without any consequences for the traffic.

Our MRC approach is threefold. First, we create a set of backup configurations, so that every network component is excluded from packet forwarding in one configuration. Second, for each configuration, a standard routing algorithm like OSPF is used to calculate configuration specific shortest paths and create forwarding tables in each router, based on the configurations. The use of a standard routing algorithm guarantees loop-free forwarding within one configuration. Finally, we design a forwarding process that takes advantage of the backup configurations to provide fast recovery from a component failure.

In our approach, we construct the backup configurations so that for all links and nodes in the network, there is a configuration where that link or node is not used to forward traffic. Thus, for any single link or node failure, there will exist a configuration that will route the traffic to its destination
on a path that avoids the failed element. Also, the backup configurations must be constructed so that all nodes are reachable in all configurations, i.e., there is a valid path with a finite cost between each node pair. Shared Risk Groups can also be protected, by regarding such a group as a single component that must be avoided in a particular configuration. Here we formally describe MRC and how to generate configurations that protect every link and node in a network. Using a standard shortest path calculation, each router creates a set of configuration-specific forwarding tables. For

Simplicity, we say that a packet is forwarded according to a configuration, meaning that it is forwarded using the forwarding table calculated based on that configuration. In this paper we talk about building a separate

forwarding table for each configuration, but we believe that more efficient solutions can be found in a practical implementation.

When a router detects that a neighbour can no longer be reached through one of its interfaces, it does not immediately inform the rest of the network about the connectivity failure. Instead, packets that would normally be forwarded over the failed interface are marked as belonging to a backup configuration, and forwarded on an alternative interface towards its destination. The selection of the correct backup configuration, and thus also the backup next-hop, is detailed in .The packets must be marked with a configuration identifier, so the routers along the path know which configuration to use. Packet marking is most easily done by using specific values in the DSCP field in the IP header. If this is not possible, other packet marking strategies like IPv6 extension headers or using a private address space and tunnelling could be used.

If a failure lasts for more than a specified time interval, a normal re-convergence will be triggered. MRC does not interfere with this convergence process, or make it longer than normal. However, MRC gives continuous packet forwarding during the convergence, and hence makes it easier to use mechanisms that prevents micro-loops during convergence,
at the cost of longer convergence times . If a failure is deemed permanent, new configurations must be generated based on the altered topology.

REVIEW OF LITERATURE

PAPER1: Routing of multipoint connections by author B. M. Waxman.

The author addresses the problem of routing connections in a large-scale packet-switched network supporting multipoint communications. He gives a formal definition of several versions of the multipoint problem, including both static and dynamic versions. He looks at the Steiner tree problem as an example of the static problem and considers the experimental performance of two approximation algorithms for this problem. A weighted greedy algorithm is considered for a version of the dynamic problem which allows endpoints to come and go during the life of a connection. One of the static algorithms serves as a reference to measure the performance of the proposed weighted greedy algorithm in a series of experiments

PAPER2: The Design Philosophy of the DARPA Internet Protocols by author D.D. Clark

The Internet protocol suite, TCP/IP, was first proposed fifteen years ago. It was developed by the Defense Advanced Research Projects Agency (DARPA), and has been used widely in military and commercial systems. While there have been papers and specifications that describe how the protocols work, it is sometimes difficult to deduce from these why the protocol is as it is. For example, the Internet protocol is based on a connectionless or datagram mode of service. The motivation for this has been greatly misunderstood. This paper attempts to capture some of the early
reasoning which shaped the Internet protocols.

PAPER3: Delayed Internet routing convergence by author C. Labovitz, A. Ahuja , A. Bose, and F. Jahanian

This paper examines the latency in Internet path failure, failover, and repair due to the convergence properties of inter domain

routing. Unlike circuit-switched paths which exhibit failover on the order of milliseconds, our experimental measurements show that inter domain routers in the packet-switched Internet may take tens of minutes to reach a consistent view of the network topology after a fault. These delays stem temporary routing table fluctuations formed during the operation of the Border Gateway Protocol (BGP) path selection process on Internet backbone routers. During these periods delayed convergence, we show that end-to-end Internet paths will experience intermittent loss of connectivity, as well as increased packet loss and latency. We present a two-year study of Internet routing convergence through the experimental instrumentation of key portions of the Internet infrastructure, including both passive data collection and fault-injection machines at Internet exchange points. Based on data from the injection and measurement of several hundred thousand inter domain routing faults, we describe several unexpected properties of convergence and show that the measured upper bound on Internet inter domain routing convergence delay is an order of magnitude slower than previously thought. Our analysis also shows that the upper theoretic computational bound on the number of router states and control messages exchanged during the process of BGP convergence is factorial with respect to the number of autonomous systems in the Internet. Finally, we demonstrate that much of the observed convergence delay stems from specific router vendor implementation decisions and ambiguity in the BGP specification.

PAPER4: An approach to alleviate link overload as observed on an IP backbone by author S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot.

Shortest path routing protocols may suffer from congestion due to the use of a single shortest path between a source and a destination. The goal of our work is to first understand how links become overloaded in an IP backbone, and then to explore if the routing protocol, -either in its existing form, or in some enhanced form could be made to respond immediately to overload and reduce the likelihood of its occurrence. Our method is to use extensive measurements of Sprint's backbone network, measuring 138 links between September 2000 and June 2001. We find that since the backbone is designed to be over provisioned, link overload is rare, and when it occurs, 80% of the time it is caused due to link failures. Furthermore, we find that when a link is overloaded, few (if any) other links in the network are also overloaded. This suggests that deflecting packets to less utilized alternate paths could be an effective method for tackling overload. We analytically derive the condition that a network, which has multiple equal length shortest paths between every pair of nodes (as is common in the highly meshed backbone networks) can provide for loop-free deflection paths if all the link weights are within a ratio 1 + 1/(d- I) of each other; where d is the diameter of the network. Based on our measurements, the nature of the backbone topology and the careful use of link weights, we propose a deflection routing algorithm to tackle link overload where each node makes local decisions. Simulations suggest that this can be a simple and efficient way to overcome link overload, without requiring any changes to the routing protocol.

SYSTEM DESIGN

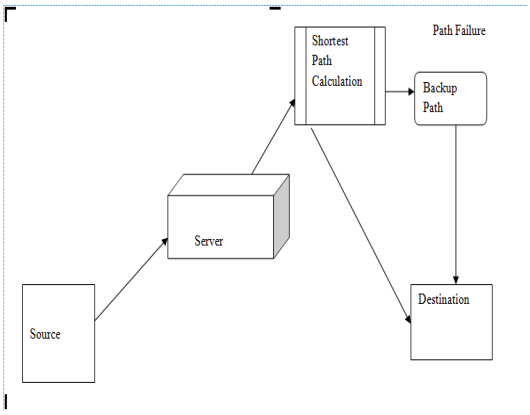
4.1 Existing System:

In Existing system, node failure and link failure problem handle while run time of the process. So time taken for recovery is very high. And then packet loss, packet reorder, delays occurring on the packet transmission. Recover the Node failure problem after packet dropped. Because backup link is not available. Choosing Alternate link select by send control packets through other links. Suppose alternate link also has same failure node means total time spending for select alternate link is waste.

4.2 Proposed System:

In proposed scheme guarantees recovery in all single failure scenarios, using a single mechanism to handle both link and node failures, and without knowing the root cause of the failure. Here we maintain backup link for each link. We present MRC, and analyze its performance with respect to scalability, backup path lengths, and load distribution after a failure. We also show how an estimate of the traffic demands in the network can be used to improve the distribution of the recovered traffic, and thus reduce the chances of congestion when MRC is used.

4.3 Architecture Diagram:

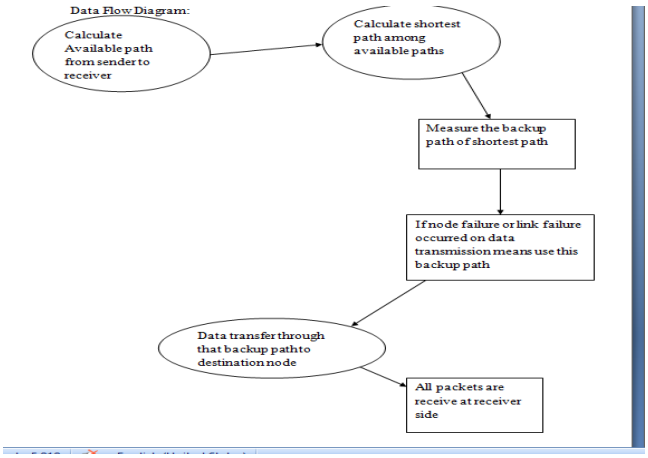


Working Procedure

Step1: Source sends the data to server.
Step2: Server updates the path information to the routing table and calculates shortest paths and backup paths between different nodes.
Step 3: Finally data is sent from source to destination.

The data flow diagram below gives step by step procedure in corcern to above architecture.

Data Flow Diagram



SYSTEM IMPLEMENTATION

5.1 Requirement Analysis:

Software Requirements

Java1.5
Java Swing
Sql Server 2000
Windows Xp.

Hardware Requirements

Hard disk : 60GB
RAM : 1GB
Processor : P IV

5.2 SOFTWARE REQUIRMENTS

Java Technology

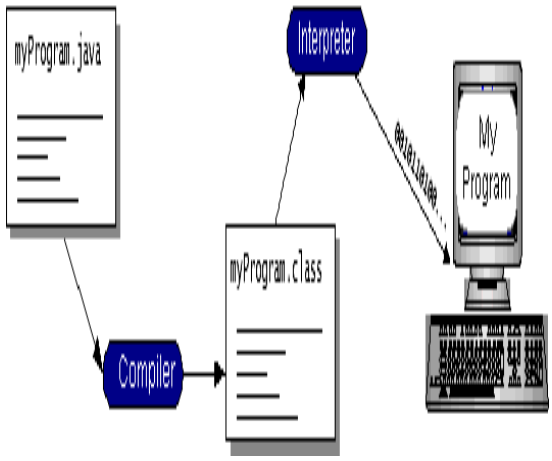
Java technology is both a programming language and a platform.

The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

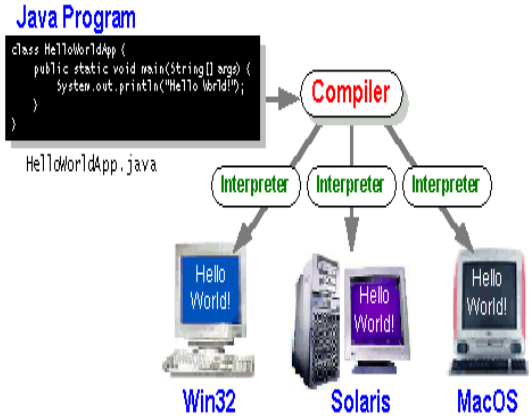
- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



You can think of Java bytecodes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java bytecodes help make "write once, run anywhere" possible. You can compile your program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac

.



The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

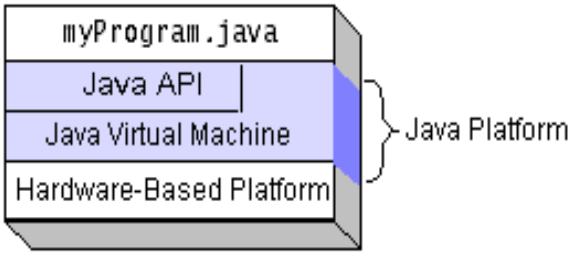
The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do?, highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability.

How does the API support all these kinds of programs? It does so with packages of software components that provide a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by applets.
- **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Data

gram Protocol) sockets, and IP (Internet Protocol) addresses.

- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security**: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components**: Known as JavaBeans™, can plug into existing component architectures.
- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC™)**: Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.

ODBC

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for application developers and database systems providers. Before ODBC became a *de facto* standard for Windows programs to interface with database systems, programmers had to use proprietary languages for each database they wanted to connect to. Now, ODBC has made the choice of the database system almost irrelevant from a coding perspective, which is as it should be. Application developers have much more important things to worry about than the syntax that is needed to port their program from one database to another when business needs suddenly change.

Through the ODBC Administrator in Control Panel, you can specify the particular database that is associated with a data source that an ODBC application program is written to use. Think of an ODBC data source as a door with a name on it. Each door will lead you to a particular database. For example, the data source named Sales Figures might be a SQL Server database, whereas the Accounts Payable data source could refer to an Access database. The physical database referred to by a data source can reside anywhere on the LAN.

The ODBC system files are not installed on your system by Windows 95. Rather, they are installed when you setup a separate database application, such as SQL Server Client or Visual Basic 4.0. When the ODBC icon is installed in Control Panel, it uses a file called ODBCINST.DLL. It is also possible to administer your ODBC data sources through a stand-alone program called ODBCADM.EXE. There is a 16-bit and a 32-bit version of this program, and each maintains a separate list of ODBC data sources.

From a programming perspective, the beauty of ODBC is that the application can be written to use the same set of function calls to interface with any data source, regardless of the database vendor. The source code of the application doesn't change whether it talks to Oracle or SQL Server. We only mention these two as an example. There are ODBC drivers available for several dozen popular database systems. Even Excel spreadsheets and plain text files can be turned into data sources. The operating system uses the Registry information written by ODBC Administrator to determine which low-level

ODBC drivers are needed to talk to the data source (such as the interface to Oracle or SQL Server). The loading of the ODBC drivers is transparent to the ODBC application program. In a client/server environment, the ODBC API even handles many of the network issues for the application programmer.

The advantages of this scheme are so numerous that you are probably thinking there must be some catch. The only disadvantage of ODBC is that it isn't as efficient as talking directly to the native database interface. ODBC has had many detractors make the charge that it is too slow. Microsoft has always claimed that the critical factor in performance is the quality of the driver software that is used. In our humble opinion, this is true. The availability of good ODBC drivers has improved a great deal recently. And anyway, the criticism about performance is somewhat analogous to those who said that compilers would never match the speed of pure assembly language. Maybe not, but the compiler (or ODBC) gives you the opportunity to write cleaner programs, which means you finish sooner. Meanwhile, computers get faster every year.

JDBC

In an effort to set an independent database standard API for Java, Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of "plug-in" database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

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

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

JDBC Goals

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

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

SQL Level API

The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to "generate" JDBC code and to hide many of JDBC's complexities from the end user.

1. ***SQL Conformance***

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

2. ***JDBC must be implemental on top of common database interfaces***

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

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

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

4. ***Keep it simple***

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

5. ***Use strong, static typing wherever possible***

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

6. ***Keep the common cases simple***

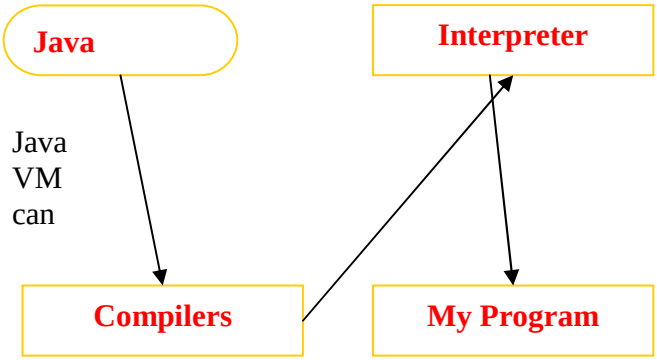
Because more often than not, the usual SQL calls used by the programmer are simple SELECT's, INSERT's, DELETE's and UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

Finally we decided to proceed the implementation using Java Networking. And for dynamically updating the cache table we go for MS SQL database. Java ha two things: a programming language and a platform.

Java is also unusual in that each Java program is both compiled and interpreted. With a compile you translate a Java program into an intermediate language called Java byte codes the platform-independent code instruction is passed and run on the computer.

Compilation happens just once; interpretation occurs each time the program is executed. The figure illustrates how this works.

You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The



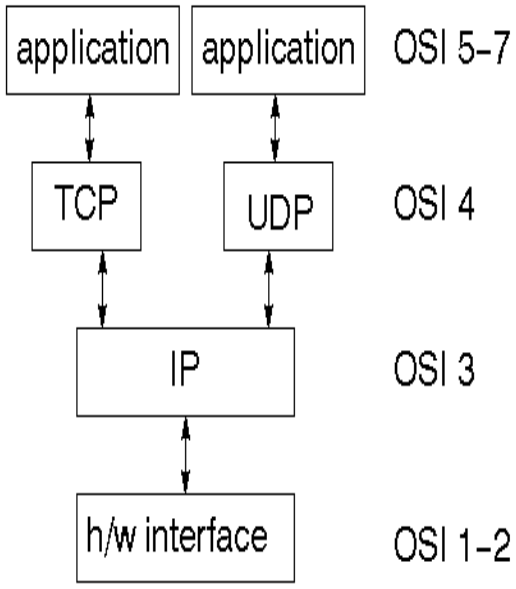
also be implemented in hardware.

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

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 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 addresses

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.

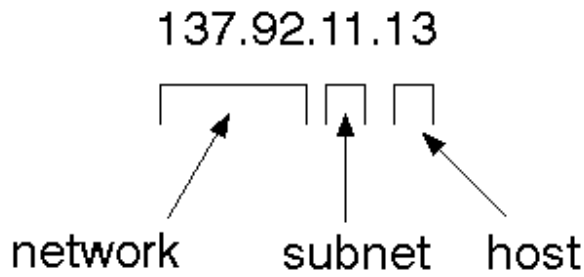
Subnet address

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 addresses

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".

Sockets

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call socket. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with Read File and Write File functions.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int family, int type, int protocol);
```

Here "family" will be AF_INET for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

5.3 System Modules With Code:

- Topology Construction
- Link Failure Detection
- Node Failure Detection
- Backup Path Transmission

Topology Construction:

In this module we design a topology to overcome the link failure and node failure problem. In the network, numerous nodes are interconnected and exchange data or services directly with each other nodes. Each node has Connection with other nodes. Each node details are maintained in the server system. Link details also maintain in the server system

Node Failure Detection:

In this module we find the Node failure by using send control packets through links.

If any node failure means acknowledgement is not there. So we easily find the node failure. Data does not reach destination. Then we solve this problem by using alternate node selection in the link.

Link Failure Detection:

In this module we find the Link failure by using Node failure result.. Each node has more than one path. We find the shortest path by using cost based technique and use that shortest path. .If data does not reach destination through this Link means that is called link failure .we can solve this problem by using backup path a

Backup Path Transmission:

In this module we find more than one shortest path for each transmission. And use this path like Backup path for data transmission. Suppose any problem in data transmission means that sender use this alternate path for transmission of data to receiver node. So time consuming for alternate path selection is reduced.

PROJECT CODES

Login Code

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;

public class login extends JFrame
implements MouseListener
{

        // Variables declaration
        private JLabel login;
        private JLabel nuser;
        private JLabel cancel;
   private JLabel jLabel5;
        private JLabel jLabel7;


        private JPanel contentPane;
        public static ServerSocket ssoc1;

        public static Socket sousoc1,ss1;
  static   node sf1;
 public static String username="";
 static int portno;
        public login() throws Exception
        {
                super();
                initializeComponent();

      this.setVisible(true);
        }


        public void initializeComponent()
        {
                nuser= new JLabel("New
User");
                login = new JLabel("Login");
                cancel = new
JLabel("Cancel");
                jLabel5 = new JLabel();
                jLabel7 = new JLabel(new
ImageIcon("system.jpg"));

                contentPane =
(JPanel)this.getContentPane();



login.setCursor(Cursor.getPredefinedCursor(
Cursor.HAND_CURSOR));

      login.addMouseListener(this);


        nuser.setCursor(Cursor.getPredefined
Cursor(Cursor.HAND_CURSOR));
```

```
nuser.addMouseListener(this);


cancel.setCursor(Cursor.getPredefine
dCursor(Cursor.HAND_CURSOR));


cancel.addMouseListener(this);


contentPane.setLayout(null);

contentPane.setBackground(new
Color(255, 255, 255));
addComponent(contentPane, login,
25,25,70,18);
addComponent(contentPane,
nuser, 125,25,70,18);
addComponent(contentPane, cancel ,
227,25,70,18);

addComponent(contentPane,
jLabel5, -1,203,495,31);
addComponent(contentPane,
jLabel7, 10,50,500,200);



this.setTitle("login");
this.setLocation(new Point(19,
37));
this.setSize(new
Dimension(500, 350));

this.setDefaultCloseOperation(DO_N
OTHING_ON_CLOSE);
this.setResizable(false);
}

/** 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);
}


public void
mousePressed(MouseEvent m)
{
try
{
if(m.getSource()==login)
{

new userlogin(portno);
dispose();
}
if(m.getSource()==nuser)
{
new newuser(portno);
dispose();
}
if(m.getSource()==cancel)
{

System.exit(0);
}
}
catch (Exception e)
{
e.printStackTrace();
}

}
public void
mouseClicked(MouseEvent m1)
{

}
public void
mouseEntered(MouseEvent m2)
{
```

```
        }
        public void
mouseExited(MouseEvent m3)
        {

        }
        public void
mouseReleased(MouseEvent m4)
        {

        }


 //==============================
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.setDefaultLookAndFeelDecor
ated(true);

        JDialog.setDefaultLookAndFeelDeco
rated(true);
                try
                {
      sf1=new node();
                new login();
                portno=new sender().login();
ssoc1=new ServerSocket(portno);

                while(true)
                {
                ss1=ssoc1.accept();

                sf1.receiver();
                }

                }

                catch (Exception ex)
                {

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

                }
 }
```

NewUser Code

```
 import java.awt.*;
 import java.awt.event.*;
 import javax.swing.*;
 import java.net.*;
 import java.util.*;
 /**
  * Summary description for login
  *
  */
 public class newuser extends JFrame
 {

        // Variables declaration
        private JLabel userlabel;
        private JLabel nodelabel;
        private JLabel passwordlabel;
        private JLabel jLabel5;
        private JTextField userfield;
        private JPasswordField
jPasswordField1;
                private JComboBox nodes;

        private JButton login;
        private JButton back;
        private JButton connection;
```

```
        private JPanel contentPane;
         public static ServerSocket ssoc1;

        public static Socket sousoc1,ss1;
        // End of variables declaration
static   sender sf1;
 public static String
username="",pword="",nodename="";
 static int portno;
 Vector nodes1;
        public newuser(int portno) throws
Exception
        {
                super();
                 sf1=new sender();
                 sf1.portnodetails(portno);
        nodedetails();
                initializeComponent();

      this.setVisible(true);
        }
 public void nodedetails()throws Exception
         {
          nodes1=new Vector();
         nodes1=sf1.ndetails();
         }

        public void initializeComponent()
        {
                userlabel = new JLabel();
                nodelabel = new JLabel();
                passwordlabel = new JLabel();
                jLabel5 = new JLabel(new
ImageIcon("user.jpg"));
                userfield = new JTextField();
                jPasswordField1 = new
JPasswordField();
                nodes=new
JComboBox(nodes1);
                login = new JButton();
                back = new JButton();
                connection=new JButton();
                contentPane =
(JPanel)this.getContentPane();

        userlabel.setHorizontalAlignment(Sw
ingConstants.CENTER);

        userlabel.setHorizontalTextPosition(S
wingConstants.CENTER);
                userlabel.setText("Node
Name");

        passwordlabel.setHorizontalAlignmen
t(SwingConstants.CENTER);

        passwordlabel.setHorizontalTextPosit
ion(SwingConstants.CENTER);
                passwordlabel.setText("Pass
word");

nodelabel.setHorizontalAlignment(SwingCon
stants.CENTER);

        nodelabel.setHorizontalTextPosition(
SwingConstants.CENTER);
                nodelabel.setText("Nodes");


                //
                // userfield
                //

        userfield.addActionListener(new
ActionListener() {
                        public void
actionPerformed(ActionEvent e)
                        {

        userfield_actionPerformed(e);
                        }

                });
                //
                // jPasswordField1
                //
```

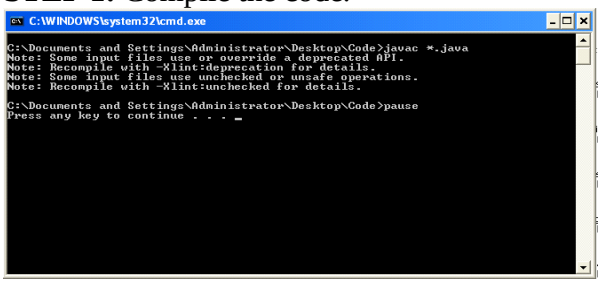
```
        jPasswordField1.addActionListener(n
ew ActionListener() {
                        public void
actionPerformed(ActionEvent e)
                        {

        jPasswordField1_actionPerformed(e);
                        }

                });
                //
                // login
                //
                login.setText("Submit");
                login.addActionListener(new
ActionListener() {
                        public void
actionPerformed(ActionEvent e)
                        {

        login_actionPerformed(e);
                        }

                });
                //
                // clear
                //
                back.setText("Back");
                back.addActionListener(new
ActionListener() {
                        public void
actionPerformed(ActionEvent e)
                        {

        back_actionPerformed(e);
                        }

                });


                connection.setText("Path");

        connection.addActionListener(new
ActionListener() {
                                public void
actionPerformed(ActionEvent e)
                                {
                                        try
                                        {


        if(!username.equals(nodename))
                                                {

        sf1.connection(username,nodename);

        connection.setEnabled(false);

                                                }
                                                else
                                                {

        JOptionPane.showMessageDialog(nul
l,"Please Enter the Proper value");

                                                }
                                        }
                                        catch
(Exception e2)
                                        {

        e2.printStackTrace();
                                        }

                                }


                        });


connection.setEnabled(false);
                        nodes.setBackground(new
Color(251, 250, 250));
                        nodes.addActionListener(new
ActionListener() {
                                public void
actionPerformed(ActionEvent e)
                                {
```

```
        nodes_actionPerformed(e);
                        }

                });

                //
                // contentPane
                //
                contentPane.setLayout(null);
                addComponent(contentPane,
userlabel, 106,150,70,28);
                addComponent(contentPane,
nodelabel, 106,230,70,28);
                addComponent(contentPane,
passwordlabel,106,190,70,28);



                addComponent(contentPane,
userfield, 232,150,100,25);
                addComponent(contentPane,
nodes, 232,230,100,25);

        addComponent(contentPane,jPasswor
dField1, 232,190,100,25);

                addComponent(contentPane,
jLabel5, 380,150,100,200);



                addComponent(contentPane,
login, 50,300,100,28);
                addComponent(contentPane,
back, 150,300,100,28);
                addComponent(contentPane,
connection, 250,300,100,28);



                this.setTitle("User Account
Creation");
                this.setLocation(new Point(19,
37));
                this.setSize(new
Dimension(500, 370));
                this.setDefaultCloseOperation(DO_N
OTHING_ON_CLOSE);
                        this.setResizable(false);
        }

        /** 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);
        }



        private void
userfield_actionPerformed(ActionEvent e)
        {

        System.out.println("\nuserfield_action
Performed(ActionEvent e) called.");



        }

        private void
jPasswordField1_actionPerformed(ActionEv
ent e)
        {

        System.out.println("\njPasswordField
1_actionPerformed(ActionEvent e) called.");

        }

        private void
login_actionPerformed(ActionEvent e)
        {
                try
                {

        username=userfield.getText();
```

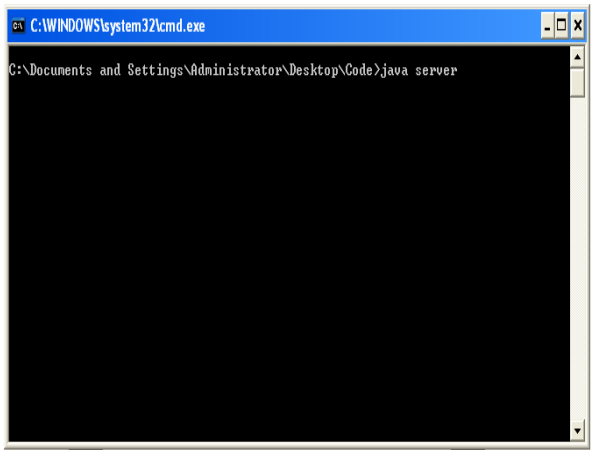
```
        pword=jPasswordField1.getText();
sf1.userlogin(username,pword);
login.setEnabled(false);
    }
                catch (Exception e1)
                {
                }


        System.out.println("\nlogin_actionPer
formed(ActionEvent e) called.");
                // TODO: Add any handling
code here
        }

        private void
back_actionPerformed(ActionEvent m)
        {

                try
                        {
new login();
dispose();
                        }
                        catch(Exception e)
                        {
                        }
        }

        private void
nodes_actionPerformed(ActionEvent e)
        {

        System.out.println("\njComboBox1_a
ctionPerformed(ActionEvent e) called.");

                Object o =
nodes.getSelectedItem();
                System.out.println(">>" +
((o==null)? "null" : o.toString()) + " is
selected.");
                // TODO: Add any handling
code here for the particular object being
selected
                        nodename=o.toString();
                        if
(!nodename.equals("select"))
                        {

        connection.setEnabled(true);
                        }
                        else
                        {

        connection.setEnabled(false);
                        }
                }
 }
```

RESULT ANALYSIS

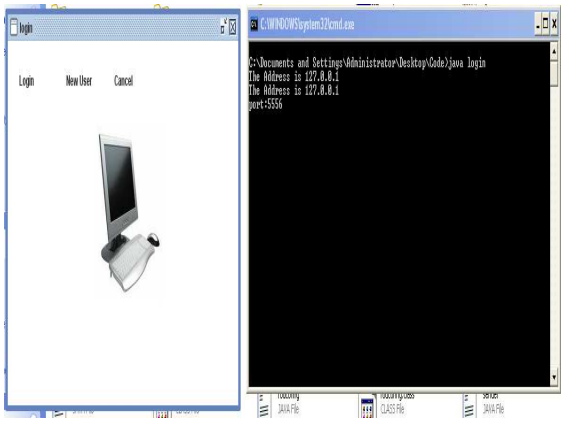
STEP 1: Compile the code.



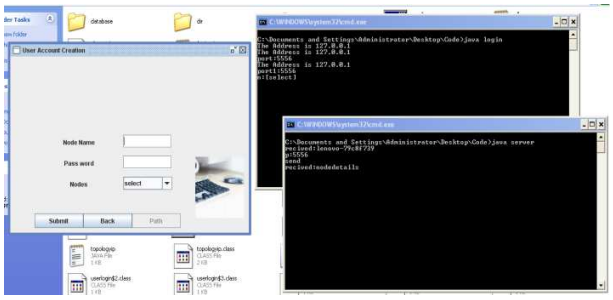
STEP 2: Run the server.



STEP 3: Login 1st node.

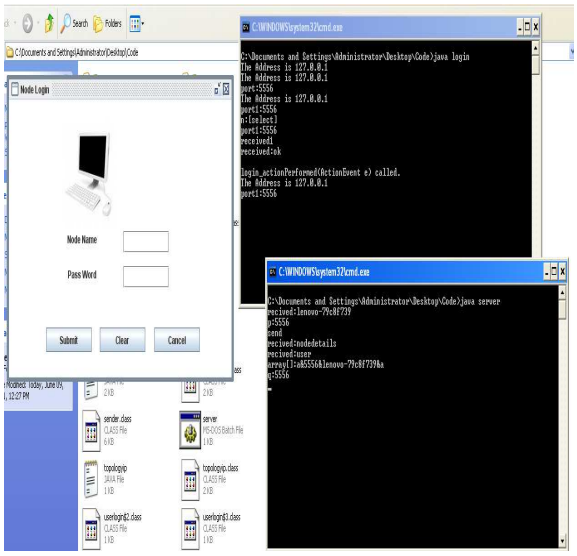


STEP 4: Create user account as new user. We get the following windows.

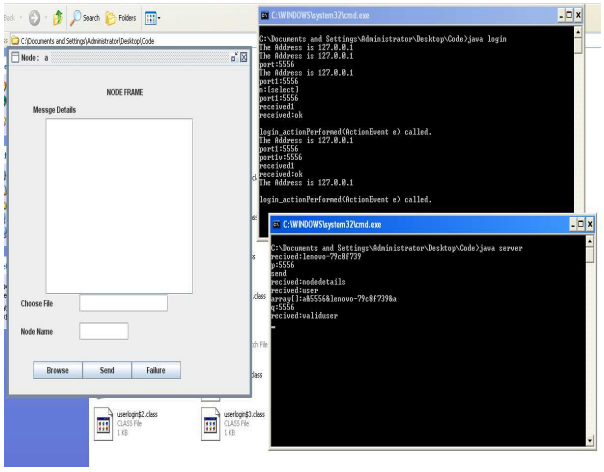


The user and server prompt updated as shown above as you enter user details.

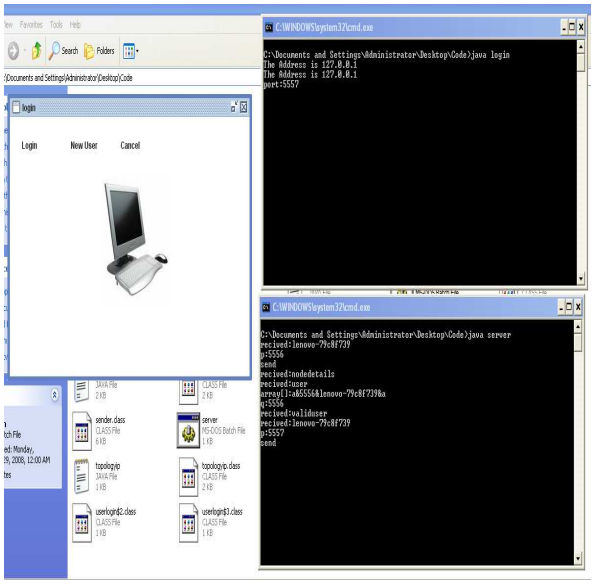
STEP 5: After entering user details login the node with same details. As shown below.



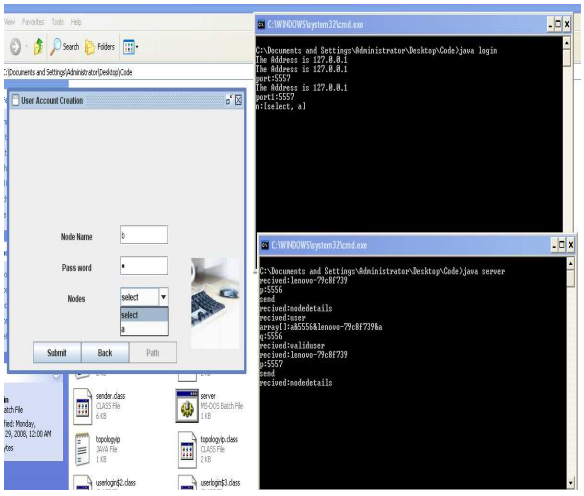
The port no: of the node which has loggined is sent to the server. Thus the 1st node named as "node a" is created. The frame is shown below.



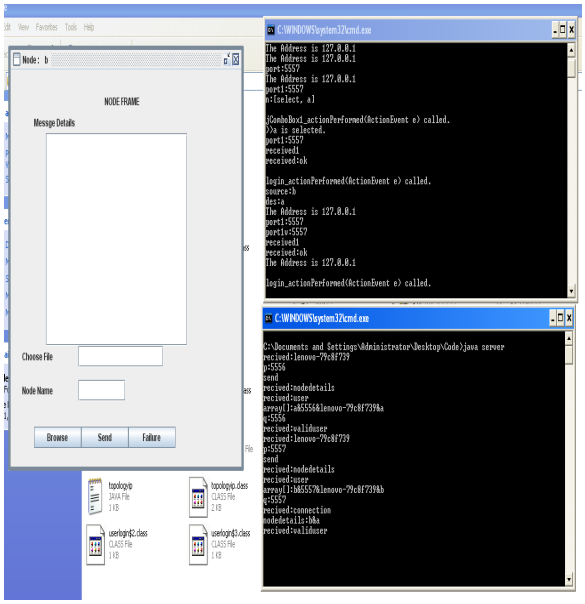
STEP 6: Now the 2nd user is logged in with the "node b". And the server gets the port no: of this new node. The updates are as shown below.



This node can link to the “node a” with the help of select option and then path is formed by selecting path button in the frame shown below**.**

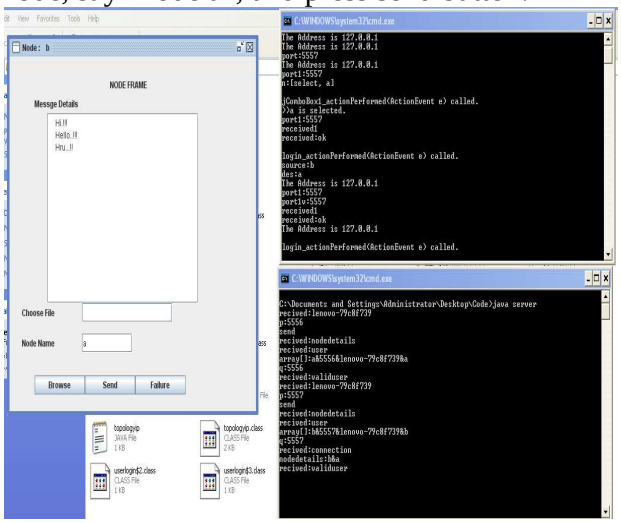


STEP 7: Next login the “node b” we get the fame as shown below.

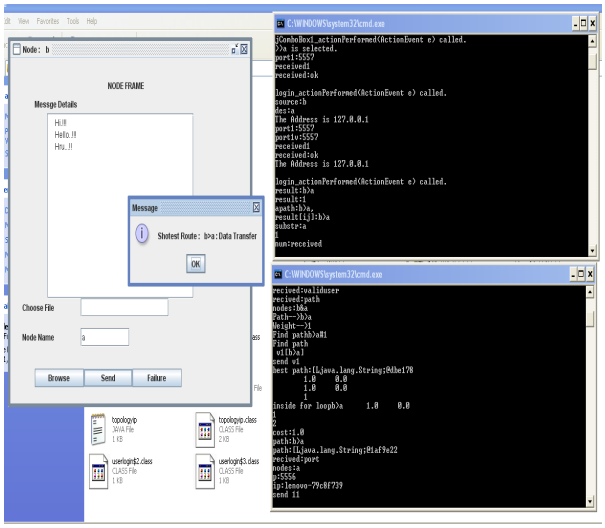


STEP 8: Similarly you can create any no: of nodes dynamically.

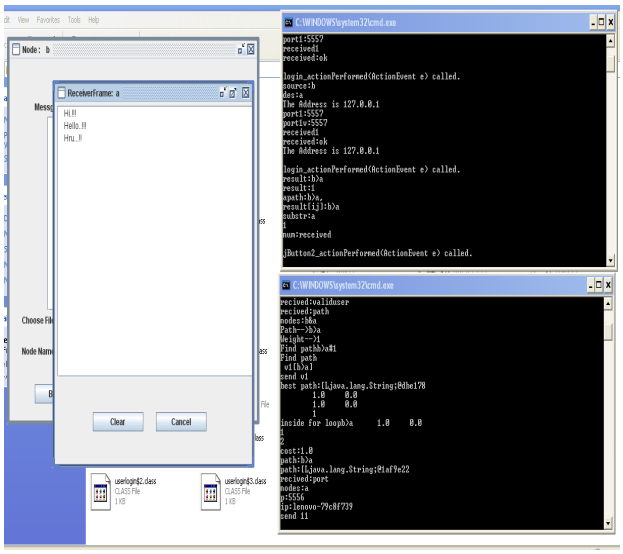
STEP 9: Now select the any source node, say “node b” and enter any data or message as shown below. Also select any destination node, say “node a”, and press send button.



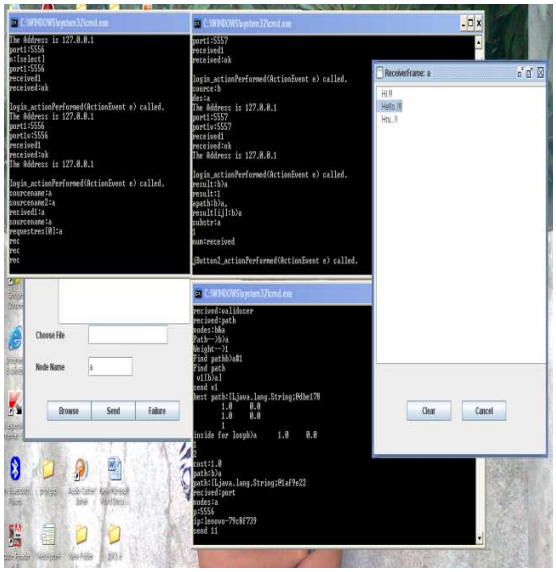
The shortest path from source to destination is chosen.



STEP 10: The data is moved from source to destination along the shortest path. The received frame at "node a" is shown below.



The updates at server, source and destination nodes after data transfer is shown below.



ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- Since no global re-routing is performed fast failure detection mechanism is done.
- MRC guarantees recovery from any single link or node failure.
- MRC makes no assumptions with respect to the root cause of the failure.

DISADVANTAGES

- There will not be any assurance for information to arrive at the target.
- Somehow it is time taking practice.
- It will not be having a accurate information of breakdown spots.

CHAPTER 8:
CONCLUSION

We have presented Multiple Routing Configurations as an approach to achieve fast

recovery in IP networks. MRC is based on providing the routers with additional routing configurations, allowing them to forward packets along routes that avoid a failed component. MRC guarantees recovery from any single node or link failure in an arbitrary bi-connected network. By calculating backup configurations in advance, and operating based on locally available information only, MRC can act promptly after failure discovery.

MRC operates without knowing the root cause of failure, i.e., whether the forwarding disruption is caused by a node or link failure. This is achieved by using careful link weight assignment according to the rules we have described. The link weight assignment rules also provide basis for the specification of a forwarding procedure that successfully solves the last hop problem.

The performance of the algorithm and the forwarding mechanism has been evaluated using simulations. We have shown that MRC scales well: 3 or 4 backup configurations is typically enough to isolate all links and nodes in our test topologies. MRC backup path lengths are comparable to the optimal backup path lengths—MRC backup paths are typically zero to two hops longer. We have evaluated the effect MRC has on the load distribution in the network while traffic is routed in the backup configurations, and we shave proposed a method that minimizes the risk of congestion after a link failure if we have an estimate of the demand matrix. In the COST239 network, this approach gave a maximum link load after the worst case link failure that was even lower than after a full IGP re-convergence on the altered topology. MRC thus achieves fast recovery with a very limited performance penalty.

REFERENCES

1. D. D. Clark, “The design philosophy of the DARPA internet protocols,” *SIGCOMM*, Computer Communications Review, vol. 18, no. 4, pp. 106–114, Aug. 1988.

2. A. Basu and J. G. Riecke, “Stability issues in OSPF routing,” in Proceedings of *SIGCOMM*, San Diego, California, USA, Aug. 2001, pp. 225–236.

3. C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, “Delayed Internet Routing Convergence,” *IEEE/ACM* Transactions on Networking, vol. 9, no. 3, pp. 293–306, June 2001.

4. C. Boutremans, G. Iannaccone, and C. Diot, “Impact of link failures on VoIP performance,” in Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video, 2002, pp. 63–71.