AN EXPLANATORY CASE STUDY ON HOW C++ PROGRAMMING LANGUAGE CAN BE USED TO IMPLEMENT IP ROUTING.

IBRAHIM UMAR FAROUK

2005/23563EE

ELECTRICAL ENGINEERING DEPARTMENT,SCHOOL OF ENGINEERING AND ENGINEERING TECHNOLOGY,FEDERAL UNIVERSITY OF TECHNOLOGY MINNA,NIGER STATE.

NOVEMBER, 2010

AN EXPLANATORY CASE STUDY ON HOW C++ PROGRAMMING LANGUAGE CAN BE USED TO IMPLEMENT IP ROUTING

IBRAHIM UMAR FAROUK

2005/23563EE

A THESIS SUBMITTED TO DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA.

NOVEMBER 2010

DEDICATION

I dedicate this work to my mother Fatima Ibrahim Umar.

DECLARATION

Ibrahim Umar Farouk, declare that this was done by me and has never been presented elsewhere for the award of a degree to the best of my knowledge. I also relinquish the copyrights of this project to the Federal University of Technology, Minna.

Ibrahim Umar Farouk

(Name of Student)

5/11/2010

(Signature and Date)

Mr.Steven S. Oyewobi

(Name of Supervisor)

5/12/2010.

(Signature and Date)

Engr. A.G Raji

(Name of H.O.D)

(Jan. 11, 2011)

(Signature and Date)

Engr Dr. G. I. Ibrahim

(Name of External Examiner)

6/12/10

(Signature and Date)

ACKNOWLEDGMENT

Glory be to AlmightyAllah whom out of His infinite mercy made it possible for this work to be completed. I wish to acknowledge the efforts of the lecturers Electrical Engineering Department F.U.T Minna, for their efforts in ensuring adequate capacity building throughout the years.

Special thanks to my family, friends and colleagues for the wonderful times we had, and for the moral support they always gave even at difficult times.

ABSTRACT

This project utilizes a computer programming based approach to provide an efficient means of routing packets of data in a network which will ease the problem of congestion being experienced by inter-network users.

This project attempts to follow a step by step approach of how a computer program (C++ in this case) is developed in such a way as to run a network router to perform IP routing and also another program to run a traditional computer as a router in any given network.

TABLE OF CONTENTS

CHAPTER ONE

GENERAL INTRODUCTION

1.1 INTRODUCTION

The ultimate goal of engineering in particular is to build real physical systems to perform given tasks. There are two methods of approach to this:

I. The analytical method
II. The empirical method

.The Analytical Method

The analytical method consists of four steps:

i. Modelling.
ii. Setting up Mathematical equation.
iii. Analysis.
iv. The Design.

The first two steps are closely related. If we use simple mathematics, then the model chosen must be correspondingly simple. If we use sophisticated mathematics then the model can be more complex and realistic. Modelling is the most critical step in analytical design. If a physical system is in correctly modelled, subsequent study will be useless.

Process modelling is the art or activity of building a mathematical model of a process by describing its fundamental physical and or chemical relationships without specifying how they are to be solved while process simulation is merely one of the activities that you can perform with that process model. Simulation is often an exercise and as such is often performed by relatively junior engineers.

The Empirical Method

The empirical method relies heavily on past experience and repeated experiments. This approach is carried out by trial and error. Although it is carried out by trial and error, it has been used successfully to design many systems. However, it is inadequate if there is no past experience to draw from or if experimentation is not feasible because of high cost or risk

This project looks into how a programming language (C++) is used to create software that can route packets of data in a network through the best possible route.

1.2 SIGNIFICANCE OF THE STUDY.

In the modern world, communication via the internet is growing rapidly. Vital information are sent and received constantly via this medium. Although the internet provides an easy means of communication, it also has its problems like any other means of communication.

The major problem faced by network by internet users and internet service providers is the inability to send and receive packets (information) smoothly. IP Routing provides a very effective and efficient solution to this problem whereby a network router selects the best possible route for a packets from its source to its destination which makes the exchange of packets (information) smooth and fast.

1.3 AIM AND OBJECTIVES.

AIM.

To study into how IP routing in a simple network can be accomplished by programming a hardware device..

OBJECTIVES.

1. To identify the function of a router.
2. To identify the different types of routing.
3. To investigate the procedures involved in each type of routing.
4. To select the best type of routing for this project.

5. To coordinate the procedures of the type of routing selected to form an algorithm for the proposed program.

1.4 METHODOLOGY.

This project will attempt to write two C++ programs that will run hardware devices to perform IP routing in a given network.

The first router for which the first program was written accepts an IP address as input from a computer or another router and looks up a stored table which is computed by the system administrator to see the least distance to that IP address and then forwards traffic through the best route to the requested address.

The second program was written to run a computer as a router whereby a given a computer in any kind of network can be chosen to act as host and route packets of data within the given network.

1.5 SCOPE AND LIMITATION.

This project will attempt to write two C++ programs that will route packets of data in a given network. The first program is for a simple network router that implements static routing whereby a system administrator is responsible for inputting the routes in the network manually while the second program is for a computer that will perform routing in a given network by accepting an IP address and sending packets to it.

But it does not cover the dynamic routing whereby routers on a given network communicate with each other and share information regarding the topology of the network thereby adjusting their routing tables automatically whenever there is a change in the network topology. The type of routing used in this project cannot be used for very large networks as the routes are computed manually.

CHAPTER TWO

LITERATURE REVIEW

2.1 EMPIRICAL METHOD IN SOFTWARE ENGINEERING RESEARCH.

Despite widespread interest in empirical software engineering, there is little guidance on which research methods are suitable to which research problems, and how to choose amongst them. Many researchers select inappropriate methods because they do not understand the goals underlying a method or possess little knowledge about alternatives. As a first step in helping researchers select an appropriate method, this chapter discusses key questions to consider in selecting a method, from philosophical considerations about the nature of knowledge to practical considerations in the application of the method. We characterize key empirical methods applicable to empirical software engineering, and explain the strengths and weaknesses of each.

Software engineering is a multi-disciplinary field, crossing many social and technological boundaries. To understand how software engineers construct and maintain complex, evolving software systems, we need to investigate not just the tools and processes they use, but also the

social and cognitive processes surrounding them. This requires the study of human activities. We need to understand how individual software engineers develop software, as well as how teams and organizations coordinate their efforts.

Because of the importance of human activities in software development, many of the research methods that are appropriate to software engineering are drawn from disciplines that study human behaviour, both at the individual level (e.g. psychology) and at the team and organizational levels (e.g. sociology).These methods all have known flaws, and each can only provide limited, qualified evidence about the phenomena being studied. However, each method is flawed differently (McGrath, 1995) and viable research strategies use multiple methods, chosen in such a way that the weaknesses of each method are addressed by use of complementary methods (Creswell, 2002). Describing in detail the wide variety of possible empirical methods and how to apply them is beyond the scope of

the chapter. Instead, we identify the five classes of research method that we believe are most relevant to software engineering:

- Controlled Experiments (including Quasi-Experiments);
- Case Studies (both exploratory and confirmatory);
- Survey Research;
- Ethnographies;
- Action Research.

2.2 COMPUTER PROGRAMMING

A computer is an electronic device that can input data, process data and output data, accurately and at great speed. Data are any kind of information that can be codified in some manner and input into the computer. Normally, we think of data as facts and numbers such as a person's name and address or the quantity or cost of an item purchased. However, data can also be graphical images, sound files, movies and more.

A computer is capable of inputting information such as the number of students in a class and their test scores. Processing data means to do something with it. Often we think of processing as performing some kind of calculations. If the number of students and their test scores have been input, then the obvious calculation would be finding the average score of the students. However, processing data can mean more than just calculations. If the names of the students have been entered, processing the data can also mean sorting the students' names into alphabetical order. Finally, to be useful, the computer needs to be able to output information, the results, to the user in an accurate and timely manner. The user is anyone that is making use of the results that the computer is producing.

A computer program is a series of instructions that tell the computer every step to take in the proper sequence in order to solve a problem for a user. A programmer is one who writes the computer program. When the computer produces a wrong result, it
can be traced to an improper sequence of instructions or incorrect data being input to the program. That is, the responsibility or blame lies on either the original programmer who wrote out the instructions for the computer to follow or the user who has entered incorrect data.

2.3 SOFTWARE DEVELOPOMENT

Software designers create new programs by using special application programs often called utility programs or development programs. A programmer uses another type of program called text editor to write the new program in a special notation called a programming language. With the text editor, the programmer creates a new text file which is an ordered list of instructions that make up the program source file. The machine dual instructions that make up the source file is called the source code. At this point a special applications program translates the source code into machine language or object code- a format that the operating system will recognize as a proper program to execute.

Three types of applications programs translate from source code to object code; compilers, interpreters and assemblers. The three operate differently and on different types of programming languages, but they serve the same purpose of translating from programming language to machine language (Marshall, 1997).

COMPILER

A compiler translates text files from within a high-level programming languag such as FORTRAN, C, PASCAL, from the source code to the object code all at once. This differs from the approach taken by interpreted languages such as BASIC, APL, and LISP in which a program is translated into object code statement by statement as each instruction is executed. The advantage of compiled languages over interpreted languages is that, compiled languages are compiled only once and thus can be executed by the computer much more quickly than interpreted languages. For this reason compiled languages are most common and are almost always used in professional and scientific applications.

Programs are often written as smaller pieces with each piece representing some aspect of the overall program. After each piece has been compiled separately, a program called a linker combines all of the translated pieces into a single executable program. Programs seldom work correctly the first time, so a program called debugger is often used to help find problems called bugs. Debugging programs usually detect an event in the executing program and point the programmer back to the origin of the event in the program source code (Marshall, 1997).

2.4 C++

C++ is a language used to program computers to perform specific tasks. C++, as the name implies, is essentially based on the C programming language. Therefore, it seems prudent to begin with a brief history of C. The C programming language was devised in the early 1970s at Bell Laboratories by Dennis Ritchie. It was designed as a system implementation language for the Unix operating system. The history of C and UNIX are closely related. For this reason a lot of UNIX programming is done with C. To some extent, C was originally based on the type less language BCPL; however it grew well beyond that.

The C++ programming language was invented by Bjarne Stroustroup. Work on what would become C++ began in 1979. The initial version was called "C with Classes." That name did not work out well, and was replaced with C++. The first version of C++ was used internally in AT&T in August 1983. The first commercial implementation was released in 1985. The C++ language standards are now handled by the American National Standards Institute (ANSI), and the International Standards Organization (ISO). This is why you often hear pure C++ referred to as ANSI Standard C++, or ISO Standard C++.

C programming language was developed first, C++ was developed later. C++ is essentially C taken to the next level. The most obvious difference between the two is that C++ supports object orientation. However, C++ sports many other improvements over C. For example, C++ handles strings better than C, and has a more robust exception handling. (Exception handling refers to a program's ability to handle unexpected errors).

C code will compile fine in most C++ compilers, but the reverse is not true. C++ code will not necessarily compile in a C compiler. Code is essentially the series of programming commands that a programmer writes. All the commands that make up a program are the source code for that program. C++ supports all C commands and also has many additions.

2.5 THE STEPS NEEDED TO WRITE A COMPUTER PROGRAM

Step 1. Fully understand the problem to be solved. Begin by looking over the output, what the program is supposed to be producing. Then look over the input that the program will be receiving. Finally, determine what general processing steps are going to be needed to turn that input into the required output.

Step 2. Design a solution using paper and pencil. Write out on paper the precise steps needed to solve the problem in the precise sequence. This is often called pseudo code. It is done by using English and perhaps some C++ like statements.

Step 3. Thoroughly desk check the solution. Desk check means to play computer and follow precisely the steps written down in the solution. You are looking for errors at this point.

Step 4. Code the solution into the programming language, C++ in our case.

Step 5. Compile the program.

Step 6. Test the program with one set of data. Try inputting one set of test data only. Examine the output and verify it is correct.

Step 7. Thoroughly test the program. At this point, one tests the program thoroughly and completely.

Step 8. Put the program into production. In the real world, this means that the program is given to the users who now run it to solve their problems. In the case of student programs, they are handed in to be graded by the instructor who plays the role of the user.

When writing a computer program it is absolutely vital that the problem to be solved is fully understood. A good percentage of large programming projects run into trouble owing to a misunderstanding of what is actually required. The specification of the problem is crucial. It is usual to write a detailed 'spec' of the problem using a 'natural language' (e.g., English). Unfortunately, it is very easy to write an ambiguous specification, which can be interpreted in a number of ways by different people. In order to combat this, a number of Formal Methods of specification have been developed. These methods are often high-level abstract mathematical languages, which are relatively simple to convert to high-level programs; in some cases this can be done automatically. Examples of such languages are Z and VDM. simple. Once the problem has been specified, it needs to be broken into small steps towards the solution, in other words an

algorithm should be designed. This is known as pseudo-code. The next two phases go hand-in-hand, they are often known as the code-test-debug cycle and it is often necessary to perform the cycle a number of times. It is very rare that a program is written correctly on the first attempt. It is common to make typographical errors which are usually unearthed by the compiler. Once the typos have been removed, the program will be able to be compiled and an executable image generated. Again, it is not uncommon for execution to expose more errors or bugs. Execution may either highlight run-time errors which occur when the program tries to perform illegal operations (e.g., divided by zero) or may reveal that the program is generating the wrong answers. The program must be thoroughly tested to demonstrate that it is indeed correct. The most basic goal should be to supply test data that executes every line of code. There are many software tools that generate statistical reports of code coverage, such as the UNIX tcov utility or the more comprehensive LDRA Test bed (Marshall, 1997).

2.6 NETWORK ROUTER.

A router is a device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in large collection of interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

A router is a networking device whose software and hardware are customised to the tasks of routing and forwarding information. A router has two or more network interfaces, which may be to different types of network (such as copper cables, fibre or wireless) or different network standards. Each network interface is a small computer specialized to convert electric signals from one form to another.

In the original 1960s-era of routing, general-purpose computers served as routers. Although general-purpose computers can perform routing, modern high-speed routers are highly specialised computers, generally with extra hardware added to accelerate both common routing functions such as packet

forwarding and specialised functions such as IPsec encryption . Other changes also improve reliability, such as using battery rather than mains power, and using solid-state rather than magnetic storage. Modern routers have thus come to resemble telephone switches, whose technology they are currently converging with and may eventually replace. The first modern (dedicated, standalone) routers were the Fuzzball routers.

Functionality of Router

A router must be connected to at least two networks, or it will have nothing to route. A special variety of router is the one-armed router used to route packets in a virtual LAN environment. In the case of a one-armed router the multiple attachments to different networks are all over the same physical link. A router which connects end-users to the Internet is called Edge router; A router which serves to transmit data between other routers is called Core router. A router creates and/or maintains a table, called a "routing table" that stores the best routes to certain network destinations and the "routing metrics" associated with those routes.

Routers connect two or more logical subnets, which do not share a common network address. The subnets in the router do not necessarily map one-to-one to the physical interfaces of the router. The term "layer 3 switching" is used often interchangeably with the term "routing". The term switching is generally used to refer to data forwarding between two network devices that share a common network address. This is also called layer 2 switching or LAN switching.

Conceptually, a router operates in two operational planes (or sub-systems) :

-:Control plane: where a router builds a table (called routing table) as how a packet should be forwarded through which interface, by using either statically configured statements (called static routes) or by exchanging information with other routers in the network through a dynamical routing protocol;

-Forwarding plane: where the router actually forwards traffic (called packets in IP) from ingress (incoming) interfaces to an egress (outgoing) interface that is appropriate for the destination address that the packet carries with it, by following rules derived from the routing table that has been built in the control plane.

A router has two key jobs:

- The router ensures that information doesn't go where it's not needed. This is crucial for keeping large volumes of data from clogging the network.
- The router makes sure that information does make it to the intended destination.

In performing these two jobs, a router joins the two networks, passing information from one to the other and, in some cases, performing translations of various protocols between the two networks. It also protects the networks from one another, preventing the traffic on one from unnecessarily spilling over to the other. This process is known as routing.

Routing is a function associated with the Network layer (layer 3) in the Open Systems Interconnection (OSI) model. Routers use network layer protocol headers, such as IP header where the source and destination addresses are included, and routing tables to determine the best path to forward the packets. For communication between routers and to decide the best route between any two

FIG. 1. Linksys Wireless-G Broadband Router

A router may create or maintain a table of the available routes and their conditions and use this information along with distance and cost algorithms to determine the best route for a given packet. Typically, a packet may travel through a number of network points with routers before arriving at its destination.

2.7 ROUTING TABLE.

A routing table is present on all IP nodes. The routing table stores information about IP networks and how they can be reached (either directly or indirectly). Because all IP nodes perform some form of IP routing, routing tables are not exclusive to IP routers. Any node loading the TCP/IP protocol has a routing table. There are a series of default entries according to the configuration of the node and additional entries can be entered either manually through TCP/IP utilities or dynamically through interaction with routers.

When an IP packet is to be forwarded, the routing table is used to determine:

1. The forwarding or next-hop IP address:
   For a direct delivery, the forwarding IP address is the destination IP address in the IP packet. For an indirect delivery, the forwarding IP address is the IP address of a router.
2. The interface to be used for the forwarding:
   The interface identifies the physical or logical interface such as a network adapter that is used to forward the packet to either its destination or the next router.

IP Routing Table Entry Types

An entry in the IP routing table contains the following information in the order presented:

Network ID. The network ID or destination corresponding to the route. The network ID can be class-based, subnet, or supernet network ID, or an IP address for a host route.

Network Mask. The mask that is used to match a destination IP address to the network ID.

Next Hop. The IP address of the next hop.

Interface. An indication of which network interface is used to forward the IP packet.

Metric. A number used to indicate the cost of the route so the best route among possible multiple routes to the same destination can be selected. A common use of the metric is to indicate the number of hops (routers crossed) to the network ID.

Routing table entries can be used to store the following types of routes:

Directly Attached Network IDs. Routes for network IDs that are directly attached. For directly attached networks, the Next Hop field can be blank or contain the IP address of the interface on that network.

Remote Network IDs. Routes for network IDs that are not directly attached but are available across other routers. For remote networks, the Next Hop field is the IP address of a local router in between the forwarding node and the remote network.

Host Routes. A route to a specific IP address. Host routes allow routing to occur on a per-IP address basis. For host routes, the network ID is the IP address of the specified host and the network mask is 255.255.255.255.

Default Route. The default route is designed to be used when a more specific network ID or host route is not found. The default route network ID is 0.0.0.0 with the network mask of 0.0.0.0.

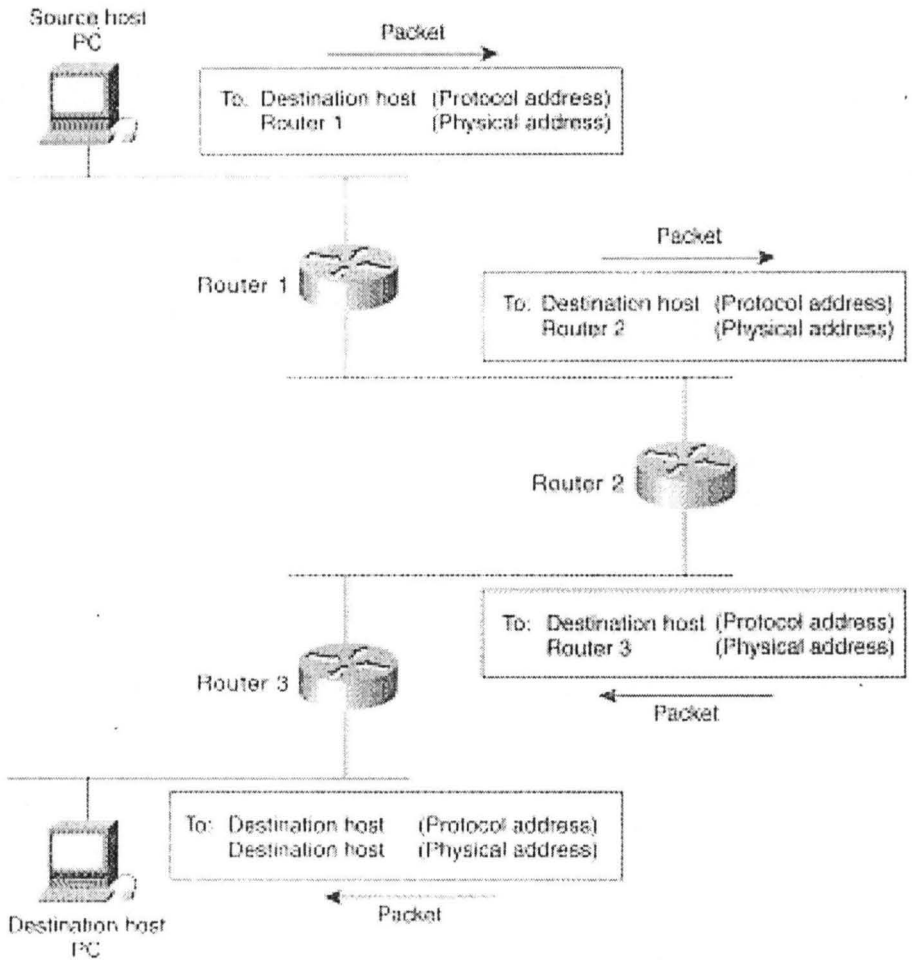
2.8 ROUTING.

Routing is the process of selecting paths in a network along which to send network traffic. Routing is performed for many kinds of networks, including the telephone network, electronic data networks (such as the internet), and transportation networks. This project is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding, the transit of logically addressed packets from their source toward their ultimate destination through intermediate nodes; typically hardware devices called routers, bridges, gateways, firewalls, or switches. General purpose computers can also forward packets and perform routing, though they are not specialised hardware and may suffer from limited performance. The routing process usually

directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the routers' memory, is very important for efficient routing. Most routing algorithms use only one network path at a time, but multipath routing techniques enable the use of multiple alternative paths.

Routing, in a more narrow sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Because structured addresses allow a single routing table entry to represent the route to a group of devices, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging) in large networks, and has become the dominant form of addressing on the internet, though bridging is still widely used within localised environments.



A Simple Example Of Routing.

Routing can be classified as static routing and or dynamic routing;

-Static routing: is a data communication concept describing one way of configuring path selection of routers in computer networks. It is the type of routing characterised by the absence of communication between routers regarding the current topology of the network. This is achieved by manually adding routes to the routing table.

In these systems, routes through a data network are described by fixed paths (statically). These routes are usually entered into the router by the system administrator. An entire network can be configured using static routes, but this type of configuration is not fault tolerant. When there is a change in the network or a failure occurs between two statically defined nodes, traffic will not be rerouted. This means that anything that wishes to take an affected path will either have to wait for the failure to be repaired or the static route to be updated by the administrator before restarting its journey. Most requests will time out before these repairs can be made. There are however times when static routes make sense and can even improve the performance of a network. The opposite of static routing is dynamic routing, sometimes also referred to as adaptive routing.

-Dynamic routing: Dynamic Routing performs the same function as static routing except it is more robust. Static routing allows routing tables in specific routers to be set up in a static manner so network routes for packets are set. If a router on the route goes down the destination may become unreachable. Dynamic routing allows routing tables in routers to change as the possible routes change. There are several protocols used to support dynamic routing. Routing Information Protocol (RIP): It helps routers dynamically adapt to changes of network connections by communicating information about which networks each router can reach and how far away those networks are. Open Shortest Path First (OSPF): It fixes many of the issues with RIP and allows routes to be selected dynamically based on the current state of the network, not just a static picture of how routers are connected. It also includes numerous advanced features, including support for a hierarchical topology (or a tree network) and automatic load sharing amongst routes.

-Advantages Of Static Routing Over Dynamic Routing.

1. Static routing has some enormous advantages over dynamic routing. Chief among these advantages is predictability. Because the network administrator computes the routing table in advance, the path a packet takes between two destinations is always known precisely, and can be controlled exactly. With dynamic routing, the path taken depends on which devices and links are functioning, and how the routers have interpreted the updates from other routers.
2. Additionally, because no dynamic routing protocol is needed, static routing doesn't impose any overhead on the routers or the network links.
3. Finally, static routing is easy to configure on a small network. The network administrator simply tells each router how to reach every network segment to which it is not directly attached.

-Disadvantages Of Static Routing.

1. The price of its simplicity is a lack of scalability. For five network segments on three routers, computing an appropriate route from every to every destination is not difficult. However, for much larger networks like for 200 network segments interconnected by more than a dozen routers, pre-computing routing tables quickly becomes a burden and is error prone.
2. When a network segment moves or is added, you would have to update the configuration for every router on the network. If you miss one, segments attached to that router will be unable to reach the moved or added segment and this may end up affecting many routers.
3. Finally, because static routing is by definition static, it cannot use redundant network links to adapt to a failure in the network.

-Advantages Of Dynamic Routing Over Static Routing.

1. The chief advantages of dynamic routing over static routing are scalability and adaptability. A dynamically routed network can grow more quickly and larger, and is able to adapt to changes in the network topology brought about by this growth or by the failure of one or more network components.

2. With a dynamic routing protocol, routers learn about the network topology by communicating with other routers. Each router announces its presence, and the routes it has available, to the other routers on the network. Therefore, if you add a new router, or add an additional segment to an existing router, the other routers will hear about the addition and adjust their routing tables accordingly. This reduces the chances that errors will occur.
3. The ability to learn about changes to the network's configuration has implications beyond adding new segments or moving old ones. It also means that the network can adjust to failures. If a network has redundant paths, then a partial network failure appears to the routers as if some segments got moved (they are now reached via alternate paths), and some segments have been removed from the network (they are now unreachable).

-Disadvantages Of Dynamic Routing.

1. Chief among the disadvantages is an increase in complexity.
2. In order to communicate information about the topology of the network, routers must periodically send messages to each other using a dynamic routing protocol. These messages must be sent across network segments just like any other packets. But unlike other packets in the network, these packets do not contain any information to or from a user. Instead, they contain information that is only useful to the routers. Thus, from the users' point of view, these packets are pure overhead. On a low-speed link, these messages can consume much of the available bandwidth, especially if the network is large or unstable.
3. Finally, some or all of the machines in a network may be unable to speak any dynamic routing protocol, or they may not speak a common protocol. If that is the case, static routing may be your only option.

CHAPTER THREE

METHODOLOGY

3.1 WRITING THE C++ PROGRAM ALGORITHM IMPLEMENTING IP ROUTING OF A NETWORK ROUTER.

The process of program development includes a number of subtasks. To be able to develop a program you must have an editor, a compiler and a linker . In modern program development platforms, these subtasks are integrated and the entire process is very transparent and informative. Such platforms are known as Integrated Development Environments (IDEs). Most of the modern C++ packages (the software that you will use to develop C++ programs) provide an IDE. Some of the commercially available packages include Turbo C++, Borland C++, C++ Builder and Visual C++.In this project we will be considering the visual C++ package.

The tasks of editing, compiling and linking are basically the tasks needed to successfully develop a C++ program that can be used as a software to run a hardware device.

3.2 Editing.

The first step in preparing your program is to use some kind of editor to type your program. Not every editor is suitable for this purpose. The edit program of DOS and the Notepad editor of Windows are two suitable editors. Integrated Development Environments (IDE) that are part of C++ packages provide built-in editors known as text editors. At the end of the editing session you must store the content of the a editor into a file.

3.3 Compiling.

The second step is to compile the source file. For this purpose, a special program known as a compiler is used. As part of the compiler, a program named the pre-processor is invoked. This takes place before the actual compilation of your program code. The pre-processor attends to your source code statements that start with the '#' sign. These statements are referred to as compiler directives. The pre-processor takes action as directed by these statements and will modify your original program file. At the end of pre-processing, all lines starting with the '#' sign will have been processed and eliminated.

3.4 Linking

The program that bridges all the gaps and completes assembly of the program is known as the linker. It will search all the object files and the libraries to find the missing sets of instructions. Sometimes the linker must be told to search certain libraries and object files. The linker automatically searches the libraries and object files that come with the C++ software. The linker will insert the missing sets of instructions into appropriate places to form a file that has a good execution path. This process is known as linking. At the end of the linking process, we have a file the PC can execute, known as an executable file.

The two programs written for this project(i.e program implementing IP routing) are divided into three basic groups which are; The header files, The source files and The resource files.

3.5 THE HEADER FILES.

The header files (having the extension .h) are text files that contain C++ programming statements, most of which do not form executable program statements. Not all statements in your program are executable. However, the statements in header files play a major role in the preparation of your program. The majority of the statements in a header file assist the compiler to carry out a thorough check of the program statements you write in your program. Once the header files are written and tested, we do not change them. If the compiler issues error or mismatch messages, then we must change our program not the header file.

The header files also known as pre-processor directives start with a '#' and take up the whole line. The pre-processors are supplied by the compiler vendors. The directive causes

direct text Substitution. The pre-processor fetches the whole file whose name is specified as the directive

Argument and replaces the directive by the contents of the file taken. This is used to combine several source files into one source file that is then compiled as a whole. The use of this directive is to include function headers that describe functions used by the source code.

The header files for the first program written which implement IP routing of a simple network router that takes an IP address as input and checks a stored table of IP addresses to find the best route to send packets to the given IP address in are;

- The Address Resolution Protocol (ARP) header files:

```
#if !defined(AFX_ARP_H__894E8CE4_001A_4D91_8D28_E612579AAF18__INCLUDED_)

#define AFX_ARP_H__894E8CE4_001A_4D91_8D28_E612579AAF18__INCLUDED_

#if _MSC_VER > 1000

#pragma once

#endif // _MSC_VER > 1000

#include <snmp.h>

#pragma        comment(lib, "snmpapi.lib")
```

- The ARP Table header files:

```
#if
!defined(AFX_ARPTABLE_H__DEC8DE47_07A5_4A87_AAC1_7329EB0BE0DD__INCL
UDED_)
#define
AFX_ARPTABLE_H__DEC8DE47_07A5_4A87_AAC1_7329EB0BE0DD__INCLUDED_

#if _MSC_VER > 1000

#pragma once
#endif // _MSC_VER > 1000
```

```
#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"
```

- The IP header files:

```
#if
!defined(AFX_IPMACDLG_H__F7DD2CFA_1EF7_4BCC_9184_F02B0AEE510F__INCL
UDED_)
#define
AFX_IPMACDLG_H__F7DD2CFA_1EF7_4BCC_9184_F02B0AEE510F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// IPMACDlg.h : header file
```

- The Resource header files:

```
#define IDM_ABOUTBOX            0x0010
#define IDD_ABOUTBOX            100
#define IDS_ABOUTBOX            101
#define IDD_ARPTABLE_DIALOG     102
#define IDR_MAINFRAME           128
#define IDD_IP_MAC              129
#define IDC_ARPTABLELIST        1000
#define IDC_REFRESH             1001
#define IDC_ADD                 1002
#define IDC_ENTRIES             1003
#define IDC_ADAPTERS            1004
```

```
#define IDC_REMOVE              1005
#define IDC_EDIT                1006
#define IDC_IPADDRESS           1007
#define IDC_MAC0                1008
#define IDC_MAC1                1009
#define IDC_MAC2                1010
#define IDC_MAC3                1011
#define IDC_MAC4                1012
#define IDC_MAC5                1013
#define IDC_TYPESTATIC          1014

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE    130
#define _APS_NEXT_COMMAND_VALUE     32771
#define _APS_NEXT_CONTROL_VALUE     1015
#define _APS_NEXT_SYMED_VALUE       101
#endif
#endif
```

- The stdAfx header files:

```
#if
!defined(AFX_STDAFX_H__CAFA0D97_1960_4AB3_AE88_C2FB4DA02075__INCLUD
ED_)
#define
AFX_STDAFX_H__CAFA0D97_1960_4AB3_AE88_C2FB4DA02075__INCLUDED_

#if _MSC_VER > 1000
#pragma once
```

```
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>           // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
```

The header files for the second program written which sends packets to an IP address of the users choice through the best possible route are:

- The source header files

```
#include <windows.h>
#include <wininet.h>
#include <stdio.h>

#pragma comment(lib, "Ws2_32.lib")
#pragma comment(lib, "wininet.lib")

#define AUTHOR "bigbang"
```

- The resource header files

3.6 THE SOURCE FILES.

The source files (having the extension .cpp) represent the solving of the programming application which in this case is the IP routing of a network router and consists of all the necessary files to implement it. These files were compiled and debugged separately because the project was implemented in stages with each source file representing a different stage.

The source files are the most important files of the program and hence creating these files was the most demanding task of this project. This task was started by establishing a detailed plan of the program. The development of the program was then carried out in a number of manageable steps (the different stages being referred to in the preceding paragraph). At each step the part of the program code to that point were compiled and debugged.

The different stages and their corresponding program codes for the first written program are as follows;

- The address resolution protocol source file.

```
CARP::CARP()
{
        // Load dynamic library: inetmib1.dll
        hMIBLibrary  = LoadLibrary(TEXT("inetmib1.dll"));

        // If library loaded, get addresses of (SnmpExtensionInit, pfnSnmpExtensionQuery)
functions
        if (hMIBLibrary)
        {
                pfnSnmpExtensionInit        =        (PFNSNMPEXTENSIONINIT)
        GetProcAddress(hMIBLibrary, "SnmpExtensionInit");
                pfnSnmpExtensionQuery       =        (PFNSNMPEXTENSIONQUERY)
        GetProcAddress(hMIBLibrary, "SnmpExtensionQuery");
```

```
            // If success get addresses and initialize SNMP, bInitialized = true
            if (pfnSnmpExtensionInit && pfnSnmpExtensionQuery)
            {
                HANDLE                  hPollForTrapEvent;
                AsnObjectIdentifier     aoiSupportedView;

                bInitialized            =           pfnSnmpExtensionInit(0,
&hPollForTrapEvent, &aoiSupportedView);
            }
        }
        else
        {
            // If fail to get addresses, bInitialized = false
            bInitialized            = FALSE;
            AfxMessageBox(_T("Load library fail"));
        }
}

CARP::~CARP()
{
        // If library loaded, free it
        if (hMIBLibrary)
            FreeLibrary(hMIBLibrary);
}

//--------------------------------------------------------------------
// Function:    GetEntries: Read ARP table for specific NIC interface.
//
// Parameters:
//      pTable              Pointer to array of arpTable struct
//      TableLength         Length of the array
```

```
//      AdapterIndex  NIC Adapter index number
//
// Returns:
//                              Number of read ARP entries
//-----------------------------------------------------------------------
int CARP::GetEntries(arpTable* pTable, int TableLength, int AdapterIndex)
{
        // Be sure initialize SNMP true
        if (!bInitialized)
                return 0;

        SnmpVarBindList         SVBList[3];
        SnmpVarBind             SVBVars[3];
        UINT                    OID[3][10];
        AsnInteger32        aiErrorStatus[3], aiErrorIndex[3];
        AsnObjectIdentifier AsnOID0 = {sizeof(OID[0])/sizeof(UINT), OID[0]};
        AsnObjectIdentifier AsnOID1 = {sizeof(OID[1])/sizeof(UINT), OID[1]};
        AsnObjectIdentifier AsnOID2 = {sizeof(OID[2])/sizeof(UINT), OID[2]};
        unsigned long       pIPAddress;
        unsigned long       pMACAddress;
        int                             iEntries;


        //-----------------------------------------------------------------
        //      Fill array of 3 OIDs
        //
        //      OID[0]:     "1.3.6.1.2.1.4.22.1.1", ipNetToMediaIfIndex
        //                      The interface on which this entry's equivalence is
effective
        //
        //      OID[1]:     "1.3.6.1.2.1.4.22.1.2", ipNetToMediaPhysAddress
```

```
        //                          The media-dependent 'physical' address
        //
        //      OID[2]:         "1.3.6.1.2.1.4.22.1.4", ipNetToMediaType
        //                          Entry type: 1:Other, 2:Invalid(Remove), 3:Dynamic,
4:Static
        //
        for (int count=0; count<3; count++)
        {
                OID[count][0]       = 1;
                OID[count][1]       = 3;
                OID[count][2]       = 6;
                OID[count][3]       = 1;
                OID[count][4]       = 2;
                OID[count][5]       = 1;
                OID[count][6]       = 4;
                OID[count][7]       = 22;
                OID[count][8]       = 1;

                switch(count)
                {
                case 0:
                        // Adapter interface
                        OID[count][9] = 1;
                        break;

                case 1:
                        // MAC address
                        OID[count][9] = 2;
                        break;

                case 2:
```

```
                // Entry Type
                OID[count][9] = 4;
                break;
        }
}

ZeroMemory(pTable, sizeof(arpTable)*TableLength);

SVBList[0].len          = 1;
SVBList[0].list         = &SVBVars[0];
SnmpUtilOidCpy(&SVBVars[0].name, &AsnOID0);

SVBList[1].len          = 1;
SVBList[1].list         = &SVBVars[1];
SnmpUtilOidCpy(&SVBVars[1].name, &AsnOID1);

SVBList[2].len          = 1;
SVBList[2].list         = &SVBVars[2];
SnmpUtilOidCpy(&SVBVars[2].name, &AsnOID2);

iEntries                = 0;
do
{
        aiErrorStatus[0]        = 0;
        aiErrorIndex[0]                 = 0;
        aiErrorStatus[1]        = 0;
        aiErrorIndex[1]                 = 0;
        aiErrorStatus[2]        = 0;
        aiErrorIndex[2]                 = 0;

        // Query information of 3 OIDs
```

```
            if (pfnSnmpExtensionQuery(SNMP_PDU_GETNEXT, &SVBList[0],
&aiErrorStatus[0], &aiErrorIndex[0]))
                if (pfnSnmpExtensionQuery(SNMP_PDU_GETNEXT, &SVBList[1],
&aiErrorStatus[1], &aiErrorIndex[1]))
                    if (pfnSnmpExtensionQuery(SNMP_PDU_GETNEXT,
&SVBList[2], &aiErrorStatus[2], &aiErrorIndex[2]))
                        if (aiErrorStatus[0] ==
SNMP_ERRORSTATUS_NOERROR &&
                            aiErrorStatus[1] ==
SNMP_ERRORSTATUS_NOERROR &&
                            aiErrorStatus[2] ==
SNMP_ERRORSTATUS_NOERROR) // Check for error
                        {
                            //-----------------------------------------------------
-------------
                            // From MSDN Help:
http://msdn2.microsoft.com/en-us/library/aa378021.aspx
                            //
                            // If the extension agent cannot resolve the
variable bindings on a Get Next request,
                            // it must change the name field of the
SnmpVarBind structure to the value of the object
                            // identifier immediately following that of the
currently supported MIB subtree view.
                            // For example, if the extension agent supports
view ".1.3.6.1.4.1.77.1", a Get Next
                            // request on ".1.3.6.1.4.1.77.1.5.1" would result
in a modified name field of ".1.3.6.1.4.1.77.2".
                            // This signals the SNMP service to continue the
attempt to resolve the variable bindings
                            // with other extension agents
```

```
//-------------------------------------------------------
------------

if(SnmpUtilOidNCmp(&SVBVars[0].name,
&AsnOID0, AsnOID0.idLength))
        break;
if(SnmpUtilOidNCmp(&SVBVars[1].name,
&AsnOID1, AsnOID1.idLength))
        break;
if(SnmpUtilOidNCmp(&SVBVars[2].name,
&AsnOID2, AsnOID2.idLength))
        break;

// Verify selected Adapter interface
if (AdapterIndex == SVBList[0].list-
>value.asnValue.number)
{
        // pIPAddress get pointer ro IP Address
        pIPAddress = (unsigned
long)SVBList[1].list->name.ids;
        pTable[iEntries].IPAddress[0] =
*(unsigned char *)(pIPAddress + 44);
        pTable[iEntries].IPAddress[1] =
*(unsigned char *)(pIPAddress + 48);
        pTable[iEntries].IPAddress[2] =
*(unsigned char *)(pIPAddress + 52);
        pTable[iEntries].IPAddress[3] =
*(unsigned char *)(pIPAddress + 56);

        // pIPAddress get pointer ro MAC
Address
```

```
                                        pMACAddress             = (unsigned
long)SVBList[1].list->value.asnValue.string.stream;
                                        if (pMACAddress)
                                        {
                                            pTable[iEntries].MACAddress[0]
    = *(unsigned char *)(pMACAddress + 0);
                                            pTable[iEntries].MACAddress[1]
    = *(unsigned char *)(pMACAddress + 1);
                                            pTable[iEntries].MACAddress[2]
    = *(unsigned char *)(pMACAddress + 2);
                                            pTable[iEntries].MACAddress[3]
    = *(unsigned char *)(pMACAddress + 3);
                                            pTable[iEntries].MACAddress[4]
    = *(unsigned char *)(pMACAddress + 4);
                                            pTable[iEntries].MACAddress[5]
    = *(unsigned char *)(pMACAddress + 5);
                                        }

                                        // Entry Type
                                        pTable[iEntries].Type=      (unsigned
long)SVBList[2].list->value.asnValue.number;

                                        // Type must be one of (1, 2, 3, 4)
                                        if    (pTable[iEntries].Type>=1    &&
pTable[iEntries].Type<=4)
                                            iEntries++;          // Move to
next array position
                                    }
                                }
                                else
                                    break; // If error exit do-while
```

```
        }
        while(iEntries<TableLength);

        // Frees the memory allocated for the specified object identifiers
        SnmpUtilOidFree(&SVBVars[2].name);
        SnmpUtilOidFree(&SVBVars[1].name);
        SnmpUtilOidFree(&SVBVars[0].name);

        return iEntries;        // Return number of Entries
}

//--------------------------------------------------------------------
// Function:    EditEntry: Add/Modify/Remove ARP entry for specific NIC interface.
//
// Parameters:
//      IPAddress           Array of 4 BYTES, 4 octs of IP Address
//      MACAddress          Array of 4 BYTES, 6 octs of MAC Address
//      Type                Entry type (2:Remove, 3:Dynamic, 4:Static)
//      AdapterIndex  NIC Adapter index number
//
// Returns:
//                                  TRUE if set successfully, FALSE otherwise.
//--------------------------------------------------------------------
BOOL CARP::EditEntry(unsigned char IPAddress[4], unsigned char MACAddress[6],
unsigned long Type, int AdapterIndex)
{
        if (!bInitialized)
                return 0;

        SnmpVarBindList         SVBList;
        SnmpVarBind             SVBVars[4];
```

```
UINT                    OID[4][10];
AsnInteger32            aiErrorStatus, aiErrorIndex;
BOOL                    bReturn     = FALSE;

//--------------------------------------------------------------------
//      Fill array of 4 OIDs
//
//      OID[0]:     "1.3.6.1.2.1.4.22.1.1", ipNetToMediaIfIndex
//                      The interface on which this entry's equivalence is effective
//
//      OID[1]:     "1.3.6.1.2.1.4.22.1.2", ipNetToMediaPhysAddress
//                      The media-dependent 'physical' address
//
//      OID[2]:     "1.3.6.1.2.1.4.22.1.3", ipNetToMediaNetAddress
//                      The IpAddress corresponding to the media-dependent 'physical' address
//
//      OID[3]:     "1.3.6.1.2.1.4.22.1.4", ipNetToMediaType
//                      Entry type: 1:Other, 2:Invalid(Remove), 3:Dynamic, 4:Static
//--------------------------------------------------------------------
for (int count=0; count<4; count++)
{
        OID[count][0]       = 1;
        OID[count][1]       = 3;
        OID[count][2]       = 6;
        OID[count][3]       = 1;
        OID[count][4]       = 2;
        OID[count][5]       = 1;
        OID[count][6]       = 4;
```

```
OID[count][7]        = 22;
OID[count][8]        = 1;
OID[count][9]        = 1 + count;

switch(count)
{
case 0:
        //      OID[0]:      "1.3.6.1.2.1.4.22.1.1", ipNetToMediaIfIndex
        //                   The interface on which this entry's
equivalence is effective
        SVBVars[count].value.asnType                    =
ASN_INTEGER;
        SVBVars[count].value.asnValue.number            = AdapterIndex;
        break;

case 1:
        //      OID[1]:      "1.3.6.1.2.1.4.22.1.2",
ipNetToMediaPhysAddress
        //                   The media-dependent 'physical' address
        SVBVars[count].value.asnType                    =
ASN_OCTETSTRING;
        SVBVars[count].value.asnValue.string.stream     = MACAddress;
        SVBVars[count].value.asnValue.string.length     = 6;  //     MAC
Address length
        SVBVars[count].value.asnValue.string.dynamic= FALSE;
        break;

case 2:
        //      OID[2]:      "1.3.6.1.2.1.4.22.1.3", ipNetToMediaNetAddress
        //                   The IpAddress corresponding to the
media-dependent 'physical' address
```

```
                    SVBVars[count].value.asnType                    =
ASN_IPADDRESS;
                    SVBVars[count].value.asnValue.string.stream     = IPAddress;
                    SVBVars[count].value.asnValue.string.length     = 4;    //      IP
Address length
                    SVBVars[count].value.asnValue.string.dynamic= FALSE;
                    break;

            case 3:
                    //      OID[3]:     "1.3.6.1.2.1.4.22.1.4", ipNetToMediaType
                    //                  Entry type: 2:Remove, 3:Dynamic,
4:Static
                    SVBVars[count].value.asnType                    =
ASN_INTEGER;
                    SVBVars[count].value.asnValue.number            = Type;
                    break;
            }
            AsnObjectIdentifier AsnOID      =       {sizeof(OID[count])/sizeof(UINT),
OID[count]};
            SnmpUtilOidCpy(&SVBVars[count].name, &AsnOID);
        }

        SVBList.len         = 4;
        SVBList.list    = SVBVars;

        aiErrorStatus   = 0;
        aiErrorIndex    = 0;

        // Set information of entry (4 OIDs)
        if  (pfnSnmpExtensionQuery(SNMP_PDU_SET,    &SVBList,   &aiErrorStatus,
&aiErrorIndex))
```

```
        if (aiErrorStatus == SNMP_ERRORSTATUS_NOERROR)
            bReturn = TRUE; // If success set bReturn = true

    // Frees the memory allocated for the specified object identifiers
    SnmpUtilOidFree(&SVBVars[3].name);
    SnmpUtilOidFree(&SVBVars[2].name);
    SnmpUtilOidFree(&SVBVars[1].name);
    SnmpUtilOidFree(&SVBVars[0].name);

    return bReturn;          // TRUE if set successfully, FALSE otherwise.
}
```

- The ARP table source files.

```
#ifdef _AFXDLL
    Enable3dControls();          // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();    // Call this when linking to MFC statically
#endif

    CARPTableDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
    }
    else if (nResponse == IDCANCEL)
    {
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
```

```
        return FALSE;
}
```

- The MAC address source files.

```
CIPMACDlg::CIPMACDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CIPMACDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CIPMACDlg)
        m_Statbic               = FALSE;
        //}}AFX_DATA_INIT
        m_EnableStatbic         = TRUE;

        m_MACAddress[0]   = _T("");
        m_MACAddress[1]   = _T("");
        m_MACAddress[2]   = _T("");
        m_MACAddress[3]   = _T("");
        m_MACAddress[4]   = _T("");
        m_MACAddress[5]   = _T("");

        int count;
        for(count=0; count<4; count++)
                m_IPAddr[count]       = 0;
        for(count=0; count<6; count++)
                m_MACAddr[count]= 0;
}

void CIPMACDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CIPMACDlg)
```

```
    DDX_Control(pDX, IDC_IPADDRESS, m_IPAddress);
    DDX_Check(pDX, IDC_TYPESTATIC, m_Statbic);
    //}}AFX_DATA_MAP
    DDX_Text(pDX, IDC_MAC0, m_MACAddress[0]);
    DDV_MaxChars(pDX, m_MACAddress[0], 2);
    DDX_Text(pDX, IDC_MAC1, m_MACAddress[1]);
    DDV_MaxChars(pDX, m_MACAddress[1], 2);
    DDX_Text(pDX, IDC_MAC2, m_MACAddress[2]);
    DDV_MaxChars(pDX, m_MACAddress[2], 2);
    DDX_Text(pDX, IDC_MAC3, m_MACAddress[3]);
    DDV_MaxChars(pDX, m_MACAddress[3], 2);
    DDX_Text(pDX, IDC_MAC4, m_MACAddress[4]);
    DDV_MaxChars(pDX, m_MACAddress[4], 2);
    DDX_Text(pDX, IDC_MAC5, m_MACAddress[5]);
    DDV_MaxChars(pDX, m_MACAddress[5], 2);
}

BEGIN_MESSAGE_MAP(CIPMACDlg, CDialog)
    //{{AFX_MSG_MAP(CIPMACDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIPMACDlg message handlers

BOOL CIPMACDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here
    m_IPAddress.SetAddress(m_IPAddr[0], m_IPAddr[1], m_IPAddr[2], m_IPAddr[3]);
```

```
    for(int count=0; count<6; count++)
        m_MACAddress[count].Format(_T("%02X"), m_MACAddr[count]);

    GetDlgItem(IDC_TYPESTATIC)->EnableWindow(m_EnableStatbic);

    UpdateData(FALSE);
    return TRUE;  // return TRUE unless you set the focus to a control
        // EXCEPTION: OCX Property Pages should return FALSE
}

int CIPMACDlg::GetIPAddress(BYTE& nField0, BYTE& nField1, BYTE& nField2,
BYTE& nField3)
{
    nField0     = m_IPAddr[0];
    nField1     = m_IPAddr[1];
    nField2     = m_IPAddr[2];
    nField3     = m_IPAddr[3];

    return 0;
}

int CIPMACDlg::GetMACAddress(BYTE& nField0, BYTE& nField1, BYTE& nField2,
BYTE& nField3, BYTE& nField4, BYTE& nField5)
{
    nField0     = m_MACAddr[0];
    nField1     = m_MACAddr[1];
    nField2     = m_MACAddr[2];
    nField3     = m_MACAddr[3];
    nField4     = m_MACAddr[4];
    nField5     = m_MACAddr[5];
```

```
	return 0;
}

BOOL CIPMACDlg::IsTypeStatic()
{
	return m_Statbic;
}

void CIPMACDlg::OnOK()
{
	// TODO: Add extra validation here
	UpdateData();

	if (Validate())
		CDialog::OnOK();
}

BOOL CIPMACDlg::Validate()
{
	// Validate IP Address
	if (m_IPAddress.IsBlank())
	{
		AfxMessageBox(_T("Please, Enter IP Address"));
		return FALSE;
	}
	else
		m_IPAddress.GetAddress(m_IPAddr[0], m_IPAddr[1], m_IPAddr[2],
m_IPAddr[3]);

	// Validate MAC Address
```

```
bool    bValidMAC=true;
for(int count=0; count<6; count++)
{
        int             len;

        m_MACAddress[count].TrimLeft();
        m_MACAddress[count].TrimRight();
        m_MACAddress[count].MakeUpper();

        len     = m_MACAddress[count].GetLength();
        if (len==0 || len>2)
        {
                bValidMAC   = false;
                break;
        }

        TCHAR       ch;
        m_MACAddr[count]  = 0;
        for(int i=0; i<len; i++)
        {
                ch      = m_MACAddress[count].GetAt(i) - '0';
                if (ch>9)
                        ch -= 7;
                if (ch<0x00 || ch>0x0F)
                {
                        bValidMAC   = false;
                        break;
                }

                m_MACAddr[count]  = m_MACAddr[count]*0x10 + ch;
        }
```

```
        }
        if (bValidMAC == false)
        {
                AfxMessageBox(_T("Invalid MAC Address"));
                return FALSE;
        }

        return TRUE;
}
```

-The source file for the second program is:

```
int initWinSock()
{
        WORD wVersionRequested;
        WSADATA wsaData;
        int err;

        wVersionRequested = MAKEWORD(1, 1);
        err = WSAStartup(wVersionRequested, &wsaData);

        if (err != 0)
        {
                return -1;
        }
        return 0;
}

void GetIP()
{
        HINTERNET hInet, hUrl;
```

```
	DWORD	dwCode, dwLen;

	if	(NULL	==	(hInet	=	InternetOpen(TEXT("InetURL:/1.0"),
INTERNET_OPEN_TYPE_PRECONFIG, NULL, NULL, 0)))
	printf("Unsuccessful InternetOpen\r\n");

	else
	{
		printf("");

		if	(NULL	!=	(hUrl	=	InternetOpenUrl(hInet,
"http://iptohost.ip.funpic.de/ipcheck.php", NULL, 0, INTERNET_FLAG_RELOAD, 0)))
		{
			dwCode = 0;
			dwLen  = sizeof(dwCode);

			HttpQueryInfo(hUrl,	HTTP_QUERY_STATUS_CODE,	&dwCode,
&dwLen, 0);
			if ((dwCode == '002') || (dwCode == '203'))
			{
				if (InternetReadFile(hUrl, cIP, 256, &dwLen))
				{
					cIP[12] = '\0';
				}
			}
			InternetCloseHandle(hUrl);
		}
		else
		printf ("Apparently no valid url!\r\n");
	}
	InternetCloseHandle(hInet);
```

```
}

int main()
{
        system("color 9F");
        printf("+---------------------+\n");
        printf("+ IPtoHOST by %s +\n", AUTHOR);
        printf("+---------------------+\n\n");

        GetIP();

  struct hostent *host;
  struct in_addr addr;

  if (initWinSock())
        {
    printf("Error! Can't initialize winsock\n");
    return 1;
  }

  addr.s_addr = inet_addr(cIP);
  if (addr.s_addr == INADDR_NONE)
        {
    host = gethostbyname(cIP);
  }
  else
        {
    host = gethostbyaddr((const char *)&addr, sizeof(struct in_addr), AF_INET);
  }

  if (host == NULL)
```

```
        {
    printf("Cannot resolve address %s. Error code: %d\n", GetIP, WSAGetLastError());
    return 1;
  }

  printf("IP: %s", cIP);
      printf("\nHost Name: %s", host->h_name);

      WritePrivateProfileString("IP Info", "IP",  cIP,       "c:\\IPtoHOST.ini");
      WritePrivateProfileString("IP Info", "HOST", host->h_name, "c:\\IPtoHOST.ini");

      host_name = (char *)malloc(sizeof(char)*(strlen(host->h_name)+1));
      strcpy(host_name, host->h_name);

      strcpy(strchr(host_name, '-'), strchr(host_name, '.'));

      printf("\nShortHost Name: %s", host_name);

  WSACleanup();

      WritePrivateProfileString("IP Info", "SHORT HOST", host_name,
"c:\\IPtoHOST.ini");
      MessageBox(NULL, "IP Info has been printed in C:\\IPtoHOST.ini", "IP Check",
MB_OK);

  return 0;
}
```

3.7 THE RESOURCE FILES.

These are a set of program files; the .ico file which stores an icon for the application and the .rc file that records the resources for the application.

-The resource file (.rc) for the first program written:

```
1 TEXTINCLUDE DISCARDABLE
BEGIN
   "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
   "#include ""afxres.h""\r\n"
   "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
   "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
   "#define _AFX_NO_OLE_RESOURCES\r\n"
   "#define _AFX_NO_TRACKER_RESOURCES\r\n"
   "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
   "\r\n"
   "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
   "#ifdef _WIN32\r\n"
   "LANGUAGE 9, 1\r\n"
   "#pragma code_page(1252)\r\n"
   "#endif //_WIN32\r\n"
   "#include ""res\\ARPTable.rc2""  // non-Microsoft Visual C++ edited resources\r\n"
   "#include ""afxres.rc""         // Standard components\r\n"
   "#endif\r\n"
```

```
    "\0"
END

#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\ARPTable.ico"

////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 235, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About ARPTable"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT           "ARPTable Version 1.0",IDC_STATIC,40,10,119,8,
                    SS_NOPREFIX
    LTEXT           "Copyright (C) Eng. Usama El-Mokadem\t1992-2008",
                    IDC_STATIC,40,25,188,8
    LTEXT           "\t\t\tEmail: musama@hotmail.com",IDC_STATIC,40,40,188,8
```

```
  DEFPUSHBUTTON   "OK",IDOK,178,7,50,14,WS_GROUP
END

IDD_ARPTABLE_DIALOG DIALOGEX 0, 0, 358, 210
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION |
WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "ARPTable"
FONT 8, "MS Sans Serif"
BEGIN
  LTEXT         "Adapters:",IDC_STATIC,7,7,45,8
  COMBOBOX      IDC_ADAPTERS,55,7,296,37,CBS_DROPDOWNLIST | CBS_SORT |
                WS_VSCROLL | WS_TABSTOP
  DEFPUSHBUTTON   "&Refresh",IDC_REFRESH,301,34,50,14
  PUSHBUTTON      "&Add...",IDC_ADD,301,66,50,14
  PUSHBUTTON      "&Edit...",IDC_EDIT,301,83,50,14
  PUSHBUTTON      "Re&move...",IDC_REMOVE,301,100,50,14
  LTEXT         "ARP Table:",IDC_STATIC,7,23,45,8
  LTEXT         "",IDC_ENTRIES,55,23,238,8
  CONTROL       "List1",IDC_ARPTABLELIST,"SysListView32",LVS_REPORT |
                LVS_SINGLESEL | LVS_SHOWSELALWAYS | LVS_NOSORTHEADER |
                WS_BORDER | WS_TABSTOP,7,34,286,158
  PUSHBUTTON      "&Close",IDCANCEL,301,178,50,14
  LTEXT         "By Eng. Usama El-Mokadem: \tmusama@hotmail.com",
                IDC_STATIC,7,194,286,9
END

IDD_IP_MAC DIALOG DISCARDABLE  0, 0, 172, 79
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Enter IP and MAC Addresses"
FONT 8, "MS Sans Serif"
```

```
BEGIN
    LTEXT           "IP Address: ",IDC_STATIC,7,7,39,8
    CONTROL         "IPAddress2",IDC_IPADDRESS,"SysIPAddress32",WS_TABSTOP,
                    69,7,96,13
    LTEXT           "MAC Address: ",IDC_STATIC,7,25,48,8
    EDITTEXT        IDC_MAC0,69,25,16,14,ES_CENTER | ES_UPPERCASE
    EDITTEXT        IDC_MAC1,85,25,16,14,ES_CENTER | ES_UPPERCASE
    EDITTEXT        IDC_MAC2,101,25,16,14,ES_CENTER | ES_UPPERCASE
    EDITTEXT        IDC_MAC3,117,25,16,14,ES_CENTER | ES_UPPERCASE
    EDITTEXT        IDC_MAC4,133,25,16,14,ES_CENTER | ES_UPPERCASE
    EDITTEXT        IDC_MAC5,149,25,16,14,ES_CENTER | ES_UPPERCASE
    CONTROL         "&Static",IDC_TYPESTATIC,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,69,43,34,10
    DEFPUSHBUTTON   "OK",IDOK,69,58,44,14
    PUSHBUTTON      "Cancel",IDCANCEL,121,58,44,14
END


#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,0
 PRODUCTVERSION 1,0,0,0
 FILEFLAGSMASK 0x3fL
qq`#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
```

```
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
  BLOCK "StringFileInfo"
  BEGIN
    BLOCK "040904B0"
    BEGIN
      VALUE "CompanyName", "Eng. Usama El-Mokadem\0"
      VALUE "Comments", "By Eng. Usama El-Mokadem: musama@hotmail.com\0"
      VALUE "Email", "musama@hotmail.com\0"
      VALUE "FileDescription", "ARP Table\0"
      VALUE "FileVersion", "1.0.0.0\0"
      VALUE "InternalName", "ARPTable.exe\0"
      VALUE "LegalCopyright", "Copyright © 1992-2008, Eng. Usama El-Mokadem\0"
      VALUE "LegalTrademarks", "Eng. Usama El-Mokadem: musama@hotmail.com -
0020 10 1289308\0"
      VALUE "Mobile", "0020 (10) 1289308\0"
      VALUE "OriginalFilename", "ARPTable.exe\0"
      VALUE "ProductName", "ARPTable\0"
      VALUE "ProductVersion", "1.0.0.0\0"
      VALUE "Web", "http://musama.tripod.com\0"
    END
  END
  BLOCK "VarFileInfo"
  BEGIN
    VALUE "Translation", 0x409, 1200
  END
END
```

```
#endif    // !_MAC


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 228
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END

    IDD_ARPTABLE_DIALOG, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 351
        TOPMARGIN, 7
        BOTTOMMARGIN, 203
    END

    IDD_IP_MAC, DIALOG
    BEGIN
        LEFTMARGIN, 7
```

```
    RIGHTMARGIN, 165
    TOPMARGIN, 7
    BOTTOMMARGIN, 72
  END
END
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
  IDS_ABOUTBOX          "&About ARPTable..."
END

#endif    // English (U.S.) resources
////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
```

```
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif //_WIN32
#include "res\ARPTable.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#endif

/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

-The second program has only an icon resource file.

CHAPTER FOUR.

ANALYSES AND COMPARISM OF RESULTS

Observing the two programs written, at first glance one might think that developing such programs is a very complex process but once you develop a program using the visual C++ package you will be able to see how easy developing such programs that serves as software that run on hardware devices can be. The cumbersome nature of the programs and why they may appear complex is due to the library support the Visual C++ package provides.

Comparing the two programs written in this project to accomplish network routing, you can observe that there is a big and clear difference between them.

Considering the header files of the two programs, the first program written has five header files corresponding to the five source files that contain the codes written to implement the IP routing of packets by a simple network router while the second program has a single header file corresponding to the source file that contains the codes of that particular program.

As said in the above paragraph, the first program has five source files and this is due to the fact that if a program is a bit complex, then writing it in a number of stages with each source file representing a different stage is the best way to go about such a problem. The second program has a single source file that provide the codes for the program.

CHAPTER FIVE

RECOMMENDATIONS AND CONCLUSIONS

5.1 RECOMMENDATIONS

After careful observations, due consultations, and technical assessment of this project work the following recommendations were made.

1. The project can be written with a more modern and user friendly language which will help in understanding the program better.
2. The project can be made to include dynamic form of routing not only the static form. This will make routing to large number of routes easier.
3. Modern routing protocols such as Routing Information Protocol (RIP) that increases routing efficiency can be implemented.
4. Recently developed software such as Visual Studio 2008 can be used for the simulation process which will be more informative and self explanatory.
5. The routing algorithm can be made to use multipath roting techniques in order to give multiple alternative paths for routing packets.
6. In addition to routing, the program can be made to accomplish bridging so as to cater for localised environments where bridging is widely used.
7. This project can be made a Masters' degree project and be expanded in such a way that processes like simulation can be added to solve various setbacks being experienced when routing packets of data.

5.2 CONCLUSION.

By carefully observing the routing function of a network router demonstrated by this project, one can see how the routing process makes sending and receiving packets of data easy and fast when a router is used.

By comparing the two programs written it can be concluded that developing software that runs on a computer to perform a given task can be much easier than developing the software for a device that performs a single function as seen in this project where the program that runs a computer as a router is less complex that the program that runs a network router to perform the function of routing.

Also writing the program in stages as was done with the first program allows for easy reconfiguration of the program to may be solve a given problem or improve efficiency.

By carefully analysing this project you can see the beauty of IP routing is that no matter how many more routes we might decide to put into this example, the process would never change! The packet is just sent from hop to hop until it reaches the destination address.

REFERENCES

1. Vic, Broquard. (2003). *C++ for computer science and engineering.* Broquard eBooks, East Peoria, U.S.A, pp. 35-624.

2. Victor, Shtern. (2000). *Core C++ A software engineering approach.* Prentice Hall PTR, U.S.A, pp. 110-169.

3. Katupitiya, J. And Kim, Bently. (2006). *Interfacing with C++.* Springer Berlin, Netherlands, pp. 75-99.

4. Jeffery, Beasly. (2009). *Networking.* Edwards Brothers, Michigan, U.S.A. pp. 224-244.

5. Ivor, Horton. (2008). *Beginning visual C++.* Wiley publishing, Indiana, Canada, pp. 97-112.

6. "*C++ programming fundamentals* [CD]. Chuck, Eastorn, Charles river media 2003.

7. Bjarne, Stroustrup. (2000). *The C++ programming language.* Addison-Wesley, Boston, U.S.A, pp. 691-723.

8. *www.wikepedia.com*

9. *www.cl.cam.ac.uk/ip-routing*

10. *www.techrepublic.com*

11. *www.freelancer.com/routing-algorithm*

12. *www.3rad.com/networks*

13. *www.computersimulationencyclopedia.com*

14. *www.musama.tripod.com*

15. *Mannir.com/farouk.zip*