University of Southern Queensland
Faculty of Engineering & Surveying

Design and Implementation of a Network Adrress Translator

A dissertation submitted by

K-J. Beasley

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Computer Systems Engineering) / Bachelor of Information Technology (Applied Computer Science)

Submitted: October, 2004

Abstract

A continuously increasing demand for Internet Protocol (IP) Addresses was something that was not considered at the time when the Internet was first designed. The argument was actually quite the opposite and most experts pooh-poohed the idea of the internet ever growing to beyond 100,000 networks. However, the 100,000th network was connected to the internet in 1996 (Tanenbaum 2003). The Explosive growth of the Internet has resulted in a shortage of the number of available IP Addresses. As this growth continues the shortage will increase and a new form of Internet Addressing will need to be established. The current form, IPv6 has been under development for some time now and has not gathered wide industry support. Obviously a temporary solution must be established to overcome the shortage of IP Addresses in the immediate future until permanent solutions can be achieved.

On the other hand the TCP Protocol was established as an end-to-end connection for reliable communication and makes use of its own 16-bit port number. This allows for up to 65,535 unique port numbers for TCP communication. Most hosts never maintain 65,535 end-to-end connections and this allows for a technology called Network Address Translation (NAT) to save on the number of IP Addresses required on the Internet by multiplexing many IP Sources onto one or more IP Addresses using unique TCP port numbers for each data stream.

The ultimate aim of this project is to produce a small prototype Network Address Translator and discuss further improvements necessary for its use in a production environment.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

K-J. Beasley

00110217980

Signature

Date

Acknowledgments

I would like to thank all those who have supported my endeavors over the past five years. Special thanks goes to all the staff from the Faculty of Engineering and Surveying and the Department of Maths and Computing at the Faculty of Science. Students are incredibly grateful for the time you take to pass on your knowledge and experience.

I would like to offer special thanks to my supervisor, Dr John Leis. After debugging a single line of code for what seemed like a month, losing all motivation and wondering why I ever suggested this project topic, it was often his support and desire to see the project succeed that positioned me back on track.

Sincere appreciation to my family. I have no idea where I would be right now if it were not for my Mum constantly looking over my shoulder checking if my project was up to date and my assignments completed. It was Dad's seemingly infinite technical knowledge that convinced me to study Engineering. Sadly this knowledge is much diminished due to a recent debilitating stroke. Lastly to my darling sister, she may be public enemy number one for telling Mum that I worked on project/assignments when indeed I played computer games, however her love and support during this time is greatly appreciated. The completion of this project and my successful graduation is testimony to your contributions and continued support.

Thankyou All.

K-J. Beasley

University of Southern Queensland

October 2004

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Explosive growth of computer networks, in particular the Internet has seen the Internet become an integral part of everyday life. Many of the tasks traditionally left for the Mail network are now being done via e-mail. Phone conversations are increasingly being replaced by Internet Chat and slowly voice chat and webcams are entering the market while available bandwidth is making this technology viable.

There is a significant trend towards internet connectivity for devices which previously would never have been considered relevant to the Internet. For example Internet Refrigerators and Air Conditioners which can be activated from a remote device are becoming popular. As development continues more and more devices will use an Internet connection as part of their operation.

The Internet currently operates based on two important protocols collectively referred to as TCP/IP. These two protocols are actually the Transmission Control Protocol and Internet Protocol. Unfortunately the designers of IP overlooked the commercial viability of the Internet and suspected it would never become anything more than a research network connecting universities and a few other large companies such as Military Research and Development.

This line of thought lead to the development of the Internet Protocol utilizing an Addressing system consisting of 32 bit addresses. That is addresses consist of 32 binary numbers each of which can only be on (1) or off (0). In an ideal world this offers $2^{32} = 4,294,967,296$ possible IP Addresses. Given that there are in excess of 6 billion people in the world this number is never going to survive in the long term when a very large proportion of the population will hopefully be online. In addition many people use more than one IP Address, for example if they have more than one computer connected directly to the internet or have a computer at work and at home which may both be connected at the same time. Finally we do not live in the ideal world and IP addresses are wasted both through requirements for divisions and wastage due to over-allocation or private requirements.

The obvious solution to this would be to allow the IP Address to be larger, maybe double or quadruple its current size. Such a protocol is being developed and has already been implemented in some areas. Unfortunately there has not been a great deal of industry support for the new protocol and some believe it may lose all levels of support and be forgotten before it is implemented. However if this happens we will still be left to face the problem of how to spread some four billion IP Addresses across the globe in a fair and equal manner. In addition some companies and universities who have purchased large ranges of the IP address space will not likely give up their range unless presented with sufficient financial incentive to do so. As the remainder of a particular resource decreases its value normally increases and this certainly could be the case with IP Addresses.

Unfortunately the lack of IP Addresses is a real difficulty affecting people across the Internet at this time. The problem is not going to wait for a solution to be developed, implemented and tested. Therefore alternatives must be developed quickly, must require little testing and must be reliable. One such solution is called Network Address Translation (NAT) and is already implemented in many forms. In Microsoft Windows 98 and above the solution is commonly known as Internet Connection Sharing (ICS). However ICS is a poor solution to the problem which works in some cases but excludes any sort of special protocols such as Video Conferencing (NetMeeting) and active File Transfer Protocol (FTP). Due to the ICS code forming a integral part of the Operating System it cannot be reverse engineered or modified and has very few security features which has allowed other products to enter the market.

The aim of this project is to develop the basics of a new type of Network Address Translator. Ideally the NAT will undergo further development after the initial project is complete and will contain security features and policies rigid enough to satisfy even the most security conscious Network Administrator while having the flexibility to be used by even the most application intensive home Internet user regardless of which applications they may wish to use.

1.1 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 reviews the beginning of the Internet and some of the network models used in designing the communications protocols used today.

Chapter 3 discusses the development of a Network Address Translator (NAT), the programming methodology followed and the Programming Languages available for Implementation.

Chapter 4 examines the Internet Protocol (IP) used in network communications. The Internet Protocol forms part of the network hierarchy and is the first layer involved in Network Address Translation.

Chapter 5 details the Transmission Control Protocol (TCP) which completes the TCP/IP Protocol Suite used on the Internet. The use of Transmission Control Protocol features in Network Address Translation are also discussed.

Chapter 6 critically examines existing Network Address Translators including popular features for Home and Business users and cost to purchase. New Network Address Translator features which may be well received by Home and Business users are also discussed.

Chapter 7 introduces the C#.NET programming language and development environment. The most important part of this project, communication sockets, are introduced including advanced features required by this project.

Chapter 8 concludes the dissertation and suggests further work in the area of 'z'.

Chapter 2

Network Reference Models

2.1 Chapter Overview

Looking at the history of Network Design, many networks were mainly hardware oriented with the software as an afterthought. This strategy is no longer suitable for today's high speed networking interfaces. This chapter will examine the software structuring in some detail.

2.2 Networking History

At the height of the Cold War in the late 1950's one line of thought was the vulnerability of the telephone network to Nuclear War. (Baran 1964) Referring to Figure 2.1(b) reveals that the destruction of a few key points could fragment the telephone network into small isolated islands.

Around 1960, the Department of Defense (DoD) awarded a contract to RAND Corporation for the development of a solution to this vulnerability. Paul Baran, an employee of RAND Corporation, developed a proposed solution depicted in Figure 2.1(c). Unfortunately When the DoD took the idea to the U.S. national telephone provider `AT&T` the idea was dismissed as a concept which could not be constructed. It is believed that `AT&T`

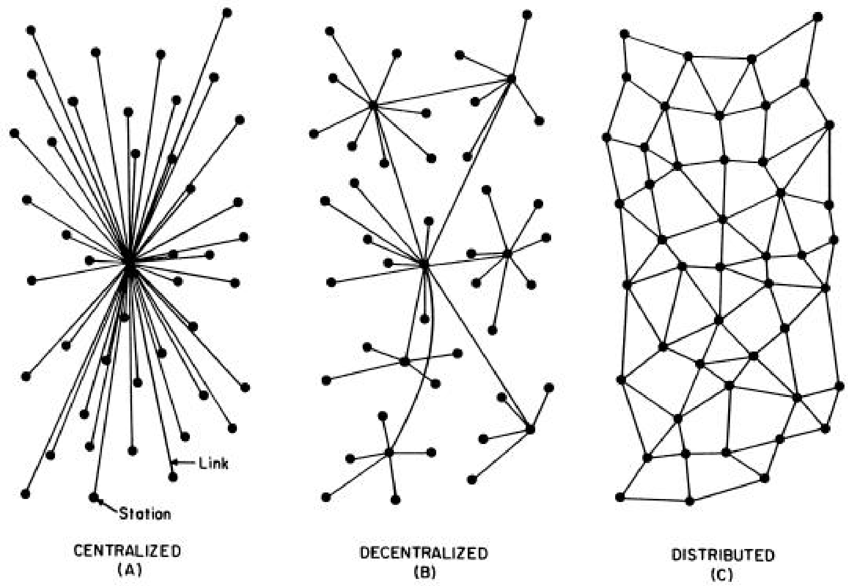

Figure 2.1: (a) Structure of a Switching office. (b) Structure of the telephone system. (c) Baran's proposal for a distributed switching system. (adapted from (Baran 1964)).

did not actually want to admit that Paul Baran had succeeded in developing a network concept where AT&T had failed, effectively dooming the idea. (Tanenbaum 2003)

In 1967 the Advanced Research Projects Agency director, Larry Roberts, turned the sights back onto networking. He worked with Wesley Clark who again suggested a packet-switched subnet communicating via routers. Roberts presented a vague paper on the packet-switching idea at the Symposium on Operating System Principles (SIGOPS) in Gatlinburg. (Roberts 1967) A similar paper at the conference described a system that had not only been designed, but actually implemented at the National Physics Laboratory in England. In 1968 BBN, a consulting firm in Cambridge, Massachusetts was awarded a contract to build what became known as the ARPANET (Tanenbaum 2003)

In the early 1980's ARPANET protocols were eventually replaced by the Transmission Control Protocol/Internet Protocol (TCP/IP) which will be discussed separately in 4 and 5. Several contracts were also awarded to BBN and the University of California at Berkeley forming the Berkeley UNIX company. Berkeley students wrote a program interface for networking called Berkeley Sockets (or simply sockets) and developed

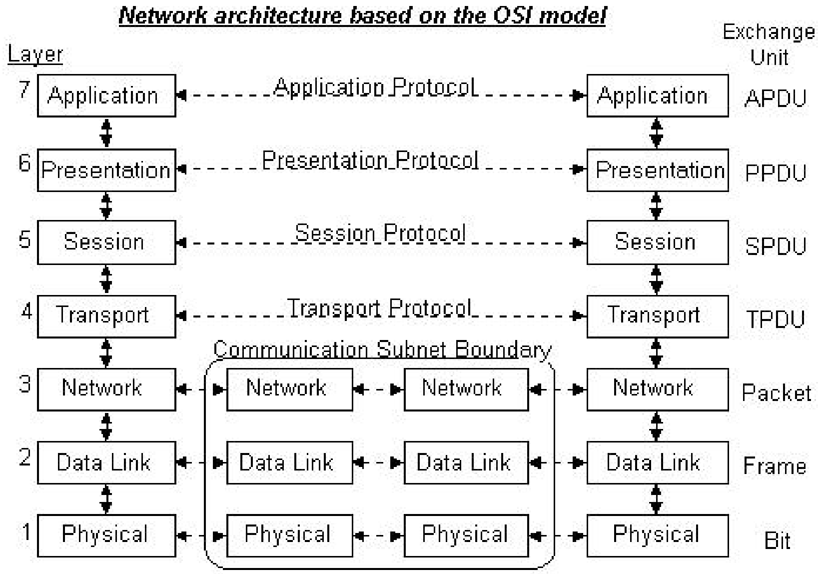

Figure 2.2: The Open Systems Interconnection Reference Model. (adapted from (Day & Zimmermann 1983)).

many applications, utilities and management programs to ease the burden of Network Administration. (Tanenbaum 2003)

2.3 The Open Systems Interconnection Reference Model

The Open Systems Interconnection (OSI) model is shown in Figure 2.2. It was based on a International Standards Organization (ISO) proposal aimed at international standardization of networking protocols.

The OSI Model was defined using five basic principles as follows:

1. A layer should add a level of abstraction to the communications architecture.
2. Each layer should provide well defined functionality.
3. The functionality of each layer should work towards an international standard.
4. Minimal information should flow across the boundaries between layers, particularly control information.

5. There should be sufficient layers that the distinct functionality of each layer is not compromised however few enough layers that the architecture is not unwieldy.

2.3.1 The Physical Layer

This layer is concerned with transmitting raw bits over a communication channel. It is concerned with issues such as timing, the standards of the physical interface, what constitutes the correct receival of a on or off bit and generally anything to do with the electrical or mechanical characteristics of the medium used for transmission.

In recent time the physical layer has undergone significant changes due to new technology in use for physically providing the data connection. Wireless Networking is an example of one entirely new physical mediums now used in computer networks. (Day & Zimmermann 1983)

2.3.2 The Data Link Layer

The data link layer is concerned with synchronization, reliability and framing which refers to organizing each chunk of data into a packet. Frames are ordered to prevent data from arriving in the wrong order and may be acknowledged in the case of a reliable service. However this service is not concerned with ensuring reliable data streams and frames may still be corrupted or not delivered.

In the data link layer on a broadcast based network an additional issue is addressed; how to control access to the shared channel. A special sub-layer which does not form part of the OSI Model deals with this issue and is called the medium access control sub-layer. (Tanenbaum 2003)

2.3.3 The Network Layer

The main task of the network layer is to control operation of a subnet. The key design issue is routing packets from source to destination. Routing can be based on various methods, ranging from static tables (which form a core part of the router and only change if a major reconfiguration is detected) through dynamic session based (changing when a connection is established or closed) to highly dynamic (determining different routing information for each packet based on current network load).

The Network Layer is also responsible for congestion control on the local subnet, to provide any Quality of Service controls (delay, transmit time, jitter, etc) required on the subnet and to allow the interconnection of heterogeneous networks including packet fragmentation.

Broadcast based networks often use a very small Network Layer or may not contain this layer at all. (Tanenbaum 2003)

2.3.4 The Transport Layer

The transport layer accepts data from the upper layer services, splits the data into smaller units, if necessary, and passes the chunks to the network layer. It is responsible for error detection and overall sequencing to ensure the ordering of messages is not changed in the transmission. The most important aspect of the Transport Layer is to shield the upper layer services from inevitable changes to the networking hardware.

The transport layer offers multiple types of services to the session layer. The most common service is an error-free point-to-point protocol. However other options such as best-effort transmission or real time (approximately) transmission may also be available. Obviously if the underlying layers offer broadcast and multicast services these will likely also be offered to the session layer.

The transport layer is the first end-to-end layer. Lower layers operate between neighboring machines or routers that form part of the network. The transport layer only operates on the source and destination machines which carry on the conversation. (Day & Zimmermann 1983)

2.3.5 The Session Layer

The basic function of the network layer is to allow users on different machines to establish sessions between them. The sessions offer special functionality such as dialog control (taking turns to transmit and receive), token management (preventing two machines from accessing the same area of memory or from updating the same information at the same time) and download resume features (Internet Explorer resuming half-way through a download even after a disconnection or crash). (Tanenbaum 2003)

2.3.6 The Presentation Layer

The presentation layer defines the syntax and semantics of information exchange. For example transmitting plain text in American Standard Code for Information Interchange (ASCII) or Unicode format. This includes dealing with differing methods of storing information and agreeing on a standard during transmission. A simple example might be the different methods of storing a date between America (12/31/2004) or Australia (31/12/2004). (Tanenbaum 2003)

2.3.7 The Application Layer

The application layer contains the range of higher-level protocols used by users of the Internet. These protocols include features such as file transfer, electronic mail, news servers and chat services. One of the most common protocols used for delivery of almost all internet web pages is HyperText Transfer Protocol (HTTP) which defines how to request and receive pages written in HyperText Markup Language (HTML). (Day & Zimmermann 1983)

2.4 The TCP/IP Reference Model

The TCP/IP Reference Model was designed in response to the need for the seamless interconnection of multiple networks. A common misconception is that the TCP/IP

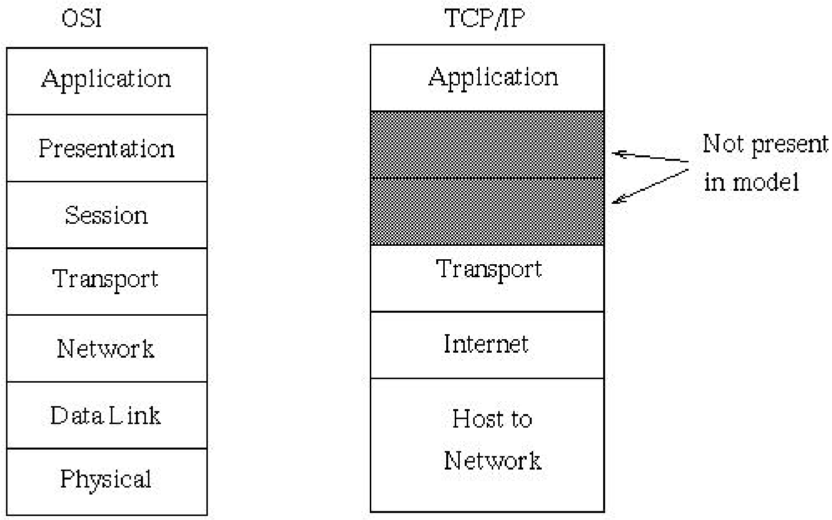

Figure 2.3: The TCP/IP reference model.

Reference Model was designed to smooth over some issues found in the OSI Reference Model. This is not true as the TCP/IP Reference Model was defined several years before the introduction of the OSI Reference Model (Cerf & Kahn 1974). The TCP/IP Reference Model, as shown in Figure 2.3, was designed mostly to satisfy Department of Defense requirements that end-to-end connections remained intact as long as the source and destination machines were functioning. This relied on the assumption that a functioning route between the source and destination existed, however, the main idea was that the actual path this connection followed could change in response to individual transmission links being decommissioned. As a result the TCP/IP Reference Model did not define anything below the Internet layer in detail. (Tanenbaum 2003)

2.4.1 The Host-to-Network Layer

The host-to-network layer is the great void left below the Internet layer. Most references on the TCP/IP reference model do not discuss this layer, however, it is included here for completeness. The main concept here is that each host must connect to the network and this will involve some protocol to encapsulate the Internet Layer, however the details of this protocol are not covered.

2.4.2 The Internet Layer

The internet layer is a connectionless internetwork that forms the basis of the TCP/IP Reference Model. Its function is to allow packets from any network to travel independently to a destination which may be separated from the packet source by many different networks. The ordering of packets may be altered during transit and each packet may follow a completely different path to the destination. This concept of following a pathway to the correct destination is known as routing.

The internet layer defines the protocol known as the Internet Protocol (IP) and associated format for an IP Header. Obviously Routing and congestion avoidance are the major issues at this level which leads to the association between the TCP/IP internet layer and the OSI network layer. (Cerf & Kahn 1974)

2.4.3 The Transport Layer

The transport layer is designed to allow a source and destination entities to undertake a conversation. The functionality here is virtually the same as in the OSI Model, the difference being that the TCP/IP Reference Model actually defines two end-to-end transport protocols. These protocols are called Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

TCP is a reliable, connection oriented protocol which means it requires an end-to-end connection to be established and maintained. This is achieved by requiring an acknowledgement for each data segment or packet. The source is limited in how many packets can be transmitted before waiting for acknowledgements to be received. The main concept is to allow a byte-stream to be delivered from source to destination without error or corruption. TCP splits the byte-stream into fragments or discrete messages, adds error checking, sequencing and flow control information and passes the packet to the internet layer.

UDP is a unreliable, connectionless protocol meaning it does not require an end-to-end connection and could continue sending a flood of packets to a destination even though the path may have become unavailable. Packets are not acknowledged and may arrive corrupted or may not arrive at all. Packets may also not arrive in the same order as they are sent. The purpose of providing this service is to allow application developers to implement their own sequencing and checksum's or if the application warrants, to exclude such features completely. Some applications which generally do not need sequencing or reliability through checksums are real-time voice and video transmission. (Tanenbaum 2003)

2.4.4 The Application Layer

Another void is found between the transport and application layers. The OSI Models session and presentation layers where not perceived as necessary by the developers of the TCP/IP model and were not included. Fortunately experience with the OSI Model has shown that the session and presentation layers are practically of no use to many applications.

The application layer contains all the higher-level protocols. The early internet protocols included virtual terminal (TELNET), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) and HyperText Transfer Protocol (HTTP). A number of additional protocols have been developed for real-time chat and voice over IP. Continuing protocol development is expected as system integrators find new and interesting ways of using the internet to make life easier. (Cerf & Kahn 1974)

2.5 Chapter Summary

The internet came from fairly humble beginnings as a small connection of four computers to form the first version of the ARPANET but not before major troubles were overcome. Developing a packet-switched subnet was something that had not been attempted and most telecommunications providers did not like the idea of some young hot shot researcher telling them how to construct their networks. However once the ARPANET began to grow, adopted more scalable protocols and became used by a large range of people the concept of a internet became possible.

A large effort began to standardize how the Internet would work. Two models were developed to address this issue. The OSI model is more generic and can be applied to almost any form of network. Years of experience have shown that the number of layers in the OSI model are slightly excessive and some layers are generally not used in Internet applications. The TCP/IP model used less layers but is more specific to the internet and can be fairly abstract when referring to lower level network features and interface. As the concepts behind software development focus more on semantics and syntaxes of the language or protocol the OSI model may gain popularity again, however, at the moment the TCP/IP model is generally most relevant to internet development.

The constant development of new protocols is a constant challenge to developers focusing on TCP/IP as new protocols may not always let TCP and IP handel the issues of routing, addressing and reliability themselves. An Early example of this was FTP where the IP address and TCP Port was embedded in the FTP data stream. As application developers develop new ways of using the Internet it is important to observe the layering of the Internet and try to avoid breaching these layers when designing new software and standards. This can particularly be a problem for applications such as Network Address Translators where failure to correct a reference to the IP address or TCP port in packet header or data can corrupt the entire process of translation.

Chapter 3

Design Specification

3.1 Chapter Overview

Many software development projects have been known to incur extensive and costly design errors. The most expansive errors are often introduced early in the development process. This underscores the need for better requirement definition and software design methodology. Software design is an important activity as it determines how the whole software development task would proceed including the system maintenance. The design of software is essentially a skill, however, it usually requires a structure which will provide a guide or a methodology for this task.

3.2 Design Methodology

3.2.1 Extreme Programming

Extreme Programming is a deliberate and disciplined approach to software Development. It has been developed over a period of about 8 years and has proven successful in companies of various sizes.

Extreme Programming is oriented towards customer satisfaction. It aims to deliver the required software on time even when changing requirements complicate the process. Team work is central to the methodology in order to achieve this goal. (Wells 2003)

Despite these advantages, Extreme Programming is not a good choice in this project because the customer who requires the product is also the programmer, therefore changing requirements are not likely to occur. There is also no development team involved in this project so the use of GroupWise development would be a waste of effort.

3.2.2 Scrum

Scrum is an agile, lightweight process used in Product Development, particularly control and management of software projects. Scrum focuses on traditional iterative, incremental programming methods while wrapping existing engineering methodologies such as Extreme Programming and Rational Unified Process to allow agile development and simple implementation.

Scrum significantly decreases development time and has faster benefit implementation while allowing adaptive, empirical system development. (Advanced Development Methods Inc 2004)

Unfortunately Scrum is a highly commercialised development process which requires employment of a certified ScrumMaster or participation in a two day course to become a ScrumMaster. Most of these courses are only available in the United States and are financially expensive. Scrum will not be implemented as part of this project.

3.2.3 Feature Driven Development

Feature Driven Development (FDD) is a process of software development aimed at delivering requested or required features in the shortest possible time period. After the overall project is identified and a feature list is created, each identified feature is fully designed and then implemented into the system.

FDD allows a software development team to remain highly focused and greatly increases production and improves team spirit by delivering entire fully featured prototypes throughout the development cycle.

Any software development suffers from exponential development times. As the project nears completion, the amount of work completed decreases for the same amount of time spent on development. FDD suffers greatly from this problem because each feature suffers from exponential development times.

FDD is particularly common in projects which are in trouble and have deadlines and milestones to be met. By focusing only on critical required features the project can often be saved. FDD will only be used in this project if development falls behind schedule.

3.2.4 STEPWISE

STEPWISE is a software development process designed to overcome limitations of the ISO 10303-11 EXPRESS model by automating software development. EXPRESS is used to represent product and process data in standard data stores, to increase data value and decrease data management costs.

STEPWISE features an enhanced architecture to support automation of EXPRESS for implementing high-level procedural interfaces, storage representations and interchange formats. (Kahn 2000)

STEPWISE is designed for high-level applications and is probably more suited to 4th and 5th level languages such as Structured Query Language. Despite its improvements over C++, C#.NET is still a 3rd level language and is not particularly suited for STEPWISE.

3.2.5 Rational Unified Process

The Rational Unified Process (RUP) was developed by the same people who originally created Unified Modeling Language (UML). UML is a single complete notation for describing object models and is extensively used in Software Engineering. RUP is a software development process providing a framework that can be used to describe specific development processes.

The essence of RUP is iteration and RUP was developed with the goal that each iteration ends in a deliverable (prototype, fully featured class, etc). RUP involves extensive Risk Management, particularly of the risk that development will fall behind schedule. RUP acknowledges that project plans do not define what will be produced, but a statement of how to manage risk. A plan of action will inevitably fail while a plan of contingencies will eventually succeed. (Sharon 1999)

However as a major part of the Engineering Program, this project cannot be allowed to fail. Some aspects of the program development may be optional, provided that the overall deliverables are provided. For this reason the contingencies are somewhat limited and not particularly suited to RUP.

3.2.6 Waterfall Models

The waterfall model was originally developed as a series of discontinuous phases involving Conception, Requirements, Architectural Design, Detailed Design, Coding and Development and Testing and Implementation.

Several variations on this system interpose their own advantages and weaknesses into the model. These variations include the Spiral Model, Modified Waterfall Model, Evolutionary Prototyping, Code-and-Fix, Staged Delivery and Evolutionary Delivery.

- The Spiral Model breaks a software project up into mini-projects, each addressing a major risk. This ensures that total project risk is inversely proportional to cost at each step in the development process.
- The Modified Waterfall Model is potentially the same as the Waterfall Model, however it is not done in Discontinuous steps. This enables the phases to overlap where needed allowing requirements to be gathered while overall project progress is still proceeding.

- Evolutionary Prototyping involves multiple iterations of requirements gathering. Iterations produce individual prototypes to be presented to the customer to stimulate further feedback and discussion of requirements.

- Code-and-Fix is the typical approach to avoiding the complexities of a development methodology. It is only useful for small, throw away projects and is dangerous because it offers no Quality Assurance or Risk Management.

- Staged Delivery involves breaking design, coding, testing and deployment into separate stages which are useful to the customer. Each stage must function independently of other stages.

- Evolutionary Development straddles evolutionary prototyping and staged delivery. Initial development is on lower-level functions which will hopefully remain independent of changing customer requirements. (Business ESolutions 2002)

The Modified Waterfall Model is prefered in the project and will be employed as long as the project remains on schedule. Failsafe will be provided by Feature Driven Development if the project schedule is not fulfilled.

3.3 Programming Language

Business today demands sophisticated computing capabilities. Even the software products used for office automation (word processors, spreadsheets, etc.) have become large and complex in the process of meeting user needs. The issues involved in creating large, complex software are many and varied. However one issue continues to cause controversy and seldom results in agreement between programmers. This issue is the choice of Programming Language. Some alternative programming languages are presented and discussed in this section.

3.3.1 Visual Basic.NET

`VB.NET` was developed as part of Microsoft's Visual Studio solution and represents the next generation of language and tools for rapidly building Microsoft Windows and Web applications. `VB.NET` has a very clean interface for designing a Graphical User Interface (GUI) making it extremely popular when developing such applications.

Unfortunately `VB.NET` is not a common tool in most Computer Science and Engineering applications. Some features are useful for special Computer Science applications, however it is more generally regarded as a business programming language. For this reason `VB.NET` was disregarded despite supporting the necessary socket operations required for this development project.

3.3.2 ASP.NET

ASP.NET is derived from the Active Server Pages language used to create dynamic web pages. This is not exactly suitable for a Network Address Translator which normally works at a much lower layer than ASP.NET. However ASP.NET would serve a useful function for this project as a web interface for management of the Network Address Translator (NAT). A particularly useful aspect of ASP.NET would be that a web interface could be used to allow an Application Layer Gateway (ALG) to report an anticipated incoming port. This may not be useful in all cases because the requested port may already be in use. However it may be used to attempt a repair of some applications which do not normally work under NAT such as Active FTP.

ASP.NET will not form part of this project however because it is unlikely to reach the application layer. The most important issue is to achieve a simple working NAT for use in future development.

3.3.3 C

C evolved from a language called B, written by Ken Thompson. C is a simple and small language, which can be translated with simple, small compilers. Today it is among the languages most commonly used throughout the computer industry.

There is no particular reason why this project could not have been developed in C. It has a clean interface, is easy to use and supports all the necessary sockets operations. The only downside is the difficulty in generating a Graphical User Interface (GUI) using appropriate libraries. Writing a windows service in C can be difficult as a number of hooks need to be developed. These hooks are for use by the operating system in starting and stopping the service.

Although C was not used in this project it was excluded only because of time considerations in GUI development and some difficulties in writing a Windows Service. The final product could be ported back to C as a future project.

3.3.4 C++/Visual C++.NET

C++ is a rewritten and improved version of C. The major focus in developing C++ was to enable the development of object oriented programs. C++ also showcases a variety of other features not found in C while still maintaining the basic syntax and semantics found in C. Visual C++.NET is a particular implementation of C++ by Microsoft. Visual C++.NET includes integrated GUI development tools which make developing a GUI type interface much easier than using standard libraries in C or other C++ environments.

Again there is no specific reason why C++ could not be used to develop this project. The added benefit of integrated GUI development tools in Visual C++.NET only adds to the reasons for using C++. However Visual C++ still requires complex methods to support running as a Windows Service. In addition the Microsoft Foundation Class (MFC) is extremely difficult to master and takes a lot of time to set up properly.

Future work could involve porting the application back to Visual C++ or developing a user interface using standard C++ libraries.

3.3.5 Java

Java is an object oriented programming language which uses a Java Virtual Machine (JVM) to run Java programs. The JVM is cross-platform capable having been ported to many variants of Windows and Unix. The Java Application Programming Interfaces are a set of pre-built classes that can be used in program development.

In C++, memory has to be explicitly requested when required. Likewise, when finished with the memory, it has to be explicitly returned to the operating system. Although this process sounds simple, it is easy to create a memory leak, which is when your application requests memory and forgets to release the allocated memory. Over time, the application grows in size, slows down the system greatly and eventually crashes. Java implements a feature called Garbage Collection which automatically recovers memory that can no longer be referenced by the program. The result is that memory leaks cannot occur, to an inexperienced programmer it looks like every variable is causing a memory leak, however, behind the scenes the garbage collector is searching for any piece of memory which is no longer required and reclaiming it to be reassigned to another variable or even another program. (Campione, Walrath & Huml 2000)

Despite the feature of garbage collection in Java it was not used in this project. Java only supports two types of sockets, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). A NAT requires a lower level socket known as a RAW socket which receives an entire packet with the TCP and Internet Protocol (IP) headers intact. Despite some later references to using a lower level protocol to inject packets at the data link layer the decision to reject Java as a programming language had already been made.

3.3.6 C#.NET

C#.NET, a Java-like programming language, was developed by Microsoft and submitted to the ECMA standards group for approval. Although the language is Java-like its syntax and semantics are remarkably similar to C/C++. The likeness to Java comes from many features which are implemented in Java such as garbage collection and the implementation of a hashing-table in standard functions for almost every variable type.

C#.NET had several other advantages over most other languages. It supported creating a Windows Service using its own implementation. The only feature left for development was code to start and stop the service. All the necessary hooks were linked to the developers functions. In addition C#.NET supports a very wide range of sockets including RAW sockets which can send and receive packets including the full IP header. The GUI development platform used in `VB.NET` is reproduced almost perfectly in C#.NET enabling the simple development of a simple management interface for the project in the same language in which the service was developed.

C#.NET was my final language of choice for this project. Although there have been critics of C#.NET my decision was to use it in a kernel level project. The language is quickly becoming popular and has successfully been used in a large number of commercial projects.

3.4 Chapter Summary

This project followed the Evolutionary Development Model. By developing prototypes commencing at low-level functionality and increasing this functionality until the overall requirements are met, project progress can be measured through the attainment of milestones and project research can be broken into individual concepts.

C#.NET is not the most common language for low level networking applications. However it was the programming language with the most features that were of practical use in this project. Any other language would have required significantly longer periods of time for development however C#.NET was similar enough in syntax to C/C++ that the learning curve of the language was very short.

Chapter 4

Internet Protocol

4.1 Chapter Overview

No Network Address Translator (NAT) could be developed without a full understanding of the Internet Protocol (IP). Already IP has been mentioned in several sections of this book. It has been largely undefined, except to say it is a protocol developed and used in the Internet. However, the use of IP is not limited to the Internet, many smaller networks also use IP as an underlying communication protocol. This is testimony to the robustness of IP, it can be used to deliver packets to the next cubical in the office, across the street, across the country or around the world. This chapter examines the reasons for developing the Internet Protocol, why it is so useful in computer communications and the reason such a protocol would need added functionality in the form of a NAT. Finally, the report shall breifly discuss what is currently being developed to remove the need for a NAT and return IP to a completely independent protocol.

4.2 IP Functionality

The Internet Protocol is the glue that holds the Internet together. It was designed explicitly for the task of internetworking or connecting many different types of network. Its primary function is to provide a best-effort attempt to deliver segments of data

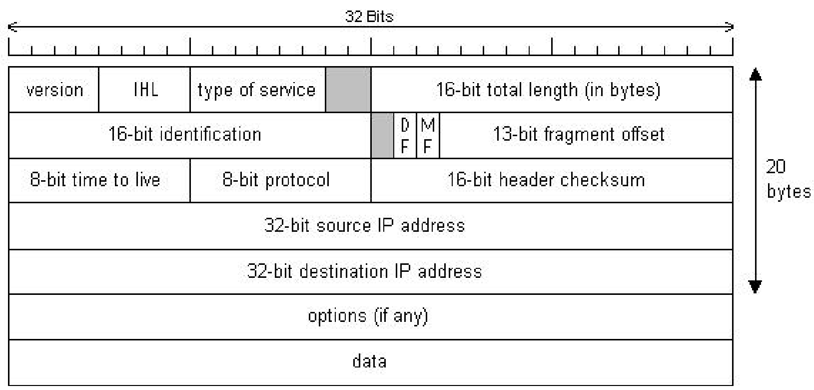

Figure 4.1: The IPv4 (Internet Protocol) header. (adapted from (Tanenbaum 2003)).

called datagrams from source to destination with no regard for where the source and destination are, be it on the same corporate Local Area Network, different Local Area Networks that are interconnected by a router or on two completely different networks separated by a multitude of other networks which are connected by many different paths, commonly called a route. (Tanenbaum 2003)

4.2.1 Looking at an IP Header

Each datagram being sent via IP has a special IP Header added. The header is used to identify the packet, set some control information, record the length of the datagram, record error checking information (error checking is only for the header itself, not for any data the header may contain) and indicate the source and destination of the datagram. The header consists of a 20-byte fixed part an a variable length optional part. There are two ways of looking at an IP Header. The first method is to review the theoretical contents of the header and learn what each part indicates. The second method is to examine a real IP packet to observe the practical application of the theory. This is important in the design and implementation of a NAT because the header will not label its contents adequately to assist the user in determining what the numbers actually mean. The structure of a IP Header is shown in Figure 4.1.

Quickly Summarizing the meaning of the individual fields in an IP Header:

- The *Version* field indicates which version of the protocol the datagram belongs to allowing transitions to new versions to roll-out over many years.

- The *IHL* or Internet Header Length is provided to define how long the header is in 32 bit words. The default and minimum value is 5 indicating a header with no options. The maximum value is 15 indicating a header containing 40 bytes of options.

- The *Type of Service* field distinguishes between different classes of service. For example real-time voice requires fast delivery, however, this is not concerned with reliability. In fact for most voice applications reliability and error checking cause more problems than they solve. For data downloads however reliability is more important than throughput (despite what some users would suggest). The actual use of this field will not be explained here as in practice most routers ignore its contents anyway.

- The *Total Length* indicates the length of the entire datagram which may consist of a IP header, options and data. The absolute maximum Total Length is 65,535 bytes which is currently suitable for 1500 byte maximum ethernet frames, however this is not ideal for new multi-gigabit ethernet connections.

- The *Identification* field allows the destination to determine which datagram a IP fragment belongs to. All IP fragments that originate from the same IP packet have the same Identification Number.

- *DF* is a single bit which stands for "Don't Fragment" and is an instruction that the packet must not be fragmented (normally because the destination does not have a full IP Stack loaded and cannot reassemble packets). IP requires that every participating network accepts frames of 576 bytes or less.

- *MF* means "more fragments". If an IP packet is fragmented all pieces except for the last will have this bit set.

- The *Fragment offset* indicates the relative position of the current fragment in the fully assembled packet. The offset is given as a number of 8-byte fields which offset the current fragment. As 13-bits is being used, 8192 fragments can occur. This allows complete fragmentation of a 65,536 byte packet (one byte larger than allowed by the IP protocol).

- *Time to live* is a counter of packet hops. It is decremented once by each router the packet passes through. If the value reaches zero the packet is discarded and the host warned. This system prevents a routing loop from buggy router configurations from crashing several backbone routers as packets flood into the loop but never leave.

- RFC1700 was the first global definition of transport level protocols. The globally accepted list is kept at `http://www.iana.org/assignments/protocol-numbers`. The *Protocol* field may contain any number from this web page. It is used at the destination and in some other circumstances such as Network Address Translators to determine what data to expect following the IP Header and Options.

- As stated earlier IP is a best effort protocol which will attempt to deliver datagrams, however makes no guarantee that individual datagrams will not become corrupted or not reach their destination. The *Header checksum* is used to verify that the data contained in the IP header is not corrupt. Higher level protocols often also use a checksum that verifies the entire packet, including the data, has not been corrupted. The IP checksum guards against routers with bad memory modules and ensures that when IP reports the source to a higher layer the address is correct.

- The *Source Address* is a 32 bit field used to identify the sending host.

- The *Destination Address* is a 32 bit field identifying the receiving host. More information on IP Addresses will be provided in a following section.

- A large range of special features have been defined for use in the IP header through the use of the *Options* field. The current list is kept current at `http://www.iana.org/assignments/ip-parameters`. The use of this field will not be discussed and has become depreciated due to the limited size of the field and the size of the global internet. (Tanenbaum 2003)

Table 4.1 shows the output of the debugger which has captured an IP packet. It demonstrates how IP Headers are practically used, however it would be a dunting task to actually decode the meaning of a packet without the details provided by the explanation.

Table 4.1: Contents of a real IP packet.

Buffer Position	Byte Contents	Explanation
[0]	69	Version = 4, IHL = 5
[1]	0	Type of service = 0 = Normal
[2]	0	
[3]	45	Total length = 45
[4]	159	
[5]	52	ID = 40756
[6]	64	
[7]	0	Don't Fragment
[8]	128	TTL = 128
[9]	6	Protocol = 6
[10]	0	
[11]	15	Checksum = 15
[12]	192	
[13]	168	
[14]	0	
[15]	3	Source = 192.168.0.3 (My Computer)
[16]	207	
[17]	46	
[18]	106	
[19]	173	Destination = 207.46.106.173
[20]	7	The rest of the packet can just be considered as data.
[21]	105	
[22]	7	
[23]	71	
[24]	231	
[25]	219	
[26]	240	
[27]	123	
[28]	109	

Table 4.1: (continued)

[29]	112	
[30]	168	
[31]	186	
[32]	80	
[33]	24	
[34]	65	
[35]	89	
[36]	213	
[37]	88	
[38]	0	
[39]	0	
[40]	80	
[41]	78	
[42]	71	
[43]	13	
[44]	10	

4.2.2 Fragmentation

IP was designed to work over many different types of networks with various hardware because it must accommodate for differences in the maximum frame sizes due to different underlying networks. The maximum frame size of the underlying network topology is called the maximum transmission unit (MTU).

Suppose in Figure 4.2 that Host A wants to send a large amount of data to Host C, hence the IP Protocol creates a packet which is 1500 bytes long (1480 bytes of data + 20 bytes for the IP Header). The router at Host B receives the packet and assuming the Don't Fragment bit is not set, it will create two packets both destined for Host C. The first packet will contain 976 bytes of data (the maximum multiple of 8 bytes + 20 bytes for the IP Header that can fit on the second network), will have the More

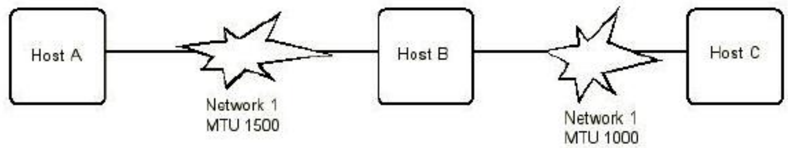

Figure 4.2: An Example of connecting two networks with differing MTU values. (adapted from (Feit 1998)).

Fragments (MF) bit set and will have a *fragment offset* of zero. The second packet will contain the remaining 504 bytes of data, will not have the More Fragments (MF) bit set however will have a *fragment offset* of 976. Host C will then have sufficient information to reassemble both fragments and receive the correct data. (Tanenbaum 2003)

4.2.3 IP Addresses

An IP Address is a 32 bit number used in the source and destination fields of an IP header. Each IP address consists of a network number and host number. In principle no two machines on the internet can have the same IP address (in practice this situation could occur however one host will not receive the packets unless the routing tables are configured incorrectly). It is worth mentioning that one of my home computers is a router and therefore receives two globally unique IP addresses, one for each of the external interface to the internet and internal interface for the intranet (internal network). When these issues are considered, the diminishing number of unused IP addresses is not surprising.

IP Addresses are usually written in dotted decimal notation. For example the 32-bit Hexadecimal address C0A80003 (or in decimal 3232235523) is written as 192.168.0.3. Several IP addresses also have special meanings. For example the IP address 0.0.0.0 refers to the current host. In the above mentioned network the address 0.0.x.x refers to the host with the given IP address on the local network. The address 255.255.255.255 indicates the broadcast address which means all hosts on the local network. The broadcast address on a remote network in the same class as above would be x.x.255.255 (though most administrators disable such addresses because they are a security risk).

The IP address 127.x.x.x always refers to the loopback device. The loopback device is a method of sending packets to the local machine without putting the packet onto the physical wire. The loopback address is also valid for a device which may not have any network interface installed to ease testing requirements. (Feit 1998)

4.2.4 Subnets

As mentioned earlier IP addresses encode the network and host number. For example the 192.168.0.3 IP address given consists of the network address, 192.168 and the host address, 0.3. Any computer in the same network must also have the same network address with a different host address. Varying each part of the host address from 0 to 255 gives 65025 addresses. Even though some addresses are reserved for the special purposes there are over 60000 usable IP addresses in the given address range. Unfortunately Ethernet was designed with much stricter limits of only 1024 hosts per network. The problem is that the networking authority will not give out two networks of 65025 hosts each simply because the underlying network was not scalable (especially when IP Addresses are already becoming scarce).

The solution was to allow networks to be split into several sub-networks or subnets although still appear as a single large network to the wider Internet. In the above example a subnet address would be specified to segregate the larger network into several smaller networks. For example a subnet address of 255.255.255.0 would allow 256 subnetworks (in some cases subnet masks of all zeros or all ones in the address cannot be used reducing this value to 254) each containing up to 254 hosts (0 and 255 are reserved in each subnetwork as mentioned previously). (Feit 1998)

4.2.5 Classless InterDomain Routing

Classless InterDomain Routing (CIDR) is a solution analogous to scraping the very last soup from the bottom of the pan. CIDR suggests allocating remaining IP addresses in variable sized blocks. The blocks are not completely free from restrictions, they still need to be allocated in blocks of 2^x however this allows the last of the IP address space to be allocated based on the current proven needs of an organization rather than anticipated future needs. Research has shown that over 50% of all networks supporting 64K hosts actually have less than 50 hosts. Under the old method of IP Address allocation these networks could have been given a network address supporting only 254 hosts and wasted IP Address space would have been avoided.

With CIDR each routing table entry has a 32-bit mask added. When a packet is recieved by the router the destination address is extracted and each address in the routing table is compared to the destination address which is masked by the correct mask (one bits in the mask allow the equalivent bit in the destination address while zero bits hide it) from the routing table. When a match is found the correct forwarding interface is looked up in the table and the packet transmitted to the next hop. Because multiple entries may match the destination address due to different masks the longest masks are used first. A long mask (a large number of the 32 bits are ones) indicate a very specific network whereas shorter masks indicate a more general case.

Fortunately the routing does get easier after the specific cases. Special cases are normally caused when the packet is very close to its destination. Consider the extreme example when the next hop will be to the final host, the router must decide which of several ports leads to the destination and will forward the packet on that port only. Now consider the other case where the destination is a long way from the current location (in terms of number of hops). When this occurs the routers may generalize specific cases. For example imagine that all internet addresses starting with 203.2.x.x belong to Australians. A router in Los Angeles does not need entries for 203.2.1.1 and then 203.2.1.2 when it can have a single entry for 203.2.0.0 with a mask of 255.255.0.0. All packets matching this routing entry would likely be forwarded by a transpacific ocean cable which might be called pacf01 by the router. (Tanenbaum 2003)

4.3 Network Address Translation

Despite the efforts to stop wasting IP Address space using solutions such as CIDR and requiring organizations to prove the need for new IP Addresses there has been little to no attempts to resolve the problem. New Internet designs with much larger IP Addresses are being tested, however final commercial implementation will be years away, if ever. A quick fix is required that can be implemented anywhere across the Internet at any moment. This quick fix has been developed in the form of Network Address Translation (NAT).

4.3.1 Overview of NAPT

Two types of NAT have been developed. The first version, Traditional NAT does not save IP addresses for end users. It can however be implemented by an ISP to save on the IP addresses wasted by allowing many small CIDR subnets (remember that a subnet with 2 IP addresses actually consumes 4 IP Addresses). Traditional NAT simply translates the users IP address to a globally unique IP address for packets destined for the internet. Hence if a user needs 4 IP addresses they are allocated 4 private IP addresses each being mapped to a global IP address by the service provider.

The second method of NAT is more accurately called Network Address Port Translation (NAPT). NAPT is concerned with actually reducing the need for multiple IP addresses. The idea is to multiplex a number of access requests to the external network onto a single globally unique IP Address (although there is no rule stating that only one global IP address could be used for NAPT). The incoming packets to the NAPT are then demultiplexed back to the original source. Internally this is accomplished by mapping tuples of the type (local IP addresses, local TU port number) to tuples of the type (registered IP address, assigned TU port number) where TU is the transport layer unit. Supported TUs are normally Transmission Control Protocol (see 5), User Datagram Protocol and Internet Control Message Protocol query's only. Limited inbound access can be provided by statically mapping a known TU port service to a specific local IP address. (Srisuresh & Egevang 2001)

4.3.2 Address Binding

In NAPT implementations, binding would take place between the tuple of (private address, private TU port) and the tuple of (assigned address, assigned TU port). This binding is created when the outgoing session commences.

4.3.3 Address Unbinding

NAPT may unbind the tuple of (assigned address, assigned TU port) when the last connection closes. However because the situation of a host crashing must be handled a timer is required to unbind addresses after a period of inactivity.

4.3.4 Header Manipulation

In the IP layer every packet header must be modified. These modifications are for the source IP address for outbound packets and destination address for inbound packets. The IP checksum must also be updated.

For TCP/UDP protocols the source port must be updated for outbound packets and restored for the destination port of inbound packets. The checksum must also be updated remembering that a pseudo header including the IP addresses forms part of the checksum. As an exception UDP packets with zero checksum should not be updated.

ICMP packets must also be specifically updated for the purposes of NAPT. The required updates are to the ICMP Query ID and ICMP checksum. The Query ID must be modifed from an internal ID to assigned ID for outbound packets and assigned ID to internal ID for inbound packets. (Srisuresh & Egevang 2001)

Sometimes it may be necessary to add a specific protocol to a NAPT. The implementation details are not defined by the NAT standards and will vary depending on the requirements of the protocol and the best practices for translating the protocol at the time the supporting software is written.

4.3.5 Incremental Checksum Adjustment

IP, TCP, UDP and ICMP headers all use the same form of checksum. Unfortunately the calculation of these checksums from scratch is computationally expensive. Fortunately there is a better method of calculating an incremental checksum for an existing header. There are two advantages of using this incremental checksum. First updating the checksum based on modifications to the header is less computationally expensive than recalculating from scratch. Second this form of update avoids the need to check for packet corruption. If the received packet is corrupt and the checksum is recalculated the next router or the destination machine will incorrectly believe the packet is correct because the recalculated checksum will equal the true checksum of the now corrupt packet. However incremental checksum on a corrupt packet will not correct for the corruption so the next router or destination that attempts to verify the checksum will fail and drop the packet. (Rijsinghani 1994)

4.3.6 ICMP error packet modifications

There is a slight difficulty in NAPT caused by ICMP error packets. The ICMP error packet may contain an embedded IP packet (normally one which caused an error) and this IP packet will contain source and destination addresses and possibly TCP/UDP port numbers which need to be updated to ensure end-to-end transparency of the NAT system. In addition the checksums of all embedded packets must also be updated to reflect changes made by modification of the ICMP error packet. More details on ICMP packets can be found at `ftp://ftp.rfc-editor.org/in-notes/rfc792.txt`.

4.3.7 FTP Support

NAPT requires special consideration when using File Transfer Protocol (FTP) in combination with NAPT. This is because FTP encodes several pieces of IP data into control packets. This data can include IP Addresses and TCP Port numbers. Additionally FTP can negotiate to open another port for the following file transfer. NAPT needs to recognize this expected incoming port or packets will be lost as they are destined to a (assigned address, assigned TU port) tuple that NAPT did not assign.

More details on the exact requirements of an Application Layer Gateway to overcome this problem are provided in Srisuresh & Egevang (2001).

4.3.8 Using IP Options

Although IP Options are depreciated and seldom used there is a possibility that private addresses contained in an IP Option would remain untranslated in a NAT packet traversing the Internet. This problem is not addressed in many implementations of NAT because routers using IP options should only consider the next-hop and the presence of a private IP address would be overlooked.

4.3.9 Recommendations for Private IP Address Range

Organizations using NAT are recommended to make use of the three private IP Address ranges provided by the Internet Assigned Numbers Authority (IANA). These Addresses are 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16. If these numbers are not used one possible problem that can occur is the local machine completes a DNS lookup and finds a local IP address is the destination. However the NAPT will be unable to differentiate between a request to the local IP address and one that should have been translated to the Internet.

4.3.10 Privacy and Security

NAPT provides a privacy and security mechanism by shielding internal clients from any unexpected inbound packets. Inbound packets are only possible if a port in use by NAPT is selected for the attack and the attack source uses the same IP address as the external machine. Launching such an attack is extremely difficult because both details are normally stored on the NAPT servers memory and are not available to users or external computers. Also because use of an invalid IP address is normally required by the attacking machine two way communication cannot normally be established.

However NAPT does have the undesirable impact on internal policing. If an internal client uses NAPT to shield their attack on the Internet the owner of the NAPT server will normally be blamed for the attack. Unless detailed logs are available the offending person cannot usually be identified.

4.3.11 Fragmented Packets

NAPT will never be able to successfully translate outbound TCP/UDP fragments. This failure results in the TCP/UDP header being contained in one of the fragments and not in any other fragments. Normally the IP Fragment number would be used in this situation however there is no guarantee that two client machines will not use the same fragment identifiers and result in corruption. The only solution is to have NAPT reassemble fragmented IP packets before allowing translation. This solution is also recommended for the purposes of enabling Secure IP (IPSec) over NAT. (Srisuresh & Egevang 2001)

4.4 IPv6

One of the newest major standards on the horizon is IPv6. Although IPv6 has not officially become a standard, it is worth some overview, especially since the final introduction of IPv6 will likely make the outcomes of this project worthless. It is very possible that this information will change as IPv6 moves closer to standardization, so this is a guide into IPv6, not definitive information. (Tanenbaum 2003)

Some of the benefits of IPv6 include greater addressing space, built-in Quality of Service (QoS), and better routing performance and services. However, a number of barriers must be overcome before the implementation of IPv6. The biggest will be what the business need is for moving from current IPv4 to IPv6. The killer application for IPv4 has not appeared yet, it may not appear at all. However IPv6 will gain momentum quickly if such an application is developed. The total lack of IP addresses may eventually force the role over without commercial support. Companies will follow more out of need for compatibility than some great new web application.

4.5 Chapter Summary

In summary IP is a protocol developed for use in interconnected packet-switched networks. It provides the underlying structure necessary to obtain information from one host to another while dealing with anything that might separate the source and destination hosts.

This chapter examined the IP Header and its data contents including some of the meaning of the data. Also some of the special functions of IP to deal with common situations with internetworking. An overview of some of the issues in distributing IP addresses to various organizations was given. The issues of Network Address Translation were then presented and reviewed in detail. This section is particularly complex and borrows some details from the next chapter on the Transmission Control Protocol (TCP). Readers who found this section difficult should not be disheartened as it requires a great deal of technical knowledge. It is recommended that the chapter on TCP is reviewed before returning to NAT.

Readers who have followed the contents of the current chapter will find the next two chapters on the Transmission Control Protocol and some existing implementations of NAT much lighter reading.

Chapter 5

Transmission Control Protocol

5.1 Chapter Overview

Transmission Control Protocol (TCP) builds on the IP layer to provide end-to-end connectivity for packet-switched networks. TCP must compensate for the lack of reliability in the IP layer and must operate in such a fashion as not to overload the underlying protocols or core network routers.

This chapter will review the major features of TCP including the header structure. The concepts of connections and establishing connections will be examined and a brief overview of another Transport layer protocol called User Datagram Protocol.

5.2 TCP Functionality

TCP is a connection-oriented reliable service designed as part of the Transport layer. TCP is connection oriented, with each application employing a TCP connection between itself and the opposing TCP end-point. This must occur before data may be sent or recieved. The common term for this type of connection is a state based service because the state of the connection is maintained and determines the ability of applications to use the service.

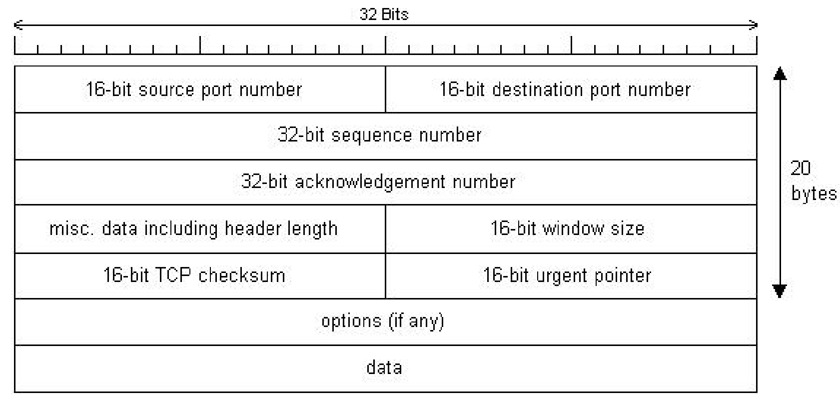

Figure 5.1: The IPv4 (Internet Protocol) header. (adapted from (Tanenbaum 2003)).

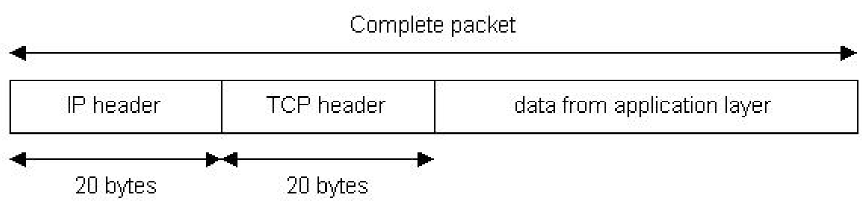

Figure 5.2: Combining TCP and IP to encapsulate data.

Figure 5.1 shows the structure of a TCP header while Figure 5.2 shows how a TCP header is added to the data which is then passed to the IP layer where an IP header is added.

Quickly Summarizing the meaning of the individual fields in an TCP Header:

- The *Source Port* identifies the end-point of the connection. The end-point must be identified to enable the TCP layer to determine which upper layer service requiring the data. Common port numbers are defined at `http://www.iana.org/assignments/port-numbers`.
- The *Destination Port* identifies the target end-point. A host's IP address and TCP Port together form a 48-bit unique end-point. Together the unique source and destination end-points identify the connection.
- The *Sequence Number* identifies the TCP segment to enable packets to be re-assembled to form a reliable byte stream.

- *Acknowledgement Number* is used to specify the *Sequence Number* of the next TCP segment expected (not the last segment that was correctly received).

- The misc data field starts off with a 4-bit *TCP Header Length*. This is followed by 6-bits which are not used and must always be zero. Finally there are six 1-bit flags used to signal important events to the TCP receiver. In order these are the *Urgent flag* (URG) which indicates that the Urgent Pointer is in use. The *Acknowledgement flag* (ACK) is set if the Acknowledgment Number is valid or cleared if the packet does not contain an Acknowledgement. The *Push flag* (PSH) is used to push the packet through the TCP layer immediately even if this means avoiding buffering. The *Reset flag* (RST) is used to reset a connection for which state information is invalid. This may be due to a host crash, to reject invalid segments or refuse a connection request. The *Synchronize flag* (SYN) is used to establish a connection. If the SYN bit is set and the ACK bit is cleared the request is to establish a connection. If both bits are set the packet represents acceptance of the connection. Finally the *Finish flag* (FIN) terminates a connection. However after sending a FIN packet the TCP port must remain open to incoming data indefinitely (assuming data continues to arrive).

- The *Window Size* is used for flow control as an indicator of how many bytes may be transmitted starting from the acknowledged segment.

- A TCP *Checksum* provides extra reliability. The TCP header, data and pseudo header participate in the checksum calculation. The checksum must be calculated with the checksum field set to zero. The pseudo header contains the IP source and destination addresses, a byte of 0 bits, a byte containing the protocol number (for TCP the protocol this is the binary representation of 6) and the TCP segment length (the IP total length minus the IP header length). Using the pseudo header violates the independence between the IP and TCP layers, however it offers additional protection against misdelivery of packets. The same concept of a pseudo header is used in the User Datagram Protocol (UDP).

- The *Urgent Pointer* indicates the byte offset from the current sequence number to find what is marked as urgent data. Urgent data is similar to computer interrupts, allowing the sender to send a type of interrupt signal to the receiver without TCP being aware of the reason for the urgent data.

- The *Options* field in TCP offers similarity to the IP Options field which allowed the implementation of a large range of special features. However unlike IP the TCP *Options* field is extensively used by features that support newer modern networking hardware and software. (Tanenbaum 2003)

5.2.1 Addressing

TCP provides its own addressing mechanism. Unlike IP however TCP addresses are not normally used by intermediate routers and are only meaningful to the end-host. Addressing is required to identify the application or service to which the data belongs. (Day & Zimmermann 1983) Some applications have default destination addresses such as HTTP. When requesting a web page we do not need to specify the use of port 80. However `http://www.google.com.au:80` is a valid reference to the Google website. With 65025 possible port numbers an internet host would need to run several thousand simultaneous web applications to use all possible ports. This knowledge allows the implementation of a Network Address Port Translator which uses ports to identify different hosts, a job normally left to the IP layer.

5.2.2 Reliability

TCP offers a reliable end-to-end byte stream which means each byte must be delivered once, in order and without error. However the IP layer does not guarantee any of these features. IP packets may be duplicated, lost, corrupted or delivered out of order. The TCP protocol uses sequence numbers, checksums that include the data and acknowledgement to ensure that received data is correct. Any TCP segment received with an incorrect checksum is immediately discarded. Such segments cannot be requested again at the time they are discarded as the checksum error could be due to corruption of the port numbers or sequence number. Sequence Numbers are also used to ensure correct ordering of the data and for the purposes of Acknowledgements. Acknowledgements may only be sent for segments which have been successfully received in order, TCP may internally buffer segments which are separated from the current sequence by a gap however may not transmit acknowledgments until the gap has been correctly filled. TCP may request that segments causing gaps in the stream are resent explicitly or request that all segments following a specific sequence number are retransmitted. However the sending host is responsible for retransmitting packets based on a timer. Support of requests from the opposing TCP end-point is optional. (Tanenbaum 2003)

5.2.3 Congestion Control

Internally the TCP recognises two types of congestion. Network Capacity is a response to an overflow of packets on the network which will cause routers to run out of buffer space and internally discard packets before they are delivered to the destination. Receiver Capacity is the amount of data the opposing TCP end-point is prepared to receive before it requires time to process the received data and pass it to the upper layers.

Internally TCP maintains two windows. One is the amount of data the receiver is prepared to accept, the second is the congestion window. TCP will never send more data than is indicated by the minimum of these two windows. The receiver window is controlled by the remote end-point. However controlling congestion on the network is slightly more involved.

Congestion control uses the congestion window and a threshold to find an optimal amount of data to fill the network without causing congestion. The threshold is initially set at 64 KB and TCP is allowed to transmit 1 KB of data. The TCP retransmission timer is started and TCP waits for acknowledgements or a timeout. Whenever the amount of data sent is all acknowledged without an error or timeout, TCP is allowed to transmit double the amount of data as on the previous attempt unless the amount has reached the threshold. After the amount of data sent on the previous attempt is equal to the threshold both grow linearly at a much slower rate (about 2 KB for each successful transmission). It is important to note that the amount of data transmitted may never exceed the threshold as the threshold grows at the same linear rate. When a timeout occurs the threshold is halved and the process repeats with TCP being allowed to send 1 KB which is doubled on every successful attempt until the threshold is reached. The process repeats indefinitely ideally with an average of about the correct amount of data the Internet can correctly handel at the time. At all times TCP may never send a burst larger than the minimum of the congestion control size and the receiver window size. (Tanenbaum 2003)

5.2.4 Connection Management

TCP needs to initialise and maintain some state based information including local and remote end-point information, local and remote sequence numbers and window sizes. Each TCP connection must be initiated and initialized by the transfer of state information. For example TCP hosts must agree on window sizes and starting sequence numbers when the connection is created. When the connection is closed the memory used for this state information may be freed for other applications or connections. All connection management data transfer is also checked and TCP must recover from any error such as lost connection requests in a timely manner. (Robison 2002)

5.3 Data Connections

5.3.1 Three Way Handshaking

All TCP connections begin passively with one side waiting for a connection. This involves a blocking call to the LISTEN or ACCEPT methods specifying a particular source address or accepting connections from any source address. The call is named as blocking because the application cannot proceed until a connection is made, the execution of the code is blocked until a connection request is received.

The second step is for an application to execute the CONNECT method specifying the end-point of the connection (IP Address and TCP Port), the window size and optionally any user data to be used when establishing the connection (a username and password for authentication perhaps). TCP then sends this information out in a packet with the SYN flag set. The recieving machine checks that an application has executed a LISTEN or ACCEPT method on the specified port. If this does not occur, the connection is

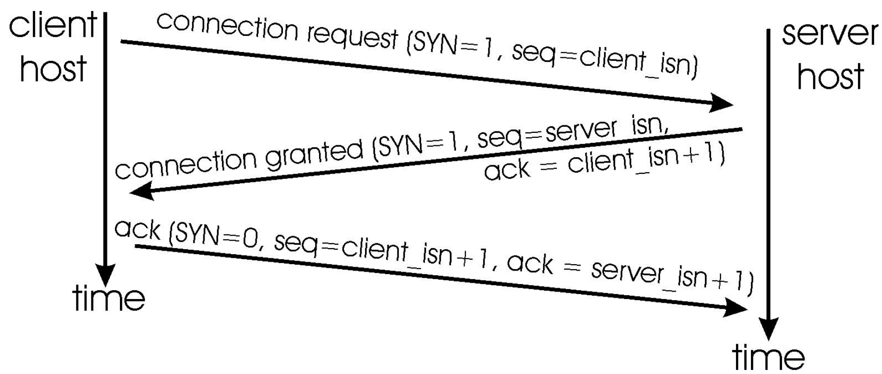

Figure 5.3: TCP connection establishment.

rejected by replying with a packet that has the RST flag set. Otherwise, the application that executed the LISTEN or ACCEPT method receives the connection request. This is accomplished by sending a reply packet with both the SYN and ACK flags set and the acknowledgement number being the sequence number used in the connection request plus one to indicate the next segment. Finally the session is fully opened by a reply with the next sequence number and the ACK flag set for the response. (Tanenbaum 2003) This process is depicted in Figure 5.3.

5.3.2 Simultaneous Open

In TCP there is the possibility that two hosts will simultaneously attempt to establish a connection. If this occurs only one connection must result. TCP handles this situation by ordering the connection end-points. Hence both connections will result in the connection (x,y) never (y,x) and TCP will only record one table entry for the connection. Each host will reply by resending their initial SYN while including an ACK for the opposite host's request. This situation is shown in Figure 5.4.

5.3.3 Active Close

TCP Connections may be closed by two methods. The first is an active close where the client closes the TCP application causing a FIN segment to be sent. The client may then receive a FIN only segment indicating a simultaneous close which is similar

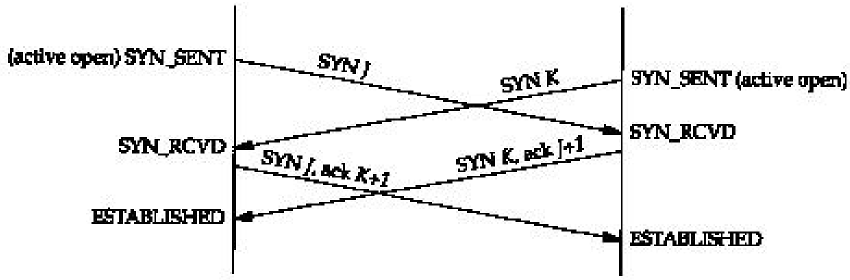

Figure 5.4: TCP Simultaneous Open. (adapted from (Tanenbaum 2003))

to a simultaneous open. Both applications then send ACK segments to finalize the connection. Alternatively the client may receive a FIN packet also Acknowledging its FIN. The client will send its final Acknowledgement and close the connection. Finally the client may recieve an Acknowledgement to its FIN without a FIN from the opposing end-point. This indicates that the opposite end-point may have more data or has kept the connection open for some other reason. The client must keep to its promise that it has completed sending data, however must continue to accept data until the opposite end-point agrees that it is also Finished (FIN) at which point the connection may be closed. In all cases a closed connection must be retained in memory in case any lingering packets related to the connection arrive.

5.3.4 Passive Close

Passive Close is the alternative method of ending a connection. A Passive Close occurs when the host receives a FIN before the application terminates. The host replies with an ACK segment for the FIN, however it must then wait for the application to complete its communications which may include sending more data before the return FIN can be transmitted. Once this FIN is Acknowledged the connection may be released.

5.4 User Datagram Protocol / Real-Time Transport Protocol

The User Datagram Protocol offers the benefits of IP communication without the strict connection establishment, state and release requirements involved in TCP. The main reason to offer this service is the addition of the port information found in the UDP header. Without this port information the transport layer would not be able to determine the appropriate higher layer destination of the information.

UDP segments can choose to use a checksum to ensure reliability. This option can also be disabled by setting the checksum to zero. Sometimes using the checksum is of little use due to the fact that the data stream is real-time and although the processing algorithms can deal with corrupt data it cannot wait for the data to be re-transmitted.

Another important feature of UDP is the lack of flow control, error control or retransmission. Particularly the lack of flow control means data is pumped onto the network as fast as the network layer can manage. This is particularly important for real-time communication.

The advantages of UDP in real-time applications lead to the development of an additional underlying protocol called the Real-Time Transport Protocol (RTP). The basic function of RTP is to multiplex several real-time data streams onto a single outgoing stream. RTP works on top of UDP which does raise some questions regarding as to which layer of the OSI or TCP/IP Reference Models it belongs.

RTP supports features such as time-stamping, encoding identification and sequencing. Sequencing is only concerned with the correct order of packets and helps the target applications account for lost data. For example if a video frame is lost it might be better not to update the video feed until the next frame arrives as opposed to blanking it. However, if the video is encoded, special consideration to prevent the corruption spreading across future frames may be required. There is also a Real-Time Transport Control Protocol which handles feedback from the RTP protocol. (Perkins 2002)

5.5 Chapter Summary

This chapter looked at Transport Layer protocols, an important link in computer communication. TCP provides a full duplex, reliable, flow controlled service to higher layers. TCP establishes and maintains connections on behalf of the higher layer service and then allows the transfer of data with the remote end-point. The data is delivered at the remote end-point and organized into correct order with any duplicates removed. TCP also handles connection release when both applications have finished using the communications channel.

TCP also plays a important role in the successful implementation of a Network Address Translator (NAT). By recognizing that TCP port numbers can be used by the NAT to represent internal end-points we can identify several different connections by varying the port number used for communication and translating this port number to the correct IP Address and Port Number at the NAT boarder.

It is important to recognize that several other transport protocols exist such as UDP. For a complete Network Address Translator we need to make as many of these protocols as possible function correctly under NAT. This is especially important for protocols which are in common use over the Internet so that NAT can provide functionality which approximates true IP Routing as closely as possible.

Chapter 6

Existing Network Address Translators

6.1 Chapter Overview

Part of implementing a new and successful Network Address Translator (NAT) is to understand current NAT implementations, the reason they are sucessful, in what manner they could be improved to become widely adopted and the purchase price. If a NAT can be developed to include better features, not require additional improvement and/or is cheap to customers, it will almost certainly gain a share of the market. If several of these objectives can be acheived, the current market leaders sales' would diminish as the new product prospered.

Obviously it is not the objective of this project to create a NAT which is ready for commercial distribution, however the issue of possible future commercialization of the product must be addressed for the lifetime of the code to extend over several years. If this issue is not considered the outcome of this project may not be suitable for commercial production environments and the entire project will become a throw-away prototype.

6.2 Windows

6.2.1 NAT32E

NAT32E is an enhanced IP Router allowing all private hosts on one or more Local Area Networks (LAN) to access the internet. NAT32E supports a range of connection interfaces including Dial Up Networking (DUN), Cable Modems, Asynchronous Digital Subsciber Line (ADSL) interfaces or Remote Access Service (RAS) interfaces. Configuration is automatic on most systems while manual web based configuration is also supported.

NAT32E supports a new feature called Connection Aggregation. This allows the NAT server to split data requests among two or more dial up modems. This feature may appear very useful, however it does not provide any level of competition to new Broadband services which operate without exclusive use of a phone line and cost approximately 50% less for the same speed.

NAT32E has many of the features that make NAT software most popular with home and small office users. It retails for US$50 for the more advanced version and US$25 for the single network only version.(NAT Software 2004)

6.2.2 BrowseGate 3 NAT/Proxy server and firewall

Browsegate 3 provides easy to use access to the Internet for all networked PC's. This includes all common services such as Web, Post Office Protocol (POP) and Simple Network Mail Protocol (SNMP) e-mail, Network News Transfer Protocol (NNTP), File Transfer Protocol (FTP) downloads or uploads and streaming video, audio and chat programs.

Browsegate 3 includes an integrated Firewall to stealth selected inbound or outbound access on specific ports. This type of technology is popular for larger organizations because it provides a higher measure of security. By controlling which services clients can access at a single point (the Firewall) the organization can make user-wide policy changes through a single update to the settings.

BrowseGate 3 is more commonly associated with large business as is indicated by its pricing schedule and enhanced security features. Pricing ranges from US$114.95 for a 5 computer licence to US$1100 for a unlimited computer version.(NetcPlus Internet Solutions 2004)

6.3 Linux

6.3.1 IP Masquerading

IP Masquerading is a form of Network Address Translation developed for Linux. The goal of the package is to provide the features of high priced routers and NAT servers without the high cost. IP Masquerading maps packets from the company intranet to the Internet and maps the responses from the Internet to the company intranet.

IP Masquerading has been developed over several years and is fairly secure and stable. It is currently being used with excellent results and any new bugs are quickly fixed by the Linux development community.

Because IP Masquerading forms part of the Linux kernel it is distributed in a number of flavors of Linux which can be downloaded freely from the Internet or purchased at minimal cost in a boxed set.

6.3.2 IP Tables

IP Tables is really just a newer version of IP Masquerading used in Linux Kernels 2.4.x and above. The purpose was to create an integrated NAT and Firewall environment including the ability to forward inbound services on static ports all as part of one large system configuration. In pervious versions of IP Masquerading the NAT was generally independent from the Firewall which was independent from Port Forwarding.

IP Tables used a large part of the stable and secure IP Masquerading system which has resulted in few bugs or errors. It is more popular among high-end System Administrators who find the integrated Firewall, NAT and Port Forwarding an advantage in easing System Administration burdens.

Again this system forms part of the newer Linux kernels and is distributed freely across the Internet.

6.4 Chapter Summary

Although there are a number of free, well developed NAT Servers available, a large majority of these support only Linux based Operating Systems. Most NAT Servers developed for Windows have a financial cost associated which may vary depending on the number of machines requiring simultaneous access to the Internet through the NAT.

Most NAT Servers for windows have some features in common. They are easy to set up with few user configured options. This can be achieved in most cases by guessing which interface is to a private network and which interface is to the Internet Provider. Most feature automatic setup of clients through the use of Dynamic Host Configuration Protocol (DHCP) to assign private IP addresses to internal clients and provide information to help clients access the NAT and other services such as Domain Name Service (DNS).

The key to success of a new NAT is to ask the question, "What services could be provided by NAT which are not currently supported by existing clients?" Most NAT Servers offer a wide range of services, so what more features can the user desire? The answer to this question is largely subjective and can be broken down into two sections. Features that Business Users seek and features that Home Users desire.

Security is the most important feature to Business Users. They require internal clients to be secure from the dangers of the Internet without constant monitoring of every computer. Business Users are therefore interested in services such as Authentication (Client with IP address X cannot use the NAT until user of client X identifies himself as a valid member of the business or organization). Sometimes this leads to the second requirement of exclusion. (If user of client X has made extensive use of Internet resources then prevent user of client X from accessing the Internet for the remainder of the day.)

Home Users favour ease of use and wide application support. Application Layer Gateways for non-NAT compatible applications and special modules, to identify specific modifications required to make packets compatible with NAT, are popular among such users. Any attempts to automatically configure the NAT to support a specific application will be well regarded by home users. High autonomy of security features and configuration will also be a popular program feature.

Obviously the requirements of Home and Business Users are very different. They may appreciate separate applications or at least different configurations based on a selection during installation. However, by tailoring the features of a NAT to more groups, increased sales and market share can be achieved. This will allow for lower cost software to compete with alternative NAT's to be viable.

Chapter 7

Network Address Translator Implementation

7.1 Chapter Overview

Implementation of the Network Address Translator (NAT) required knowledge of the C#.NET programming language and a knowledge of the Windows Socket Application Programming Interface (API) including special RAW Sockets to enable to inclusion of TCP/IP headers in the packets being sent and received. Additionally, knowledge of the function of a Network Address Translator in relation to the IP and TCP protocols is also necessary. This information concerning NAT was covered in Chapter 4.

This chapter will cover other aspects of implementation such as using the C#.NET language and Windows API Sockets.

7.2 C#.NET Basics

The C#.NET programming language is similar to an amalgamation of the C/C++ Programming Language and the Java programming language. The main difference in C#.NET is the input/output. In most C/C++ programs input/output is from the

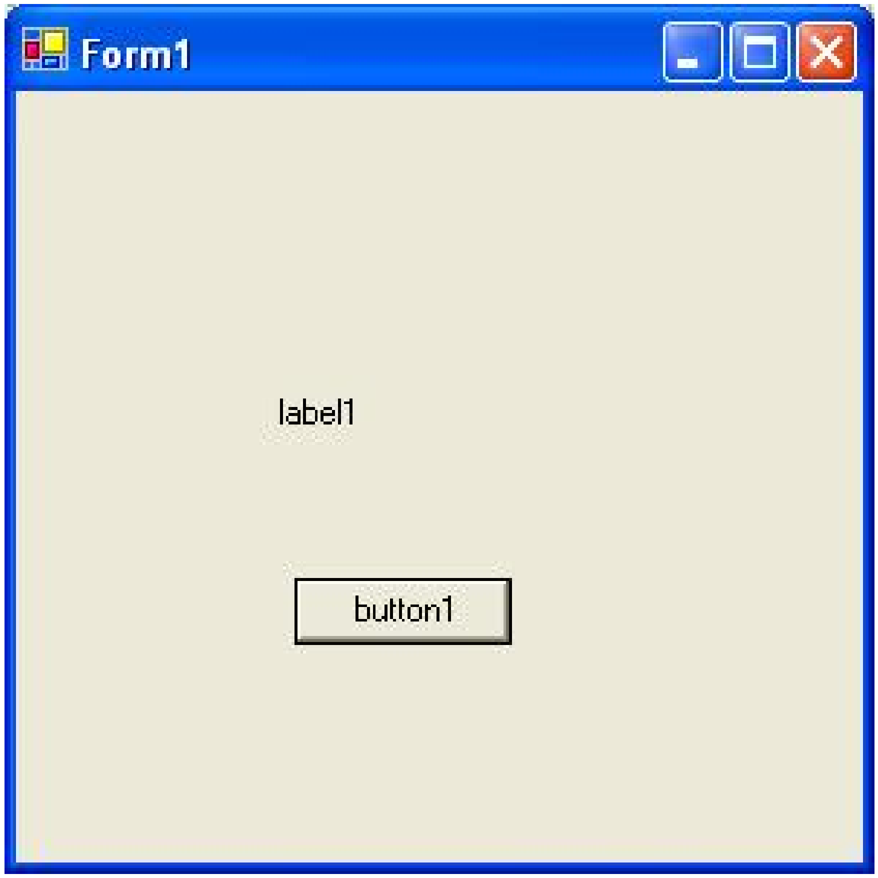

Figure 7.1: A simple C#.NET form.

console. In normal C#.NET programs there is a Graphical User Interface (GUI) and as such most input/output is from/to GUI controls. For example suppose a C#.NET Form is created as in Figure 7.1. Button1 is a input control which may be linked to some code. If we linked this button to the code in Listing 7.1 the result shown in Figure 7.2 will be displayed.

Listing 7.1: Hello World Code

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Dissertation
{
        /// <summary>
        /// Summary description for Form1.
        /// </summary>
        public class Form1 : System.Windows.Forms.Form
        {
                private System.Windows.Forms.Label label1;
                private System.Windows.Forms.Button button1;
```

```
            /// <summary>
            /// Required designer variable.
            /// </summary>
            private System.ComponentModel.Container
                components = null;

            public Form1()
            {
                    //
                    // Required for Windows Form Designer
                       support
                    //
                    InitializeComponent();

                    //
                    // TODO: Add any constructor code
                       after InitializeComponent call
                    //
            }

            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            protected override void Dispose( bool
                disposing )
            {
                    if( disposing )
                    {
                            if (components != null)
                            {
                                    components.Dispose();
                            }
                    }
                    base.Dispose( disposing );
            }

            #region Windows Form Designer generated code

            /// <summary>
            /// The main entry point for the application.
            /// </summary>
            [STAThread]
            static void Main()
            {
                    Application.Run(new Form1());
            }

            private void button1_Click(object sender,
                System.EventArgs e)
            {
                    label1.Text = "Hello World";
            }
    }
}
```

Although this is a simple example it does demonstrate some of the most basic features of C#.NET and should be easily understood by those who have previously developed in C++.

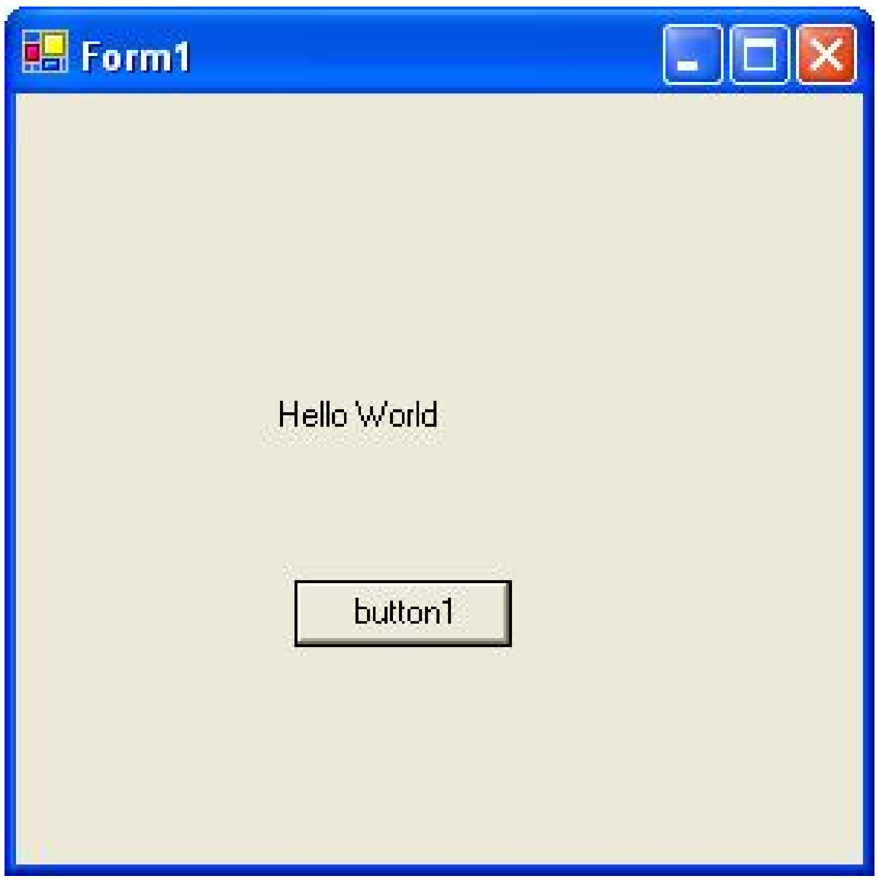

Figure 7.2: The result of Hello World Code execution on the C#.NET form.

7.3 Using Sockets

Understanding the most basic use of C#.NET is only the first step in developing a complex internet application. The next step is to understand the use of Windows Socket. Traditionally a socket was the end-point of a transport layer protocol such as TCP. However a socket became known as the end point for any protocol and the term RAW Socket was coined to describe a socket working below the IP Protocol, that is receiving or transmitting packets with the IP header intact and no IP error checking. (Robison 2002)

7.3.1 Application Programming Interface

In order for a third party application to use a core part of the Operating System it must follow a standard for calling system functions. In windows this standard is called the Application Programming Interface (API). The API controls all sockets as a part of the Operating System function. The socket may be controlled through a number of functions available to the programmer. By calling these functions correctly a program can create a socket connection and send or receive data.

7.3.2 Windows Sockets

The first step in using the API to creating a working socket is to create a socket descriptor. The descriptor is similar to unix files and is usually stored simply as an integer which has special meaning to the operating system. A socket descriptor is created by the code Socket nameofsocket = **null**; . Next, the socket descriptor is linked to a real socket. This is achieved by the code nameofsocket = **new** Socket(AddressFamily. InterNetwork, SocketType.Raw, ProtocolType.IP);. Obviously this socket is for Inter-Networks and is a RAW socket from the IP Protocol.

Once a socket is obtained the C#.NET development studio makes it easier by showing the methods implemented for the socket. This interface is shown in Figure 7.3. If the socket command to be accessed is known, then it is only necessary to type the first few

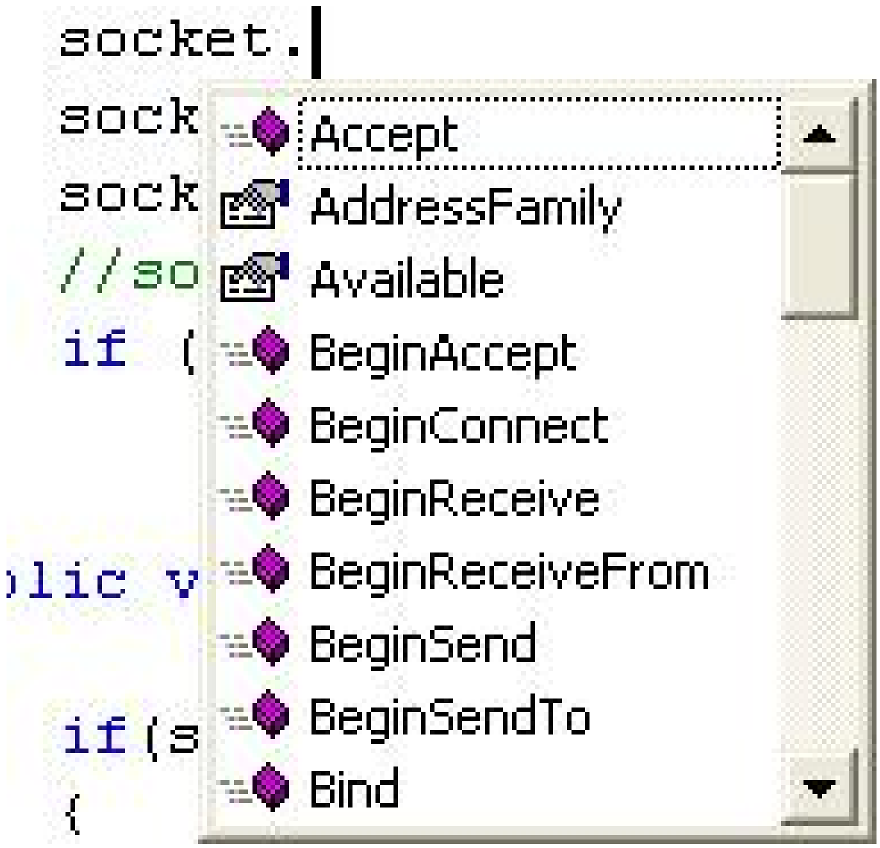

Figure 7.3: The Visual Studio Development Environment.

letters of the name and push the tab key, the Development Studio addresses the rest. It also provides dynamic help based on which command is being used and supports automatic selection for complex or well known choices.

7.3.3 Advanced Socket Control

There are a few special features of sockets required in this project. First there is the requirement to send packets including the IP Header without allowing the system to generate its own IP Header. Secondly, the NAT needs to check all packets as they are received, to ensure that the packets should be sent onto the internet. These packets will not be explicitly sent to the NAT Server because they will contain the address of the remote end-point. This type of receiving is called promiscuous mode and is not supported under all hardware and software configurations.

The solution to the first problem is fairly simple. C#.NET allows an option to be set on each socket to include the IP Header as in the code nameofsocket.SetSocketOption (SocketOptionLevel.IP, SocketOptionName.HeaderIncluded, 1);. Setting up Promiscuous mode however, requires a small block of code which requests the appropriate settings and checks the response to ensure the operation completed successfully indicating that the software can support this operation. Listing 7.2 shows the required code segment.

Listing 7.2: Promiscous Mode Sockets

```
private bool SetSocketOption()
{
        bool ret_value = true;
        try //.NET Exception handeling
        {
                byte []IN = new byte[4]{1, 0, 0, 0};
                byte []OUT = new byte[4];
                int SIO_RCVALL = unchecked((int)0x98000001);
                    // Control code for SIO_RCVALL documented
                    on MSDN.
                // See http://msdn.microsoft.com/library/
                    default.asp?url=/library/en-us/winsock/
                    winsock/wsaioctl_2.asp for details.
                int ret_code = socket.IOControl(SIO_RCVALL, IN
                    , OUT); //receive all IP packets on the
                    network.
                ret_code = OUT[0] + OUT[1] + OUT[2] + OUT[3];
                    //Check that operation suceeded.
                if(ret_code != 0) ret_value = false; //If not
                    return error.
        }
        catch(SocketException)
        {
                ret_value = false; //If any of the above
                    caused an exception, return an error.
        }
        return ret_value;
}
```

Using these advanced features and standard socket operations all the features required for this project were implemented.

7.4 Putting it all Together

Once the Fundamental aspects of working with the RAW Sockets API were understood the final program could be written. This included development of an overview of the solution and finally implementation as a Windows Service.

7.4.1 Pseudocode

Listing 7.3 shows the functioning of the NAT. Obviously this is made very simple in C#.NET by the functionality of Hashing Tables and the RAW Sockets class I wrote which managed all of the IP header and checksum details internally.

Listing 7.3: PseudoCode for Main NAT Function

```
ForEach Packet
    if Packet Source = Internal Network & Packet
        Destination = External Network
        Packet Source = Global Internet Address of NAT
        if Hash Table Result = Port Number
            Packet Source Port = Hash Table Result
        else
            Packet Source Port = New Hash Table Result
    if Packet Destination = Global Internet Address of NAT
         & Destination Port = Reverse Hash Table Result
        Packet Destination = Reverse Hash Table Result
        Packet Destination Port = Original Port Number
```

7.4.2 A Windows Service

Listing 7.4 shows the core code required for an implementation of a Windows Service in C#.NET. It is obvious from this code that the main points are to declare the required variables, set up some constantly looping decision making functions and clean up any variables and persistent code when stopping the service. Attention is drawn to the TODO: labels indicating areas where the user needs to add code. This code is completely generated by C#.NET without any user input except to request the creation of a Windows Service.

Listing 7.4: Implementing a Windows Service

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
```

```
using System.ServiceProcess;

namespace WindowsService1
{
        public class Service1 : System.ServiceProcess.
            ServiceBase
        {
                /// <summary>
                /// Required designer variable.
                /// </summary>
                private System.ComponentModel.Container
                    components = null;

                public Service1()
                {
                        // This call is required by the
                            Windows.Forms Component Designer.
                        InitializeComponent();

                        // TODO: Add any initialization after
                            the InitComponent call
                }

                // The main entry point for the process
                static void Main()
                {
                        System.ServiceProcess.ServiceBase[]
                            ServicesToRun;

                        // More than one user Service may run
                            within the same process. To add
                        // another service to this process,
                            change the following line to
                        // create a second service object. For
                             example,
                        //
                        //   ServicesToRun = New System.
                            ServiceProcess.ServiceBase[] {new
                            Service1(), new
                            MySecondUserService()};
                        //
                        ServicesToRun = new System.
                            ServiceProcess.ServiceBase[] { new
                             Service1() };

                        System.ServiceProcess.ServiceBase.Run(
                            ServicesToRun);
                }

                /// <summary>
                /// Required method for Designer support - do
                    not modify
                /// the contents of this method with the code
                    editor.
                /// </summary>
                private void InitializeComponent()
                {
                        components = new System.ComponentModel
                            .Container();
                        this.ServiceName = "Service1";
                }

                /// <summary>
                /// Clean up any resources being used.
                /// </summary>
```

```
                protected override void Dispose( bool
                    disposing )
                {
                        if( disposing )
                        {
                                if (components != null)
                                {
                                        components.Dispose();
                                }
                        }
                        base.Dispose( disposing );
                }

                /// <summary>
                /// Set things in motion so your service can
                    do its work.
                /// </summary>
                protected override void OnStart(string[] args)
                {
                        // TODO: Add code here to start your
                            service.
                }

                /// <summary>
                /// Stop this service.
                /// </summary>
                protected override void OnStop()
                {
                        // TODO: Add code here to perform any
                            tear-down necessary to stop your
                            service.
                }
        }
}
```

7.5 Chapter Summary

This chapter reviewed some of the significant features of C#.NET that were important in development of the Network Address Translator. By understanding these important concepts the development of the RAW IP Receiver and Sender could be completed. Once this was achieved the main task was to provide an interface for accessing the TCP/IP Header fields for updates and maintaining important information such as the Checksum's updated without involving the user.

The final code implementation is included in Appendix B. There are a few features of this code that have not been discussed here. However the main outcome of this project was a fully features packet class which can be used to receive packets completely, make modifications and send the packet. This class has several other uses in products such as usage meters, network bridges and routers. One important feature of this device is that a router could exist on a network without consuming an IP address while still offering all necessary routing features.

Chapter 8

Conclusions and Further Work

8.1 Achievement of Project Objectives

The following objectives have been addressed:

History of the Internet and Network Reference Models Understanding the motivation behind creating the Internet and the considerations made before its inception, is the first step in developing any type of Internet enabled application. The major focus was on the TCP/IP Protocol suite which is used for most Internet communications. The networking hierarchy was studied in detail to determine the contribution each layer made to the overall communication structure. Two networking hierarchy models were presented, the TCP/IP Reference Model and the Open Systems Interconnect (OSI) Reference Model. These models also discussed common network problems and how they can be overcome. Chapter 2 presented these important details.

Design and System Specification In Chapter 3 the reasons for key design choices were addressed. This project was not intended to be simply thrown away at completion and therefore, needed to adhere to strict communications standards. It was necessary for a design methodology to be chosen to maintain the project time line. Additionally the choice of programming language was discussed and chosen. Although the use of C#.NET was controversial for this type of project its selection was justified for the special coding features it contained.

The Internet Protocol Chapter 4 expands on a concept raised in the TCP/IP Reference Model from Chapter 2. The idea being to have the layer use a mesh of interconnecting networks to attempt transmission of a packet from a source host through the mesh onto another destination host. At this point Network Address Translator (NAT) implementation begins, however Transport layers also form part of the NAT service delivery and are discussed in Chapter 5.

The Transmission Control Protocol Chapter 5 offers greater detail on the Transport Layer which defines an end-to-end connection between two hosts. The layer provides the error control, retransmission and packet ordering expected by higher level services.

Existing Technology Current implementations of Network Address Translators were discussed in Chapter 6. The importance of utilizing popular features while implementing new ones was also detailed. The costs associated with purchasing some of these existing NATs were also addressed.

Implementation The concepts of C#.NET required in the implementation of a Network Address Translator and other low-level Network applications were addressed in Chapter 7. The topics ranged from a simple greeting program to complex socket operations. Full implementation code is provided in Appendix B.

Although the overall Network Address Translator is incomplete, the major objectives of the project have been achieved. A large amount of research was completed to understand core concepts of Internet communication protocols. Knowledge was gathered concerning features of a chosen programming language appropriate for NAT implementation. The concepts of NAT implementation were comprehensively researched. The code to deal with low level networking interfaces was implemented and the ability to develop NAT in the selected language was proven.

It is anticipated that future project development could use this research and development to implement a fully featured Network Address Translator with security and compatibility features aspired by consumers.

8.2 Further Work

The immediate future of this project requires research into problems experienced with connection stability. The existing code supports full TCP three way handshaking however the connection then resets resulting in client and host confusion. It is suspected that the RAW Socket code may allow the packet to also be passed to higher level layers on the NAT Server which then rejects the connection as unestablished.

In the longer term it would be recommended to implement security and authentication features to ensure only registered users are accessing the NAT server. This could be coupled with packet shaping (slowing the maximum throughput of a client) or connection severing (disconnect clients with no remaining quota). Home users would prefer features that detected expected incoming connections and attempt to compensate at the NAT Server. If such functionality is important sequence numbers could be used to further de-multiplex connections however this could quickly become computationally expensive to the NAT Server and is not a guaranteed method of delivery.

Other features that would integrate well with a NAT project are Firewalls to block unexpected or distrusted inbound connections, Proxy servers to cache web page requests and replies and dynamic port forwarding that open NAT ports when a service such as a Web Server starts and close the ports when the service is stopped.

Testing needs to be undertaken to determine if the NAT and other featured applications would require a multi-threaded approach to ensure maximum user throughput. At the time of development it was considered the development of multiple threads of control within the application were unnecessary. Under heavy load however, a multi-threaded applications may support more clients than a single-threaded application could handle.

References

Advanced Development Methods Inc (2004), *Scrum Development Process*, World Wide Web, United States of America.
`http://www.controlchaos.com/entry.htm`
current May 2004.

Baran, P. (1964), *Distributed Communications*, RAND Corporation.
`http://www.rand.org/publications/RM/RM3420/RM3420.chapter1.html`
current October 2004.

Business ESolutions (2002), *Project Lifecycle Models: How they differ and when to use them*, World Wide Web, Caifornia.
`http://www.business-esolutions.com/islm.htm`
current May 2004.

Campione, M., Walrath, K. & Huml, A. (2000), *The Java Tutorial: A Short Course on the Basics*, third edn, Addison-Wesley.

Cerf, V. & Kahn, R. (1974), *A Protocol for Packet Network Interconnection*, Vol. COM-22, IEEE Trans. on Commun., pp. 637–648.

Day, J. & Zimmermann, H. (1983), *The OSI Reference Model*, Vol. 71, The Institute of Electrical and Electronics Engineers, pp. 1334–1340.

Feit, D. S. (1998), *TCP/IP Architecture, Protocols and Implementation with IPv6 and IP Security*, second edn, McGraw-Hill.

Kahn, H. (2000), *STEPWISE Project*, World Wide Web, Manchester.

http://www.stepwise.org
current May 2004.

NAT Software (2004), *NAT32 Home Page*, NAT Software Germany.
http://www.nat32.com/
current October 2004.

NetcPlus Internet Solutions (2004), *BrowseGate 3 NAT/Proxy server and firewall*, NetcPlus Internet Solutions.
http://www.netcplus.com/browsegate.html
current October 2004.

Perkins, C. E. (2002), *RTP: Audio and Video for the Internet*, Addison-Wesley, Boston.

Rijsinghani, A. (1994), *Computation of the Internet Checksum via Incremental Update*, Network Working Group.
ftp://ftp.rfc-editor.org/in-notes/rfc1624.txt
current October 2004.

Roberts, L. G. (1967), *Multiple Computer Networks and Intercomputer Communication*, Proc. First Symp. on Operating Systems Prin., ACM.

Robison, W. (2002), *Pure C#: A Code-Intensive Premium Reference*, Sams Publishing.

Sharon, Y. (1999), *(ootips) The Rational Unified Process*, World Wide Web, United States of America.
http://ootips.org/rup.html
current May 2004.

Srisuresh, P. & Egevang, K. (2001), *Traditional IP Network Address Translator (Traditional NAT)*, Network Working Group.
ftp://ftp.rfc-editor.org/in-notes/rfc3022.txt
current October 2004.

Tanenbaum, A. S. (2003), *Computer Networks*, fourth edn, Prentice Hall PTR.

Wells, D. (2003), *Extreme Programming*, World Wide Web, Utah.
http://www.extremeprogramming.org/index.html
current May 2004.

Appendix A

Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG 4111/4112 Research Project
PROJECT SPECIFICATION

FOR: **Kevin-John BEASLEY**

TOPIC: Design and Implementation of a Network Address Translator

SUPERVISORS: Dr. John Leis

PROJECT AIM: The project aims to investigate the core aspects of computer networking and TCP/IP Sockets so an efficient and scalable Network Address Translator (NAT) can be designed and implemented for both home and business use.

PROGRAMME: **Issue B, 1 October 2004**

1. Research information on computer networking including history and development.

2. Investigate the TCP/IP Reference Model and TCP/IP Protocols to understand how computer networking relates to the Internet

3. Utilise RFC's relating to NAT Functionality and Implementation.

4. Research appropriate Programming languages for Implementation of a NAT Server, especially the availability of Sockets Programming.

5. Design appropriate sockets interfaces to enable incoming TCP/IP packets to be captured, translated and sent. This includes updating the TCP and IP checksums where necessary.

6. Design a simple NAT server based on the above socket interface to prove the ability to implement NAT in the chosen language.

As time permits:

7. Improve on the NAT implementation to include more features and greater scalability. Support more types of Transport Layer translation.

AGREED: _____________ (student) _______________ (Supervisor)

___ / ___ / ___ ___ / ___ / ___ (date)

Appendix B

Project Source Code

B.1 NATService.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Net;
using System.Net.Sockets;
using System.ServiceProcess;
using Microsoft.Win32;
using System.Security;
using System.IO;
using System.Extended.Collections;

namespace NAT_Service
{
        public class Service1 : System.ServiceProcess.ServiceBase
        {
                /// <summary>
                /// Required designer variable and Raw Socket class.
                /// </summary>
                private System.ComponentModel.Container components = null;
                RawSender myRawSend;
                RawSender myIntRawSend;
                RawSender[] RawSenders = new RawSender[100];
                RawSocket ExternalRawSock;
                RawSocket InternalRawSock;
                ushort[] ReservedPorts = new ushort[100];
                bool[] PortinUse = new bool[100];
                BidirHashtable twowayhash = new BidirHashtable();
                string ReversedGlobalIP;
                string ReversedLocalIP;
                int numberofports;

                public Service1()
                {
                        // This call is required by the Windows.Forms Component Designer.
                        InitializeComponent();

                        // TODO: Add any initialization after the InitComponent call
                }

                // The main entry point for the process
                static void Main()
                {
                        System.ServiceProcess.ServiceBase[] ServicesToRun;

                        // More than one user Service may run within the same process. To add
                        // another service to this process, change the following line to
                        // create a second service object. For example,
                        //
                        //   ServicesToRun = New System.ServiceProcess.ServiceBase[] {new Service1(), new MySecondUserService()};
                        //
```

```
            ServicesToRun = new System.ServiceProcess.
                ServiceBase[] { new Service1() };

            System.ServiceProcess.ServiceBase.Run(
                ServicesToRun);
        }

        /// <summary>
        /// Required method for Designer support - do not
            modify
        /// the contents of this method with the code
            editor.
        /// </summary>
        private void InitializeComponent()
        {
            components = new System.ComponentModel.
                Container();
            this.ServiceName = "Service1";
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        /// <summary>
        /// Set things in motion so your service can do
            its work.
        /// </summary>
        protected override void OnStart(string[] args)
        {
            // TODO: Add code here to start your service
                .
            for(int i = 0; i<100; i++)
            {
                ReservedPorts[i] = 0;
                PortinUse[i] = false;
            }
            RegistryKey key;
            try
            {
                RegistryKey softwareKey = Registry.
                    LocalMachine.OpenSubKey( "Software
                    " );
                if( softwareKey == null )
                {
                    EventLog.WriteEntry( "Unable to
                        open the registry Software
                        key for 'USQ NAT Project'" )
                        ;
                    return;
                }

                key = softwareKey.OpenSubKey( "
                    NATUSQProj" );
```

```
                if( key == null )
                {
                    EventLog.WriteEntry( "Unable to open the registry NATUSQProj for 'USQ NAT Project'" );
                    return;
                }
            }
            catch( ArgumentNullException argNullExp )
            {
                EventLog.WriteEntry( "Argument null exception thrown " + argNullExp.Message );
                return;
            }
            catch( ArgumentException argExp )
            {
                EventLog.WriteEntry( "Argument exception thrown " + argExp.Message );
                return;
            }
            catch( IOException ioExp )
            {
                EventLog.WriteEntry( "IO Exception thrown " + ioExp.Message );
                return;
            }
            catch( SecurityException secExp )
            {
                EventLog.WriteEntry( "Security exception thrown " + secExp.Message );
                return;
            }

            string LocalIP;
            string GlobalIP;
            int temp = 0;
            ushort testport = 9000;
            LocalIP = key.GetValue( "LocalIP" ).ToString();
            GlobalIP = key.GetValue( "GlobalIP" ).ToString();
            string[] tempIP;
            tempIP = GlobalIP.Split(".".ToCharArray(),4);
            ReversedGlobalIP = tempIP[3] + "." + tempIP[2] + "." + tempIP[1] + "." + tempIP[0];
            tempIP = LocalIP.Split(".".ToCharArray(),4);
            ReversedLocalIP = tempIP[3] + "." + tempIP[2] + "." + tempIP[1] + "." + tempIP[0];
            numberofports = (int)key.GetValue( "InitialPorts" );
            do
            {
                bool exception;
                do
                {
                    exception = false;
                    try
                    {
                        myRawSend=new RawSender();
```

```
                        myRawSend.StartSender (GlobalIP, testport);
                    }
                    catch (SocketException ex)
                    {
                        if (ex.ErrorCode == 10048)
                        {
                            exception = true;
                        }
                    }
                    testport = testport++;
                }while(exception || myRawSend.WasError);
                RawSenders[temp] = myRawSend;
                ReservedPorts[temp] = (ushort)(testport - 1);
            } while(numberofports > temp);
            myRawSend=new RawSender();
            myRawSend.StartSender (GlobalIP, 9258);
            myIntRawSend=new RawSender();
            myIntRawSend.StartSender (LocalIP, 0);
            if (myRawSend.WasError)
            {
                EventLog.WriteEntry("Critical Error: Socket Failed.");
                return;
            }
            if (myIntRawSend.WasError)
            {
                EventLog.WriteEntry("Critical Error: Socket Failed.");
                return;
            }
            string IPString="10.10.10.10";
            IPHostEntry HosyEntry = Dns.Resolve(Dns.GetHostName());
            if(HosyEntry.AddressList.Length > 0)
            {
                foreach(IPAddress ip in HosyEntry.AddressList)
                {
                    IPString=ip.ToString();
                }
            }

            ExternalRawSock=new RawSocket();
            ExternalRawSock.StartSocket (LocalIP, 0, true);
            ExternalRawSock.PacketArrival += new RawSocket.PacketArrivedEventHandler(ExternalDataArrival);
            if (ExternalRawSock.ErrorOccurred)
            {
                EventLog.WriteEntry("Critical Error: Socket Failed.");
                return;
            }
            InternalRawSock=new RawSocket();
            InternalRawSock.StartSocket (GlobalIP, 0, true);
            InternalRawSock.PacketArrival += new RawSocket.PacketArrivedEventHandler(
```

```
                InternalDataArrival);
            if (InternalRawSock.ErrorOccurred)
            {
                EventLog.WriteEntry("Critical Error: Socket Failed.");
                return;
            }
            ExternalRawSock.KeepRunning = true; //Want to recieve all incomming packets.
            ExternalRawSock.Run ();
            InternalRawSock.KeepRunning = true;
            InternalRawSock.Run();
        }

        private void InternalDataArrival(Object sender, PacketArgs e)
        {
            IPAddress test, test2;
            test = new IPAddress((e.source));
            test2 = new IPAddress(e.destination);
            if((e.destination == (uint)((System.Net.IPAddress.Parse(ReversedGlobalIP).Address))) && (e.tcpdestination == 9258))
            {
                ulong returnaddr;
                returnaddr = (ulong)twowayhash.ReverseLookup((ulong)((e.destination << 16) + e.tcpdestination));
                e.destination = (uint)(returnaddr >> 16) & 0xFFFFFFFF;
                e.tcpdestination = (ushort)(returnaddr & 0xFFFF);
                myIntRawSend.send(e);
                if (!myIntRawSend.WasError)
                {
                    // If adding a debuging mode report that all went well.
                }
                else
                {
                    EventLog.WriteEntry("Error: Return Packet Failed.");
                }
            }
        }

        private void ExternalDataArrival(Object sender, PacketArgs e)
        {
            IPAddress test, test2;
            test = new IPAddress((e.source));
            test2 = new IPAddress(e.destination);
            if(e.source == (uint)((System.Net.IPAddress.Parse(ReversedLocalIP).Address)))
            {
                if (e.tcpdestination == 80)
                {
                    int selectport = -1;
                    for(int i = 0; i<numberofports - 1; i++)
```

```
                        {
                            if (PortinUse[i] == false)
                                selectport = i;
                                break;
                        }
                        if ((selectport == -1) && (PortinUse[selectport] == true))
                        {
                            EventLog.WriteEntry("Insufficient Ports Allocated.");
                            //Replace this line with code to dynamically allocate more ports in future.
                            return;
                        }
                        else
                        {
                            PortinUse[selectport] = true;
                            //Eventually need timers to reclaim closed ports.
                            //Only working with one host at this time so automatic reuse will occure.
                        }
                        twowayhash[((e.source << 16) + e.tcpsource)] = (ulong)(((uint)(System.Net.IPAddress.Parse(ReversedGlobalIP).Address) << 16) + ReservedPorts[selectport]);
                        e.source = (uint)((System.Net.IPAddress.Parse(ReversedGlobalIP).Address));
                        e.tcpsource = ReservedPorts[selectport];
                        myRawSend.send(e);
                        if (!myRawSend.WasError)
                        {
                            //Debug mode will report sucessful translation here.
                        }
                        else
                        {
                            EventLog.WriteEntry("Error: Cannot Transmit Packet.");
                        }
                    }
                }
            }

            /// <summary>
            /// Stop this service.
            /// </summary>
            protected override void OnStop()
            {
```

```
                        // TODO: Add code here to perform any tear-
                            down necessary to stop your service.
                        ExternalRawSock.KeepRunning = false;
                        InternalRawSock.KeepRunning = false;
                }
        }
}
```

B.2 RawSocket.cs

```
// RawSocket Class


namespace NAT_Service
{
        using System;
        using System.Net;
        using System.Net.Sockets;
        using System.Runtime.InteropServices;


        [StructLayout(LayoutKind.Explicit)]
        public struct IPHeader
        {
                [FieldOffset(0)] public byte    ip_verIHL; //IP Version (4 bits) + IHL (Header Length) (4 bits)
                //IHL of 5 indicates no options. IHL of 15 indicates 40 bytes of options.
                [FieldOffset(1)] public byte    ip_tos; //Type of Service + Empty (2 bits)
                [FieldOffset(2)] public ushort  ip_totallength; //Total Packet Length
                [FieldOffset(4)] public ushort  ip_id; //Unique IP ID
                [FieldOffset(6)] public ushort  ip_DFMFoffset; //Empty (1 bit) + Don't Fragment (1 bit) + More Fragments (1 bit) Flags + Offset (13 bits)
                [FieldOffset(8)] public byte    ip_ttl; //Time To Live
                [FieldOffset(9)] public byte    ip_protocol; //Protocol (TCP, UDP, ICMP, Etc.)
                [FieldOffset(10)] public ushort ip_checksum; //IP Header Checksum
                [FieldOffset(12)] public uint   ip_srcaddr; //Source IP Address
                [FieldOffset(16)] public uint   ip_destaddr; //Destination IP Address
                // IP Options go here but since we don't know if any exist we won't include them here.
        }

        [StructLayout(LayoutKind.Explicit)]
        public struct TCPHeader
        {
                [FieldOffset(0)] public ushort      tcp_srcport; //Source TCP Port Number.
                [FieldOffset(2)] public ushort      tcp_destport; //Destination TCP Port Number.
                [FieldOffset(4)] public uint  tcp_sequence; //TCP Sequence Number
                [FieldOffset(8)] public uint  tcp_acknowledgement; //TCP Acknowledgement Number;
                [FieldOffset(12)] public byte tcp_headerlength; //TCP Header Length (4 bits) + Empty (4 bits)
                [FieldOffset(13)] public byte tcp_flags; //Empty (2 bits) + URG (1 bit) + ACK (1 bit) + PSH (1 bit) + RST (1 bit) + SYN (1 bit) + FIN (1 bit)
                [FieldOffset(14)] public ushort     tcp_windowsize; //TCP Window Size
                [FieldOffset(16)] public ushort tcp_checksum; //TCP Checksum
```

```
            [FieldOffset(18)] public ushort tcp_urgentpointer; //TCP Urgent Pointer
            // TCP Options go here but since we don't know if any exist we won't include them here.
        }
        public class PacketArgs : EventArgs
        {
            public PacketArgs(byte [] buf) //Initialise.
            {
                int messagelength;
                int IHL;
                //IPHeader * head = (IPHeader *) fixed_buf; // Assign IP Header from recieve buffer.
                //this.prottype = head->ip_protocol;
                this.MainHeader = new IPHeader();
                //Array.Copy(buf, 0, this.MainHeader, 0, 20);
                this.MainHeader.ip_verIHL = buf[0];
                this.MainHeader.ip_tos = buf[1];
                this.MainHeader.ip_totallength = (ushort)((buf[2] << 8) + buf[3]);
                this.MainHeader.ip_id = (ushort)((buf[4] << 8) + buf[5]);
                this.MainHeader.ip_DFMFoffset = (ushort)((buf[6] << 8) + buf[7]);
                this.MainHeader.ip_ttl = buf[8];
                this.MainHeader.ip_protocol = buf[9];
                this.MainHeader.ip_checksum = (ushort)((buf[10] << 8) + buf[11]);
                this.MainHeader.ip_srcaddr = (uint)((buf[12] << 24) + (buf[13] << 16) + (buf[14] << 8) + buf[15]);
                this.MainHeader.ip_destaddr = (uint)((buf[16] << 24) + (buf[17] << 16) + (buf[18] << 8) + buf[19]);
                IHL = (this.MainHeader.ip_verIHL & 0x0F);
                if (IHL > 5)
                {
                    this.IPOptions = new byte[(IHL - 5) * 4];
                    Array.Copy(buf, 20, this.IPOptions, 0, (IHL - 5) * 4);
                }
                switch (this.MainHeader.ip_protocol)
                {
                    case 6:
                        this.TCPHead = new TCPHeader();
                        this.TCPHead.tcp_srcport = (ushort)((buf[IHL * 4] << 8) + buf[IHL * 4 + 1]);
                        this.TCPHead.tcp_destport = (ushort)((buf[IHL * 4 + 2] << 8) + buf[IHL * 4 + 3]);
                        this.TCPHead.tcp_sequence = (uint)((buf[IHL * 4 + 4] << 24) + (buf[IHL * 4 + 5] << 16) + (buf[IHL * 4 + 6] << 8) + buf[IHL * 4 + 7]);
                        this.TCPHead.tcp_acknowledgement = (uint)((buf[IHL * 4 + 8] << 24) + (buf[IHL * 4 + 9] << 16) + (buf[IHL * 4 + 10]
```

```
                                        << 8) + buf[IHL * 4 + 11]);
                                this.TCPHead.tcp_headerlength = buf[IHL * 4 + 12];
                                this.TCPHead.tcp_flags = buf[IHL * 4 + 13];
                                this.TCPHead.tcp_windowsize = (ushort)((buf[IHL * 4 + 14] << 8) + buf[IHL * 4 + 15]);
                                this.TCPHead.tcp_checksum = (ushort)((buf[IHL * 4 + 16] << 8) + buf[IHL * 4 + 17]);
                                this.TCPHead.tcp_urgentpointer = (ushort)((buf[IHL * 4 + 18] << 8) + buf[IHL * 4 + 19]);
                                if (this.tcpheaderlength > 5)
                                {
                                    this.TCPOptions = new byte[(TCPHead.tcp_headerlength - 5) * 4];
                                    Array.Copy(buf, (IHL * 4) + 20, this.TCPOptions, 0, (TCPHead.tcp_headerlength - 5) * 4);
                                }
                                messagelength = this.MainHeader.ip_totallength - (IHL * 4) - (this.tcpheaderlength * 4);
                                this.RemainingData = new byte[messagelength];
                                Array.Copy(buf, (IHL * 4) + (this.tcpheaderlength * 4), this.RemainingData, 0, messagelength);
                                break;
                            default: //For any other packets (if implmented) copy everything after the IP Header as data.
                                messagelength = this.MainHeader.ip_totallength - (IHL * 4);
                                this.RemainingData = new byte[messagelength];
                                Array.Copy(buf, IHL * 4, this.RemainingData, 0, messagelength);
                                break;
                    }
            }

            private UInt16 incrementalchecksum(UInt16 original, UInt16 updated)
            {
                    Int32 cksum = Convert.ToInt32(((~(UInt16)this.checksum) & 0xFFFF));
                    cksum += Convert.ToInt32(((~(UInt16)original) & 0xFFFF));
                    cksum += Convert.ToInt32(updated);
                    cksum = (cksum >> 16) + (cksum & 0xffff);
                    cksum += (cksum >> 16);
                    return (UInt16)((~cksum));
            }
```

```
        private UInt16 incrementalTCPchecksum(UInt16
            original, UInt16 updated)
        {
            Int32 cksum = Convert.ToInt32(((~(UInt16)
                this.tcpchecksum) & 0xFFFF));
            cksum += Convert.ToInt32(((~(UInt16)original
                ) & 0xFFFF));
            cksum += Convert.ToInt32(updated);
            cksum = (cksum >> 16) + (cksum & 0xffff);
            cksum += (cksum >> 16);
            return (UInt16)((~cksum));
        }

        /*public static UInt16 calcchecksum( Byte[] buffer
            , int size )
        {
            Int32 cksum = 0;
            int counter;
            counter = 0;
            while ( size > 0 )
            {
                UInt16 val = (ushort)((buffer[counter]
                    << 8) + buffer[counter+1]);
                cksum += Convert.ToInt32( val );
                counter += 2;
                size -= 2;
            }
            cksum = (cksum >> 16) + (cksum & 0xffff);
            cksum += (cksum >> 16);
            return (UInt16)(~cksum);
        }*/

        public byte version
        {
            get {return (byte)((this.MainHeader.
                ip_verIHL & 0xF0) >> 4);}
            set
            {
                ushort temp = (ushort)((this.
                    MainHeader.ip_verIHL << 8) + this.
                    MainHeader.ip_tos);
                this.MainHeader.ip_verIHL = (byte)((
                    this.MainHeader.ip_verIHL & 0x0F)
                    + ((value & 0x0F) << 4));
                this.checksum = incrementalchecksum(
                    temp, (ushort)((this.MainHeader.
                    ip_verIHL << 8) + this.MainHeader.
                    ip_tos));
            }
        }

        public byte IHL
        {
            get {return (byte)(this.MainHeader.ip_verIHL
                & 0x0F);}
            set
            {
                ushort temp = (ushort)((this.
                    MainHeader.ip_verIHL << 8) + this.
                    MainHeader.ip_tos);
                ushort tcplength = (ushort)(this.
                    totallength - (this.IHL * 4));
```

```
                    this.MainHeader.ip_verIHL = (byte)((this.MainHeader.ip_verIHL & 0xF0) + (value & 0x0F));
                    this.checksum = incrementalchecksum(temp, (ushort)((this.MainHeader.ip_verIHL << 8) + this.MainHeader.ip_tos));
                    this.tcpchecksum = incrementalTCPchecksum(tcplength, (ushort)(this.totallength - (this.IHL * 4)));
                }
            }

            public byte TOS
            {
                get {return (byte)((this.MainHeader.ip_tos) >> 2);}
                set
                {
                    ushort temp = (ushort)((this.MainHeader.ip_verIHL << 8) + this.MainHeader.ip_tos);
                    this.MainHeader.ip_tos = (byte)((value & 0xFF) << 2);
                    this.checksum = incrementalchecksum(temp, (ushort)((this.MainHeader.ip_verIHL << 8) + this.MainHeader.ip_tos));
                }
            }

            public ushort totallength
            {
                get {return this.MainHeader.ip_totallength;}
                set
                {
                    ushort temp = this.MainHeader.ip_totallength;
                    ushort tcplength = (ushort)(this.MainHeader.ip_totallength - (this.IHL * 4));
                    this.MainHeader.ip_totallength = value;
                    this.checksum = incrementalchecksum(temp, this.MainHeader.ip_totallength);
                    this.tcpchecksum = incrementalTCPchecksum(tcplength, (ushort)(this.MainHeader.ip_totallength - (this.IHL * 4)));
                }
            }

            public ushort ID
            {
                get {return this.MainHeader.ip_id;}
                set
                {
                    ushort temp = this.MainHeader.ip_id;
                    this.MainHeader.ip_id = value;
                    this.checksum = incrementalchecksum(temp, this.MainHeader.ip_id);
                }
```

```
        }

        public bool DF
        {
            get {if((this.MainHeader.ip_DFMFoffset & 0x4000) == 0) return false; else return true; }
            set
            {
                ushort temp = this.MainHeader.ip_DFMFoffset;
                if(value)
                    this.MainHeader.ip_DFMFoffset = (ushort)(this.MainHeader.ip_DFMFoffset | 0x4000);
                else
                    this.MainHeader.ip_DFMFoffset = (ushort)(this.MainHeader.ip_DFMFoffset & 0x3FFF);
                this.checksum = incrementalchecksum(temp, this.MainHeader.ip_DFMFoffset);
            }
        }

        public bool MF
        {
            get {if((this.MainHeader.ip_DFMFoffset & 0x2000) == 0) return false; else return true; }
            set
            {
                ushort temp = this.MainHeader.ip_DFMFoffset;
                if(value)
                    this.MainHeader.ip_DFMFoffset = (ushort)(this.MainHeader.ip_DFMFoffset | 0x2000);
                else
                    this.MainHeader.ip_DFMFoffset = (ushort)(this.MainHeader.ip_DFMFoffset & 0x5FFF);
                this.checksum = incrementalchecksum(temp, this.MainHeader.ip_DFMFoffset);
            }
        }

        public ushort Offset
        {
            get {return (ushort)(this.MainHeader.ip_DFMFoffset & 0x1FF);}
            set
            {
                ushort temp = this.MainHeader.ip_DFMFoffset;
                this.MainHeader.ip_DFMFoffset = (ushort)((this.MainHeader.ip_DFMFoffset & 0x7E00) + (value & 0x1FFF));
                this.checksum = incrementalchecksum(temp, this.MainHeader.ip_DFMFoffset);
            }
```

```
        }

        public byte TTL
        {
            get {return this.MainHeader.ip_ttl;}
            set
            {
                ushort temp = (ushort)((this.MainHeader.ip_ttl << 8) + this.MainHeader.ip_protocol);
                this.MainHeader.ip_ttl = value;
                this.checksum = incrementalchecksum(temp, (ushort)((this.MainHeader.ip_ttl << 8) + this.MainHeader.ip_protocol));
            }
        }

        public byte Protocol
        {
            get {return this.MainHeader.ip_protocol;}
            set
            {
                ushort temp = (ushort)((this.MainHeader.ip_ttl << 8) + this.MainHeader.ip_protocol);
                this.MainHeader.ip_protocol = value;
                this.checksum = incrementalchecksum(temp, (ushort)((this.MainHeader.ip_ttl << 8) + this.MainHeader.ip_protocol));
                //Could muck around updating the TCP checksum here but why bother since protocol != 6 doesn't have a TCP header anyway.
            }
        }

        private ushort checksum
        {
            get {return this.MainHeader.ip_checksum;}
            set {this.MainHeader.ip_checksum = value;}
        }

        public uint source
        {
            get {return this.MainHeader.ip_srcaddr;}
            set
            {
                ushort temp = (ushort)((this.MainHeader.ip_srcaddr & 0xFFFF0000) >> 16);
                ushort temp2 = (ushort)(this.MainHeader.ip_srcaddr & 0x0000FFFF);
                this.MainHeader.ip_srcaddr = value;
                this.checksum = incrementalchecksum(temp, (ushort)((this.MainHeader.ip_srcaddr & 0xFFFF0000) >> 16));
                this.checksum = incrementalchecksum(temp2, (ushort)(this.MainHeader.ip_srcaddr & 0x0000FFFF));
                this.tcpchecksum = incrementalTCPchecksum(temp, (
```

```
                        ushort)((this.MainHeader.ip_srcaddr & 0xFFFF0000) >> 16));
                    this.tcpchecksum = incrementalTCPchecksum(temp2, (ushort)(this.MainHeader.ip_srcaddr & 0x0000FFFF));
                }
            }

            public uint destination
            {
                get {return this.MainHeader.ip_destaddr;}
                set
                {
                    ushort temp = (ushort)((this.MainHeader.ip_destaddr & 0xFFFF0000) >> 16);
                    ushort temp2 = (ushort)(this.MainHeader.ip_destaddr & 0x0000FFFF);
                    this.MainHeader.ip_destaddr = value;
                    this.checksum = incrementalchecksum(temp, (ushort)((this.MainHeader.ip_destaddr & 0xFFFF0000) >> 16));
                    this.checksum = incrementalchecksum(temp2, (ushort)(this.MainHeader.ip_destaddr & 0x0000FFFF));
                    this.tcpchecksum = incrementalTCPchecksum(temp, (ushort)((this.MainHeader.ip_destaddr & 0xFFFF0000) >> 16));
                    this.tcpchecksum = incrementalTCPchecksum(temp2, (ushort)(this.MainHeader.ip_destaddr & 0x0000FFFF));
                }
            }

            public ushort tcpsource
            {
                get {return this.TCPHead.tcp_srcport;}
                set
                {
                    ushort temp = this.TCPHead.tcp_srcport;
                    this.TCPHead.tcp_srcport = value;
                    this.tcpchecksum = incrementalTCPchecksum(temp, this.TCPHead.tcp_srcport);
                }
            }

            public ushort tcpdestination
            {
                get {return this.TCPHead.tcp_destport;}
                set
                {
                    ushort temp = this.TCPHead.tcp_destport;
                    this.TCPHead.tcp_destport = value;
                    this.tcpchecksum = incrementalTCPchecksum(temp, this.TCPHead.tcp_destport);
                }
```

```
        }

        public uint tcpsequence
        {
            get {return this.TCPHead.tcp_sequence;}
            set
            {
                ushort temp = (ushort)((this.TCPHead.tcp_sequence & 0xFFFF0000) >> 16);
                ushort temp2 = (ushort)(this.TCPHead.tcp_sequence & 0x0000FFFF);
                this.TCPHead.tcp_sequence = value;
                this.tcpchecksum = incrementalTCPchecksum(temp, (ushort)((this.TCPHead.tcp_sequence & 0xFFFF0000) >> 16));
                this.tcpchecksum = incrementalTCPchecksum(temp2, (ushort)(this.TCPHead.tcp_sequence & 0x0000FFFF));
            }
        }

        public uint tcpacknowledgement
        {
            get {return this.TCPHead.tcp_acknowledgement;}
            set
            {
                ushort temp = (ushort)((this.TCPHead.tcp_acknowledgement & 0xFFFF0000) >> 16);
                ushort temp2 = (ushort)(this.TCPHead.tcp_acknowledgement & 0x0000FFFF);
                this.TCPHead.tcp_acknowledgement = value;
                this.tcpchecksum = incrementalTCPchecksum(temp, (ushort)((this.TCPHead.tcp_acknowledgement & 0xFFFF0000) >> 16));
                this.tcpchecksum = incrementalTCPchecksum(temp2, (ushort)(this.TCPHead.tcp_acknowledgement & 0x0000FFFF));
            }
        }

        public byte tcpheaderlength
        {
            get {return (byte)((this.TCPHead.tcp_headerlength & 0xF0) >> 4);}
            set
            {
                ushort temp = (ushort)((this.TCPHead.tcp_headerlength << 8) + this.TCPHead.tcp_flags);
                this.TCPHead.tcp_headerlength = (byte)((value & 0x0F) << 4);
                this.tcpchecksum = incrementalTCPchecksum(temp, (ushort)((this.TCPHead.tcp_headerlength << 8) + this.
```

```
                TCPHead.tcp_flags));
        }
    }

    public bool tcpUGR
    {
        get {if((this.TCPHead.tcp_flags & 0x20) ==
            0) return false; else return true; }
        set
        {
            ushort temp = (ushort)((this.TCPHead.
                tcp_headerlength << 8) + this.
                TCPHead.tcp_flags);
            if(value)
                this.TCPHead.tcp_flags = (byte)(
                    this.TCPHead.tcp_flags | 0
                    x20);
            else
                this.TCPHead.tcp_flags = (byte)(
                    this.TCPHead.tcp_flags & 0
                    x1F);
            this.tcpchecksum =
                incrementalTCPchecksum(temp, (
                ushort)((this.TCPHead.
                tcp_headerlength << 8) + this.
                TCPHead.tcp_flags));
        }
    }

    public bool tcpACK
    {
        get {if((this.TCPHead.tcp_flags & 0x10) ==
            0) return false; else return true; }
        set
        {
            ushort temp = (ushort)((this.TCPHead.
                tcp_headerlength << 8) + this.
                TCPHead.tcp_flags);
            if(value)
                this.TCPHead.tcp_flags = (byte)(
                    this.TCPHead.tcp_flags | 0
                    x10);
            else
                this.TCPHead.tcp_flags = (byte)(
                    this.TCPHead.tcp_flags & 0
                    x2F);
            this.tcpchecksum =
                incrementalTCPchecksum(temp, (
                ushort)((this.TCPHead.
                tcp_headerlength << 8) + this.
                TCPHead.tcp_flags));
        }
    }

    public bool tcpPSH
    {
        get {if((this.TCPHead.tcp_flags & 0x08) ==
            0) return false; else return true; }
        set
        {
            ushort temp = (ushort)((this.TCPHead.
                tcp_headerlength << 8) + this.
                TCPHead.tcp_flags);
```

```
                if(value)
                    this.TCPHead.tcp_flags = (byte)(
                        this.TCPHead.tcp_flags | 0
                        x08);
                else
                    this.TCPHead.tcp_flags = (byte)(
                        this.TCPHead.tcp_flags & 0
                        x37);
                this.tcpchecksum =
                    incrementalTCPchecksum(temp, (
                    ushort)((this.TCPHead.
                    tcp_headerlength << 8) + this.
                    TCPHead.tcp_flags));
            }
        }

        public bool tcpRST
        {
            get {if((this.TCPHead.tcp_flags & 0x04) ==
                0) return false; else return true; }
            set
            {
                ushort temp = (ushort)((this.TCPHead.
                    tcp_headerlength << 8) + this.
                    TCPHead.tcp_flags);
                if(value)
                    this.TCPHead.tcp_flags = (byte)(
                        this.TCPHead.tcp_flags | 0
                        x04);
                else
                    this.TCPHead.tcp_flags = (byte)(
                        this.TCPHead.tcp_flags & 0
                        x3B);
                this.tcpchecksum =
                    incrementalTCPchecksum(temp, (
                    ushort)((this.TCPHead.
                    tcp_headerlength << 8) + this.
                    TCPHead.tcp_flags));
            }
        }

        public bool tcpSYN
        {
            get {if((this.TCPHead.tcp_flags & 0x02) ==
                0) return false; else return true; }
            set
            {
                ushort temp = (ushort)((this.TCPHead.
                    tcp_headerlength << 8) + this.
                    TCPHead.tcp_flags);
                if(value)
                    this.TCPHead.tcp_flags = (byte)(
                        this.TCPHead.tcp_flags | 0
                        x02);
                else
                    this.TCPHead.tcp_flags = (byte)(
                        this.TCPHead.tcp_flags & 0
                        x3D);
                this.tcpchecksum =
                    incrementalTCPchecksum(temp, (
                    ushort)((this.TCPHead.
                    tcp_headerlength << 8) + this.
                    TCPHead.tcp_flags));
```

```
            }
        }

        public bool tcpFIN
        {
            get {if((this.TCPHead.tcp_flags & 0x01) == 0) return false; else return true; }
            set
            {
                ushort temp = (ushort)((this.TCPHead.tcp_headerlength << 8) + this.TCPHead.tcp_flags);
                if(value)
                    this.TCPHead.tcp_flags = (byte)(this.TCPHead.tcp_flags | 0x01);
                else
                    this.TCPHead.tcp_flags = (byte)(this.TCPHead.tcp_flags & 0x3E);
                this.tcpchecksum = incrementalTCPchecksum(temp, (ushort)((this.TCPHead.tcp_headerlength << 8) + this.TCPHead.tcp_flags));
            }
        }

        public ushort tcpwindow
        {
            get {return this.TCPHead.tcp_windowsize;}
            set
            {
                ushort temp = this.TCPHead.tcp_windowsize;
                this.TCPHead.tcp_windowsize = value;
                this.tcpchecksum = incrementalTCPchecksum(temp, this.TCPHead.tcp_windowsize);
            }
        }

        private ushort tcpchecksum
        {
            get {return this.TCPHead.tcp_checksum;}
            set {this.TCPHead.tcp_checksum = value;}
        }

        public ushort tcpurgent
        {
            get {return this.TCPHead.tcp_urgentpointer;}
            set
            {
                ushort temp = this.TCPHead.tcp_urgentpointer;
                this.TCPHead.tcp_urgentpointer = value;
                this.tcpchecksum = incrementalTCPchecksum(temp, this.TCPHead.tcp_urgentpointer);
            }
        }

        //private byte prottype;
```

```
        public IPHeader MainHeader; //Since I wrote methds
            for accessing this it should really be
            private.
        // But I don't have time to deal with the changes
            this requires or to write similar methods for
            the
        //TCP header.
        public TCPHeader TCPHead;
        public byte [] IPOptions;
        public byte [] TCPOptions;
        public byte [] RemainingData;
        //public int acalculatedchecksum;
        public UInt32 test1;
        public UInt32 test2;
    }

    public class RawSender
    {
        private bool error_in_send;
        private static int len_send_buf; //Max packet size
            for send.
        byte [] send_buf_bytes; //Data to send.
        private Socket sender = null; //Socket to send via
            .
        public ushort truecheck;
        public ushort reportedcheck;

        public RawSender() //Constructor
        {
            error_in_send=false;
            len_send_buf = 65535; //Can't send a larger
                packet. Windows would crash out.
            send_buf_bytes = new byte[len_send_buf];
        }

        public void StartSender(string IP, int port)
        {
            sender = new Socket(AddressFamily.
                InterNetwork, SocketType.Raw,
                ProtocolType.IP);
            sender.Blocking = false;
            sender.Bind(new IPEndPoint(IPAddress.Parse(
                IP), port));
            if (SetSenderOption()==false) error_in_send=
                true;
        }

        public void ShutdownSender()
        {
            if(sender != null)
            {
                sender.Shutdown(SocketShutdown.Both);
                sender.Close();
            }
        }

        private bool SetSenderOption()
        {
            bool ret_value = true;
            try //.NET Exception handeling
            {
                sender.SetSocketOption(
                    SocketOptionLevel.IP,
                    SocketOptionName.HeaderIncluded,
                    1);
```

```
                                sender.SetSocketOption(SocketOptionLevel.Socket, System.Net.Sockets.SocketOptionName.ReuseAddress, 1);
                                //sender.SetSocketOption(SocketOptionLevel.IP, SocketOptionName.SendBuffer, 100000);

                                //int FIONBIO = unchecked ((int)0x0004CB34);
                                //int FIONBIO = unchecked((int)2147772030);
                                //byte []IN = new byte[4]{1, 0, 0, 0};
                                //byte []OUT = new byte[4];
                                // See http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/wsaioctl_2.asp for details.
                                //int ret_code = sender.IOControl(FIONBIO, IN, OUT); //receive all IP packets on the network.
                                //ret_code = OUT[0] + OUT[1] + OUT[2] + OUT[3]; //Check that operation suceeded.
                                //if(ret_code != 0) ret_value = false; //If not return error.
                        }
                        catch(SocketException)
                        {
                                ret_value = false; //If any of the above caused an exception, return an error.
                        }
                        return ret_value;
                }

                public bool WasError
                {
                        get // Let main program check for errors.
                        {
                                return error_in_send;
                        }
                }

                public static UInt16 calcchecksum( Byte[] buffer, int size )
                {
                        Int32 cksum = 0;
                        int counter;
                        counter = 0;
                        while ( size > 0 )
                        {
                                UInt16 val = (ushort)((buffer[counter] << 8) + buffer[counter+1]);
                                cksum += Convert.ToInt32( val );
                                counter += 2;
                                size -= 2;
                        }
                        cksum = (cksum >> 16) + (cksum & 0xffff);
                        cksum += (cksum >> 16);
                        return (UInt16)(~cksum);
                }

```

```
        public UInt16 TCPChecksum( Byte[] buffer, int size
            , int IHL )
        {
            byte[] biggerbuffer;
            int length = size - (IHL * 4) + 12;
            //this.error_occurred = false;
            if (length%2 == 1)
            {
                //this.error_occurred = true;
                length++;
            }
            biggerbuffer = new byte[length];
            //biggerbuffer[0] = buffer[12];
            Array.Copy(buffer, 12, biggerbuffer, 0, 8);
            biggerbuffer[8] = 0;
            biggerbuffer[9] = 6;
            biggerbuffer[10] = (byte)(((size - (IHL * 4)
                ) & 0xFF00) >> 8);
            biggerbuffer[11] = (byte)((size - (IHL * 4))
                & 0x00FF);
            Array.Copy(buffer, IHL*4, biggerbuffer, 12,
                size - (IHL * 4));
            Int32 cksum = 0;
            int counter;
            counter = 0;
            while ( length > 0 )
            {
                UInt16 val = (ushort)((biggerbuffer[
                    counter] << 8) + biggerbuffer[
                    counter+1]);
                cksum += Convert.ToInt32( val );
                counter += 2;
                length -= 2;
            }
            cksum = (cksum >> 16) + (cksum & 0xffff);
            cksum += (cksum >> 16);
            return (UInt16)(~cksum);
        }

        public void send(PacketArgs args)
            //(IPHeader MainHeader)
        {
            int messagelength;
            ushort mychecksum;
            //int test = MainHeader.ip_verIHL;
            //listBox1.Items.Add(source + "        \t
                    " + destination);
            send_buf_bytes[0] = args.MainHeader.
                ip_verIHL;
            send_buf_bytes[1] = args.MainHeader.ip_tos;
            send_buf_bytes[2] = (byte)((args.MainHeader.
                ip_totallength & 0xFF00) >> 8);
            send_buf_bytes[3] = (byte)(args.MainHeader.
                ip_totallength & 0x00FF);
            send_buf_bytes[4] = (byte)((args.MainHeader.
                ip_id & 0xFF00) >> 8);
            send_buf_bytes[5] = (byte)(args.MainHeader.
                ip_id & 0x00FF);
            send_buf_bytes[6] = (byte)((args.MainHeader.
                ip_DFMFoffset & 0xFF00) >> 8);
            send_buf_bytes[7] = (byte)(args.MainHeader.
                ip_DFMFoffset & 0x00FF);
```

```
send_buf_bytes[8] = args.MainHeader.ip_ttl;
send_buf_bytes[9] = args.MainHeader.ip_protocol;
// Don't calculate checksum yet. We want to check both ways until we are certain it works.
send_buf_bytes[10] = 0;
send_buf_bytes[11] = 0;
send_buf_bytes[12] = (byte)((args.MainHeader.ip_srcaddr & 0xFF000000) >> 24);
send_buf_bytes[13] = (byte)((args.MainHeader.ip_srcaddr & 0x00FF0000) >> 16);
send_buf_bytes[14] = (byte)((args.MainHeader.ip_srcaddr & 0x0000FF00) >> 8);
send_buf_bytes[15] = (byte)(args.MainHeader.ip_srcaddr & 0x000000FF);
send_buf_bytes[16] = (byte)((args.MainHeader.ip_destaddr & 0xFF000000) >> 24);
send_buf_bytes[17] = (byte)((args.MainHeader.ip_destaddr & 0x00FF0000) >> 16);
send_buf_bytes[18] = (byte)((args.MainHeader.ip_destaddr & 0x0000FF00) >> 8);
send_buf_bytes[19] = (byte)(args.MainHeader.ip_destaddr & 0x000000FF);
//this.MainHeader.ip_srcaddr = (uint)((buf[12] << 24) + (buf[13] << 16) + (buf[14] << 8) + buf[15]);
int IHL = (args.MainHeader.ip_verIHL & 0x0F);
if (IHL > 5)
{
    Array.Copy(args.IPOptions, 0, send_buf_bytes, 20, (IHL - 5) * 4);
}
mychecksum = calcchecksum(send_buf_bytes, (send_buf_bytes[0] & 0x0F)*4);
send_buf_bytes[10] = (byte)((args.MainHeader.ip_checksum & 0xFF00) >> 8);
send_buf_bytes[11] = (byte)(args.MainHeader.ip_checksum & 0x00FF);
if (args.Protocol == 6)
{
    send_buf_bytes[args.IHL * 4] = (byte)((args.TCPHead.tcp_srcport & 0xFF00) >> 8);
    send_buf_bytes[args.IHL * 4 + 1] = (byte)(args.TCPHead.tcp_srcport & 0x00FF);
    send_buf_bytes[args.IHL * 4 + 2] = (byte)((args.TCPHead.tcp_destport & 0xFF00) >> 8);
    send_buf_bytes[args.IHL * 4 + 3] = (byte)(args.TCPHead.tcp_destport & 0x00FF);
    send_buf_bytes[args.IHL * 4 + 4] = (byte)((args.TCPHead.tcp_sequence & 0xFF000000) >> 24);
    send_buf_bytes[args.IHL * 4 + 5] = (byte)((args.TCPHead.tcp_sequence & 0x00FF0000) >> 16);
```

```
                    send_buf_bytes[args.IHL * 4 + 6] = (byte)((args.TCPHead.tcp_sequence & 0x0000FF00) >> 8);
                    send_buf_bytes[args.IHL * 4 + 7] = (byte)(args.TCPHead.tcp_sequence & 0x000000FF);
                    send_buf_bytes[args.IHL * 4 + 8] = (byte)((args.TCPHead.tcp_acknowledgement & 0xFF000000) >> 24);
                    send_buf_bytes[args.IHL * 4 + 9] = (byte)((args.TCPHead.tcp_acknowledgement & 0x00FF0000) >> 16);
                    send_buf_bytes[args.IHL * 4 + 10] = (byte)((args.TCPHead.tcp_acknowledgement & 0x0000FF00) >> 8);
                    send_buf_bytes[args.IHL * 4 + 11] = (byte)(args.TCPHead.tcp_acknowledgement & 0x000000FF);
                    send_buf_bytes[args.IHL * 4 + 12] = args.TCPHead.tcp_headerlength;
                    send_buf_bytes[args.IHL * 4 + 13] = args.TCPHead.tcp_flags;
                    send_buf_bytes[args.IHL * 4 + 14] = (byte)((args.TCPHead.tcp_windowsize & 0xFF00) >> 8);
                    send_buf_bytes[args.IHL * 4 + 15] = (byte)(args.TCPHead.tcp_windowsize & 0x00FF);
                    send_buf_bytes[args.IHL * 4 + 16] = 0;
                    send_buf_bytes[args.IHL * 4 + 17] = 0;
                    send_buf_bytes[args.IHL * 4 + 18] = (byte)((args.TCPHead.tcp_urgentpointer & 0xFF00) >> 8);
                    send_buf_bytes[args.IHL * 4 + 19] = (byte)(args.TCPHead.tcp_urgentpointer & 0x00FF);
                    if (args.tcpheaderlength > 5)
                    {
                        Array.Copy(args.TCPOptions, 0, send_buf_bytes, (args.IHL * 4) + 20, (args.tcpheaderlength - 5) * 4);
                    }
                    messagelength = args.totallength - (args.IHL * 4) - (args.tcpheaderlength * 4);
                    Array.Copy(args.RemainingData, 0, send_buf_bytes, (args.IHL * 4) + (args.tcpheaderlength * 4), messagelength);
                }
                else
                {
                    messagelength = args.totallength - (args.IHL * 4);
                    Array.Copy(args.RemainingData, 0, send_buf_bytes, (args.IHL * 4), messagelength);
```

```
                }
                //sender.SendTo(send_buf_bytes, args.totallength, System.Net.Sockets.SocketFlags.None, new IPEndPoint(args.destination, 0));
                if (mychecksum == args.MainHeader.ip_checksum)
                {
                    if (args.Protocol == 6)
                    {
                        mychecksum = TCPChecksum(send_buf_bytes, args.totallength, args.IHL);
                        error_in_send = false;
                        send_buf_bytes[args.IHL * 4 + 16] = (byte)((args.TCPHead.tcp_checksum & 0xFF00) >> 8);
                        send_buf_bytes[args.IHL * 4 + 17] = (byte)(args.TCPHead.tcp_checksum & 0x00FF);
                        this.truecheck = mychecksum;
                        this.reportedcheck = args.TCPHead.tcp_checksum;
                        sender.Blocking = false;
                        //sender.BeginSendTo
                        sender.Connect(new IPEndPoint((long)args.destination, args.tcpdestination));
                        while (!sender.Connected)
                        {
                        }
                        /*while (sender.Blocking)
                            {
                            }*/
                        sender.Send(send_buf_bytes, 0, args.totallength, System.Net.Sockets.SocketFlags.None);
                        //sender.SendTo(send_buf_bytes, args.totallength, System.Net.Sockets.SocketFlags.None, new IPEndPoint(args.destination, args.tcpdestination));
                    }
                    else
                    {
                        this.truecheck = 0;
                    }
                }
                else
                {
                    error_in_send = true;
                }
            }

        }

        public class RawSocket
        {
            private bool error_occurred;
            public bool KeepRunning; //Keep recieving packets?
            private static int len_receive_buf; //Size of recieve buffer.
```

```
        byte [] receive_buf_bytes; //Buffer for packets.
        private Socket socket = null; //Socket for recieving.
        public UInt16 checksum1;
        public UInt16 checksum2;

        public RawSocket() //Constructor
        {
            error_occurred=false;
            len_receive_buf = 65535; // Absolute maximum IP packet size.
            // Be careful of Buffer Overruns. WIndows won't recieve a packet that violates this number.
            receive_buf_bytes = new byte[len_receive_buf];
        }

        public void StartSocket(string IP, int port, bool promiscuous)
        {
            socket = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.IP);
            socket.Blocking = false;
            socket.Bind(new IPEndPoint(IPAddress.Parse(IP), port));
            //socket.Bind(new IPEndPoint(IPAddress.Parse("202.173.149.32"), 0));
            if (SetSocketOption(promiscuous)==false) error_occurred=true;
        }

        public void ShutdownSocket()
        {
            if(socket != null)
            {
                socket.Shutdown(SocketShutdown.Both);
                socket.Close();
            }
        }

        private bool SetSocketOption(bool promiscuous)
        {
            bool ret_value = true;
            try //.NET Exception handeling
            {
                socket.SetSocketOption(SocketOptionLevel.IP, SocketOptionName.HeaderIncluded, 1);
                if (promiscuous)
                {
                    byte []IN = new byte[4]{1, 0, 0, 0};
                    byte []OUT = new byte[4];
                    int SIO_RCVALL = unchecked((int)0x98000001); // Control code for SIO_RCVALL documented on MSDN.
                    // See http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/wsaioctl_2.asp for
```

```
                                        details.
                                int ret_code = socket.IOControl(SIO_RCVALL, IN, OUT); //receive all IP packets on the network.
                                ret_code = OUT[0] + OUT[1] + OUT[2] + OUT[3]; //Check that operation suceeded.
                                if(ret_code != 0) ret_value = false; //If not return error.
                        }
                }
                catch(SocketException)
                {
                        ret_value = false; //If any of the above caused an exception, return an error.
                }
                return ret_value;
        }

        public bool ErrorOccurred
        {
                get // Let main program check for errors.
                {
                        return error_occurred;
                }
        }

        public static UInt16 calcchecksum( Byte[] buffer, int size )
        {
                Int32 cksum = 0;
                int counter;
                counter = 0;
                while ( size > 0 )
                {
                        UInt16 val = (ushort)((buffer[counter] << 8) + buffer[counter+1]);
                        cksum += Convert.ToInt32( val );
                        counter += 2;
                        size -= 2;
                }
                cksum = (cksum >> 16) + (cksum & 0xffff);
                cksum += (cksum >> 16);
                return (UInt16)(~cksum);
        }

        public UInt16 TCPChecksum( Byte[] buffer, int size, int IHL )
        {
                byte[] biggerbuffer;
                int length = size - (IHL * 4) + 12;
                //this.error_occurred = false;
                if (length%2 == 1)
                {
                        //this.error_occurred = true;
                        length++;
                }
                biggerbuffer = new byte[length];
                //biggerbuffer[0] = buffer[12];
                Array.Copy(buffer, 12, biggerbuffer, 0, 8);
                biggerbuffer[8] = 0;
```

```
                    biggerbuffer[9] = 6;
                    biggerbuffer[10] = (byte)(((size - (IHL * 4)
                        ) & 0xFF00) >> 8);
                    biggerbuffer[11] = (byte)((size - (IHL * 4))
                        & 0x00FF);
                    Array.Copy(buffer, IHL*4, biggerbuffer, 12,
                        size - (IHL * 4));
                    Int32 cksum = 0;
                    int counter;
                    counter = 0;
                    while ( length > 0 )
                    {
                            UInt16 val = (ushort)((biggerbuffer[
                                counter] << 8) + biggerbuffer[
                                counter+1]);
                            cksum += Convert.ToInt32( val );
                            counter += 2;
                            length -= 2;
                    }
                    cksum = (cksum >> 16) + (cksum & 0xffff);
                    cksum += (cksum >> 16);
                    return (UInt16)(~cksum);
            }

            private void Receive(byte [] buf, int len)
            {
                    //byte temp_protocol=0;
                    //uint temp_version=0;
                    //uint temp_ip_srcaddr=0;
                    //uint temp_ip_destaddr=0;
                    //short temp_srcport=0;
                    //short temp_dstport=0;
                    //IPAddress temp_ip;

                    UInt16 RecievedChecksum;
                    byte temp1, temp2;
                    RecievedChecksum = (ushort)((buf[10] << 8) +
                        buf[11]);
                    temp1 = buf[10];
                    temp2 = buf[11];
                    buf[10] = 0;
                    buf[11] = 0;
                    if (calcchecksum(buf, (buf[0] & 0x0F)*4) ==
                        RecievedChecksum)
                    {
                            buf[10] = temp1;
                            buf[11] = temp2;
                            int IHL = (buf[0] & 0x0F);
                            if (buf[9] == 6) //This causes us to
                                only recieve TCP packets which is
                                fine for now.
                            {
                                    temp1 = buf[IHL * 4 + 16];
                                    temp2 = buf[IHL * 4 + 17];
                                    RecievedChecksum = (ushort)((
                                        temp1 << 8) + temp2);
                                    buf[IHL * 4 + 16] = 0;
                                    buf[IHL * 4 + 17] = 0;
                                    if (TCPChecksum(buf, (ushort)((
                                        buf[2] << 8) + buf[3]), IHL)
                                         == RecievedChecksum)
                                    {
```

```
                        buf[IHL * 4 + 16] = temp1;
                        buf[IHL * 4 + 17] = temp2;
                        PacketArgs e;
                        e=new PacketArgs(buf);
                        OnPacketArrival(e);
                    }
                    checksum1=TCPChecksum(buf, (
                        ushort)((buf[2] << 8) + buf
                        [3]), IHL);
                    checksum2 = RecievedChecksum;
                }
                //PacketArgs e;

                //fixed(byte *fixed_buf = buf)
                //{
                //IPHeader * head = (IPHeader *)
                    fixed_buf; // Assign IP Header
                    from recieve buffer.
                //e=new PacketArgs(buf);
                // head->ip_protocol, ((uint)(head->
                    ip_verIHL & 0x0F) << 2) * 4
                //Array.Copy(buf, 0, e.MainHeader, 0,
                    20);

                //e.MainHeader = head;
                //e.SubHeader = fixed_buf[((uint)(head
                    ->ip_verIHL & 0x0F) << 2) * 4];
                /*e.HeaderLength=(uint)(head->
                    ip_verIHL & 0x0F) << 2; //Header
                    Length from IHL (bits 5-8).

                    temp_protocol = head->
                        ip_protocol;
                    switch(temp_protocol)
                    {
                        case 1: e.Protocol="ICMP
                            :";      break;
                        case 2: e.Protocol="IGMP
                            :";      break; //Don't
                             need IGMP in NAT
                        case 6: e.Protocol="TCP:";
                                break;
                        case 17: e.Protocol="UDP
                            :";      break;
                        default: e.Protocol= "
                            UNKNOWN"; break;
                        // See http://www.iana.org
                            /assignments/protocol-
                            numbers for details.
                    } // Use this in future version
                        to properly decode TCP or
                        ICMP headers.

                    temp_version =(uint)(head->
                        ip_verIHL & 0xF0) >> 4; //
                        Version from verIHL (bits
                        1-4).
                    e.IPVersion = temp_version.
                        ToString();

                    temp_ip_srcaddr = head->
                        ip_srcaddr; //Decode IP
                        addresses to strings.
                    temp_ip_destaddr = head->
                        ip_destaddr;
```

```
                    temp_ip = new IPAddress(temp_ip_srcaddr);
                    e.OriginationAddress =temp_ip.ToString();
                    temp_ip = new IPAddress(temp_ip_destaddr);
                    e.DestinationAddress = temp_ip.ToString();

                    // This is a very bad idea as it defeats the purpose of seperate IP and TCP layers.
                    // Will reprogram it later when support is added for total recognition of TCP header.
                    // Would cause problems if IP Options were used. (Could use e.HeaderLength to fix)
                    temp_srcport = *(short *)&fixed_buf[e.HeaderLength];
                    temp_dstport = *(short *)&fixed_buf[e.HeaderLength+2];
                    e.OriginationPort=IPAddress.NetworkToHostOrder(temp_srcport).ToString();
                    e.DestinationPort=IPAddress.NetworkToHostOrder(temp_dstport).ToString();

                    e.PacketLength =(uint)len;
                    e.MessageLength =(uint)len - e.HeaderLength;

                    e.ReceiveBuffer=buf;
                    Array.Copy(buf,0,e.IPHeaderBuffer,0,(int)e.HeaderLength);
                    Array.Copy(buf,(int)e.HeaderLength,e.MessageBuffer,0,(int)e.MessageLength); //Copy remaining data to message buffers.*/
                //}

                //OnPacketArrival(e); //Call processing functions.
            }
        }

        public void Run()
        {
            IAsyncResult ar = socket.BeginReceive(receive_buf_bytes, 0, len_receive_buf, SocketFlags.None, new AsyncCallback(CallReceive), this);
        }

        private void CallReceive(IAsyncResult ar)
        {
            int received_bytes;
            received_bytes = socket.EndReceive(ar);
            Receive(receive_buf_bytes, received_bytes);
            if (KeepRunning) Run(); //Keep recieving more packets.
        }
```

```


            public delegate void PacketArrivedEventHandler(
                Object sender, PacketArgs args);

            public event PacketArrivedEventHandler
                PacketArrival;

            protected virtual void OnPacketArrival(PacketArgs
                e)
            {
                if (PacketArrival != null)
                {
                    PacketArrival(this, e);
                }
            }
        }
}
```

B.3 BidirHashtable.cs

```
using System;
using System.Collections;
//using System.Runtime.Serialization;

namespace System.Extended.Collections
{
	/// <summary>
	/// BidirHashtable is a simple, bidirectional data structure
	/// designed around Hashtables and accessed like a more robust Hashtable.
	/// Internally it just contains two hashtables:
	/// one maps from key to value, the other maps from value to key.
	/// Lookup in either direction is quick;
	/// changes take twice as long since two Hashtables are accessed.
	/// Forward lookup is just through the [] as in Hashtable.
	/// Reverse lookup is through ReverseLookup().
	/// Adding and setting elements is done with forward syntax identical to
	/// in Hashtable, but both internal Hashtables are affected.
	/// </summary>
	public class BidirHashtable : IDictionary, ICollection, IEnumerable,
		ICloneable
	{
		private Hashtable m_htFwd = null;
		private Hashtable m_htBkwd = null;

		/// <summary>
		/// Just create a Two-Way Hash Table.
		/// </summary>
		public BidirHashtable()
		{
			m_htFwd = new Hashtable();
			m_htBkwd = new Hashtable();
		}

		/// <summary>
		/// Somewhat similar to a Copy Constructor in C++
		/// </summary>
		public BidirHashtable(IDictionary dict)
		{
			m_htFwd = new Hashtable();
			m_htBkwd = new Hashtable();

			foreach( object key in dict.Keys )
			{
				this[key] = dict[key];
			}
		}

		/// <summary>
		/// Use an existing Hash Table and map the reverse lookups.
		/// </summary>
		private BidirHashtable(Hashtable ht, byte bytDummyIndicatesAttach)
		{
```

```
            m_htFwd = ht;
            m_htBkwd = new Hashtable();

            foreach( object key in ht.Keys )
            {
                m_htBkwd[ht[key]] = key;
            }
        }

        public int Count {get { return m_htFwd.Count;  } }
        public bool IsSynchronized {get { return m_htFwd.IsSynchronized; } }
        public object SyncRoot {get { return m_htFwd.SyncRoot; } }
        public void CopyTo(
            Array array,
            int index
            )
        {
            m_htFwd.CopyTo( array, index );
        }

        public void CopyValuesTo(
            Array array,
            int index
            )
        {
            m_htBkwd.CopyTo( array, index );
        }

        public void Add( object key, object val )
        {
            m_htFwd.Add( key, val );
            m_htBkwd.Add( val, key );
        }

        public void Remove( object key )
        {
            object val = m_htFwd[key];
            m_htFwd.Remove( key );
            m_htBkwd.Remove( val );
        }

        public void Clear()
        {
            m_htFwd.Clear();
            m_htBkwd.Clear();
        }

        /// <summary>
        /// Forward lookup or set.
        /// </summary>
        public object this[ object key ]
        {
            get {  return m_htFwd[key];  }
            set
            {
                if ( m_htFwd.ContainsKey(key) )
                {
                    m_htBkwd.Remove( m_htFwd[key] );
                }
                m_htFwd[key] = value;
                m_htBkwd[value] = key;
            }
```

```
        }

        /// <summary>
        /// Reverse Lookup works at normal hashtable speeds.
        /// </summary>
        public object ReverseLookup( object val )
        {
            return m_htBkwd[val];
        }

        public bool IsFixedSize
        {
            get {  return m_htFwd.IsFixedSize;  }
        }

        public bool IsReadOnly
        {
            get {  return m_htFwd.IsReadOnly;  }
        }

        /// <summary>
        /// Do Not Use this, should be made private in future.
        /// </summary>
        public ICollection Keys
        {
            get {  return m_htFwd.Keys;  }
        }

        /// <summary>
        /// Do Not Use this, should be made private in future.
        /// </summary>
        public ICollection Values
        {
            get {  return m_htFwd.Values;  }
        }

        public bool Contains( object key )
        {
            return m_htFwd.Contains( key );
        }

        public bool ContainsValue( object val )
        {
            return m_htBkwd.Contains( val );
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return m_htFwd.GetEnumerator();
        }

        IDictionaryEnumerator IDictionary.GetEnumerator()
        {
            return m_htFwd.GetEnumerator();
        }

        public object Clone()
        {
            BidirHashtable bh = new BidirHashtable();
            bh.m_htFwd = (Hashtable) m_htFwd.Clone();
            bh.m_htBkwd = (Hashtable) m_htBkwd.Clone();
            return bh;
```

```
        }

        #region Explicit conversion to/from Hashtable
        public static explicit operator BidirHashtable(
            Hashtable ht)
        {
            return new BidirHashtable( ht );
        }

        public static explicit operator Hashtable(
            BidirHashtable bd)
        {
            return (Hashtable) bd.m_htFwd.Clone();
        }
        #endregion

        #region Access to private Hashtables
        /// <summary>
        /// Gives direct access for debugging only.
        /// </summary>
        public Hashtable ForwardHashtable
        {
            get { return m_htFwd; }
        }

        /// <summary>
        /// Gives direct access for debugging only.
        /// </summary>
        public Hashtable BackwardHashtable
        {
            get { return m_htBkwd; }
        }

        /// <summary>
        /// Make a copy to change without causing bugs in
            this
        /// two-way hash table.
        /// </summary>
        public Hashtable BackwardHashtableClone
        {
            get { return (Hashtable) m_htBkwd.Clone();
                }
        }
        #endregion

        #region Attach and ReverseDirection
        /// <summary>
        /// Reverse the hash table. Lookups are in
            opposite directions.
        /// </summary>
        public void ReverseDirection()
        {
            Hashtable htTemp = m_htFwd;
            m_htFwd = m_htBkwd;
            m_htBkwd = htTemp;
        }

        public static BidirHashtable Attach(Hashtable ht)
        {
            return new BidirHashtable( ht, (byte) 0 );
        }
        #endregion
    }
}
```

B.4 ProjectInstaller.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Configuration.Install;
using Microsoft.Win32;
using System.Security;
using System.IO;
using System.Windows.Forms;
using System.ServiceProcess;
using System.Diagnostics;

namespace NAT_Service
{
        /// <summary>
        /// Summary description for ProjectInstaller.
        /// </summary>
        [RunInstaller(true)]
        public class ProjectInstaller : System.Configuration.
            Install.Installer
        {
                private System.ServiceProcess.
                    ServiceProcessInstaller
                    serviceProcessInstaller1;
                private System.ServiceProcess.ServiceInstaller
                    serviceInstaller1;

                /// <summary>
                /// Required designer variable.
                /// </summary>
                //private System.ComponentModel.Container
                    components = null;
                private EventLog eventLog;

                public ProjectInstaller()
                {
                        // This call is required by the Designer.
                        InitializeComponent();
                        eventLog = new EventLog();

                        // TODO: Add any initialization after the
                            InitComponent call
                }

                /// <summary>
                /// override the install method to set up the
                    information.
                /// all thats create here is a registry key. It
                    should be noted that this function
                /// can't be debugged so catch all possible
                    exceptions
                /// </summary>
                /// <param name="iInstallData"></param>
                public override void Install( IDictionary
                    iInstallData )
                {
                        try
                        {

                                /// must call base class install first
                                base.Install( iInstallData );

                                /// just create the key the gui part
                                    of the code will set it
```

```
                RegistryKey reg = Registry.LocalMachine.OpenSubKey( "Software", true );
                if( reg == null )
                {
                    eventLog.WriteEntry( "Error trying to install 'USQ NAT Project'" );
                    return;
                }

                RegistryKey scheduleKey = reg.CreateSubKey( "NATUSQProj" );

                if( scheduleKey == null )
                {
                    eventLog.WriteEntry( "Error trying to install 'USQ NAT Project'" );
                    return;
                }

                reg.Close();
            }
            catch( ArgumentNullException argNullExp )
            {
                eventLog.WriteEntry( "Error with the argument subkey " + argNullExp.Message );
            }
            catch( SecurityException secExp )
            {
                eventLog.WriteEntry( "Error the user does not have access permission " + secExp.Message );
            }
            catch( IOException ioExp )
            {
                eventLog.WriteEntry( "Error the registry key is closed " + ioExp.Message );
            }
            catch( UnauthorizedAccessException unExp )
            {
                eventLog.WriteEntry( "Error the user does not have access permission " + unExp.Message );
            }
            catch( ArgumentException argExp )
            {
                eventLog.WriteEntry( "Error in the install data format " + argExp.Message );
            }
            catch( Exception exp )
            {
                eventLog.WriteEntry( "A problem occured with the install " + exp.Message );
            }

            return;
        }
```

```
        /// <summary>
        /// override the uninstall method and remove the registry key
        /// </summary>
        /// <param name="iInstallData"></param>
        public override void Uninstall( IDictionary iInstallData )
        {
            try
            {
                if( iInstallData == null )
                {
                    eventLog.WriteEntry( "Error unable to uninstall the application 'USQ NAT Project'" );
                }
                else
                {
                    base.Uninstall( iInstallData );

                    Registry.LocalMachine.OpenSubKey( "Software", true ).DeleteSubKeyTree( "NATUSQProj" );

                }
            }
            catch( ArgumentException argExp )
            {
                MessageBox.Show( "Error in the install data format " + argExp.Message );
            }
            catch( InstallException instExp )
            {
                MessageBox.Show( "A problem occurred with the install " + instExp.Message );
            }

            return;

        }

        #region Component Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.serviceProcessInstaller1 = new System.ServiceProcess.ServiceProcessInstaller();
            this.serviceInstaller1 = new System.ServiceProcess.ServiceInstaller();
            //
            // serviceProcessInstaller1
            //
            this.serviceProcessInstaller1.Account = System.ServiceProcess.ServiceAccount.LocalSystem;
```

```
                        this.serviceProcessInstaller1.Password =
                            null;
                        this.serviceProcessInstaller1.Username =
                            null;
                        //
                        // serviceInstaller1
                        //
                        this.serviceInstaller1.ServiceName = "
                            SchedulerService";
                        this.serviceInstaller1.StartType = System.
                            ServiceProcess.ServiceStartMode.
                            Automatic;
                        //
                        // ProjectInstaller
                        //
                        this.Installers.AddRange(new System.
                            Configuration.Install.Installer[] {



                }
                #endregion
        }
}
```

B.5 NATControl.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using Microsoft.Win32;
using System.Data;
using System.Net;

namespace NAT_Settings_Application
{
	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	public class NATSetup : System.Windows.Forms.Form
	{
		private System.Windows.Forms.Label Internal;
		private System.Windows.Forms.Label label7;
		private System.Windows.Forms.Label label8;
		private System.Windows.Forms.Label label9;
		private System.Windows.Forms.TextBox InternalSubnet4;
		private System.Windows.Forms.TextBox InternalSubnet3;
		private System.Windows.Forms.TextBox InternalSubnet2;
		private System.Windows.Forms.TextBox InternalSubnet1;
		private System.Windows.Forms.Label label10;
		private System.Windows.Forms.Label label11;
		private System.Windows.Forms.GroupBox groupBox1;
		private System.Windows.Forms.GroupBox groupBox2;
		private System.Windows.Forms.ComboBox ExternalIP;
		private System.Windows.Forms.ComboBox InternalIP;
		private System.Windows.Forms.Button NATOkButton;
		private System.Windows.Forms.Button NATCancelButton;
		private System.Windows.Forms.Label label1;
		private System.Windows.Forms.TextBox NumberofPorts;
		/// <summary>
		/// Required designer variable.
		/// </summary>
		private System.ComponentModel.Container components = null;

		public NATSetup()
		{
			//
			// Required for Windows Form Designer support
			//
			InitializeComponent();

			//
			// TODO: Add any constructor code after InitializeComponent call
			//
		}

		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
```

```
		{
			if( disposing )
			{
				if (components != null)
				{
					components.Dispose();
				}
			}
			base.Dispose( disposing );
		}

		#region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			this.Internal = new System.Windows.Forms.Label();
			this.label7 = new System.Windows.Forms.Label();
			this.label8 = new System.Windows.Forms.Label();
			this.label9 = new System.Windows.Forms.Label();
			this.InternalSubnet4 = new System.Windows.Forms.TextBox();
			this.InternalSubnet3 = new System.Windows.Forms.TextBox();
			this.InternalSubnet2 = new System.Windows.Forms.TextBox();
			this.InternalSubnet1 = new System.Windows.Forms.TextBox();
			this.label10 = new System.Windows.Forms.Label();
			this.label11 = new System.Windows.Forms.Label();
			this.groupBox1 = new System.Windows.Forms.GroupBox();
			this.InternalIP = new System.Windows.Forms.ComboBox();
			this.groupBox2 = new System.Windows.Forms.GroupBox();
			this.ExternalIP = new System.Windows.Forms.ComboBox();
			this.NATOkButton = new System.Windows.Forms.Button();
			this.NATCancelButton = new System.Windows.Forms.Button();
			this.label1 = new System.Windows.Forms.Label();
			this.NumberofPorts = new System.Windows.Forms.TextBox();
			this.groupBox1.SuspendLayout();
			this.groupBox2.SuspendLayout();
			this.SuspendLayout();
			//
			// Internal
			//
```

```
            this.Internal.Location = new System.Drawing.
                Point(32, 32);
            this.Internal.Name = "Internal";
            this.Internal.Size = new System.Drawing.Size
                (64, 16);
            this.Internal.TabIndex = 14;
            this.Internal.Text = "IP Address";
            this.Internal.TextAlign = System.Drawing.
                ContentAlignment.MiddleRight;
            //
            // label7
            //
            this.label7.Location = new System.Drawing.
                Point(192, 72);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size
                (8, 16);
            this.label7.TabIndex = 21;
            this.label7.Text = ".";
            //
            // label8
            //
            this.label8.Location = new System.Drawing.
                Point(160, 72);
            this.label8.Name = "label8";
            this.label8.Size = new System.Drawing.Size
                (8, 16);
            this.label8.TabIndex = 20;
            this.label8.Text = ".";
            //
            // label9
            //
            this.label9.Location = new System.Drawing.
                Point(128, 72);
            this.label9.Name = "label9";
            this.label9.Size = new System.Drawing.Size
                (8, 16);
            this.label9.TabIndex = 19;
            this.label9.Text = ".";
            //
            // InternalSubnet4
            //
            this.InternalSubnet4.Location = new System.
                Drawing.Point(200, 72);
            this.InternalSubnet4.MaxLength = 3;
            this.InternalSubnet4.Name = "InternalSubnet4
                ";
            this.InternalSubnet4.Size = new System.
                Drawing.Size(24, 20);
            this.InternalSubnet4.TabIndex = 18;
            this.InternalSubnet4.Text = "";
            //
            // InternalSubnet3
            //
            this.InternalSubnet3.Location = new System.
                Drawing.Point(168, 72);
            this.InternalSubnet3.MaxLength = 3;
            this.InternalSubnet3.Name = "InternalSubnet3
                ";
            this.InternalSubnet3.Size = new System.
                Drawing.Size(24, 20);
            this.InternalSubnet3.TabIndex = 17;
            this.InternalSubnet3.Text = "";
            //
            // InternalSubnet2
```

```
            //
            this.InternalSubnet2.Location = new System.Drawing.Point(136, 72);
            this.InternalSubnet2.MaxLength = 3;
            this.InternalSubnet2.Name = "InternalSubnet2";
            this.InternalSubnet2.Size = new System.Drawing.Size(24, 20);
            this.InternalSubnet2.TabIndex = 16;
            this.InternalSubnet2.Text = "";
            //
            // InternalSubnet1
            //
            this.InternalSubnet1.Location = new System.Drawing.Point(104, 72);
            this.InternalSubnet1.MaxLength = 3;
            this.InternalSubnet1.Name = "InternalSubnet1";
            this.InternalSubnet1.Size = new System.Drawing.Size(24, 20);
            this.InternalSubnet1.TabIndex = 15;
            this.InternalSubnet1.Text = "";
            //
            // label10
            //
            this.label10.Location = new System.Drawing.Point(24, 72);
            this.label10.Name = "label10";
            this.label10.Size = new System.Drawing.Size(72, 16);
            this.label10.TabIndex = 22;
            this.label10.Text = "Subnet Mask";
            this.label10.TextAlign = System.Drawing.ContentAlignment.MiddleRight;
            //
            // label11
            //
            this.label11.Location = new System.Drawing.Point(32, 32);
            this.label11.Name = "label11";
            this.label11.Size = new System.Drawing.Size(64, 16);
            this.label11.TabIndex = 23;
            this.label11.Text = "IP Address";
            this.label11.TextAlign = System.Drawing.ContentAlignment.MiddleRight;
            //
            // groupBox1
            //
            this.groupBox1.Controls.AddRange(new System.Windows.Forms.Control[] {
                this.InternalIP,
                this.label7,
                this.label8,
                this.label9,
                this.InternalSubnet3,
                this.InternalSubnet2,
                this.InternalSubnet4,
                this.label10,
                this.InternalSubnet1,
                this.Internal});
            this.groupBox1.Location = new System.Drawing.Point(56, 24);
            this.groupBox1.Name = "groupBox1";
```

```
                this.groupBox1.Size = new System.Drawing.Size(264, 104);
                this.groupBox1.TabIndex = 32;
                this.groupBox1.TabStop = false;
                this.groupBox1.Text = "Internal Interface";
                //
                // InternalIP
                //
                this.InternalIP.Location = new System.Drawing.Point(104, 32);
                this.InternalIP.Name = "InternalIP";
                this.InternalIP.Size = new System.Drawing.Size(120, 21);
                this.InternalIP.TabIndex = 23;
                //
                // groupBox2
                //
                this.groupBox2.Controls.AddRange(new System.Windows.Forms.Control[] {
                    this.ExternalIP,
                    this.label11});
                this.groupBox2.Location = new System.Drawing.Point(56, 152);
                this.groupBox2.Name = "groupBox2";
                this.groupBox2.Size = new System.Drawing.Size(264, 104);
                this.groupBox2.TabIndex = 33;
                this.groupBox2.TabStop = false;
                this.groupBox2.Text = "External Interface";
                //
                // ExternalIP
                //
                this.ExternalIP.Location = new System.Drawing.Point(104, 32);
                this.ExternalIP.Name = "ExternalIP";
                this.ExternalIP.Size = new System.Drawing.Size(120, 21);
                this.ExternalIP.TabIndex = 31;
                //
                // NATOkButton
                //
                this.NATOkButton.Location = new System.Drawing.Point(64, 328);
                this.NATOkButton.Name = "NATOkButton";
                this.NATOkButton.TabIndex = 34;
                this.NATOkButton.Text = "Ok";
                this.NATOkButton.Click += new System.EventHandler(this.NATOkButton_Click);
                //
                // NATCancelButton
                //
                this.NATCancelButton.Location = new System.Drawing.Point(216, 328);
                this.NATCancelButton.Name = "NATCancelButton";
                this.NATCancelButton.TabIndex = 35;
                this.NATCancelButton.Text = "Cancel";
                this.NATCancelButton.Click += new System.EventHandler(this.NATCancelButton_Click);
                //
                // label1
                //
```

```
            this.label1.Location = new System.Drawing.Point(64, 280);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(128, 24);
            this.label1.TabIndex = 39;
            this.label1.Text = "Number of Ports to Use:  ";
            this.label1.TextAlign = System.Drawing.ContentAlignment.MiddleRight;
            //
            // NumberofPorts
            //
            this.NumberofPorts.Location = new System.Drawing.Point(200, 280);
            this.NumberofPorts.Name = "NumberofPorts";
            this.NumberofPorts.Size = new System.Drawing.Size(112, 20);
            this.NumberofPorts.TabIndex = 38;
            this.NumberofPorts.Text = "10";
            //
            // NATSetup
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(376, 382);
            this.Controls.AddRange(new System.Windows.Forms.Control[] {
                this.label1,
                this.NumberofPorts,
                this.NATCancelButton,
                this.NATOkButton,
                this.groupBox2,
                this.groupBox1});
            this.Name = "NATSetup";
            this.Text = "NAT Setup and Control";
            this.Load += new System.EventHandler(this.NATSetup_Load);
            this.groupBox1.ResumeLayout(false);
            this.groupBox2.ResumeLayout(false);
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new NATSetup());
        }

        private void NATSetup_Load(object sender, System.EventArgs e)
        {
            IPHostEntry HosyEntry = Dns.Resolve(Dns.GetHostName());
            if(HosyEntry.AddressList.Length > 0)
            {
                foreach(IPAddress ip in HosyEntry.AddressList)
```

```
                        {
                            ExternalIP.Items.Add(ip.ToString());
                            InternalIP.Items.Add(ip.ToString());
                        }
                    }
                }

                private void NATCancelButton_Click(object sender, System.EventArgs e)
                {
                    Application.Exit();
                }

                private void NATOkButton_Click(object sender, System.EventArgs e)
                {
                    /// saftety checking code
                    if( InternalIP.SelectedIndex == -1 && ExternalIP.SelectedIndex == -1 )
                    {
                        MessageBox.Show( "You must select an Internal and External IP address" );
                        return;
                    }
                    if (!((System.Convert.ToInt32(NumberofPorts.Text) >= 1) && (System.Convert.ToInt32(NumberofPorts.Text) <= 60000)))
                    {
                        MessageBox.Show( "You must enter a number of ports to use. Currently Min = 1 and Max = 60000" );
                        return;
                    }
                    /// Edit the Registry.
                    RegistryKey reg = Registry.LocalMachine.OpenSubKey( "Software", true ).OpenSubKey( "NATUSQProj", true );
                    if( reg == null )
                    {
                        MessageBox.Show( "Error unable to create the registry key 'USQ NAT Project'" );
                        return;
                    }
                    reg.SetValue( "LocalIP", InternalIP.Text);
                    reg.SetValue( "GlobalIP", ExternalIP.Text);
                    reg.SetValue( "InitialPorts", (NumberofPorts.Text));

                    Application.Exit();
                }
        }
}
```