

School of Computer Science,
University of Birmingham



Interactive TCP/IP Simulation

Andrew J Hogan
Computer Science BSc

Supervisor: Dr Steven Vickers
8th April 2008

Abstract

The goal of this project was to create a piece of software that simulates the processes undertaken when two computers communicate over a network using specific protocols. The protocol simulated was the TCP/IP stack, a common standard for computerised communication. The report outlines the steps taken from project initiation to completion and evaluation of the software. This includes the choices made on features, functionality and their justification.

The software created was a simulation that closely adhered to the processes laid out by the TCP/IP specification. It allows the user to simulate a message being sent from one computer (the client) to a second computer (the server). The simulation illustrates the steps performed by each computer during the opening, message sending and closing of a network connection. It allows the user to specify the initial configuration settings and virtual network addresses of both computers. The user can trigger messages being transmitted to become damaged or destroyed. How TCP/IP deals with these events is then illustrated.

The simulation is intended to be used as a teaching tool to aid learning by demonstrating how network protocols interact. This is achieved by a simple colour-coded visual representation of the TCP/IP stack. Specific details, such as the structure of addresses is also covered. The software contains in-built help functions to explain the major concepts. The user is able to interact with the simulation, controlling the speed at which it runs. This allows them to review all the displayed changes before progressing to the next step.

The software created as a result of this project succeeds in demonstrating the processes performed during TCP/IP communication in an easy to understand format. The simulation is intended for use by networking students. It requires some background knowledge of the field to understand how the simulated software stack relates to network hardware.

Acknowledgements

Dr Steven Vickers – *For all his help as my supervisor over the course of this project.*
'Al' – *For travelling miles to be my patient test subject!*
Lindsey S – *For finding all (??) of my spelling mistakes.*

Table of Contents

Section	Page Number
Introduction	1
Project Goals	1
Project Structure	1
Background Research	2
What is to be investigated	2
Research Texts & Source Evaluation	2
Networking Protocols	2
TCP/IP Protocol Stack	2
Application Layer	5
Transport Layer.....	6
Network Layer.....	12
Data-Link Layer.....	14
Physical (Media-Access) Layer	15
Network Hardware	15
Summary	16
Existing Products	17
Specification	19
Product Users	19
Requirements Elicitation	19
User Requirements	20
Non-Functional Requirements	22
Outstanding Issues	23
System Evolution	23
Requirements Validation	23
Feasibility Study	23
Time Estimation (Ref)	24
Risk Management (Ref).....	24
Problem Analysis	24
Design Decisions	24
Simulation Scope	24
Level of Detail	25
Application Architecture	27
Object Orientated Programming	27
Language Selection.....	27
Model-View-Control	27
Software Engineering	28
Fall-Back Positions	28
Class Breakdown	30
Application Hierarchy	31
Class Interaction	32
Timing Interactions	32
UML Diagram	33
Project Management	34
Risk Management	34
Software Development Lifecycle	35
Time Management	38
Implementation	40
Address Classes	40
PDU Classes	43
Communication Classes	46
Stack Layer Classes	46
Intermediate Network Devices	51
View Classes	52
Design Alterations	57
Implementation Problems	58
Testing	59
Development Testing	59
Unit Testing	59
Integration Testing	59
Beta Testing	60
Release Testing	61
Improvements	62
Evaluation	62
Conclusion	64
Appendix	65
Bibliography	65
CD Contents	65
Activity on Arrow Network	66
Gantt Chart	68
Allowed IP Address Table	69

Introduction

TCP/IP is a synthesis of complementary network protocols and means Transmission Control Protocol (TCP) over Internet Protocol (IP). TCP is responsible for creating a virtual pipe for two applications, typically on different computers, to communicate through. It compensates for data being delivered in an incorrect order or being lost in transmission, as such ensuring the reliability of communication. IP encompasses the logical addressing of computers across a network. It controls how packets of data travel within and across multiple networks, and ensures corrupted packets are disregarded. Together they construct a system of common rules and procedures that defines the complex process of transferring data across a significant proportion of the world's networks, encompassing business, home and the Internet.

The TCP/IP protocol suite defines the network communication process that applications can use; regardless of the platform they operate on. It splits up the communication process into four distinct, hierarchical layers. Each layer has pre-defined input and output interfaces and is responsible for performing a specific action. An application can communicate via TCP/IP, by requesting such functionality from an enabled system. The application supplies the data and its destination to the TCP/IP stack. The entire communication process is then conducted transparently to that application and the user. The purpose of the TCP/IP standard is to ensure compatibility of all TCP/IP implementations, regardless of the system platform, network hardware or the software running on it.

Project Aims

The goal of this project was to design and implement a computerised simulation of the processes undertaken and messages passed between two computers communicating via Transmission Control Protocol over Internet Protocol. It is to be used as a teaching tool to visually illustrate the logic of the hierarchical arrangement of these interacting protocols and how they communicate with one another. The simulation displays the format and content of messages passed between these interacting protocols and how they are handled by each through the process of encapsulation. Additionally, the simulation illustrates how intermediate network hardware interacts with computers using TCP/IP.

The processes were illustrated by simulating Client and Server software. The user enters some text into the Client software to send to the Server software. The Client software is supplied with the address details and the Server software must be configured to listen for incoming connections. As the simulation progresses, the user is presented with a visual representation of the processing and message passing conducted by the interacting TCP/IP protocols until the message is successfully delivered to the Server's software. The user can configure initial settings used by each layer and can trigger messages to be damaged or destroyed while being transmitted. These problems are detected automatically by the simulated TCP/IP stacks and the actions real stacks would take are illustrated. It is possible to trigger any combination of errors and the simulation will react as a real TCP/IP communication would.

As understanding the complexities of TCP/IP is a highly specialised sub-field of networking, the target users of the simulation are university lecturers and students. As such, an initial level of the knowledge of computer science and network concepts will be expected. The goal of the simulation is to function as a teaching tool that can be displayed, for example, on a projector in a university lecture. The lecturer will be able to illustrate these processes to their students and talk through the interacting processes step by step. The simulation could also be used by individual students as a means to conduct further independent studies into the field.

Justification

Interest in this subject was sparked when the developer attended a university module on networking. All the individual components and processes utilised by TCP/IP were well understood, but it was lacking a suitable explanation of how all these processes integrated into a single communication package. As such, it was extremely difficult to understand the overall flow and processing of information between communicating computers. The module was lacking a medium through which to visually demonstrate how these entities interact. After further research, it became apparent that textbooks and existing simulations were also lacking in a succinct summary of these interactions. This unfulfilled niche had been identified and creating a device to summarise these interactions became this project's goal.

Structure

This project follows a conventional development lifecycle. Initially research was conducted into how TCP/IP works. This was critical as the simulation had to accurately illustrate these processes to be a valuable teaching tool. Research into existing similar products has been conducted to ascertain how successful they have been and to identify any design decisions that have failed.

Following the research phase, the project's requirements were identified. A feasibility study was conducted to ensure the project was achievable with the resources available to the developer and intended users.

The design phase outlined the structure and interactions of the product. It provided an overview of how the application processed information. Potential difficulties were identified and analysed in detail. Following the design, the product was implemented.

The product was tested by a group of potential end-users and evaluated against the original specification. The project management techniques utilised for this project have been identified and explained, including the software development lifecycle, risk and time management.

Background Research

What will be investigated?

The aim of this project is to simulate the processing and data passing conducted when a message is being sent from one computer to another using the TCP/IP protocol. To more accurately target the requirements of the simulation, background research has been conducted in two fields:

- The technical aspect of how TCP/IP functions. This included the type of data that can be sent, actions taken by computers in order to process and send data, and how data is formatted into messages that can be sent across a network.
- Existing products that include content that overlaps the fields outlined above. This may include other simulations of processes performed during network communication. Also, any data outputs available from computer hardware actually conducting TCP/IP communication.

Due to the complexity of the TCP/IP protocol, a volume of preliminary research had to be conducted into the technical procedures and data formatting used in the communication process. This was critical: failure to illustrate the protocol methodology accurately would fail this product in its primary role as a teaching tool, rendering it obsolete.

Research texts & source evaluation:

Several published texts, network-industry regulatory body and universities' websites were consulted during the research phase. It became clear that TCP/IP as a protocol is constantly being revised to improve quality of service and utilise new communication technologies. The most significant of these revisions is the implementation of Internet Protocol version 6, replacing version 4. However, this is still to be rolled out across much of the internet at time of writing. Version 6 extends the addressing format used at the IP layer, but is backwards compatible and does not alter the processing techniques that deal with addresses. This is just one of many examples that raise the issue of which version of TCP/IP should be implemented in this project. After consulting numerous published and university texts on the subject, the book TCP/IP Illustrated Volume 1, *The Protocols* by W.R Stevens PhD (Addison-Wesley Professional Computing Series) was selected as the core source of information for this project. This book covers the largest range of processing logic and data formats out of all the texts consulted. It also presents this information in the most consistent format. The suitability of this text was based primarily on W.R Stevens industry recognised expertise in networking. The author of 7 widely known networking texts, he also co-authored several Request for Comments (RFC) documents for the Internet Engineering Task Force – an industry body responsible for developing and promoting internet standards. Whenever possible, TCP/IP processes were followed from this text. Where this information was not available, other texts were referenced to clarify protocol functionality.

Networking Protocols

Networks are "two or more computers that are connected together to share resources such as hardware, data, and software."¹ Computers come in a variety of hardware and software implementations, created by different companies. These devices are often unable to communicate directly as the way they format and pass data is different on each system. Protocol standards have been developed so different computers can communicate. These define a strict set of services and processes which must be adhered to, to allow computers to communicate across a network. These processes are not platform-dependent, so can be implemented on a variety of hardware and software, making rivaling companies products compatible with one another. The TCP/IP suite is one of the resulting protocol standards. It has been globally adopted as a primary method for reliable data transmission across a network and is widely used on the Internet.

TCP/IP: The Protocol Stack

Protocol Stack:

TCP/IP is not one but a synthesis of two complementary network protocols and literally means Transmission Control Protocol (TCP) over Internet Protocol (IP). The two protocols then work in conjunction with others in a stack to complete data transmission from source to destination. Each of these protocols is responsible for a separate task. Collectively, the suite provides all the functionality required for message passing across a computer network.

TCP creates a virtual pipe for two applications, typically on different computers, to communicate through. It compensates for data being delivered in an incorrect order or being lost in transmission; as such it ensures reliability of communication.

IP encompasses the logical addressing of computers across a network. It controls how packets of data travel within and across multiple networks, and disregards corrupted packets. Together they construct a system of common rules and procedures that defines the process of transferring data, used across a vast proportion of the world's networks including business, home and the Internet.

¹ Kimberly Carito, *American Dynamics Website*, http://www.americandynamics.net/intellexip/primer_ip_glossary.aspx, [Accessed 16th March 2008]

- **TCP/IP Standard:** A system of rules defining communication on a TCP/IP network, such as what services should be provided. The TCP/IP standard ensures the compatibility of all TCP/IP implementations, regardless of the system platform, network hardware or the software being used.
- **TCP/IP Implementation:** A vendor-specific hardware or software component of a system that performs the functions that enable a computer to participate in a TCP/IP network; specifically how these services have been implemented on a system.

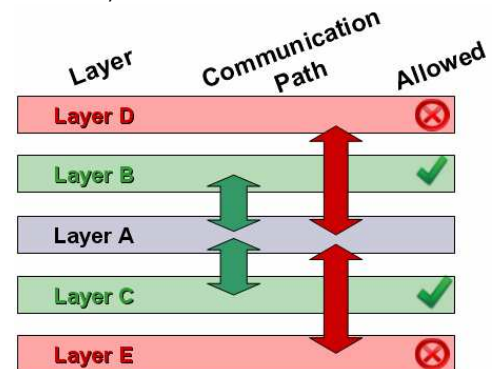
Layering:

As computer operating systems, architecture and network implementations are so diverse; TCP/IP has been designed to be modular. The communication process is split up into five distinct, hierarchical layers called the Protocol Stack. Each layer has pre-defined input and output interfaces and is responsible for providing specific services to the layer above it for the communication process. These interfaces and services are defined by the TCP/IP standard. A vendor can then create a piece of software or hardware that is compatible with an operating system or network transmission medium and be sure that the module is compatible with other systems because it conforms to the TCP/IP standard. This allows computers with different hardware and operating systems to communicate via TCP/IP on and between a variety of different types of network transmission media.

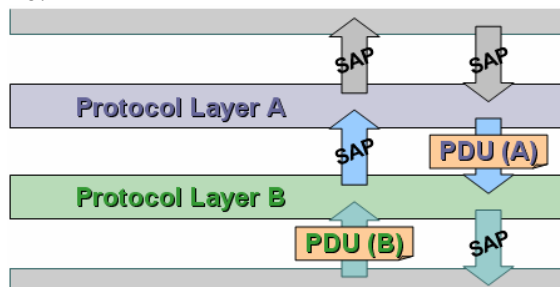
The modular interfaces have the benefit of allowing new processing and hardware solutions to be 'slotted in' at the required layer as required to suit different circumstances. For example, it is possible to send a TCP/IP transmission wirelessly, over Ethernet, optical fibre, a coaxial network or even a satellite transmission between computers running different operating systems. As each component in the communication is TCP/IP compatible, the transmission can be completed without either system having knowledge of how the other systems operate. This allows TCP/IP networks to be expanded as new hardware and software systems become available.

The most fundamental aspect of the layered approach is that each layer performs a unique task in the communication process. It operates independently and as if it was communicating directly with its corresponding layer on the target computer via the defined protocol for that layer. In reality this functionality is provided by the layer below. The stack is organised so that each layer communicates only with the layers immediately above and below it. Controlling interactions is important as it defines the scope of information that each layer has access to.

This diagram illustrates the communication paths available to layer A:



Information is passed up and down the stack between layers through defined interfaces called Service Access Points (SAP). SAPs define the services that a layer provides to the layer above it. They only accept correctly formatted information called Protocol Data Units (PDU). Each layer processes messages based on the information held within its own PDU.



The five layers of the TCP/IP stack are:

- (4) Application Layer** - Provides an interface for applications to access network functions.
- (3) Transport Layer** - Provides a reliable quality of service, including ensuring delivery and flow control.
- (2) Network Layer** - Responsible for ensuring messages are sent on the correct route through a network.
- (1) Data Link Layer** - Responsible for ensuring messages are sent & received without errors or collisions.
- (0) Physical (Media Access) Layer*** - Converts computerised data into a binary signal to be sent through the communication media.

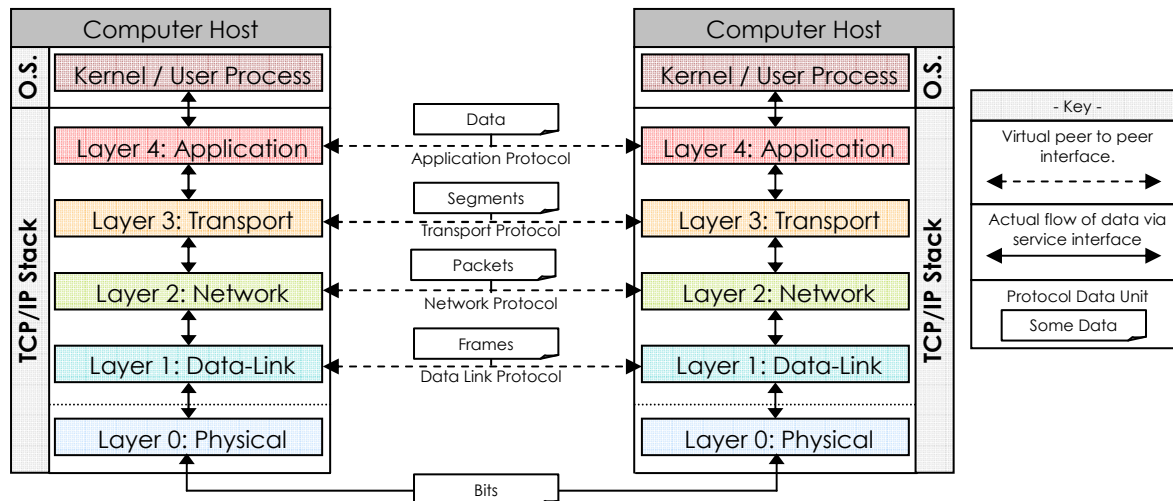
*The Physical Layer is not part of the stack. It is hardware-dependent and is responsible for taking TCP/IP-formatted data and converting it into a signal which is transmitted across network media. These tasks are performed on the network card hardware itself; the process is governed by the drivers that hardware is operated with and not the protocol standard. However, in order for a TCP/IP communication to be completed the data must be transmitted across the network. This layer has to work in conjunction with the protocol stack and drivers are written to allow this. Without this facility, TCP/IP would not be able to send messages between networks using differing media (cables) and topologies (logical layouts). As such, this layer has to be included in the project.

Each participating host in TCP/IP communication has its own stack. The layers within each stack contain variables used to manage each end of the communication process. At higher layers, these variables may be shared between the two computers to create a consistent communication path at both ends. Once the user process on one host requests network functionality from the Application Layer, the Application Layer

communicates with the corresponding Application Layer (using the Application Protocol) on the target host via a 'virtual tunnel' provided by the Transport Layer. Likewise, the Transport Layer communicates with the target host's Transport Layer via a 'virtual tunnel' provided by the Network Layer and so on. This theoretical direct communication is called the 'Peer-to-Peer Interface'.

The research indicated there is no industry-adopted colour scheme to represent each of the layers. To more clearly differentiate sections of the stack, I have selected a colour-coded scheme. These colours are used throughout the project and will be implemented in the simulation. This is to closely link the documentation with the simulation to provide a consistent, visual key to the project. This should aid in simplifying the simulation to facilitate learning.

Map of the flow of data from one host to another via the TCP/IP layers:

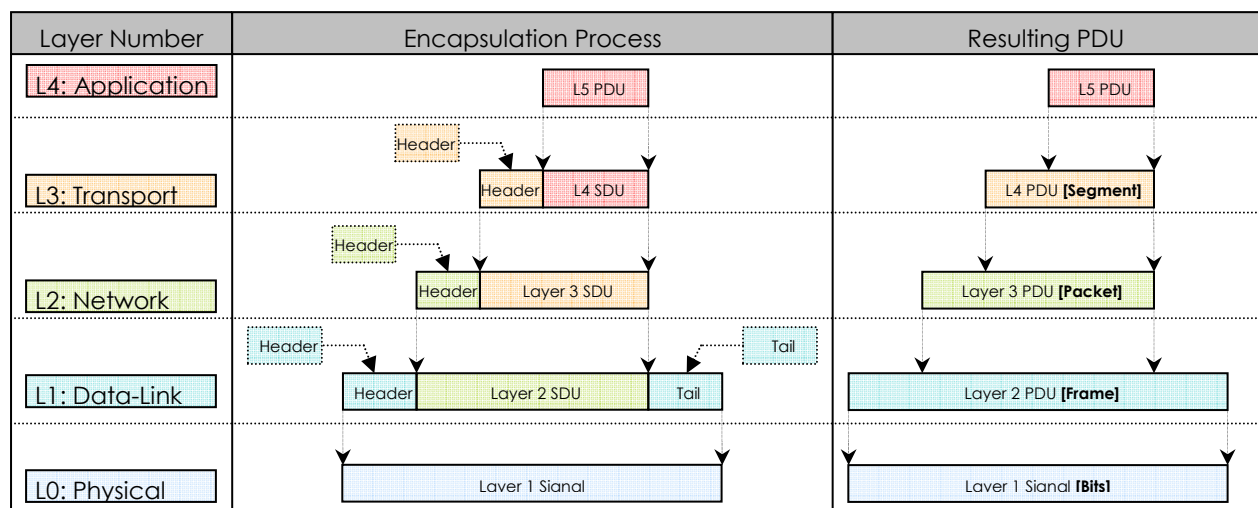


Encapsulation:

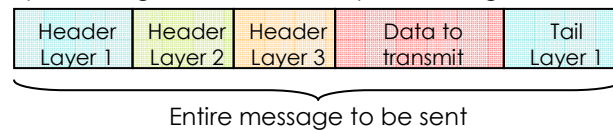
The result of the TCP/IP stack layering is that data is sent from one host's Application Layer to another host's Application Layer indirectly. On a computer sending data, the PDU from Layer N (PDU:N) is sent to layer N-1. At Layer N-1, the PDU of Layer N (PDU:N) becomes the Layer N-1 Service Data Unit (SDU). So $(PDU:N) = (SDU:N-1)$. The PDU of any given layer is the SDU of the layer immediately below. A PDU of a layer includes the information that is required by that layer to process it. This information typically comprises of the source and destination hosts' network addresses and message sequencing information.

As data is sent from an application it is encapsulated at each layer, and processing information is added to the data (SDU) so it can be correctly processed at that layer (making it a PDU). The data then drops down the stack to the lower layers, having further processing information added at each layer. It is then transmitted across some form of network media and is received by the Data-Link Layer of the receiving host's stack. The processing information of the PDU for that layer is then removed and used to process it. The remaining SDU is passed to the layer above, where it translates to the PDU of that layer and can be processed correctly. The message is passed upwards, having the processing information removed again at each layer, so the above layer can process it appropriately. This is called demultiplexing. Eventually the only part of the message left will be the original data for the receiving application to read off.

The encapsulation process of data at each layer of the TCP/IP stack: *(The counter-flow of this table would illustrate demultiplexing as data travels up the stack.)*



As a result of the encapsulation process, every message sent through the TCP/IP stack is actually an amalgamation of every layer's processing information on top of the original data to be sent.



Application Layer

Layer 4

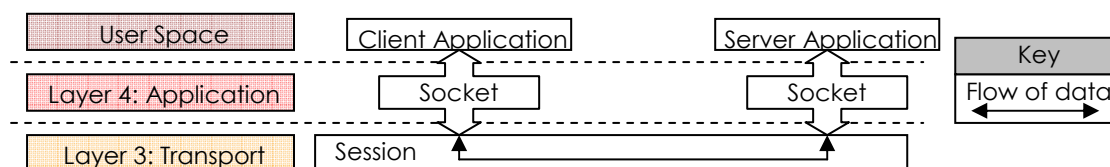
Function:

The TCP/IP stack begins with the Application Layer. It provides an interface for kernel and user processes (software) to access available network services. Any program the kernel grants network privileges to can request a service from the network via this interface, the rest of the communication process is conducted by the lower layers and appears transparent to the application. As such, the Application Layer's primary role is disguising the technicalities of networked communication from the end user and applications. The functions this layer provides are specialised and specific for the task being performed. These include all the protocols required for peer-to-peer communication. The Application Layer's scope of responsibility also includes encryption and decryption and the compression and decompression of data being transmitted.

Implementation:

The Application Layer creates an interface (called a Socket) to a communication pipe (called a Session) created by the lower Transport Layer. "Instead of using the term 'Session,' TCP/IP uses the term 'Socket' to describe the path over which cooperating applications communicate."² This is because the Socket entity through which applications send and receive data (via a Session) is provided directly to the application by Layer 4, but the task of controlling how data is processed within a Session is performed at Layer 3 (the Transport Layer). The Socket provides a virtual circuit through which applications can communicate, but the management of that communication is conducted lower down in the protocol stack.

The term virtual-circuit is used as the stack creates a virtual electrical connection between the two communicating hosts, through which they can read off and write data to. This data will then be delivered to the target host as if it were connected directly. In reality, these messages are sub-divided for transmission and may take different routes to get to their destination.



Each active Socket on the Internet (or closed network) is identified by an IP Address and Port Number for each end of a connection. These addresses are used by lower layers in the stack for processing purposes, but must be entered into the Application Layer at the top of the stack in order to be passed down to the appropriate layers for use.

The majority of Application Layer services involve sending data across the network from one application to another application. Each TCP/IP connection is initialised by one piece of software, (the Client) in order to request a service from other software (the Server). In order for a Client software to communicate via TCP/IP to a Server software, it needs a Socket to be created. To do this, the Client software requests a connection to an IP Address (a Layer 2 address) and Port Number (Layer 3 Address) via the Application Layer. The Application Layer tasks the Transport Layer with this connection and provides an 'Active Socket' to the Client software. The Transport Layer creates a Session to the Server computer's Transport Layer. Through this Session, the Client's Application Layer contacts the Server's Application Layer. If the Server software is 'listening' for incoming connections via an open 'Passive Socket', the two Sockets connect. The client and server applications can then read and write data to these Sockets, which act as a virtual circuit and deliver data to the corresponding application over the network.

It is important to note that although one piece of software will always initiate a connection, once connected, data can be passed in both directions between applications. It is possible, although very unlikely, that two applications will try to connect to each other simultaneously. Due to this possibility, "TCP was purposely designed to handle simultaneous opens and the rule is that only one connection results from this, not two connections." Stevens (1993)³ The service is requested by the Client application, then data is written in a program-specific format to the Client's Socket, it is then transmitted (passing through the TCP/IP stack) and read from the Server's Socket by the Server application.

² Cisco Systems Inc. (2002), *Understanding TCP/IP*, <http://www.cisco.com/univercd/cc/td/doc/product/iaabu/centri4/user/scf4ap1.htm> [Accessed 29th November 2007].

³ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p250.

At a higher level than Socket communication, the Application Layer provides functionality through a variety of commonly implemented protocols. Examples of these protocols include:

- **Ping & TraceRoute** – “Ping stands for Packet InterNet Groper and is used to test whether another host is reachable via the network.”⁴ A successful ping to a host will result in a ping echo reply from that host. TraceRoute is a more advanced version used to display the path datagrams take across a network to the target host.
- **HTTP** – Hyper Text Transfer Protocol. “The protocol used by the World-Wide-Web to exchange text, pictures, sounds, and other multi-media information via a graphical user interface.”⁵
- **FTP** – File Transfer Protocol is “a widely used protocol that enables a user to transfer files between two computers across a network.”⁶ Rather than just sending streams of data, FTP allows the entire file, including access and security flags to be copied.
- **SMTP** – Simple Mail Transfer Protocol is used across the internet to deliver electronic mail (e-mail). “An e-mail daemon that speaks SMTP accepts incoming connections... and copies messages from them to the appropriate mailboxes.”⁷
- **NFS** – Network File System “provides file services, and other client/server components such as X-Window”⁸ on Unix and Linux systems.
- **DNS** – Domain Name System “is a distributed database that is used by TCP/IP applications to map between hostnames and IP addresses.”⁹
- **SNMP** – Simple Network Management Protocol “is a specialized request/reply protocol... A system administrator interacts with a client program that displays information about the network.”¹⁰ This program can be used to retrieve and set configurations on hardware across the network.
- **DHCP** – Dynamic Host Configuration Protocol. A system through which a server or router will automatically assign dynamic IP Addresses to computers as they connect to the network.

Transport Layer

Layer 3

Function:

The layer immediately below the Application Layer is the Transport Layer. The two most common protocols that operate at this layer are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP).¹¹ Features of UDP:

- **Connectionless Communication** - UDP is a connectionless protocol which sends independent datagrams (messages) from Client to Server without the need to establish a ‘virtual circuit’ (connection) first.
- **Unreliable Connection** - Once a UDP datagram is sent, there is no way of knowing if it will ever reach its destination as there are no acknowledgements of receipt.
- **Not Ordered** – There is no guarantee that if multiple messages are sent, that they will arrive at the recipient in the same order that they were sent.
- **Broadcast & Multicast Compatible** – A single UDP datagram can be sent from one system and be delivered to all the systems on a network (Broadcast) or a group of pre-specified computers on a network (multicast).
- **Datagram Messages** – UDP messages are datagrams with specific bounds of the size of message they can contain per datagram. Each datagram is sent individually.

The UDP is offers “a lightweight and efficient mechanism for sending and receiving data”¹². It “provides a way for applications to send encapsulated datagrams without having to establish a connection.”¹³ This avoids the added processing complexities and time overheads associated with checking whether a datagram has reached its destination. Because of this, UDP connections are most suited to time-dependent communications where it is more important that the messages are received as quickly as possible and accepting some may be lost, rather than a transmission potentially being delayed until a lost message is re-sent (as in TCP). Applications include real-time streaming media, such a video conferencing and VoIP (Voice over IP) calls. UDP trades off occasionally losing messages such as a portion of sound or video, in order to ensure what the user receives is the most up to date information so there are no unexpected delays that would leave one recipient lagging behind in a communication. Additionally, UDP datagrams are often used for Remote Procedure Calls (RPC) by

⁴ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p85.

⁵ Peaking University. (2007), *TCP/IP for Internet demonstrators*, <http://www.pku.edu.cn/academic/research/computer-center/tc/html/TC0500.html> [Accessed 29th November 2007].

⁶ Casad, Joe (2003), *SAMS Teach Yourself TCP/IP in 24 Hours*, p234.

⁷ Tanenbaum, Andrew S.(2003), *Computer Networks Fourth Edition*, p602.

⁸ Casad, Joe (2003), *SAMS Teach Yourself TCP/IP in 24 Hours*, p113.

⁹ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p187.

¹⁰ Peterson, Larry L. & Davie, Bruce S. (2003), *Computer Networks: A Systems Approach Third Edition*, p657.

¹¹ Tanenbaum, Andrew S.(2003), *Computer Networks Fourth Edition*, p524.

¹² O'Reilly – Safari Books Online. (2007), *Optimizing NFS Performance: Tuning and Troubleshooting NFS on HP-UX Systems, Chapter 10. NFS/UDP vs. NFS/TCP*, <http://safari.oreilly.com/0130428167/ch10> [Accessed 1st December 2007].

¹³ Tanenbaum, Andrew S.(2003), *Computer Networks Fourth Edition*, p525.

networked applications, where the connectionless system avoids the increased latency that would be created by a connection-orientated implementation.

TCP is a more complex solution than UDP. It functions in a very different manner and is suitable for a different array of tasks. TCP's fundamental features are:

- **Connection-Orientated Communication** – To send TCP messages, a 'virtual circuit' connection must be initialised between the two participating computers. Once transmitted, the 'virtual circuit' is terminated.
- **Reliable Connection** – Delivery of TCP messages is acknowledged by returning a receipt of the message to the sender. Lost or damaged messages are not acknowledged. After a defined time-out, any messages sent and not acknowledged are re-transmitted. The implementation of this feature means that a host can be sure that any message sent across a TCP connection has been delivered successfully. In a worst-case scenario where the network connection is permanently disrupted and no more messages can be sent, as no acknowledgements are received, the host will be able to identify the failure of the connection and inform the application that it has failed to send the message. This means a host will always be able to ascertain whether a message has been delivered successfully or not.
- **Ordered Connection** – If multiple messages are sent across a TCP connection, the receiving application will have the messages delivered to it in the same order that they were sent.
- **Not Broadcast or Multicast Capable** – As a TCP connection is a 'virtual circuit' between only two computers, it is not possible for a TCP connection to have multiple destination computers.
- **Streaming Message** – A TCP connection acts as an unbound stream of data for the application to send information on. Messages are sent sequentially and the TCP connection will remain open until all the data has been successfully delivered.

TCP was "specifically designed to provide a reliable end-to-end byte stream over an unreliable network."¹⁴ This characteristic makes it ideal for sending computerised data across a network. For example, if a digital file was to be damaged due to errors or file portions were going missing when being sent, the file would no longer function. TCP provides a means to ensure anything transmitted is delivered in the same state it was sent in. Any missing or damaged messages would simply not be acknowledged and be continually re-sent until the entire data stream is received successfully. TCP is a two-way communication, where the machine that is having data sent to it (Server) will constantly update the sender (Client) about the speed at which it is prepared to receive data, and how much it can receive in one go. The speed of flow and buffer sizes can be altered dynamically throughout the course of transmission to counteract the altering bandwidth and latency limitations of the network, as well as the processing limitations of the receiving computer.

UDP transmissions are 'best effort' and only suitable for specialised tasks where delivery does not have to be guaranteed. When UDP sends a datagram, there is "simply no way to know if it has been lost or not."¹⁵ In systems where UDP delivery is vital, such as RPC's, the logic for checking for lost datagrams and requesting repeated messages has to be implemented as "Application logic, not Protocol logic."¹⁶ This requires more advanced applications to be coded for error checking, whereas an application that utilises TCP does not have to process this as the logic is implemented in the protocol itself. The majority of network transmissions require reliable delivery of messages, so TCP is used more frequently. The most commonly used Application Layer protocols, allowing activities such as file transfers (FTP), viewing web pages (HTTP) and sending emails (SMTP), run exclusively over TCP.

Session:

A TCP connection is called a Session. A Session is initialised by the Client, and responded to by the Server. A Session is identified by its local and destination Port Numbers, the addressing scheme used at the Transport Layer. Port numbers are 16-bit binary numbers, although are typically represented in their integer format. They range from 0 to 65535. Port numbers between 0 & 255 are reserved by IANA RFC specification 278017 for specific tasks, such as FTP. Numbers 256-1023 were reserved by UNIX convention for other tasks, but have been superseded by IANA RFC 2780.18

Typically the outgoing end of a Session is created on a randomly generated Ephemeral Port. This is because the Port Number used does not have to be known in advance. The Server end of a Session operates on a fixed Port Number. This is because the Client Session needs to know in advance which Port Number it is to connect to, whereas the Server finds out the Client's Port Number once it is connected.

The Client Application Layer initially requests a Socket connection to a destination IP Address and Port Number of the Server. If a Server application is running at the destination IP Address and has designated that Port Number for incoming connections, the Server will respond to this connection. A Session is then created between the Transport Layers of the Client and Server computers. The Client can then send data to the Server. Once complete, the Socket is closed and the Session is terminated. A Session is a communication between two Hosts,

¹⁴ Tanenbaum, Andrew S. (2003), *Computer Networks Fourth Edition*, p532.

¹⁵ Mc Quaid, Jim (2007), *Nice Guys Finish Last, Network Performance Daily: Whiteboard Series (Video Presentation)*, http://www.networkperformancedaily.com/2007/08/whiteboard_series_nice_guys_fi.html [Accessed 06th December 2007].

¹⁶ Mc Quaid, Jim (2007), *Nice Guys Finish Last, Network Performance Daily: Whiteboard Series (Video Presentation)*, http://www.networkperformancedaily.com/2007/08/whiteboard_series_nice_guys_fi.html [Accessed 06th December 2007].

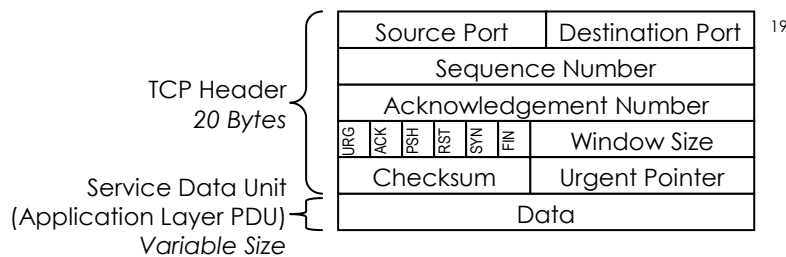
¹⁷ IANA RFC Database Website, Protocol Registries, <http://www.iana.org/protocols/>, [Accessed 19th March 2008]

¹⁸ Adapted from Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p.13

but each Host's half of the Session is individually responsible for keeping track of what messages have been transmitted, timing the communication and interacting with the neighbouring stack layers.

Segment:

The Protocol Data Unit exchanged at the Transport Layer is a TCP Segment. The logical structure of a Segment is:



The structure of a Segment is fixed and takes up 20 –bytes of information. If a section of the TCP header data contained within a Segment is empty, it will be filled with a padding of null data to maintain the header size. This ensures that no header information is confused with the data being transmitted in the segment. This is because, although there is a logical structure to the header, in reality it is a continuous sequence of binary data. The only way a computer can differentiate between the different values is by knowing how big each value is, then literally counting along the number of bits to find the required data in the stream. As the computer knows the header size, all remaining data can be assumed to be the transmitted data.

[illegible]

01010111101001010100100101010011010001010010101001010010001010010101001010010101001010100101010010101001010100101010000000000000000010101...

Above: A representation of the of a TCP segment's physical structure. Binary values are for illustrative purposes only to represent the consecutive nature of transmission.

A TCP Header contains distinct sections of data. Not all these sections will be utilised in every transmitted Segment. This is because certain pieces of data are used for initialising a connection, some for transferring data and some for closing Sessions. Some are only used under specific, unforeseen circumstances. The data contained in the Header consists of:

- **Source Port Number** - Port Number of the Sender
- **Destination Port Number** – Destination Port Number
- **Sequence Number** – Used to denote which Segment this is in the communication sequence.
- **Acknowledgement Number** – Acknowledges the Sequence Number of the last successfully received Segment on that host.
- **Window Size** – The size of the sender's available receiving window.
- **Data Area** – The SDU (PDU of Application Layer)
- **Checksum** – Used for error checking purposes.
- **Urgent Pointer** – Used to point to urgent data in the transmission.
- **Boolean (On/Off) flags: ACK, PSH, SYN, FIN, URG** - These are used to define the contents and status of the Segment.
 - **ACK** – An active ACK indicates that the Segment's Acknowledgement Number contains an Acknowledgement for previously sent Segments' Sequence Numbers.
 - **PSH** – Push indicates that the Segment contains data to be 'pushed' to the Application Layer.
 - **SYN** – Synchronize Sequence Number is used when initialising a connection. It indicates to the receiver where to start its sequence numbers from in order to correctly synchronise the TCP session with the other end.
 - **FIN** – The FIN flag will indicate to the recipient that the sender has sent all the data it intends to and is closing its end of the connection.
 - **URG** – The URG flag is used in only specialised tasks. It denotes the Urgent Pointer Field contains points to data to be processed immediately.
 - **RST** – Indicates the resetting of the TCP Session following a loss of synchronization.

Acknowledgements:

TCP Sessions use a system of acknowledgements to ascertain whether a Segment has been successfully delivered to its recipient. Once a Segment has been received and processed the recipient transmits a return Segment, called an Acknowledgement, back to the message sender. Once this has been received, the original sender knows the message has been delivered successfully.

Instead of the Server acknowledging every received Segment individually, TCP implements a system called a Delayed Acknowledgment. This reduces the total amount of messages sent, allowing more bandwidth to be used

Eg: ACK piggybacked on PSH

Port(Server)				Port(Client)			
Sequence Number (257)							
Acknowledgement Number (1153)							
URG	ACK	PSH	RST	SYN	FIN	<No Window Size>	
Checksum				<No Urgent Pointer>			
<Some Data>							

This example Segment is sending <Some Data>, in addition it is acknowledging the other host's Segment Number 1152.

¹⁹ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p. Inside Cover.

for the actual transmission of data. In addition, "it delays the ACK, hoping to have data going in the same direction as the ACK, so the ACK can be sent along with the data." This allows the ACK to be 'piggy-backed' with any data being sent in the opposite direction. Most TCP implementations delay an ACK for a maximum of 200ms before one is automatically transmitted. Data will continue to be sent and acknowledged until the Client has finished sending the entire message assigned to it by the Application Layer. At this point, it will close the Session.

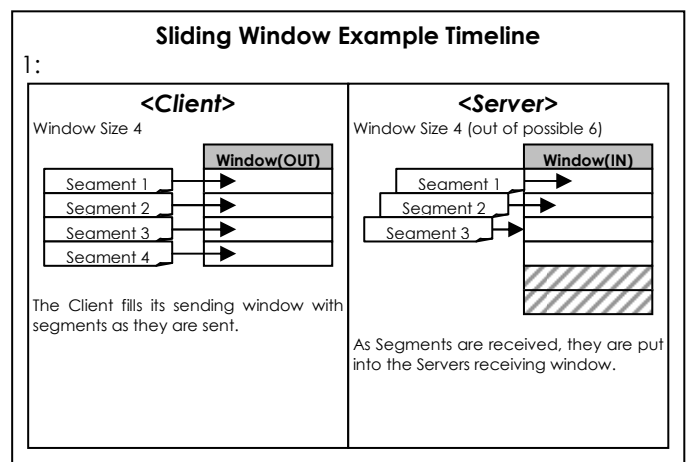
Sliding Window:

The flow of data in a TCP Session is regulated by buffers called Sliding Windows. Each participating Host has a Sliding Window which it reads incoming Segments from and a Sliding Window which it writes outgoing Segments to. For the purposes of this project, these will be named Sliding Window^(In) and Sliding Window^(Out).

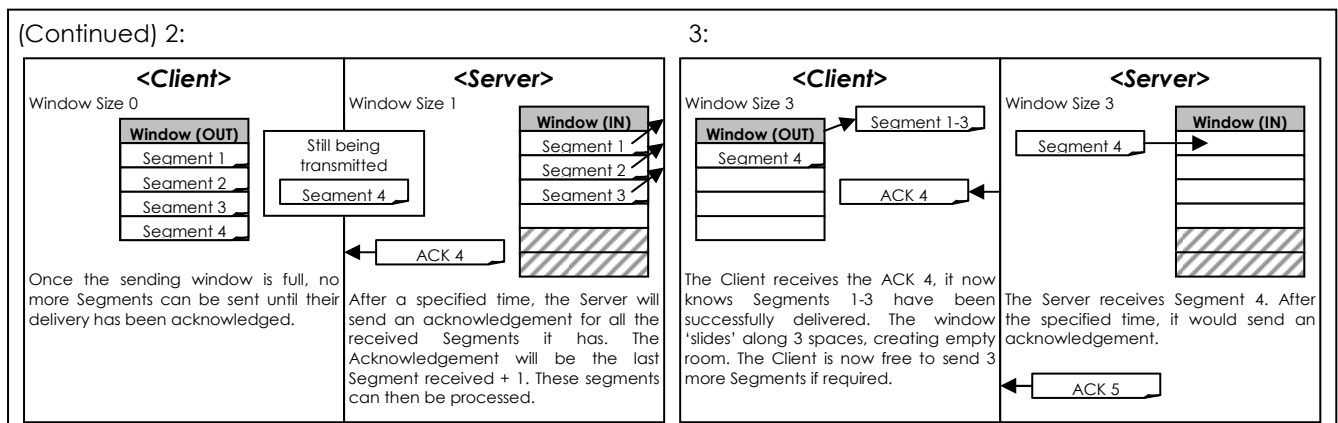
The purpose of these windows is to stop the sending computer from transmitting so many Segments that the recipient is unable to process them quickly enough, which would lead to data being lost. Each host of a TCP connection is aware of how many Segments they can process in a given time, therefore their maximum Sliding Window size. Whilst a Session is open, "the window advertised by the receiver (Sliding Window^(In) on the Server) is called the Offered Window."²⁰ The sending host (Client) compares this to its maximum sending window size (Sliding Window^(Out)). The sender then calculates the size of the Usable Window, which will be the smaller of the two window sizes. This ensures neither host can overflow the other with too much data. For example: A Server's Offered Window size is 4, the Client's maximum window size is 6, so the Usable Window size is 4. Once the Usable Window size is established, the Client can send as many Segments to the Server as there is space in the window. Similarly the Server can do the same.

As Segments are sent, they are written into the Host's Sliding Window^(Out); this window has a limited size and once it is full of sent Segments no more can be sent until the host has received an acknowledgement of their receipt. Once acknowledged, the Host knows these Segments have been delivered successfully, and they are removed from the Sliding Window^(Out). This creates more space in the window, so the Host can send more Segments.

Every Segment put into the sending window has an attribute to keep track of how long it has been in the window. When a Host, for example a Client, sends Segments they are placed in the Sliding Window^(Out). After those segments have been in the window for a defined time (called the Window Timeout) and not been acknowledged by the Server, the sender can assume that they were lost in transmission or that their acknowledgments has been lost in transmission. The Client will then re-transmit all the un-acknowledged Segments in the Sliding Window^(Out) that have exceeded the Window timeout. If they were lost in transmission, the intended recipient then has the opportunity to receive them again. The recipient will then send an acknowledgement to the Client, and sending can continue.



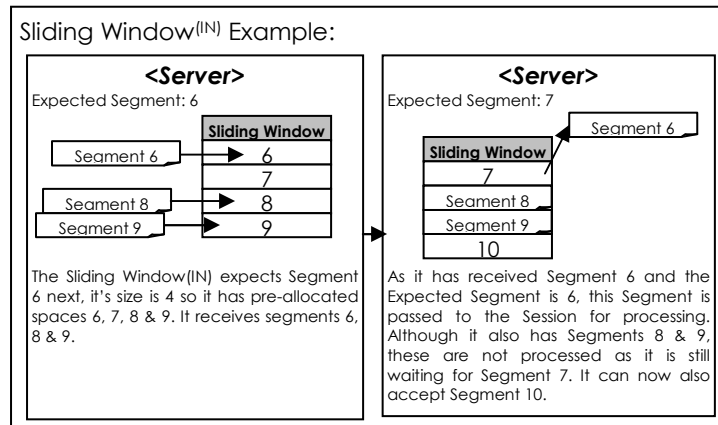
If a Segment was originally received successfully, but the acknowledgement was lost, the recipient knows it has already processed the Segments it has received again and discards them. The timeout is implemented for Acknowledgments also. If no new Segments are received by the Server after the Window Timeout, it will assume that its most recent acknowledgement has been lost and will re-transmit it to the Client.



The SlidingWindow^(Out)'s primary role is to detect when Segments have been lost in transmission and re-transmit them. The role of the SlidingWindow^(In) is different. It acts as an input buffer for the Session so the Host does not have to process each Segment it receives immediately. This allows for more efficient implementation of multiple applications operating concurrently on one computer.

²⁰ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p280.

Due to the nature of how messages are passed across networks (outlined in the Network Layer documentation), it is possible for some Segments to be lost, or for a series of Segments not to arrive in the order they were sent. The SlidingWindow^(In) acts as a buffer to help compensate for these problems. It will only allow the Session to read the Segments in the order they were originally sent, not in the order they were received, re-organising Segments that do not arrive in the correct order. It does this by keeping track of the next expected Segment Numbers, and pre-allocates spaces for these Segments. If it receives Segments out of order, it places them in the correct space in the window, leaving gaps where missing Segments that have not been received yet and will be placed in the future. It will only ever supply the Segment at the very start of the window (the Expected Segment) to the Session. At which point, the next Expected Segment will be the previous Segment's Sequence Number + 1. This way the Session will always be able to correctly process Segments even when they arrive in the wrong order.



The Sliding Window^(Out) re-sends Segments if no acknowledgment has been received by the Window Timeout period. If messages are lost in transmission, any Expected Segment that has gone missing will be re-transmitted (as no acknowledgment was ever sent for it) and received by the Sliding Window^(In). It will then be delivered to the Session, enabling the communication process to continue. The dual Sliding Window system is beneficial as it avoids either host from overflowing the other with Segments. It also allows participating hosts to monitor if Segment are lost in transmission.

By implementing this window buffer, with acknowledgements and re-transmission on timeouts, a TCP Session will be able to reliably send its entire message on an un-reliable network, as long as a link exists at all, even if it fails 90% of the time. This is because both the Client and Server will continue re-transmitting their messages again & again* until they receive the next Segment in the Session. This system ensures a reliable delivery of data in an un-reliable time span (as theoretically it could take many re-transmissions for every Segment before the message is delivered in its entirety).

*If a network link fails entirely, the hosts can identify this as they have a pre-defined maximum re-transmission threshold. After they have re-transmitted a Segment a defined number of times with no reply, it is assumed that the link has failed. At this point, the Transport Layer can inform the Application layer that it has failed to deliver the message. This differs from UDP, where there is no way to identify if a message has ever been delivered successfully.

If either side's processing capacity or network bandwidth diminishes or expands during the course of a TCP Session, it can send a message to the other host, defining an updated Offered Window size. If the newer window size is smaller, the other host will reduce its Usable Window accordingly. If the Offered Window size is larger than previously, it will increase its Usable Window size to the newer size or its maximum available size (whichever is smaller). This system allows TCP Sessions to adapt to changing networking and processing loads dynamically, allowing them to always utilise the maximum available capacity to send Segments. If a host is participating in multiple TCP Sessions simultaneously, it will have a Sliding Window for each Session, but its total available capacity will be split between each Session.

E.g. 1 Session with a window size of 6 or 3 Sessions each with a window size of 2.

Opening a connection:

In order to create a connection the Client Transport Layer will attempt to initialise a Session with the Server's Transport Layer. The opening process of creating a Session is called the 'three-way-handshake'. Until the Client and Server have completed creating the Session, they are unaware of the Offered Window size of the other host. Because of this, TCP Sessions assume a worst-case scenario and have a Usable Window size of 1 until the other host's Offered Window size can be determined.

The Client sends the first Segment to the Server. This Segment is called the Active Open. The Active Open specifies that it is requesting a Session by turning the SYN (Synchronize Sequence Number) on/off flag to on. The Sequence Number is the Initial Sequence Number (ISN) for the connection (defined by the Client). "In BSD (and in most Berkeley-derived implementations) when the system is initialised, the initial send sequence number is initialised to 1."²¹ However, the ISN can be any number and is often generated by a function of the computer's main clock, "a variable incremented by 64,000 every half-second."²² In this example, the Client's ISN will be 1. The Client's SlidingWindow^(Out) is now full

Active Open

Port(Client)		Port(Server)	
Sequence Number (1)			
<No Acknowledgement Number>			
JR	ACK	PSH	RST
	SYN	FIN	
Checksum		<Window Size 4>	
		<No Urgent Pointer>	
<No Data>			

Passive Open

Port(Server)					Port(Client)				
Sequence Number (1024)									
Acknowledgement Number (2)									
URG	ACK	RST	SYN	FIN	<Window Size 6>				
Checksum					<No Urgent Pointer>				
<No Data>									

²¹ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p232.

²² Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p233.

(contains Active Open only). The SYN Segment also contains this host's Offered Window size. The Window size is the number actual Segments that host is prepared to receive at any given time.

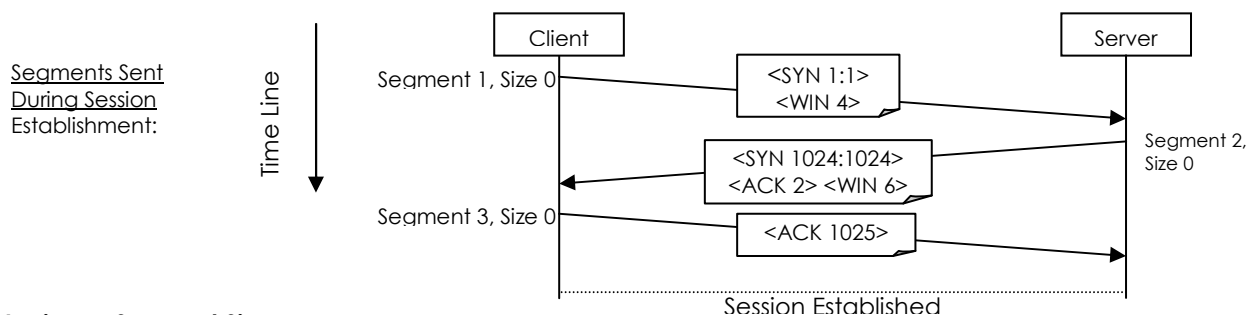
If the target host has a Server application running that is listening to the defined Port Number, it will then reply to the Active Open with a Passive Open. This Segment contains no data. It acknowledges the receipt of the first Segment by sending an acknowledgement (ACK'ing) its Sequence Number +1. This Segment also has its SYN flag activated as it contains the Server's ISN. In this example, the Server's ISN will start be1024. This will make it easier to differentiate between the different sequencing information. It also includes information on the Server's Offered Window Size. The Server's Sliding Window^(Out) is now full.

ACK for Passive Open

Port(Server)				Port(Client)			
Sequence Number (1)							
Acknowledgement Number (1025)							
JR	ACK	PSH	RST	SYN	FIN	<No Window Size>	
Checksum						<No Urgent Pointer>	
<No Data>							

The receipt of the Server's ACK clears the Clients Active Open from the Sliding Window. This allows the third message in the 'three-way-handshake' to be sent. This is the Client's ACK of the Server's Passive Open SYN. The Client's SYN is still 1, as the Sequence Number only increments as actual message data is sent, not for Acknowledgments. Segment acknowledgements containing no data do not increment their Sequence numbers because not every Acknowledgement needs to be received, if one is lost and a newer acknowledgement is received at a later time, the newer acknowledgement will supersede the older one as it identifies the receipt of all previous Segments to the acknowledged number.

The receipt of the Passive Open ACK by the Server allows the Server to clear its Sliding Window^(Out). This completes the 'three-way handshake'. A TCP Session now connects the two hosts. At this point the Usable Window size is calculated for the TCP Session. The size of the Client and Server's Sliding Windows will now be altered to the Usable Window size.



Maximum Segment Size:

The Maximum Segment Size (MSS) that a host can receive can only be transmitted in the SYN Segments sent during the initialisation of a TCP Session. "The MSS is the largest 'chunk' of data that TCP will send to the other end (of a connection)."²³ The MSS advertised by a host is related to the physical transmission limitations of the network that the host is on (some networking technologies can send more data than others in a single transmission). The higher the MSS, the more data can be sent in a single Segment. If the target host is on a different network to the first host (ie: the Session will pass through a Router), there is a possibility that the Segments will pass through different network technology to the one they were originally transmitted on. As the other networking hardware may have a smaller MSS than the first, Segments passing across multiple networks have their MSS limited to the TCP default of 536Bytes. All TCP network hardware has to be capable of processing Segments of this size.²⁴ If no MSS is sent in the SYN, the TCP default is also applied.

Transferring Data:

Once a Session has been established, the Client can start sending data to the Server. The Client will send as many Segments its sending Sliding Window^(Out) will allow. Segments containing data for the target application have the PSH or 'push' flag enabled. This signifies that the data contained in the Segment is to be 'pushed' to the Application Layer by the Transport Layer. For every Segment sent that contains data, the Sequence Number will be added to by the size of data contained in the Segment.

For example, if the first Segment to be sent over a TCP Session with an ISN of 1, and the Segment contains 128 bytes of data, the Sequence Number for the Segment will be 129 (ISN + quantity of data). The next 128 byte Segment's Sequence Number will be 257. The reason the Sequence Number raises by the quantity of data, rather than a linear progression (1, 2, 3...) is that a TCP Session may alter the size of the Segments mid-Session to counteract bandwidth limitations. By incrementing the Sequence Number by the volume of data, the receiving Transport Layer can keep track of the data contained in each Segment, without having to read the data (which is the responsibility of the Application Layer).

Closing a Session:

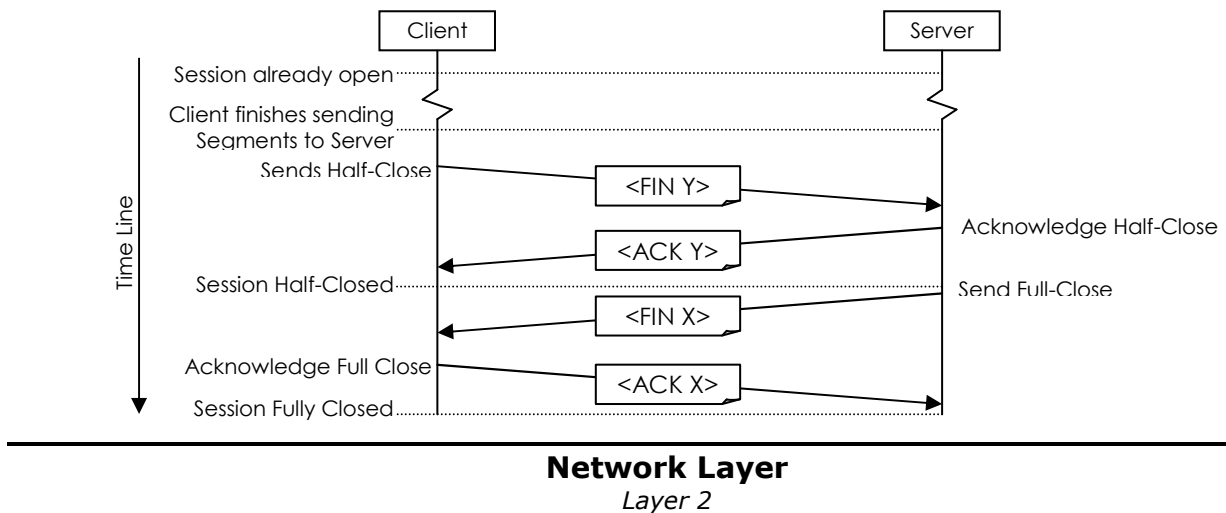
As a TCP Session is bi-directional, each participating host computer has to terminate their end of a connection independently. Once one host has finished sending the intended data Segments, they will close their end of the connection. This is called a Half-Close. The other host's connection is still open, it can still send data (which the first host will continue to send acknowledgments for) until it has completed sending its message Segments. The

²³ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p236.

²⁴ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p237.

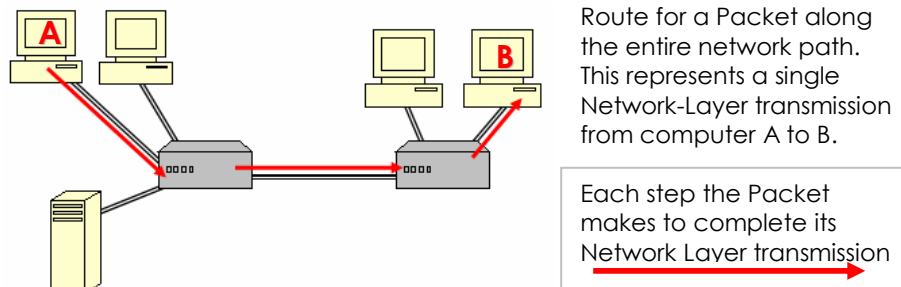
second host will then close their portion of a TCP Session, this is called the Full Close. Once both close commands are complete, the entire TCP Session is closed. To close a Session, an empty Segment is sent with its FIN flag active. This is received by the corresponding host, which then has to send an acknowledgment of this Segment back again. As both ends of a connection have to be closed, two FIN Segments are sent and two ACK Segments are sent in response to them. As the FIN and ACK Segments involved in a Session close contain no data, the Segments Sequence Number does not increase as they are sent. As with all TCP Segments, they are stored in a Sliding Window until they are acknowledged (including the standard timeouts and re-sending).

Segments Sent During Session Termination:



Function:

The Network Layer is responsible for delivering its PDU, called a Packet from the source computer to the destination computer, across one or many inter-connected networks. The Network Layer utilises a logical addressing scheme called the Internet Protocol Address (IP Address) to achieve this. Each computer on a TCP/IP network is assigned its own IP Address. Segments are encapsulated into Packets. Each Packet encloses a header, containing the source and destination computers IP Addresses, and the Segment to be sent. These addresses are utilised by the networking hardware the Packet passes through in order to 'route' the Packet along the correct path for it to reach its destination.



Delivery:

Whereas Transport Layer Segments represent each sub-section of a stream of data across a Session through a virtual-circuit communication pipe, the Network Layer is not designed to do this. It treats each Segment encapsulated as an individual message, addressing and processing each Packet independently of the others. Because of this, there is no guarantee that Packets will arrive in the same order in which they are transmitted.

Once a Packet is sent, each local computer will check its destination IP Address to see if it is intended for them, if so, it will be demultiplexed and the Segment passed up to the Transport Layer. The Network Layer's functionality primarily lies in passing Packets between interconnected networks. IP Addresses are formatted to logically portray the location of the target computer on a network. Devices called Routers connect different networks together. This information is used by Routers to decide whether a Packet should be left on the network on which it was broadcast, or whether it should be 'routed' to a neighbouring network. There it will continue its journey towards its intended destination. Routers keep track of which IP Addresses are accessible using a Routing Table. This contains a list of which IP's are accessible through each physical port on the Router. If a Packet is received and the location of its target IP Address is unknown, routers will forward the Packet through a pre-defined Default Port. Routers communicate the contents of their Routing Tables to neighbouring Routers. This creates a distributed database which allows a Packet to be sent from any IP Address to any other IP Address on a network, such as the Internet.

Once a Packet is sent, it is processed at location after location, with no messages being sent back to acknowledge its process to the sender. The Network Layer is responsible for delivering Packets to their destinations and offers no facility to confirm whether that Packet has been delivered successfully. It is possible that a Packet could be damaged or lost during the transmission process. This means that the Network Layer is an unreliable communication protocol. Because of this, Transmission Control Protocol was designed to operate

'over' IP (Hence TCP over IP). As TCP supplies a means to detect message transmission failures and re-send them, it provides a reliable delivery method over an un-reliable IP network.

IP Addresses:

IP addresses are layer 2 logical addresses of computers on a network. They are logical addresses, as they are assigned based on the 'virtual topology' of a network, not physically how computers are laid out within a building. Because of this, two computers physically next to each other may be on separate networks and have completely dissimilar IP addresses. This project is using the current IP version 4 standard.

The format of an IP address is four numbers (called octets) separated by decimal points:

Number.Number.Number.Number

Each of these numbers range can be between 0 & 255. ie: **0-255.0-255.0-255**

Each of these integer octets represents an 8-bit binary number, for example:

'174.126.164.58' is the integer representation of **'10101110.01111110.10100100.00111010'**

Network addresses have a pre-defined class A-E (although D & E are not currently used). These classes band similar logical levels of addresses together. Network classes differ by how many of the octets are reserved for the network address, and how many are assigned for host computers on that network. The class of IP address defines how many Host computers can have an IP address on that Network Address.

Class Network:Host First Octet (Binary)*

A --- N . H . H . H --- 0xxxxxxx

B --- N . N . H . H --- 10xxxxxx

C --- N . N . N . H --- 110xxxxx

*Where x signifies either a 0 or 1.

Class A networks use the first octet as the **network address**, and the remaining 3 octets for its **host computers** (N.H.H.H), class B's use two octets for the network address (N.N.H.H), class C's use three (N.N.N.H)

Because of this standard, there are fewer class A networks (approximately 255) than B (approximately 255 x 255 available addresses) and so on. Class A network addresses are reserved for top level domains such as governments, where they have many computers on a single network (255 x 255 x 255). Internet Service Providers (ISP's), for example, may have a class B address, so they can give their Network addresses available host addresses to their customers (255x255).

Some IP addresses are reserved for specific functions, and are not allowed to be allocated to hosts or networks. In order to understand these exceptions, the 8-bit binary version of the decimal octets must be viewed. This is because computers process IP addresses in their binary, not decimal format at the machine-level.

- An IP address of all 0's (00000000.00000000.00000000.00000000) or 127.0.0.0 (01111111.00000000.00000000.00000000) denote a loopback address, where any messages sent from a host with this target IP Address will be returned to that host by the Network Layer. This allows software on the same computer to communicate via the TCP/IP stack.
- An address where all the octets of the host portion of the address are 1's is a broadcast address. For example on a class B network:

N N H H
10000001.00010011.11111111.11111111 (129.19.255.255)

Broadcast addresses will be sent to every host on that network. Similarly, an IP Address consisting of all 1's would broadcast to all hosts on all networks. Although precautions are taken so as not to allow this on larger interconnection networks to avoid abuse.

- An IP Address where all the host-portion octets are zeros is the Network Address of that particular network. Any messages sent to a Network Address will be picked up by the router, and then broadcast locally to every host on that network. Example Network Address:

N N H H
10000001.00010011.00000000.00000000 (129.19.0.0)

- The network address, where all the host digits are 0's except the last, which is a 1 could be used by a host computer. However it is reserved by convention for port IP Address of the router controlling that network. For Example:

N N H H
10000001.00010011.00000000.00000001 (129.19.0.1)

Computer networks that are not connected to the Internet do not have to conform to the IP class system, as they act as an independent unit. It is possible to add to the available pool of host addresses using Network Address Translation (NAT), or to section up a network into sub-networks using Subnetting. These will use a single external IP Address for all the computers situated behind a single router. The router will manually forward received packets to the correct computer using an algorithm using local IP Addresses. Every Network connected to another (for example the Internet) has a Network IP address. This address is typically the address of the router that assigns IP addresses of the host computers on that network. The router manages the flow of data traffic between its private network and the wider Internet.

A table summarising all the possible available IP Addresses has been created and is included in the Appendix.

IP Header:

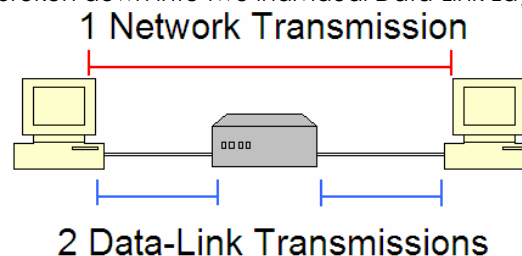
Similarly to a Segment, a Packet also has a rigid structure to format its contained information. This includes the Version of the IP Protocol implemented, The type of service the Packet is implementing, its total length. An identification number for the Packet stream, an offset to identify where in the Packet stream each is situated. The time the Packet can 'live' before delivery times out (this stops it circling the network forever if the destination IP Address is not found). A checksum for error detection. The Source & Destination IP Addresses and finally the contained information (Segment).

Version Num.(4-bit)	HeaderLength(4-bit)	Service Type (8-bit)	Total Length (16-bit)	25
Packet Identification (16-bit)			Packet Fragment Offset (16-bit)	
Time To Live (8-bit)	Protocol Version (8-bit)		Header Checksum (16-bit)	
Source IP Address (32-bit)				
Destination IP Address (32-bit)				
Packet Options (Variable)				
Data (Variable)				

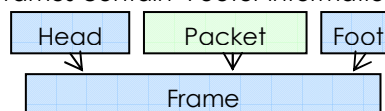
25

Data-Link Layer*Layer 1*

The Data-Link layer is the bottom layer of the TCP/IP stack; it takes Packets from the Network Layer and is responsible for converting these into PDUs that can be passed to the Physical Layer for transmission. Where as the Network Layer is responsible for delivering a message from start to finish, the Data-Link Layer is responsible for the transmission of each individual jump a message takes on route to its final destination. For example, this Network Layer transmission is then further broken down into two individual Data-Link Layer transmissions.

**Frame & Error Detection:**

The Data-Link Layer encapsulates Packets into PDU's called Frames; these are in turn transmitted across the network by the Physical Layer. Frames headers contain a source and destination Media Access Control (MAC) Address. Unlike Segments & Packets, Frames contain 'Footer information in addition to the Frame Header.



The Head contains the source & destination MAC Addresses, in addition to other required processing information. The Foot contains Cyclic Redundancy Check (CRC) check digits. These are used by the Data-Link layer by implementing a mathematical technique on them against the Frame's contents. The resulting number will indicate whether or not the Frame's binary structure has been damaged during transmission. The result of this is that this layer is responsible for the critical error detection process that ensures damaged information is not processed un-intentionally, which may lead to computer error or failure.

MAC Address:

MAC Addresses are a 'flat' addressing scheme. This means that unlike the 'logical' IP Address, the MAC address of network hardware bears no relation to where that hardware is situated on a network. Hardware is typically assigned a MAC address in the final stage of its construction. The address is 'burnt-in' to the hardware's memory and is fixed for the products lifespan. Each successive hardware component is assigned the next MAC Address in the manufacturer's library.

MAC Addresses are 48-bit binary numbers, typically formatted into six groups of two hexadecimal numbers (0-F, where the digits range from 0-9, 10 is represented by A, 11 by B and so on), separated by a colon or hyphen, for example:

94:C8:DB:91:DB:C8 or 94-C8-DB-91-DB-C8

Each hexadecimal pair has the capacity to represent numbers for 0-255, or 1 byte. Depending on the network implementation, a varying quantity of these pairs is reserved to identify the manufacturer of the device, with the rest as an individual identification number for the device itself. In Ethernet, this is typically implemented as three pairs reserved for each.

00:A0:C9:14:C8:29²⁶

²⁵ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p. Inside Cover.

²⁶ About.com Computer Networking Website, The MAC Address, <http://compnetworking.about.com/od/networkprotocolsip//aa062202a.htm>, [Accessed 22nd March 2008]

In this example, the first three hexadecimal pairs (00:A0:C9) identify the hardware manufacturer, the remainder identify the specific piece of hardware. 00:A0:C9 is one of the identifiers utilised by Intel Corporation.

Other than vendor codes, the only convention for MAC Addresses is that one consisting of all 1's is a broadcast address (similarly to IP Addresses). In its hexadecimal format, this address is FF-FF-FF-FF-FF-FF.

Address Resolution Protocol:

Unlike Packets (IP Addresses) and Segments (Port Numbers), the destination MAC Address for a Frame is not known before data transmission. In order for the Data-Link Layer to be able to encapsulate a Packet into a Frame, it must find the MAC Address associated with the Packet's destination IP Address. Address Resolution Protocol (ARP) provides the "dynamic mapping from an IP address to the corresponding hardware (MAC) address"²⁷. The term dynamic is used as the mapping is done automatically, transparently of higher layers and software.

MAC Addresses and their corresponding IP Addresses are stored in the ARP Table, a cache of records each host stores for computers it recently communicated with. Records are either static, or dynamic. Static indicates the record is fixed and will remain in the cache. These are manually-assigned by network administrators as records of computers that each host will most often communicate with. Dynamically stored records contain addresses that the computer had to resolve itself during a communication process. These addresses are stored in the cache for as long as they are deemed useful. After a timeout has passed, indicating the record has not been used for a significant time, the record is removed. This prohibits the ARP table from becoming so big that searching for records would incur a significant delay in networking speed due to increased processing time.

When a Frame needs to be sent to an unknown MAC Address, this address must be resolved. To do this, the Data-Link layer will send an ARP Request. This is a specialised Frame containing both the source MAC and IP Addresses. Its destination is the IP Address to locate, using a broadcast MAC Address. This will cause every computer on the network to read the request. If the host who is assigned the destination IP Address reads the request, it will identify it is intended for that computer and send an ARP Response to the original sender, including its own MAC Address. The sender of the request will also log the sources IP and MAC Addresses in its ARP Table. Once the response is read by the original sender, both computers will have a record of the others IP and MAC addresses. This allows Packets to be encapsulated into a Frame with a known destination MAC address and sent.

Physical (Media Access) Layer

Layer 0

The Physical Layer is not technically part of the TCP/IP stack, as the stack's responsibilities are bound by computer data structures. Instead, the stack interacts with the Physical-Layer of the computer. This is provided primarily by the network card's hardware and drivers.

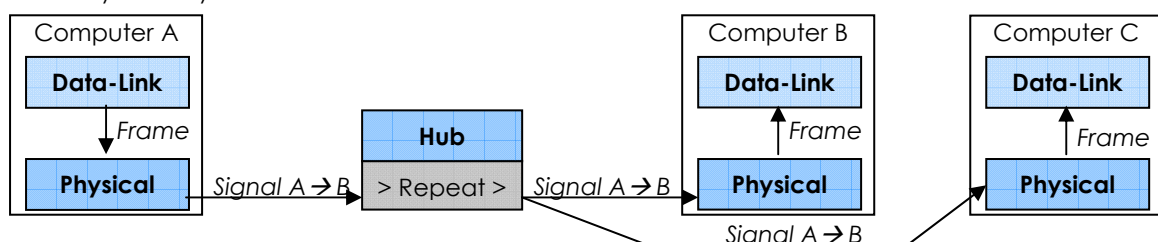
The Physical, or Media Access layer sits between the Data-Link Layer and the actual network Media on the protocol stack. It is responsible for taking Frames from the Data-Link layer and converting them into a binary signal to be transmitted across the Media. At the other end of the link, the receiving Physical Layer reads-off the signal and converts it back into a machine-readable Frame. This is then fed back into the bottom of the receiving Data-Link Layer and processed through demultiplexing at each layer. As this process is removed from the TCP/IP stack, it allows TCP/IP communications to be transmitted over any network media that drivers can be written to facilitate it. For example, TCP/IP can be sent over and between Ethernet, Wireless, optical or microwave networks. Although the process of converting Frames into signals and back again is hardware-dependant, the logic of the implemented process will not alter between network hardware.

Network Hardware

Although it is possible to create a network between two computers, this is typically not the case. Computers are connected to a network of many. It is inefficient to have a network cable connecting each computer to each other computer, so devices have been developed to manage the logical layout of a network and ensure messages are delivered to their recipients, while reducing the number of cables required to achieve this.

- **Hub**

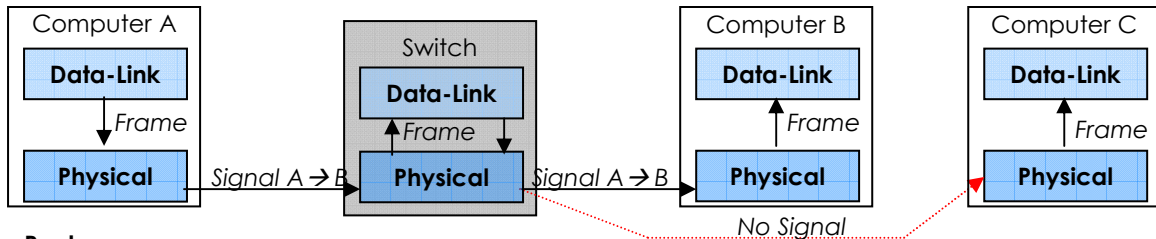
Hubs (or repeaters) are designed to simply repeat the signal they receive to every other connection on the hub. Because they do not process any data, simply mirroring the signal, they are said to operate at the Physical Layer.



²⁷ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p.54

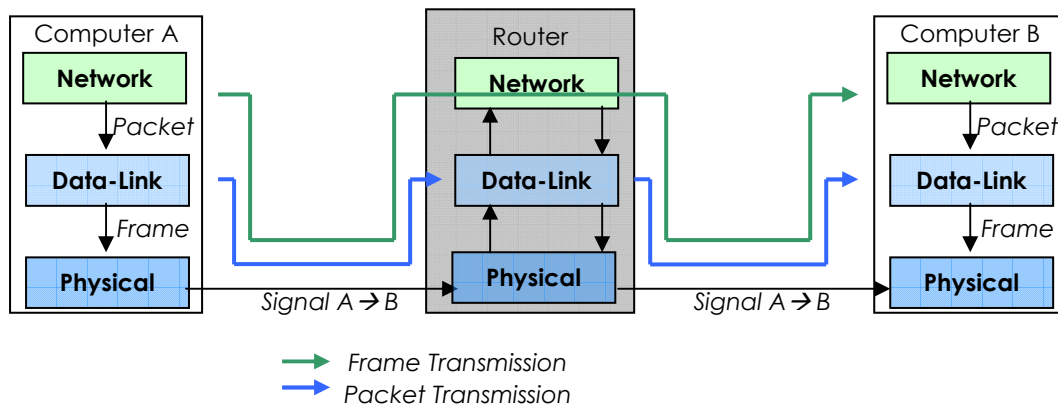
- **Switch**

Switches operate at the Data Link Layer. They receive a signal, convert it back into a Frame and then examine its destination MAC Address. Switches maintain a record of each MAC Address that is accessible through each of their physical connections, called a Switching Table. The Frame is then forwarded out of the one connection that will reach that computer.



- **Router**

These devices operate at the Network Layer. They Process messages by demultiplexing them back into Packets. Similarly to switches, they keep a record of which computers are accessible out of each connection. However, they store both MAC and IP Addresses in this table.



Computers send their Packet to the destination IP Address, however as the Data-Link layer operates over each individual jump on a Packet's journey, that Packet is encapsulated into a separate Frame at each stage of transmission.

This means that to send a Packet from computer A to B:



Because the Router acts as an intermediate device, when computer A sends an ARP request to find the MAC address associated with Computer B's IP Address, the Router will reply with a 'Proxy-ARP Response', indicating to computer A that Computer B's MAC address is in fact the MAC address on the Router that computer A is connected to. In turn, Computer B will believe that computer A's MAC Address is the MAC Address of the port it is connected to on the Router. The Router will handle the translation between the two internally.

Summary

A Summary of the processes performed at each layer:

- **Software:** Client Software Connects to Server Software and data is exchanged.
- **Application:** Provides a Socket Connection between the Client & Server.
- **Transport:** Provides a Session between the two Sockets. Each end of the Session Is identified by a Port Number. Encapsulates data into a PDU called a Segment.
- **Network:** Encapsulates Segments into PDU's called Packets. Uses IP Addresses to route Packets to their destination. Routers operate at this layer.
- **Data-Link:** Encapsulates Packets into Frames. Uses MAC Addresses and is responsible for each jump along a Packets Route. Conducts the majority of Error Detection. Switches operate at this layer.
- **Physical:** Converts a Frame into a Digital Signal to send through the network media. Hubs operate at this layer.

Existing Products

Having conducted research into existing software and demonstration techniques used to illustrate how TCP/IP works, it was not possible to find any single implementation which illustrated the entire interaction process. This indicates that the final simulated product for this project will not be appreciably similar to any existing designs. The impact of this is that there will be a reduced risk of impeding on existing designs and breaching copyright or intellectual property. Conversely, the appraisal of the simulation will be difficult as it can not be directly compared to any existing designs.

Beneficial information that was found can be broken down into three sections, documented algorithms, computer network logs and existing simulations.

Documented Algorithms:

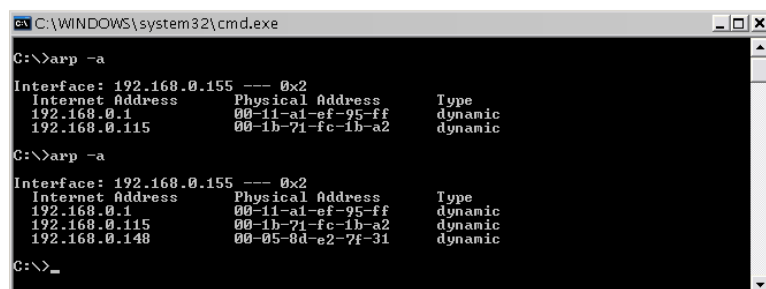
TCP/IP is an open standard; the details on how it is implemented are widely available through academic websites, publications and industry documentation. The fundamental concepts of which have been specified under Networking Protocols in the Background Research section of this report. This information will be critical to ensuring the accuracy of the simulation to the TCP/IP standard.

As outlined in TCP/IP: The Protocol Stack, the protocol standards define the interfaces and services each layer should perform, in addition to key data structures such as PDU formatting and Sliding Window design. They do not however define how the internal processing at each layer has to be conducted; this encompasses the vendor-specific implementations of the stack and means different company's implementations may be more efficient at performing the same tasks as the underlying implementation within each layer may be more efficient. The result of this is that the simulation to be designed in this project will itself represent a vendor-specific implementation. In order for it to still be valuable as a teaching tool, only the defined processes and data structures should be visible to the user. For example, any internally used algorithms that do not reflect the logical implementation of TCP/IP should not be visible within the application.

The benefits of viewing such documentation is that it provides accurate information on how TCP/IP functions. The negative aspects of this research is that reading algorithms and viewing data structures does not translate the key concepts of how to visualise such actions which is intrinsic to creating a teaching tool simulation. Presenting algorithms on their own will not function successfully as a teaching tool.

Computer Logs:

Some data structures are accessible within computer operating systems. As they provide no indication of the TCP/IP processes, they do not function as a facility to learn from on their own. They do give an indication of the data structures used within computer systems for networking processes.



```

C:\WINDOWS\system32\cmd.exe
C:\>arp -a
Interface: 192.168.0.155 --- 0x2
   Internet Address      Physical Address      Type
   -----
   192.168.0.1            00-11-a1-ef-95-ff     dynamic
   192.168.0.115          00-1b-71-fc-1b-a2     dynamic
C:\>arp -a
Interface: 192.168.0.155 --- 0x2
   Internet Address      Physical Address      Type
   -----
   192.168.0.1            00-11-a1-ef-95-ff     dynamic
   192.168.0.115          00-1b-71-fc-1b-a2     dynamic
   192.168.0.140          00-05-0d-e2-7f-31     dynamic
C:\>_
  
```

This is a screenshot of the ARP Table of a desktop computer running windows XP Professional. It was obtained by running "arp -a" from the command line. This "displays the current ARP entries by interrogating the current protocol data."²⁸ The command was run before and after connecting to another networked computer and illustrates how additional entries are added to the cache.

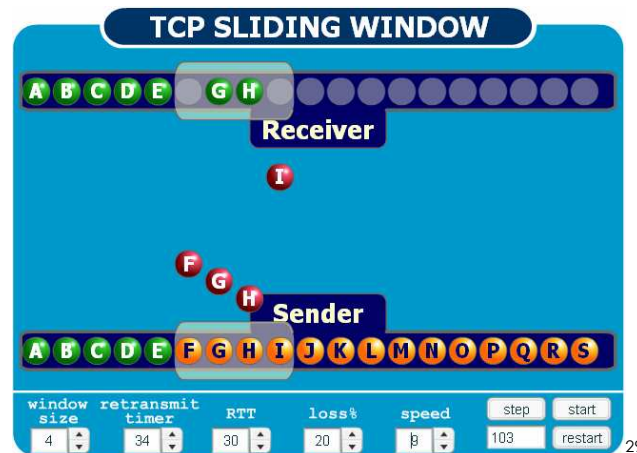
The benefits of viewing this type of source is that it illustrates how data structures change over time, although not necessarily in the most effectively portrayed format. This type of readout is also beneficial as it or similar commands are available under all major operating systems, meaning it can be run on most computers.

²⁸ Windows XP Professional Edition help comment accessed through command line using "arp -?"

Existing Simulations:

Consultation with Professors at the Computer Science department of the University of Birmingham and research in the University of Birmingham's Library revealed that there are no existing directly comparable products to this project's goal. Internet searches provided only one existing product, of similar but not exact scope of this project, a simulation of TCP/IP outbound and inbound sliding windows in operation during a Session was located.

This simulation is accessible from http://www.raduniversity.com/networks/2004/sliding_window/demo.html and is hosted under the teaching section of the RAD Data Communications company website.



This is a flash-based simulation illustrating the Outbound Sliding Window of the sender and the Inbound Sliding Window of the receiver of a Session at the Transport Layer. It does not display any technical information on how Segments are structured or how Sessions synchronize at each state of the communication process.

The good qualities of this simulation are that it has a simple, un-cluttered layout, and displays the key concepts of how sliding windows operate. The illustration style accurately portrays the covered interactions without focusing on other irrelevant variables. The display format and layout proved an effective way to present the covered concepts.

Settings relating to speed, Segment loss and window size can be altered 'on the fly'. The simulation can also be paused and resumed at any point. These features are particularly useful on a teaching tool as it allows time, for example, for a lecturer to pause the simulation and talk-through its progress up to that point, and then encourage the students to speculate as to what will happen next.

The flaws of this simulation, in relation to the goals of this project are that it only shows communication transmission logic between two sliding windows at the Transport Layer. It displays no information pertaining to a Session or its interaction with the other stack layers. Additionally, the format of the Segments is not shown. When altering the time progression in step-by-step mode, the next step has no reflection on any real-time or processing logic steps. It is simply moving the dots around the screen at a slower pace.

The highlighted beneficial techniques utilized by this simulation will be re-examined during the design phase to help structure the simulation as an effective teaching tool.

²⁹ RAD Data Communications (2008), RAD University Tutorials, http://www.raduniversity.com/networks/2004/sliding_window/demo.html [Accessed 17th January 2008].

Project Specification

The purpose of this project is to design an application to simulate a TCP/IP communication between two computers. In order to implement the simulation, a clearly defined requirements document must be constructed. This was constructed following the research phase and draws from concepts learnt. The specification will identify each of the requirements which the product must comply with in order for the finished application to fulfil the initial brief; to function as a TCP/IP teaching tool.

Product Users:

To effectively generate delivery requirements, the range of potential final users was identified. Only by knowing the intended audience is it possible to recognise the required features of the product. By identifying these users, the requirements which the product must fulfil in order to be a useful teaching tool can be better understood.

As the procedural implementation of TCP/IP is a highly technical sub-section of computer networking, it is unlikely that average pc operators will be target users for the simulation. It is more likely that this subject matter will be relevant in the fields of academia and network engineering.

It is likely that engineers will require details of the specific algorithmic implementations of TCP/IP, the overview simulation proposed would not be as beneficial to them compared with other users as it will not display the specific implementation algorithms applicable directly to hardware and software engineering. This simulated overview of the concepts, rather than technical details of specific implementations, lends itself directly to academia. Specifically for use by networking professors as a visual aid in lectures and by their students as an independent study aid, it would provide enough information to summarise the key concepts, allowing the user to conduct further reading on specific areas if required.

Requirements Elicitation:

Having identified potential end users, in order to specify requirements that will suitably fulfil their needs, a variety of elicitation techniques were identified. These techniques can be broken down into:

- **User interviews**

During the elicitation process, a group of five potential users were consulted. This group consisted of two students who were attending a networking course at the University of Birmingham at the time, two students who had completed similar networking courses previously and one networking technician trainer working at a UK-based telecommunications and signalling company. This group was selected as it represented a group of potential users (current students and networking trainer), and persons whom may have benefited from such a simulation during the course of their academic career (two graduates).

It was decided to keep the interviewees set small so as to be able to spend more time with each individual. Having a larger set would have required less flexible elicitation media such as questionnaires, with which it is difficult to obtain original ideas and perspectives. This is because questionnaires will only answer the questions the writer sets, whereas interviews are two-way communications where the conversation topic can migrate to relevant concepts, potentially unforeseen by the interviewer. These would not have been recorded using questionnaires.

The interview process consisted of speaking to each person for approximately fifteen minutes, discussing areas relating to TCP/IP that they had found most difficult to grasp, and those that were simply not explained. This process was conducted in an informal fashion as it was decided this would put interviewees at ease, potentially making their answers more honest. The use of leading questions was avoided throughout the meetings.

As the interview process was designed to gauge user opinions of what was missing from currently used learning techniques, rather than a structured data collection exercise. Statistical analysis of the results is not possible due to the collection methodology. The commonly occurring concepts from all the students were:

- Lectures on computer networking never utilised simulations to portray concepts, instead relying upon slideshows and pseudo-code summaries of algorithms to explain difficult concepts.
- In the majority of cases concepts were understood individually, but there was a lack of illustration in how each integrated with one another, and collectively how protocol theory related to actual implementation on computers i.e. where do protocols sit in relation to hardware and software.
- Textbooks on the subject material often explain the required material in a complex fashion that went beyond what the students were required to know for their module. As such, they were difficult to learn from while remaining in the bounds of the curriculum.
- Lack of explanation on how routers interact with TCP/IP communication, specifically relating to MAC Addresses at the Data-Link Layer.

These comments served as a guide for defining the system requirements. The interview with the networking technician trainer served to confirm the initial premise that such a simulation would be less beneficial to users in the engineering field, as many of the concepts displayed would already be fully understood by the technicians. However, the possibility of using such a tool to explain to clients where

faults lie on their networks was suggested. This could serve as a basis for further work upon completion of the initial implementation of the simulation.

- **Ethnography**

Previously completing an eleven-week module on computer networking provided the original design motivation for this project. The developer's personal experiences of what concepts were easily understood using traditional teaching methods, and which would benefit from a different approach, were critical in deciding the scope of the simulation. As this module had been completed several years previously to the commencement of this project, it was decided that the developer should personally attend several of the lectures on networking run within the School of Computer Science, where this simulation is developed.

Attending these lectures helped clarify teaching techniques that were most and least effective at portraying the relevant concepts, specifically areas where no teaching aids were available to aid the lecturer in explaining the subject matter. The particular areas identified during this process were:

- Illustrating how Sliding Windows function within Sessions at the Transport Layer.
- How the TCP/IP stack interacts with software on the computer.
- How the TCP/IP stack interacts with the physical network relating to signals being transmitted.

- **Prototyping**

Due to the tight and inflexible delivery deadline, it was deemed infeasible to conduct prototyping to further clarify requirements. This is primarily because the time estimated to produce a functioning prototype would place it too close to the final delivery deadline to allow significant revisions to the software.

Requirements:

After thoroughly researching the processes and interactions undertaken by computers communicating via TCP/IP, the technical aspects to be incorporated into the requirements were identified. The requirements elicitation has further highlighted specific areas of these functions which should be concentrated on to facilitate the learning of difficult concepts. These have been used to produce User Requirements.

During the development process, any previously unidentified requirements which are deemed of significant importance to the project could have been added to the specification. This would only have been conducted after a review with the Project Supervisor to confirm their relevance and that there would be enough time for them to be implemented.

User Requirements:³⁰

High-level abstract requirements of what the system should do, ranked by priority.

1. **The product must simulate TCP/IP communication between two computers.**

Priority: Mandatory

Dependencies: None

- 1.1 The simulation must include:

- 1.1.1 Software to generate data traffic between two computers

- 1.1.2 The TCP/IP stack layers

- 1.1.2.1 The responsibilities of each layer:

- 1.1.2.1.1 *The Application Layer*

- 1.1.2.1.1.1 How Sockets operate

- 1.1.2.1.2 *The Transport Layer*

- 1.1.2.1.2.1 How Sessions operate

- 1.1.2.1.2.1.1 Sliding Window functionality

- 1.1.2.1.2.1.2 Port Numbers relation to Sessions and Segments

- 1.1.2.1.2.1.3 Segment PDU structure and contents

- 1.1.2.1.3 *The Network Layer*

- 1.1.2.1.3.1 Packet PDU structure and contents

- 1.1.2.1.3.2 IP Addresses

- 1.1.2.1.3.2.1 The structure of IP Addresses

- 1.1.2.1.3.2.2 What IP addresses are allowed

- 1.1.2.1.4 *The Data-Link Layer*

- 1.1.2.1.4.1 Frame PDU structure and contents

- 1.1.2.1.4.2 MAC Addresses

- 1.1.2.1.4.2.1 The structure of a MAC Address

- 1.1.2.1.4.2.2 What MAC Addresses are allowed

- 1.1.2.2 How each layer communicates through Service Access Points

- 1.1.3 How TCP/IP interacts with the Physical Layer

- 1.1.3.1 Converting Frames into signals

- 1.1.3.2 How signals are transmitted across the network media

- 1.2 The simulation must allow the user to configure initial variables

- 1.2.1 Configure the IP Address each simulated computer uses

³⁰ Structure adapted from Ian Sommerville (2004), *Software Engineering – 7th Edition*

- 1.2.2 Configure the MAC Address each simulated computer uses
- 1.2.3 The simulated Transport Layer's Session must have configurable variables:
 - 1.2.3.1 Sliding Window size
 - 1.2.3.2 Segment Timeout Period
 - 1.2.3.3 Segment Re-Transmit Threshold
 - 1.2.3.4 Segment Size
- 1.3 The application must allow the user to damage signals being transmitted across the simulated network media
 - 1.3.1 This damage must be identified when detected
- 1.4 The simulation must progress in a stage-by-stage format, representing a small time period
 - 1.4.1 The simulation must allow the user to view all the changes since the previous stage before progressing to the next
 - 1.4.2 The simulation must have a method of automatically progressing through the time periods to illustrate TCP/IP interactions without user interaction
- 1.5 The simulation must ensure that only valid variables are used for the simulation
 - 1.5.1 Only valid IP Addresses are used where appropriate
 - 1.5.1.1 Computers can only have IP Addresses reserved for hosts
 - 1.5.1.2 Routers can only have IP Addresses reserved for routers
 - 1.5.2 Different hardware cannot have identical addresses
 - 1.5.2.1 No computer or router can have the same IP Address
 - 1.5.2.2 No computer or router can have the same MAC Address
 - 1.5.3 There must be a facility for the application to automatically generate the required initial address variables. This will allow users to view the TCP/IP simulation without requiring initial knowledge of IP and MAC Address structure and rules.
- 2 The product must display the simulated TCP/IP stack to the user.**

Priority: Mandatory

Dependencies: Requirement 1 implemented

 - 2.1 The simulation must be shown on-screen.
 - 2.1.1 The display must identify each layer of the stack individually.
 - 2.1.1.1 The display must clearly differentiate each layer of the TCP/IP stack from the others.
 - 2.1.2 The display must illustrate how data is passed between each layer through Service Access Points.
 - 2.1.3 The display must illustrate PDUs
 - 2.1.3.1 PDU structure and contents
 - 2.1.3.2 How PDUs are encapsulated and demultiplexed.
 - 2.2 The display must show all aspects of the simulation on a single screen.
 - 2.2.1 The processing methods at all layers must be displayed simultaneously.
 - 2.2.2 Where it is not possible to display it within the screen confines; detailed information on each component, should be available to the user on demand.
 - 2.3 The display must identify each component of the TCP/IP stack
 - 2.3.1 Each component must have a short explanation of its function displayed when the component is identified by the user
- 3 The product must simulate intermediate network hardware.**

Priority: High

Dependencies: Requirement 1 implemented

 - 3.1 Simulated network hardware must include:
 - 3.1.1 Physical Layer devices - Hubs or repeaters
 - 3.1.1.1 How hubs propagate signals without processing data
 - 3.1.2 Data-Link Layer devices – Unmanaged Switches
 - 3.1.2.1 How switches process Frames and forward them based on MAC Addresses
 - 3.1.3 Network Layer devices – Routers
 - 3.1.3.1 How routers process Packets and forward them based on IP Addresses
 - 3.1.3.2 Address Resolution Protocol
 - 3.1.3.2.1 How routers translate MAC Addresses of incoming Frames to outgoing Frames
 - 3.1.3.2.2 How routers send and respond to ARP Requests by-proxy

Non-Functional Requirements:**1 Usability**

- 1.1** The application must be controllable entirely from a mouse, keyboard or a combination of both. *This will ensure it can be used where not all interface peripherals are available or functional, which is a potential risk when using computers in a shared environment (e.g. lecture theatre or computer lab).*
- 1.2** If the user attempts to input invalid data, the application must inform them of why the data has not been accepted. *This will ensure the user understands why their action has not been allowed so that they can learn from it.*

2 Efficiency Requirements**2.1 Performance**

- 2.1.1** From launching the application to it being ready to use must take no longer than 30 seconds on a computer that meets the minimum system requirements. *This will ensure it can be used in a lecture environment without creating a significant disruption.*
- 2.1.2** From the application being fully loaded, it must take no longer than 10 seconds to enter in configuration information, making it ready to run the simulation. This will additionally help reduce wasted time during lectures (as with 2.1).

2.2 Space Requirements

- 2.2.1** The entire application must not exceed 1.44MB of storage space. This will allow it to be stored on a floppy disk. *By maintaining a small file size, the application can be executed from a central storage location over a network. As the application may be run over a wireless connection, ensuring a maximum file size of 1.44MB means that over the slowest typically implemented wireless connection (IEEE 802.11b 11Mb/sec, 1.375 MB/sec), the full 1.44MB can be read from the server in 1.04 seconds. This will not have a significant impact on loading times.*

2.3 Reliability

- 2.3.1** An application crash must not destabilise the system it is executed on. *As the simulation is to be implemented in the Java programming language, the compiled application will be running within the Java virtual machine. As this is an interface that is executed in the user, rather than kernel, space it is extremely unlikely that an application failure would disrupt normal computer functionality.*

2.4 Portability

- 2.4.1** The application must be executable on Windows, Linux and MAC operating systems. *This will allow it to be run on whatever platform the student or lecturer is running on.*

3 Organisational Requirements**3.1 Delivery Requirements**

- 3.1.1** The application must be complete (fulfilling the specified requirements) and tested by Monday 17th March 2008. *This will leave 4 additional days before it is demonstrated to the customer in the Presentation (outlined in the Project Management section). This time will be used to prepare for the presentation.*

3.2 Implementation Requirements

- 3.2.1** The system must be executable and fulfil the specified efficiency requirements (2.1-2.4) on the School of Computer Science's computers. This will be possible as the School's computers' hardware specifications significantly exceed the minimum requirements to execute the Java virtual machine.

3.3 Standards Requirements

- 3.3.1** The system must portray the TCP/IP protocol stack accurately to the user.
- 3.3.2** The source code must be commented. *JavaDoc will be used in the source code to explain what each method requires to execute and its function. This will allow the software to be maintained or upgraded by programmers other than the original system developer.*

3.4 External Requirements**3.4.1 Interoperability**

- 3.4.1.1** No Requirements. *As the application is executed on a single machine, it is required to fulfil no communication or security features other than those implemented by the operating system kernel of the computer it executed on.*

3.4.2 Ethical

- 3.4.2.1** No Requirements. *As the application does not access or display any private or confidential information, no ethical requirements will be implemented.*

3.4.3 Safety Requirements

- 3.4.3.1** No Requirements. *As the application does not interact with machinery that could pose a potential risk to health, nor does it display rapidly flashing images, there are no health risks associated with the software.*

Outstanding Issues – The issues identified as outstanding or unanswered by this document are:

- The availability of a computer to demonstrate the application during the software Presentation. *This is outside of the scope of the project, but not outside the responsibility of the software engineer.*
- Training. *The software will include help material to explain its core features and illustrate how to use the application. The application is intended to be viewed in conjunction with this report, which will explain the more complex concepts.*
- Setup. *The application will be written in the Java programming language. It will be compiled and packaged into a JAR file. This imitates a self-executable application on any system where the Java virtual machine has been installed.*
- Warranty. *No warranty is supplied with the application, however further support can be sought by contacting the developer. Contact details will be included within the application.*
- Maintenance. *The source code for the application will be commended with JavaDoc tags, assisting with code upgrades and maintenance by programmers other than the original developer.*

System Evolution

- It has been identified that in the future extension of the functionality of the application may be required to encompass simulation of additional computers.
In order to allow this, the application's structure must be rigidly defined and modular, allowing the addition of more computers without requiring significant rework of the existing code. Source code will be commented with JavaDoc tags to aid maintenance.
- In the future, the application may have to provide more a detailed display on individual sections of the TCP/IP stack not previously identified as areas of specific complexity or interest.
In order to allow this, the architectural model selected must clearly separate the underlying simulation from the display. This will allow modifications to the display without causing unexpected errors within the simulation model.

Requirements Validation:

User requirements of the system will be validated by a 1 week period of testing. The application will be run by the developer, the Project Supervisor and two other computer science students who have studied networking. The testing phase will be used to validate the user requirements. The simulation will be run and the functionality defined by each point will be checked-off.

Each of the non-functional requirements has been specified with quantitative validation criteria. Time in seconds was used for time-bound parameters. Specific dates have been identified for delivery deadlines. The existence of features was identified by yes/no criteria. Each of these parameters was measured and checked-off.

Any unfulfilled requirements were noted and will be discussed during the project evaluation phase.

Feasibility Study:

A feasibility study was conducted to confirm that it was possible to produce the specified application. The study focused on the available computer facilities, time considerations and risk factors.

Computer Facilities

• Running the completed application

The application was written in the Java programming language created by Sun Microsystems. In order to execute Java applications, the Java Virtual Machine (JVM) runtime environment must be installed on the computer. The minimum software requirements to run the JVM is the operating systems Solaris 7, RedHat 7.3, SuSE 8.0 or Windows 98 or newer.

The minimum hardware requirements are "Pentium 166MHz or faster processor with a minimum of 125MB free disk space and a minimum of 32MB of RAM."³¹

The JVM is available for free under the "Sun Microsystems, Inc. Binary Code License Agreement"³², so any potential user for the application will be able to install it on their machine for free with no licensing concerns.

• Application Development

To construct and test the application source code, the Netbeans Integrated Development Environment (IDE) was used. This provides coding, compilation and testing faculties within a single application. It is available for free from the Netbeans website (www.netbeans.org) under the "Lesser General Public License version 2.1 (LGPLv2)"³³

This software is available on the School of Computer Science networked computers and will be installed on the developer's home computer for the duration of the development lifecycle. The minimum system requirements vary depending on the operating system. The highest specified hardware requirements are 500 MHz Intel Pentium III or equivalent, 512 Megabytes RAM, 125 Megabytes of free disk space³⁴ All workstations at the School of Computer Science exceed these requirements by a significant margin, so no performance difficulties were expected.

³¹ Sun Microsystems Java.com Website. (2008), *Java Runtime Environment (JRE) system requirements*, <http://www.java.com/en/download/help/sysreq.xml> [Accessed 22nd March 2008].

³² Sun Microsystems Java.com Website. (2008), *Binary Code License Agreement*, http://java.sun.com/j2se/1.4.2/j2re-1_4_2_17-license.txt [Accessed 24th March 2008].

³³ Netbeans.org Website (2008) *General Public License (GPL) Version 2*, <http://www.netbeans.info/downloads/licence/nb-6.1-beta-2008-03-06-license.txt> [Accessed 24th March 2008].

³⁴ Netbeans.org Website (2008) *NetBeans™ IDE 5.0 Release Notes*, http://www.netbeans.org/community/releases/50/relnotes.html#system_requirements [Accessed 24th March 2008].

Time estimation

The project had to be completed by the deadline of 9th April 2008. There were several preceding deadlines pertaining to specific aspects of the project. These are fully discussed in the Time Management section of the Project Management chapter of this report.

It was concluded that the specified application could be completed within the defined time parameters, allowing a margin for error in case of development delays.

Risk Management

An analysis of potential risks that could have affected the success and delivery of the project was undertaken. These are specified in the Risk Analysis section of Project Management. Where risks were identified, solutions were implemented to prevent them or reduce the risk of them occurring. It was concluded that the project could proceed with these solutions in place.

Problem Analysis

Design Decisions

Modern computers are very powerful machines, capable of keeping track of millions of pieces of data concurrently. It is possible to simulate the TCP/IP stack in an identical fashion to how the processing is conducted on a real network. However, there are many sub-protocols and interacting software which are often associated with TCP/IP communication which should not be simulated. Not only would this increase the scope and workload of the project, but would also further complicate the processes to be displayed, reducing its effectiveness as a teaching tool for the fundamental concepts of TCP/IP. From a design perspective it becomes increasingly difficult to display all this information on a single computer screen in order to fulfil user requirement 2.2.

After the research stage the aspects of the TCP/IP stack that will not be included in this simulation have been narrowed down into two sections: those that are outside of the scope of this project and those that do not need to be implemented to the same level of detail. The following outlined design decisions were made in order for the application to more successfully present the logic of the simulation to the user without compromising the actual processing methodology of the TCP/IP stack.

Simulation Scope

The following items have been evaluated as outside the scope of this project. All the features outlined in this section could be added to the simulation later if they are deemed important. It is possible to do this due to the modular design of the simulated TCP/IP stack which will be implemented thoroughly.

Application Layer Protocols

The Application Layer does not provide one but many interface protocols for Software. Each of these protocols operates in a different manner, however they all communicate via the standard Application Layer interfaces. Specific details of their operation are outside the scope of this project. For the purposes of this simulation, a holistic view of the Application Layer will be taken whereby the simulation will involve a TCP/IP connection between two simulated pieces of software, communicating via socket transfer. In reality, the data communicated could be generated by any of the Application Layer protocols.

Domain Name Servers (DNS)

The DNS system is a "distributed database used by TCP/IP to map between hostnames and IP Addresses."³⁵ For example www.midnightraven.net is mapped by a DNS Server to the IP Address 193.254.210.145. It is one of the previously mentioned Application Layer protocols; however it is often thought to be an intrinsic part of TCP/IP so has been mentioned separately. For the purposes of this simulation, IP Addresses will always be used, not hostnames. This decision has been made as information about the structure and formatting of an IP Address has been defined as a user requirement in section 1.1.2.1.3.2 as an important factor to illustrate in this teaching tool. Replacing the IP Address with a word-based representation would hide the process of how IP Addresses are implemented from the user.

Dynamic Host Configuration Protocol (DHCP)

Like DNS this is also an Application Layer protocol, but has been identified as significant and will be explained separately here. DHCP is a system whereby computers joining a network are automatically assigned an IP Address from the server or router controlling the hosts on that network. It would be feasible to implement a DHCP system into the application. However, as with DNS, this would hide the process of IP Address selection from the user and reduce the quality of the learning experience. It would also require placing additional components on-screen. This would create clutter, where the goal is to create as simple a layout as possible to allow the concepts to be illustrated in an easy to understand way. Because of this, no DHCP functionality will be implemented.

Subnetting

As described in the Network Layer of the Background Research section, IP Addresses are split into Network and Host portions octet by octet. These are used to define the class of an IP Address (eg N.H.H.H is a class A address). Subnetting is a device that has been implemented on the Network Layer and allows network administrators to further sub-divide the host portion of their network logic into multiple smaller networks, each with their own hosts. This is achieved by using the original IP Address in conjunction with a similar number called a Subnet Mask.

³⁵ Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, p187.

"Subnets provide extra flexibility for the network administrator"³⁶ by providing a means to logically sub-divide an existing network. Subnetting is not required in order for the Network Layer to function and is not always implemented. Subnetting facilities will not be included within this simulation as it would create additional work load, would further complicate the simulation for new users and it is not a primary TCP/IP feature.

Physical Layer

The goal of this project is to create a simulation illustrating a TCP/IP communication process. As previously discussed, this encompasses the simulation of the four TCP/IP stack layers. In addition to this, a Software Layer must be implemented to generate data traffic for the TCP/IP stack to process. The interaction of the TCP/IP stack with the network media itself is the responsibility of the Physical Layer. Its job is to encode Data-Link Frames into signals, transmit them across the network media, then decode them back into Frames for the Data-Link Layer to process on the receiving computer. The basic processing logic of the Physical Layer does not alter from implementation to implementation, just the specific tasks required for the layer to communicate with that exact network media type (e.g. optical fibre or Category 5 cable).

The exact type of network the Physical Layer operates over is irrelevant to the TCP/IP stack's responsibilities. This is because the Physical Layer makes the delivery process transparent to the Data-Link Layer, and thus the rest of the stack. However, as the project requirements specify that the entire communication mechanism must be illustrated to the user, a Physical Layer implementation must be selected for simulation. This will be used to illustrate the Physical Layer transmission mechanisms to the user.

As TCP/IP is compatible with a large variety of network implementations, selection of a specific implementation is difficult without an existing strict set of requirements. It was decided that the most suitable selection factors would be to choose an implementation that was both simple in design and one which the user was potentially already familiar with. The simplicity of design would make it easier to illustrate the Physical Layer processing logic to the user, without having to simulate specialised techniques utilised on specific implementations (such as microwave transmissions or optical fibre sensors). If possible an implementation would be selected that users may have existing experience with. This would improve the educational value of the simulation as users would more readily relate what is being simulated with existing technologies they have experienced in the past.

The Physical Layer implementation that has been selected is Ethernet. This has been selected primarily because it is a 'simple' standard compared to more complex implementations such as wireless communication. Ethernet utilises a single communication cable connecting each computer to another computer (or other network device). The Frame is converted into a Signal and transmitted in a binary stream from one end to the other, and is then converted back into a Frame. In comparison, this implementation is of relatively low complexity, so would be simple to simulate and display, whilst still illustrating all the relevant technical processes. In addition to this, the vast majority of Local Area Networks that users are likely to have come into contact with use one of the variations of this communication standard. Previous experience can help relate what the user views on-screen to how the communication is implemented on a real TCP/IP network.

Full duplex Ethernet (data can be sent in both directions across a cable simultaneously) has been chosen over half duplex (data can only be sent one direction at a time). This has been selected primarily due to time constraints within the simulation. The process by which data can only be sent in one direction is almost identical to data being sent in both directions simultaneously; however, the simulation could potentially take far longer to send all the messages required to complete a TCP Session. Potentially, half duplex could double the time taken to send messages, slowing the process to such an extent that the user loses interest in the simulation, thereby negating its use.

Level of Detail

Items in this category have been evaluated as required in order to simulate the TCP/IP stack, though they do not need to be implemented to the same level of detail as those in an actual TCP/IP stack. These have been implemented to a lesser detail level whilst still maintaining the processing logic presented to the user. These alterations will in no way present a false version of the stack processing but will make the simulation easier for the user to understand. This will increase its effectiveness as a teaching tool.

Initial Sequence Number (ISN)

The ISN is used to number the first Segment sent by either side of a Session. By the TCP/IP standard, this number is generated by a function of the computer's main clock and incremented by 64,000 every half-second, as detailed within the Background Research section of this project. The primary reason for starting a Session with an unpredictable ISN, then amending it after a defined period of time, is to help secure the connection from TCP Sequence Prediction Attacks.³⁷ However, as these attacks will not occur within this closed simulation this is not required.

Unlike a computer built for this functionality, it would be confusing for a user to keep track of the random, rapidly increasing sequence numbers used on each Segment. It would become difficult to keep track of which Segments were sent in which order and make the simulation harder to understand. The decision has been made to start every Session's ISN at 1, and increment it each time a Segment is sent. The sequence numbers will not loop round to zero if they get to 64,000. This will make it a lot easier for the user to understand Segment transmission order as the numbers increase in a linear fashion (e.g. 1, 2, 3, 4... instead of 13,256, 27,592, 62,512,

³⁶ Cisco Systems Website (2008), <http://www.cisco.com/warp/public/535/4.html> [Accessed 26th March 2008].

³⁷ The Tech FAQ Website (2008), <http://www.tech-faq.com/tcp-sequence-prediction.shtml> [Accessed 26th March 2008].

8056). This will not compromise the logic of the Session's numbering system, but will make it significantly easier to understand the progression of Segments over the course of a Session's transmission lifespan.

Time Simulation

As the application will be simulating the two Network Layers concurrently it will have to display the data being processed within each layer and the PDUs being transmitted through the stack at the same time. Although a computer is capable of processing all this information in real-time, conducting an entire TCP/IP communication Session in seconds, it would be impossible for a user to comprehend what is being simulated at this pace. The most important design change for the simulation is to remove the real-time processing in the stack. The simulation must be presented to the user at a pace they can control, making it possible for them to read and understand the logic of what is being presented to them.

Even when able to slow down the TCP/IP stack it would still be difficult for a user to comprehend how all the processes work with the simulation presenting continuously altering data. To make it easier to see how data passes through each layer, and between communicating computers, the simulation will break down the processing into individual steps. The user can then step forwards through these time periods one at a time at their own pace. This allows the user to review everything that has changed since the previous step before continuing. This should make it easier for the user to understand everything that has been processed, enhancing the learning value of the application. As this application is intended as a teaching tool, it will also allow the teacher to progress the simulation at an appropriate pace during demonstrations, allowing them to explain each step to their students. It will also allow them to pose questions to their students, such as "what will happen next?" This simulation is intended to enhance two-way communication between teacher and student, facilitating understanding and improving the learning experience over a simple, straight demonstration that lacks the capacity for interaction. The individual time sections will allow a single piece of processing performed at each layer of the stack and for each layer to read and write some data to their neighbouring layers. The time sections will primarily be based on the step-by-step logic of the simulation, rather than a reflection on any real-world unit of time.

Error Checking

In a real TCP/IP stack error detection and correction is performed by all layers of the stack. Advanced mathematical algorithms such as Cyclic Redundancy Checks (CRC) and Hamming Code are implemented to detect errors created during the transmission process. These techniques are highly complex and could provide the basis for a project in their own right. Although they could be implemented, it is not feasible to display how these checking techniques operate within this application. The techniques for error detection fall outside the requirements for this project but error correction itself does not and so will be demonstrated to the user.

This simulation will perform a simplified error detection mechanism at the Data-Link Layer. This is the layer responsible for the majority of the error detection tasks in a real TCP/IP stack. Instead of using a complex algorithm to detect errors, Frames will contain a simple yes/no flag to indicate whether they contain an error. This will be detected at Data-Link Layer and the result displayed to the user.

Bi-Directional

In a real TCP Session it is equally possible for the Client to send or receive data with the Server as data will be passed in both directions. The goal of this Project is to simulate the method the protocol implements for the communication. The specific computer sending data (direction of communication) is not relevant to this, as it changes on a case-by-case basis. For the simulation purposes of this project, the Client will always be sending data to the receiving Server. Although the simulation is capable of sending data in both directions, the Server will only send acknowledgements to the client, not additional data. This decision was made to reduce the visual complexity of the communication process displayed on-screen. It was found to be too confusing for users to try to monitor the content of sliding windows and which acknowledgement was for which segment. As the primary goal of this simulation is as a teaching tool it was decided that creating additional complexity without introducing additional concepts would have impaired the learning experience.

Packet

As identified in the Background Research, a Packet contains many variables that are only utilised by specific sub-implementations of the Internet Protocol. As the goal of this project is to simulate the TCP/IP stack in a general form to illustrate the processes to the user, some of these variables will not be useful to display to the user. The primary reason for this is that the software will be simulating an identical Network Layer IP process on both the Client and the Host, so the specific protocol implementation variables will not be relevant. The variables selected for use in the Packet object are: Encapsulated Segment, Source and Destination IP Address, Packet Number and Packet Time to Live.

Application Architecture

Object Orientated Programming:

As this simulation will process many large data structures simultaneously, it was decided the source code should be implemented in an Object Orientated Programming (OOP) language rather than a linear one. This was selected because the OOP structures provide the capabilities for processing multiple items simultaneously. In addition, many of the facilities required for processing complex data structures and message passing are implemented natively within these languages. These features lend themselves to simulating the structure and interactions that have been specified within the requirements document.

Language Selection:

Java was selected as the programming language this software was written in. It was selected over other OOP languages as the resultant software is portable so it can be executed on a wide variety of operating systems. In contrast on a compiled application from C++ source code can typically only be executed on a single computer platform. This was an important factor in selecting the programming language as the operating systems found within the School of Computer Science and on the personal computers of Computer Science lecturers and pupils vary widely. The reasons for this variety are outside of the scope of this project. Because this software is intended to be used by precisely by this range of people, the requirement for portable code was deemed of paramount importance.

The secondary benefits of implementation in Java include availability of a wide variety of open source or General Public Licence development software. The Integrated Development Environment (IDE) Netbeans was selected as the programming platform. This application is available for free under a GNU GPL licence³⁸ for installation on the Developer's personal computer and installed on the majority of the School of Computer Science's networked machines. This allowed easy migration of software between the two sites via the use of a USB pendrive (as outlined under the Project Management section, Data Security 2.4).

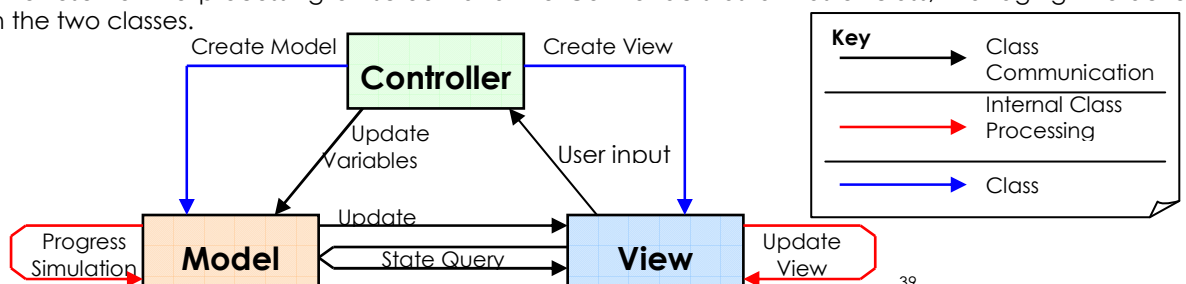
Model-View-Control:

Within this and subsequent sections, the processing of the simulation will be referred to as 'the model' and the display of the processed data will be referred to as 'the view'.

As specified in the Project Management section, although the model requirements were well-understood, the final design for the view was unknown. As a result, a phased approach was selected to develop the model, as its hierarchical structure lends itself to a modular development process. A second development lifecycle was selected for the View: the Evolutionary Model. This was chosen so the view could be implemented, evaluated, and then improved throughout the lifecycle of the development process, until its requirements were well-understood and had been implemented to satisfactory level.

In order for the view to be evaluated and changes implemented continually, the software architecture had to separate the model's processing domain logic from the view's. The intended outcome was that the view could be altered without any impact on the underlying model. This significantly reduced the development impact of view enhancements as changes only needed to be made to the view's source code. The second benefit of this implementation was that the model classes could be developed independently of the views that are to display their data.

The Model-View-Control architectural model provides this functionality. The software is partitioned into 3 class-types, the Model, View and Control. In this context, the Model provides the processing functionality, the View displays the result of the processing on-screen and the Control acts as a master class, managing interactions between the two classes.



The above diagram represents the variation of the standard MVC architectural pattern that will be implemented in this project.

In order to achieve this separation, the model classes needed to have clearly defined data structures. Each variable that may be displayed on-screen by view classes had to have a publicly accessible 'get' method, used by the view to fetch the most up-to-date value of the variable. In order for the view to detect when such a variable has changed, call its 'get' method and display the up-to-date value on-screen, the model must have a mechanism for notifying the view that a variable's state has changed. In Java, this mechanism is provided by a

³⁸ Netbeans.org Website [2008] General Public License (GPL) Version 2, <http://www.netbeans.info/downloads/licence/nb-6.1-beta-2008-03-06-license.txt> [Accessed 24th March 2008].

³⁹ Extensively adapted from original version by Fred Bradstadt (2006), Public Domain Licence, Model View Controller, <http://en.wikipedia.org/wiki/Image:ModelViewControllerDiagram.png> [Accessed 25th March 2008]

function called Observer-Observable. This functionality was acquired by importing 'java.util.Observable' & 'java.util.Observer'. These are packages available within the standard Java API.

- Observer – An Object that is responsible for a task (e.g. displaying information) when another Object changes state.
- Observable – Any object that's state may be of interest.

These facilities provide the means for one Object (the view) to be notified if the Object it is viewing (the model) has changed. When this occurs, a method is automatically run which includes the code required to process and display the updated information.

Software Engineering:

The most important aspect to the design of this application is that it must accurately reflect the structure and interactions of an actual TCP/IP stack. Processing functionality was implemented layer by layer as on a real stack. Each layer is responsible for performing the processing functionality defined by the protocol standards. Each layer was provided with only the initial parameters accessible to it on a real stack, for example a Data-Link Layer's own MAC Address but not the address of computers it will make contact with. These variables are the required information for processing to be conducted at that layer. All other information must be read from received PDUs passed between the stack layers using the defined communication protocols.

The scope of simulated variables accurately reflects those of a real stack. This has been implemented by making all variables private within Stack Layer classes. Access to variables contained in PDUs is strictly controlled by defined 'get' methods which are only accessible by the Stack Layer that PDU is currently being processed by. By employing this design, it was simpler to enforce the scope of variables and processing of each stack layer an identical manner to TCP/IP, where each layer only has access to its initially supplied parameters, and PDUs it receives from the layers immediately above and below it through Service Access Points (SAP). These design features correlate with functionality and scope of variables available through the standard OOP constructs provided by Java. Adherence to this structure automatically enforced variable scope, negating the risk of variables accidentally being used by layers that would not have access to them in a real TCP/IP stack. For the purposes of the design, the simulated software and Physical Layers will be implemented with the same structure and interfaces as the TCP/IP layers.

This structure creates a modular application, stack layers have to be capable of communicating directly with their corresponding stack layers, although this is done transparently for them by the lower layers. As the application is modular, potential errors are confined to an individual class. Future upgrades can be performed by simply removing an old class and replacing it with a newer version. No work will be required on the other classes to be compatible.

To ensure application execution efficiency, the classes were designed to perform their required processing tasks using the lowest number of computational operations as possible. This included, for example, implementing integer comparisons using Boolean operators in preference to String comparisons. Code was implemented using a standardised structure for naming variables and methods. Javadoc commenting was written for each method. The implementation of these practices is discussed in context in the Software Layer PDU Class section.

Fall-Back Positions:

It became clear to the developer during the research phase that the scope of the simulation is not clearly definable. It would be possible to continually increase the level of detail simulated by each layer and network device. Additionally, further devices and protocols could be implemented on top of the core simulation structure. For example, UDP could be implemented by simply adding functionality at the Transport Layer, as no lower stack layers would need to be changed.

The greater the scope of the simulation, the more potential benefits the simulation will be able to provide the final users. Counteracting this, the simulation could end up being so detailed that it presents too much information to the user and it becomes hard to understand. This would create a product that behaves contrary to its primary requirement to be a teaching tool.

As the entire project must be completed within a 28 week window (discussed in the Project Management section), the implementation timetable for the application was of paramount importance. If the schedule slipped some functionality may not be completed, rendering the simulation unable to fulfil some or all of its requirements. To avoid this, two steps were taken:

- **Level of Detail** - A clear level of detail was defined across the entire application to ensure it was not implemented to an unnecessary level of complexity. This helped avoid the implementation phase from taking more time than was available.
 - The structure of IP and MAC Addresses was identified as having critical importance to the simulation. This was because their structure has a direct impact on processes performed by individual stack layers.
 - The required fields for each PDU were specified. These primarily consisted of source and destination addresses, encapsulated data and any additional variables required to simulate core functionality.
 - Stack Layers' fields included each PDU read from the stack layer during the most recent time period, the processing conducted on that PDU and the PDU (if any) generated to be forwarded

to the next layer as the result. Additional information, such as the current contents of address tables and the Sliding Windows (Transport Layer only) were also required.

- **Prioritised Development Order** - A strict development order was also adhered to. This ensured critical classes and functionality was implemented as a priority. If difficulties had occurred that had adversely affected progress, the most important areas of the simulation would have been completed already, ensuring the final product was still of some value to the user.
 - *Implement Stack Layers.* The simulation of the stack is of critical importance as that is the project's primary goal. The Transport Layer encompasses the most advanced concepts to simulate, followed by the Network Layer. Lower layers are less complex as the processes they conduct are of a simpler nature.
The stack layers were implemented from the Physical Layer upwards. This allowed the simple stack layers to be implemented quickly, leaving the rest of the development period to spend on the advanced layers. This process also meant implementation lessons learnt during the development of simple classes could be drawn on during development of advanced classes, avoiding potentially time-consuming re-working to remove errors.
In a worst-case scenario where not all stack layers would have been implemented by the deadline, only the completed layers would have been used in the application. This would allow, for example, all stack processing from the Network Layer down to be simulated. Although this would not have fulfilled all the requirements of the user, limited functionality is of more use than no functionality at all.
 - *Progressive Testing* - Although each layer was implemented to communicate directly with its corresponding layer, the final version would communicate via the SAP of the layer below it. Functionality provided by previously developed layers could be tested during the development of the next layer by allowing it to communicate via the older stack layer below it. This provided a valuable period for each developed layer to be unit and integration tested during the implementation of the layer above it. This reduced the requirement for a large specific testing period on completion of the simulation, only the newest layer would need to be tested as the previous layers already had been. These features would be of benefit if the project was behind in its development progress.
 - *Simple Outputs.* Initially the model classes' functionality was tested by using simple text-based outputs. This functionality was provided by Java's 'System.out' command, allowing the developer to output data to the command line during the execution of an application. This meant that the model classes could be developed independently of their view classes. The outputs were then disabled when the model was paired with its associated view class.
If the development schedule had run over, the processing conducted by a model class could still be shown without implementing a view class. Even if not represented in the most suitable format, the core simulated processes of the model could still be illustrated, although they would not be shown in the user-friendly formatted structure provided by the view class.
 - *Customizable Variables* - The stack layers were implemented and tested using pre-defined variables initially. For example, each Network Layer had a fixed IP Address, specified within the source code, and the Transport Layer had a pre-defined message to send over the Session. This would illustrate the functionality and processes undertaken by each layer from the offset without requiring user interaction to specify variables from a view class. The ability to configure these variables was implemented later on in the development lifecycle, once all core functionality had been implemented. This was because if the development ran over, the simulation would still have been of use without user-configurable settings, which is essentially a bonus feature to enhance user interaction with the application.
 - *Intermediate Devices Last* - The priority for the project was to simulate two TCP/IP stacks communicating. Only once this was achieved, and the Client and Server computers could communicate directly over a single network cable, would the Intermediate Network Devices be implemented. This is because the simulation of these devices has been identified as not of critical importance to the user. The simulation would still be of significant academic value without them, although it would be further enhanced by illustrating how these devices interact with the stack.
 - *Code Commenting and Structure* - Throughout the application's source, code was structured in a uniform fashion. Variables were named logically, each method had Javadoc comments identifying its role in processing and complex sections of code were further described. If the simulation could not have been completed within the deadline, these features would have allowed other programmers to utilise the existing code, meaning the work completed on the project would still have potential uses.
These features, coupled with the modular nature of the application architecture, would facilitate future upgrades to the source code to simulate additional aspects of TCP/IP communication, that would be completed by programmers other than that the initial developer.
 - *View Classes* - The view classes were developed using an Evolutionary development process. Spare time remaining at the end of the implementation stage was used to further refine the layout and contents of the user interface view classes. The aim was to provide the most suitable

interface for the user to learn from the simulation. This was refined based on feedback from user-testing and the developer's personal judgement.

If all the specified requirements were fulfilled leaving spare development time, further functionality could be identified in a review session and implemented on a case-by-case basis.

Classes:

To mimic the Stack accurately, the following class types have been designed:

- **The Control Class** - This class is responsible for initialising all other classes and managing communication between them.
- **Layer Model Classes** – Each layer will be implemented in a single model class. These simulate the processing conducted at each layer of the TCP/IP Stack:
 - Software Layer
 - Application Layer
 - Transport Layer
 - Network Layer
 - Data-Link Layer
 - Physical Layer
- **Layer View Classes** – These are responsible for retrieving and displaying the data created by the Model Classes. There is one view class responsible for displaying the processing of each model class. *Each Layer Model extends Observable and has an associated Layer View which implements Observer.*
- **Bus Model Classes** – These are designed to house the PDUs passed between the Layer Model Classes during the course of the simulation, either vertically (up and down the stack) or horizontally (between the two Physical Layers). One Bus Model Class will be written and multiple instances of it created, one between each stack two layers. This reduces the amount of coding required.
- **Bus View Classes** – These are responsible for displaying the PDUs by the Bus Model Classes. *Each Bus Model extends Observable and has an associated Bus View which implements Observer.*
- **PDU Objects** – Each Model Layer has an associated PDU Object which it processes. As these objects will be displayed by either Layer or Bus view classes, they do not have their own view classes. Instead, they have two internal methods for retrieving Strings of information. These will be read by the view class and used to display an appropriate summary of the PDU:
 - Get Summary – *Returns a simple summary of the PDU (e.g. "Segment[A String]").*
 - Get Detailed – *Returns a String of detailed information about the PDU (e.g. Segment:0, Source Port:1026, Destination Port:4023, Encapsulated Data:A String).*

PDU Objects are:

- Software PDU (Software Layer → Application Layer communication)
- Application PDU (Application Layer → Transport Layer communication)
- Segment (Transport Layer → Network Layer communication)
- Packet (Network Layer → Data-Link Layer communication)
- Frame (Data-Link Layer → Physical Layer communication)
- Frame (Physical Layer → Physical Layer communication) As the signal broadcast between the two Physical Layers is just a binary representation of a Data-Link Layer Frame, the Frame Object can be reused by having a facility to provide two different 'view representations' of itself. One illustrating it as a Frame, the other in a binary format.
- **Address Objects** – For the Model Layers and PDUs to operate correctly, they require address information. Address Objects store this information. As these objects will be displayed by either Layer or Bus view classes, they do not have their own view classes. Instead, they have two internal methods for retrieving Strings of information. These will be read by the View class and used to display an appropriate summary of the Address Object.
 - Get Summary – *Returns a simple summary of the Address (e.g. IP[100.52.63.4]).*
 - Get Detailed – *Returns a String of detailed information about the address (e.g. IP Address: Decimal:100.52.63.4, Binary: 01100100.00110100.00111111.00000100, Address Class: C, Allowed: Yes).*

Address Objects are:

- Port Number
- IP Address
- MAC Address
- **Intermediate Network Device Models** – These include hardware that is placed between two or more networked computers. They are designed to allow multiple computers to be connected without requiring a network cable connecting each computer to every other. Depending on their complexity, functionality provided can include managing the flow of message traffic between specific computers based on the address structures implemented at a specific network layer. The devices include:

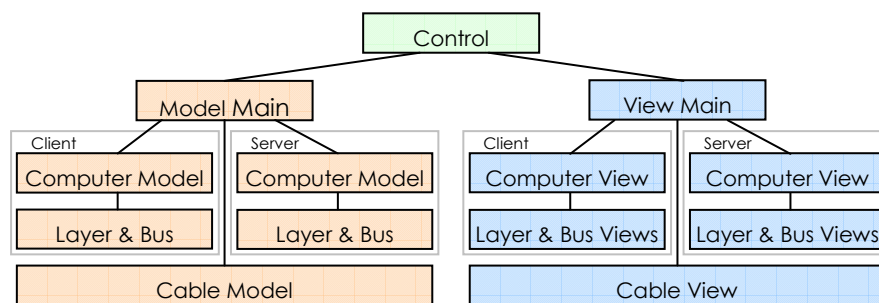
- Hub – Operates at the Physical Layer. Repeats each received signal to every other connected computer.
- Switch – Operates at the Data-Link Layer. Forwards Frames to the destination computer using MAC Addresses.
- Router – Operates at the Network Layer. Forwards Packet to the destination computer using IP Addresses.
- **Intermediate Network Device Views** – These are view classes to display the processing and internal variables used by their corresponding model classes. For each model class, a view class has been implemented. Each *Intermediate Network Device Model* extends *Observable* and has an associated *Intermediate Network Device View* which implements *Observer*.

Application Hierarchy:

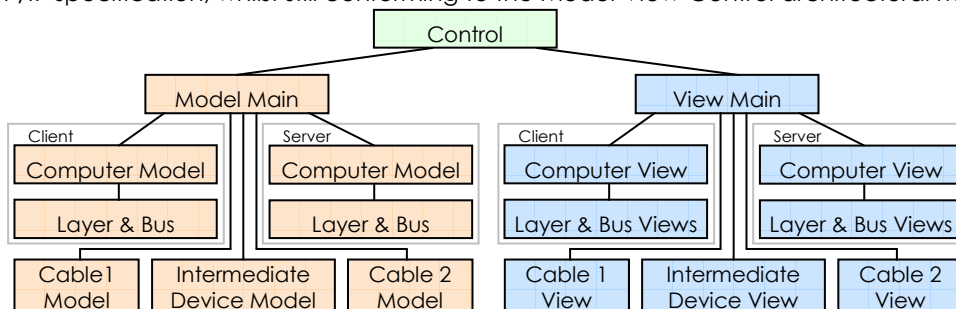
The application has been designed with a strict class hierarchy. This governs dependences and helps structure the components in a logical fashion to conform to Model-View-Control architecture.

This structure consists of:

- The *Control Class*: has a *Model Main Class* and a *View Main Class*.
 - The *Model Main Class* has a *Client Model Class* and a *Server Model Class* and a shared *Cable Model*.
 - Each of the *Client and Sever Model Classes* has a *Model Layer Class* for each of the stack layers.
 - Each of the *Client and Sever Model Classes* has *Model Communication Bus Classes* connecting each of the two neighbouring stack *Layer Classes* together.
 - Each of the *Client and Sever Model Classes' Physical Layers* has access to a shared *Cable Model*.
 - The *View Main Class* has a *Client View Class* and a *Server View Class*.
 - For each of the *Model Layer Classes* there is a *Layer View Class*.
 - For each of the *Model Communication Bus Classes* there is a *Communication Bus View Class*.
 - There is a *Cable View Class* for the *Cable Model*.



This diagram illustrates the hierarchy of the application architecture. By grouping classes together in this way, the class structure and interactions can be carefully controlled. This ensures they behave within the boundaries laid out by the TCP/IP specification, whilst still conforming to the Model-View-Control architectural model.

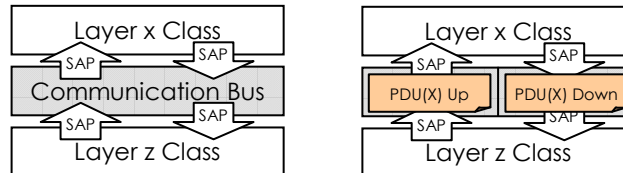


When the application is simulating TCP/IP communication via an intermediate device such as a Hub, Switch or Router, there are additional Model and View classes to simulate processing within the intermediate device. Additionally, there are two communication cables instead of one; the first to connect the Client to the Intermediate Device and the second to connect the Intermediate Device to the Server.

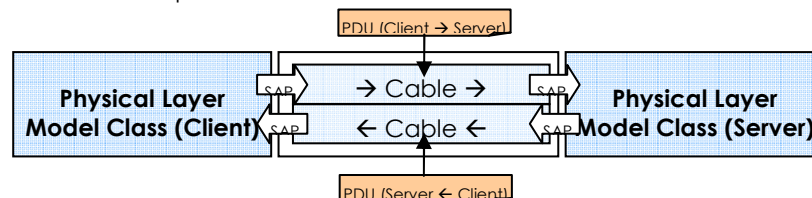
Class Interaction:

Whereas the model layers process data contained in PDUs, the buses connecting the layers together and the network cable itself do virtually no processing for the simulation. They act as communication pathways, by being a storage device for PDU Objects. PDUs are written to the bus or cable by one stack layer, then remain in the buffer until the neighbouring stack layer removes that PDU for processing. Each bus or cable has a capacity of 1 PDU for each direction of communication.

- **Communication Bus** Each layer class will communicate with neighbouring layers via Service Access Points (SAP). In order for the simulation to illustrate how data passes between layers, the SAPs will be connected to Objects that represent a communication bus between the two neighbouring layers. These can contain up to two PDUs: one PDU being sent up and the other down the stack. This will allow the user to see a PDU at each stage of it being passed between two layers. PDUs are written to and read from the communication bus via the SAPs.



- **Network Cable Model** – An Object, similar in design to the above communication bus, will connect the two Physical Layers together. Frame PDUs can be written to and read from the cable in each direction. This will simulate the Full-Duplex nature of the Ethernet Cable.



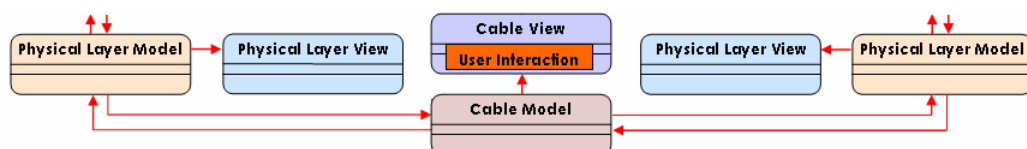
Timing interactions:

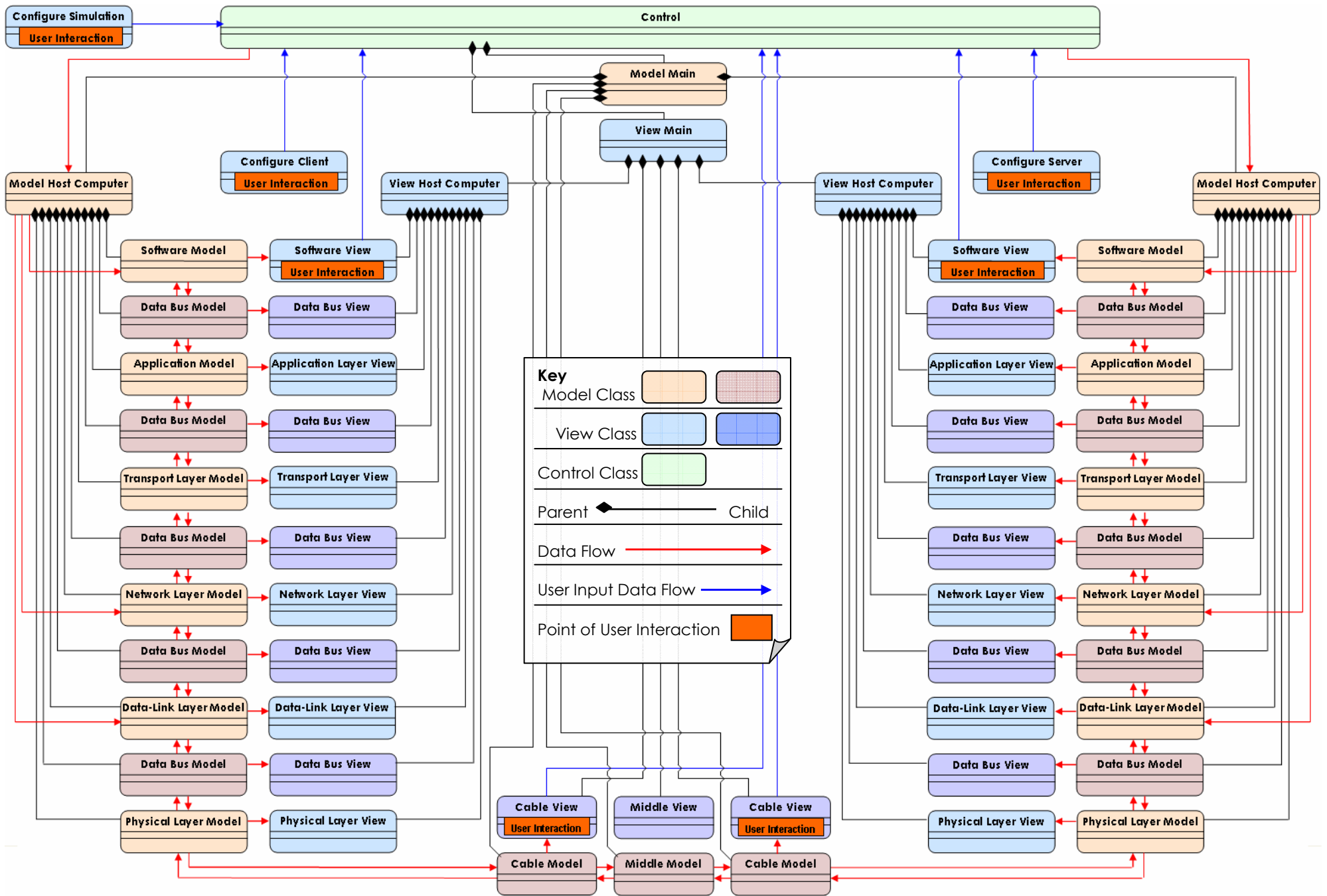
In order for the simulation to progress, each class must perform their processing tasks in a synchronised order. For example, one layer needs to write a generated PDU to a Data Bus before the other layer can read it off for processing. The simulation has been implemented in a hierarchical structure to allow this. A series of odd and even 'tick' signals are generated by the control class. These are passed to each model computer and in turn, to each of their stack layers (in addition to Data Buses, Communication Cables and Intermediate Network Devices). The 'tick' causes either a read or write method to be called on each model Object in sequence. The read method causes each Object to read off PDUs from their upper and lower Data Buses (or right/left on Physical Layer Objects). These PDUs are processed and others generated as a result. These are stored in local variables until the write method is called. This triggers each layer to write these stored PDUs to their outbound Data Buses. This process ensures that Layers communicate in sequence. This ensures processing efficiency as layers are not performing read or write tasks when there is no data to be processed.

UML:

Rather than explaining all class interactions, the diagram on the following page is a UML representation of the structure of the software, indicating dependencies and data flow of all classes when the application is simulating communication via intermediate network hardware.

The diagram below replaces the central portion of the UML diagram to represent direct communication with no intermediate network device.





Project Management

The goal of this project is to create an application and report explaining the development process undertaken to create it. This is a significant volume of work for a single developer to undertake. In order to deliver the product on deadline, appropriate project management techniques were implemented. These assessed potential risks to the project and helped work progress to a schedule. Project management also identifies development technique(s) appropriate for the application and ensure they are implemented correctly. Portions of the project management section refer to decisions made earlier in the project's lifecycle, but have been included here to collate all management decisions together.

Risk Management:

Risks are "an inherent – and inevitable – characteristic of projects."¹ As this will be the largest single project that the developer has undertaken, a lack of experience could further increase the possibility of failure. Risk assessment is critical to a project; it encompasses "obtaining a clear definition of risks, including how important that risk is to the project – what the severity of its occurrence would be, its sensitivity – and the likelihood of that risk occurring."² It is critical that an accurate risk assessment is undertaken as it will identify potential problems and allow steps to be taken in the implementation to avoid risk and reduce the impact if risk occurs.

A risk assessment was undertaken in conjunction with the project supervisor. The identified risks are rated on a scale of 1-5 for probability and impact (1 lowest, 5 highest). Their priority is then calculated by the function Probability x Impact. Appropriate steps were then taken to reduce the probability and impact of them.

Identified Risks:

Identified Risk	Probability 1-5 (Low-High)	Impact 1-5 (Low-High)	Priority (Probability x Impact)
Personal Illness	3	4	12
Data Loss	2	5	10
Programming Difficulties	3	3	9
Data Theft	1	5	5
Similar Applications	3	1	3
Hardware Failure	1	2	2

Personal Illness:

As this project is developed solely by a single engineer, the impact of personal illness would be significant. Losing time to illness would pose a serious threat, particularly as the priority for this project was to deliver the specified application by a fixed deadline. As this deadline was inflexible successful management and delivery of the project must include a degree of flexibility, providing fallback positions on deliverable requirements without compromising the technical accuracy of the simulation. These will be identified in the project specification as non-critical requirements.

Data Integrity & Versioning:

To protect against the possibility of data loss, either from hard drive failure or computer viruses, from compromising the integrity of the source files, a backup methodology was implemented throughout the lifespan of the project. An application called Microsoft Sync Toy was used at the end of every working day to create an up-to-date backup image of the computer folder containing the project files. This was copied to the hard drive of a networked computer. These backups would allow recovery of all but the most recent updates of files in event of data loss.

Weekly, the backup image would be copied to a new, dated folder on the user area of School of Computer Science's network prior to the weekly supervisor meeting. The weekly backups would provide roll-back positions in the event that an un-traceable fault in coding caused program instability. Portions of the application could be reverted to a version from a previous week.

Programming Difficulties:

As the software involves a large number of interacting program classes, it incorporates a complex architecture to logically organise the application and accurately implement suitable scope of variables. To avoid difficulties in programming, especially those found at later stages of development when class interactions are highly complex, a rigid, well-defined structure had to be defined during the design phase and adhered to. Functionality for structuring classes and managing interactions in the selected programming language had to be researched. Use of previously-implemented facilities and techniques reduced workload and complexity. In addition, by implementing appropriate existing functionality, execution efficiency was also increased.

Data Security:

¹ Field & Keller (1998), *Project Management - Risk*, p109.

² Field & Keller (1998), *Project Management - Risk*, p110.

³ Joan Jackson (2004), *An introduction to Project Management – Managing Risk*, Adapted from handout p4.

As this project is to be academically graded it is possible that students within the University, or from other academic institutions (potentially even companies wishing to utilise the code for profit) may attempt to view the application's source code to plagiarise algorithms, portions, or the code in its entirety. If source code or algorithms were copied, the project's author would be open to accusations of plagiarism. To protect the property of the University of Birmingham and preserve the designer's academic merit it is important to implement suitable security procedures. The procedures that have been implemented are:

1. To ensure the designer's full name and University student identification number is included as a comment within the source code of every file written. This will help identify any misappropriated code.
2. As application development will take place both using the School of Computer Science's networked computers and on the designer's home computer, the following steps have been taken to ensure data cannot be accessed without authorisation:
 - 2.1. All project-related files stored on the School's network will be contained within a single branch of the file structure of the designer's user account. These folders will have file permissions set on them to ensure only I have read access to the files contained within.
 - 2.2. Precautions will be taken to ensure no other users obtain access to the designer's School's network account login user name and password, as these will be required to access files outlined in 2.1.
 - 2.3. As the home-computer used for application development is not shared, less stringent security measures need to be implemented as there is little danger of the files being accessed without authorisation locally:
 - 2.3.1. To protect against the unlikely event that the files will be accessed remotely via the internet, the development computer will be protected by a router that has a hardware firewall, in addition to a software implemented firewall installed on the computer itself.
 - 2.4. When transporting files between the two locations, the files will be written to a USB pendrive. The pendrive's file system will be encrypted in by a 128-bit AES (Advanced Encryption Standard) format. "AES is a small, fast, hard-to-crack encryption standard... It has been determined as the best compromise between a combination of security, performance, efficiency, ease of implementation and flexibility."⁴ This will ensure that if the pendrive is lost or stolen, the data cannot be accessed.
3. To protect against computer viruses, a copy of ESET Nod32 Anti-Virus™ software was installed on the home development computer and kept up to date. The School of Computer Science independently maintains its own anti-virus software, protecting files on its network.

Similar Applications:

There is little justification for developing a product that is identical or inferior to previously released applications. Before the requirements for this project were finalised, research had to be undertaken to ensure the product being developed was unlike any existing applications or at least improved on features provided by existing products. If comparable applications existed, preventative steps were taken to ensure this project did not plagiarise them.

Hardware Failure:

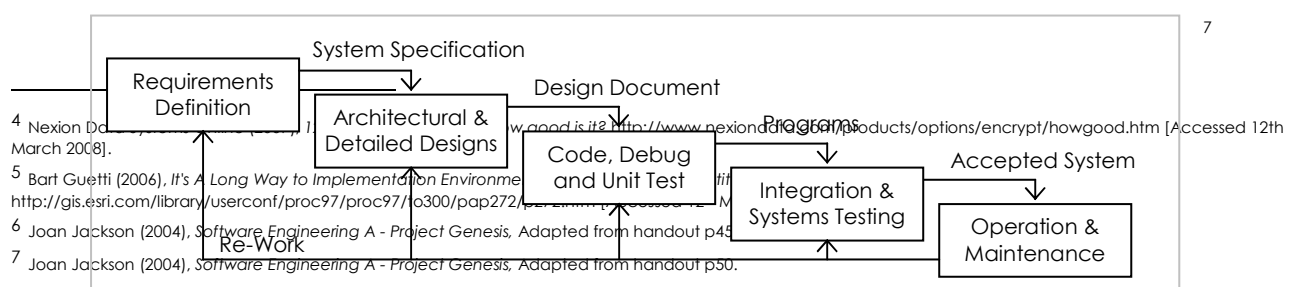
The risk of the development platform that the application and report are written on being unavailable due to hardware failure is low. This is because not only is the developer's personal computer available for work, but the School of Computer Science provides several hundred computer terminals for their students to use. Each of these computers has the required application suites and meets the minimum hardware requirements to allow work to continue. The excess of available facilities, accompanied by the previously outlined backup regime, minimises the potential impact of hardware failure seriously affecting the project progress.

Software Development Lifecycle:

For application development to be resource- and time- efficient, development methodologies were researched. "The nature of the application, and the organization developing the application, determine the model that is best suited for the situation. Also, a single model may not be appropriate for the entire application."⁵ The application was to be developed by a single individual. Initial research indicated that the structure of TCP/IP is modular, arranged in a stack. With these facts, the three most common software lifecycle models were analysed for suitability and are summarised as follows⁶:

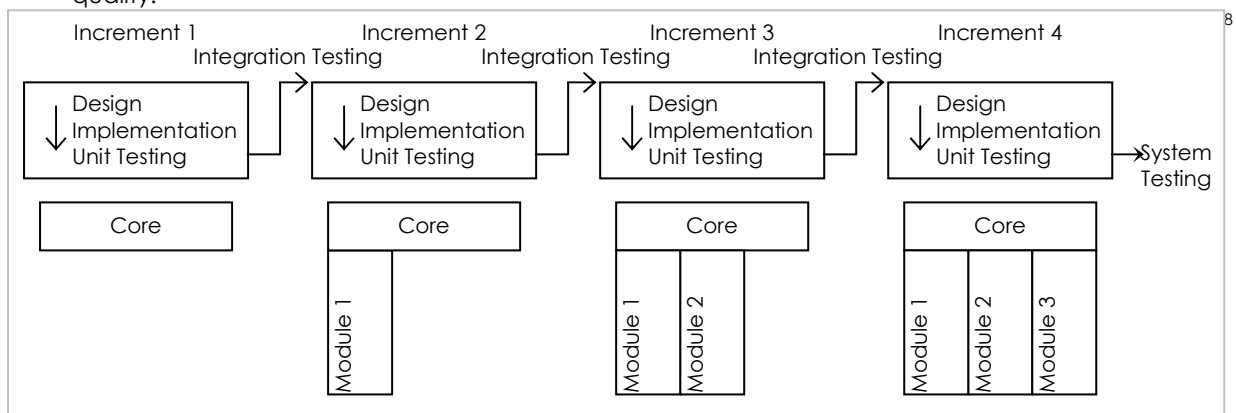
- *The Waterfall Model*

In the Waterfall Model development is split into sequential stages, where one stage must be completed to a deliverable standard before the next stage can be started. The final product can then be evaluated. If changes must be made the lifecycle flows back to where the changes are needed, then passes through every succeeding stage following that again before the product can be re-evaluated.

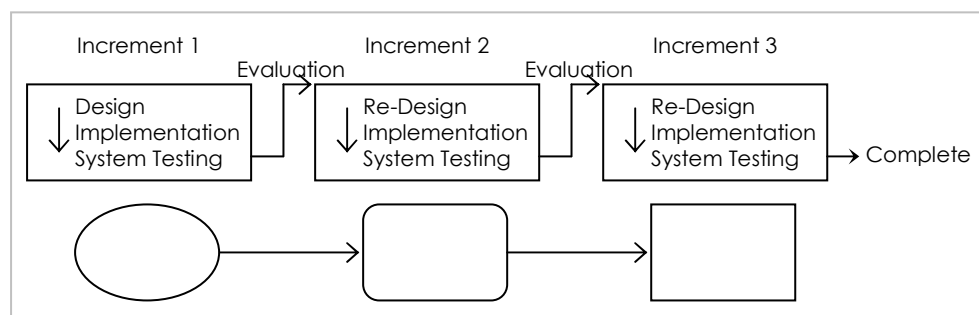


Key Factors:

- Provides a very clear framework of work to be done.
 - Iterative approach is very slow as all planning must be completed before production can commence.
 - Use is limited to projects where requirements are well-understood as changes require significant rework.
- The Incremental / Phased Approach**
The Incremental Model is useful where the project can be split into independent sub-sections, or sections with clearly defined interfaces. Each successive increment will deliver a portion of the final working system. Subsequent phases of development can implement lessons learnt in previous phases to improve quality.

Key Factors:

- Useful only where the project can be sub-divided clearly.
 - Tests modules thoroughly at each stage of development.
 - As a small development lifecycle is implemented for each increment, the overall system can take longer to develop.
- The Evolutionary Model**
The Evolutionary Model ties the development process closely to the real world. Requirements are drawn-up and a product is developed. It is then evaluated against the specification and areas for improvement are recorded. Each iteration builds on the previous one, improving areas that were not acceptable until the final product is ready for delivery. This format for development is useful where requirements are uncertain as the focus for the product can be altered with each successive iteration of development.

Key Factors:

- Each iteration improves or replaces the previous version.
 - Useful where requirements are uncertain.
 - Significant initial work has to be completed for the first increment before evaluation and re-development can be started.

Selection:

⁸ Adapted From Joan Jackson (2004), *Software Engineering A - Project Genesis*, Adapted from handout p55.

⁹ Adapted From Joan Jackson (2004), *Software Engineering A - Project Genesis*, Adapted from handout p58.

Having reviewed the three software development lifecycles it was decided that one development methodology for the entire application would be unsuitable. The 'model' portion of the application, responsible for simulating communication had clearly defined requirements and a rigid modular structure due to its close adherence to the TCP/IP protocol stack. The 'view' section had uncertain requirements because the application's front-end user interface was not finalised in the design stage. It needed a process of testing and evaluation to optimise usability.

Model Development Lifecycle:

The most suitable development methodology for the application 'model' was an Incremental Approach. This was appropriate as the structure of the 'model' adhered as closely as possible to the layered implementation of the TCP/IP protocol stack (as outlined in the research phase). Each layer of the stack had a clear processing task and well defined inputs and outputs and could be unit tested independently of the other layers using dummy inputs. Each layer of the stack was designed to communicate directly with its corresponding layer on the computer it is connected to. As each successive increment implements the next layer down the protocol stack, starting at the Software layer and working down to the Physical layer, the application could be implemented and tested layer-by-layer. Initially the two simulated Software Layers communicated directly, then the two Application layers, allowing the Software to communicate via the Application Layer functionality, then the Transport Layer and so forth.

Once each increment had been developed and unit tested, it was then integrated into the existing modules and its compatibility tested at the time. This reduced the need for a time consuming rigorous testing phase at the completion of programming. If an unforeseen event had caused the project to run out of development time, all the completed modules could have been evaluated, even if the entire 'model' had not been finished. This provided several key fall-back positions if such an event had occurred.

The developmental increments initially created a control class which managed the entire application. They then implemented the data structures (PDUs) passed between the protocol layers, and the communication media for passing these data structures (communication buses between the layers and the network cable between the host computers). Only once these were implemented was each successive protocol layer developed. The incremental stages created in turn:

1. Control Class – Manages application and creation of other classes.
2. Address Objects – Before the simulated layers could be implemented, it was important that the scope of variables available at each layer was clearly defined. By creating the IP Address, MAC Address and Port Number classes before developing the rest of the application, architectural design faults where a portion of the application would have access to variables that would be out of its scope in the real TCP/IP stack were made impossible.
3. Protocol Data Units – The objects which will be passed between each protocol layer.
4. Network Cable Class – This will act as the communication medium between the two computers. Media Access layer PDUs will be written to and read from this class.
5. Data Buses – This is the communication medium between two protocol layers within the same host computer and also has PDUs written to and read from it.
6. Simulated Software – A piece of client and server software on the two computers used by the application to simulate network traffic.
7. Application Layer
8. Transport Layer
9. Network Layer
10. Data Link-Layer
11. Physical Layer

View Development Lifecycle:

The second aspect of the application is the 'view'. This is where the data being simulated is to be displayed in a suitably formatted graphical representation of the 'model' functionality to facilitate user education. As the layout and functionality of a teaching tool are difficult to quantify within user requirements, the approach taken allowed for continual modifications to the UI to test potential improvements to the applications layout. Unlike the 'model', which is highly modular with each class having a well-defined scope, the 'view' displays the progress of the simulation in its entirety. It may have arisen that not all the information and data structures simulated within the application needed be displayed to the user in order for the application to illustrate how TCP/IP works. As such, the range of data that needs to be displayed was refined during the development process. As the 'view' requirements could not be clearly defined at this stage, and the development process called for repeated revisions of the user interface to refine the 'view', it was decided that the Evolutionary Development Model would fulfil the development requirements most successfully.

Development Phases Scheduled to implement views:

1. A user interface window, containing the outline structure of the Client and Server host computers, each with 6 vertical layers (4x TCP/IP layers plus the Media Access and Software layers), including space for displaying a cable connecting the two computers.
2. The cable connecting the two computers
3. The data buses between each TCP/IP layer
4. The Software Layer.
5. The Application Layer.
6. The Transport Layer
7. The Network Layer.
8. The Data Link Layer
9. The Media Access Layer.
10. Facilities to configure the simulation manually (e.g. configuring host IP Addresses and Session settings).

In order to create the 'view' of every model component, that component had to be completed and functioning. Only at that stage was it known what information had to be displayed on-screen. As a result, the 'view' development lifecycle was entirely dependent on the progress of the 'model' development. This development could have been run in parallel to the 'model' development, but its progress would have been based on fulfilling the model prerequisites first.

As more 'view' sections were completed, revisions had to be made to previous sections in order to maintain a coherent display format for all of the layers. Additionally, as the virtual space on-screen is limited by the resolution of the display, the area assigned to each of the vertical layers had to be modified to ensure adequate space was provided to present more detailed concepts' functionality at specific layers. The constraint of space meant that decisions had to be made over which information could be reformatted or removed from previously completed layers to allow room for more fundamental concepts. This process of development illustrates how the design evolved from a vague set of requirements.

Time Management:

Planning:

As discussed under risk management, delivery of the application on time is the most critical acceptance requirement, as serious penalties are applied for every 24 hour period of delay beyond the deadline. The time period allotted for completion of this project is twenty-eight weeks. This is split into two eleven-week University semesters, partitioned by a three-week University holiday. There is also a three-week period after the second semester before the deadline. The important deadlines set by the client were: an appraisal at week eleven, the project presentation at week twenty-five, and the report deadline at week twenty-eight.

The week eleven appraisal was an unexamined review of the project progress by a University lecturer other than the project Supervisor. This was beneficial to the developer as it provided a second opinion. The meeting helped highlight aspects of the project that could otherwise have been neglected and provided suggestions for the important focal points for the project, in order to maximise customer satisfaction on the deliverable requirements. The most important proposal highlighted during this session was that focussing on the technical accuracy of the TCP/IP stack in the 'model' sections would yield better results than spending extra time creating a more developed 'view'.

Week twenty-five required a thirty-minute presentation to be made to two University lecturers. This presentation was examined and counts towards a significant proportion of the Project's final grading. As the presentation's primary function was to demonstrate the developed application, all of the programming had to be completed and fully-tested by this point. Although the source code will not be submitted until the final deadline, it was important to have it fully commented by this point as a presentation examiner could have asked questions on the architecture of the application that the developer may not have remembered off-hand.

The deadline for the report is week 28. By this point, the report and application must be completed to the final delivery standard. The majority of the report writing will take place between weeks twenty-four and twenty eight, after completion of the source code. However, as the grading scale places significant weighting on the content of the report, far greater than the 18% (5 week) period assigned for the write-up, a log book was kept throughout the development and implementation stages. This has been used to record decisions made regarding all architectural, programming and management of the project and their justification and is being reviewed at this stage to aid in the report writing.

The project Supervisor has been available weekly for a thirty-minute meeting to discuss progress and help with technical and design difficulties each week during the two semesters. These meetings have provided two-way communication where progress can be explained and suggestions made for solutions to technical problems. It also provided the additional benefit of acting as a weekly reminder of time passing over the duration of the project. This helped the developer manage time and resource allocation in order to keep on schedule.

Critical Path:

When planning a schedule, module prerequisites and dependencies are key to scheduling the order of tasks. These relationships define what must be completed before starting an activity, what can be done once this activity has been completed and what can be done at the same time as this activity¹⁰. The culmination of

¹⁰ Adapted From Field & Keller (1998), *Project Management -Scheduling*, p186.

recording these dependencies creates the critical path, which is "the sequence of activities that takes the longest time to complete. Any delay to an activity in the critical path will cause delays to the overall project."¹¹

A heuristic explanation of the dependencies in this project is that the preparation work from the Proposal to the Requirements Definition had to be completed in sequence and before all other tasks. Then the Address, Protocol Data Unit and Data Bus classes had to be completed. Following that, each of the protocol stack layers 'model' classes could be designed, implemented and tested working from the lowest layer up the stack. Each successive 'model' had preparation design work completed on it whilst the previous layer was being completed, however implementation of the next layer could not be started until the previous layer's implementation had been completed. The 'view' of each layer could have been designed concurrently with the implementation of its corresponding 'model' layer, but similarly cannot be implemented until the 'model' it is dependent on is completed. Testing could be performed simultaneously on all modules as their result could only cause algorithmic alterations to the existing implementation, not alteration of its architecture (which would have created dependency issues if alterations to previously completed modules were undertaken during the implementation of newer modules). Development of the Control class continued until all other classes had been completed and could interact with it successfully.

*An Activity on Arrow Network has been created and is included in the appendix of this report. It was "used to identify project duration without the need to schedule against calendar time."*¹²

Estimation:

The next stage was to take the information illustrated by the Activity on Arrow Network and use it to create a real-time schedule. The exact time required to develop the application was technically difficult to quantify, so in order to create a successful plan a process of estimation was conducted on each portion (specified in the Software Development Lifecycle) of the project. This involved calculating the approximate time and resources needed (for example required hardware and reference material) to design, develop and test each section of the project. This data was then used to produce a schedule with clearly defined goals against time.

The time weighting for each section of the project was concluded after extensive research into each of the TCP/IP layers, prior to completing the project proposal and after completing a University networking module. This study was conducted initially to decide which project the developer would undertake, but served as an appropriate indication of the complexity of deliverables at each protocol layer. The developer's skill level in programming object-orientated applications in the selected language was key to estimating the development time needed to complete each section. Academic examination results and previous experience developing smaller applications were used as a guide to assessing which sections would require the largest assignment of work-time and which sections could be technically difficult to code.

The time allocation for each section was defined in weekly time units as estimating to a greater degree of accuracy would have been difficult. Due to the increased time required to complete more accurate estimations, the developer receives diminishing returns on this resource outlay as time is wasted before the development process is even undertaken. Whenever there was doubt as to the accuracy of the assessed time requirements, the period required was deliberately over-estimated. This would provide more development time if the task took longer than expected. In addition, the over-estimated time not fully expended would provide float ("The excess time available for an activity in addition to its estimated duration"¹³). This spare time could then be used later in the project's progression towards completing work that experienced unexpected delays.

A Gantt Chart has been created and is included in the appendix of this report. It was used to calculate the time to be assigned to each module of the project based on each task's estimated time weighting.

¹¹ Mike Wooldridge (Unknown), *Software Project Management*, <http://www.csc.liv.ac.uk/~mjw/teaching/soft-eng/lect05.pdf>, Lecture 5 handout, p13. [Accessed 14th March 2008].

¹² Adapted From Joan Jackson (2004), *Software Engineering A - Planning*, From handout p15.

¹³ Field & Keller (1998), *Project Management -Scheduling*, p194.

Implementation

The implementation challenge for this project was to design a way for the application to simulate all processing tasks specified at each layer of the TCP/IP stack, as defined in the research section. Each implemented layer must accept and produce PDUs of the correct format via specified SAPs. The TCP/IP specification details the processing that must be performed on these, but does not define the specific algorithms to achieve this. These are down to the responsibility of the specific vendor's implementation. For this project, it is the developer who must write algorithms to fulfil the specification's processing requirements. This section examines specific aspects of the design's implementation. It does not serve as a complete design document for the application, but focuses on areas of high complexity and those that proved difficult to translate the design into Java source code. It is presented in the order the classes were implemented.

Address Classes

Address classes are objects used by the 'model' which represent actual TCP/IP addresses. They contain only data regarding their own address and their processing capacity is limited to calculating internal variables. The each contain two methods used by 'view' classes to represent the address on-screen. The first returns a Summary String of the Address, the second returns a Detailed String. This string contains all relevant internal variables such as the address type and whether it is valid. This string is formatted in HTML to aid the view in representing the data correctly.

IP Address:

The IP Address Class creates an IP Address object from a String Parameter. This String is a text-representation of the IP Address generated by the application or supplied by the user, for example "190.54.22.12". The processing conducted within this class is primarily targeted to compute whether the IP Address is valid. If valid, the address Class and Type must be computed. These fall into the following categories:

- **Address Class:** A, B, C, D, E (All Class D & E addresses are invalid IP Addresses for hosts to hold)
- **Address Type:**
 - **Loopback** - This address returns packets to the localhost within the same machine, so is not valid as a host IP Address.
 - **Broadcast** - Broadcast addresses are used by Packets being delivered to all hosts on all connected networks, they are not valid IP Addresses for hosts.
 - **Network Address** - Network addresses are IP Addresses designated for the entire network and cannot be used to represent an individual host IP address.
 - **Local Broadcast** - Local broadcast addresses are used by Packets being delivered to all hosts on a single network, they are not valid for an individual host.
 - **Router Port Address** - By TCP/IP convention, the first available host address on a network should be reserved for the network's router's internal port. These addresses are technically valid host addresses, but should only be used by routers.
 - **Host Addresses** - These are the remaining IP Addresses, and can be assigned to host computers.

Regular Expression:

In order to perform complex analysis of the parameter String representation of the IP Address the IP Address Object should represent, a regular expression is implemented to analyse the String. This confirms whether the String supplied represents a correct set of characters in the correct order to represent an IP Address. The regular expression was written in Java utilising the regular expression facilities provided by the 'java.util.regex' package. The first regular expression implemented was:

$\Sigma:\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \bullet\}$

$$\left([1-9] \mid [1-9][0-9] \mid 1[0-9][0-9] \mid 2[0-4][0-9] \mid 25[0-5]\right) \left(\bullet([0-9] \mid [1-9][0-9] \mid 1[0-9][0-9] \mid 2[0-4][0-9] \mid 25[0-5])\right)^3$$

This regular expression yields a valid IP address of [1-255].[0-255].[0-255].[0-255], and was sufficient for validating the syntax of an IP Address.

During the testing phase, a human computer interaction-based problem was discovered that would lead to valid IP Addresses being computed as invalid. The discovered problem was that IP Addresses were being inputted with leading zeros by some users (highlighted in bold), e.g. 192.168.**015.001** instead of 192.168.15.1. These implementations were not recognised as valid by the initial regular expression, so a change was implemented to recognise addresses with leading zeros (marked in red):

$\Sigma:\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \bullet\}$

$$[0]^* \left([1-9] \mid [1-9][0-9] \mid 1[0-9][0-9] \mid 2[0-4][0-9] \mid 25[0-5]\right) \left(\bullet[0]^*([0-9] \mid [1-9][0-9] \mid 1[0-9][0-9] \mid 2[0-4][0-9] \mid 25[0-5])\right)^3$$

Once the String has been passed through the regular expression, the IP Address object can confirm whether the Address has a valid syntax. If the address does not, the IP Address object sets its *isValid* variable to false. If the syntax is valid, the address can be processed to ascertain its type.

The first stage of analysis will split up the single String into an array of 4 integers using Java's String Tokenizer ('java.util.StringTokenizer'). The Address is split up by identifying each decimal octet between the "." token. Note,

this is only possible without potentially causing an exception within Java as the syntax of the address has already been confirmed (it is not possible for incorrect characters of formatting to be entered into the Tokenizer).

e.g. **String**="79.155.134.214" → **Int Octet1**=79, **Int Octet 2**=155, **Int Octet 3** = 134, **Int Octet 4** = 214

Address Characteristics:

Once the integer array has been created, a second array is created by converting the integers into their 8-bit binary representations. (e.g. 79 is 01001111). This is performed as the logical structure of IP Addresses is only apparent when viewed in their binary format. These binary representations are then used to compute the IP Address Type with a large 'if statement'. Large if statements can become computationally inefficient if incorrectly implemented. To reduce the computational task, the structure of the statement was designed to require the fewest possible evaluations to complete the processing. This if statement will never validate more than 3 expressions deep. The first layer of the hierarchy concentrates on the first few binary digits of the first octet of the Address. A summary is presented below:

Note - 0's represent 0's, 1's represent 1's and x's represent either a 0 or 1. Address Class 'X' represents an address with an invalid class.

```

If(octet 1 starts with '0'){ Address is definitely This Host Address
  If(address = 00000000.00000000.00000000.00000000){
    Address Is: 00000000.00000000.00000000.00000000
    Address Type: This Host Address
    Address Class: X
    Is Valid Host: No
  } else if (octet 0 = 01111111){ Address is definitely Loopback Address
    Address Is: 01111111.xxxxxxxxx.xxxxxxxxx.xxxxxxxx
    Address Type: Loopback Address
    Address Class: X
    Is Valid Host: No
  } else{Address is definitely Class A NNNNNNNN.HHHHHHHH.HHHHHHHH.HHHHHHHH - 0xxxxxxxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx
    If(address = xxxxxxxx.00000000.00000000.00000000){
      Address Is: 0xxxxxxxx.00000000.00000000.00000000
      Address Type: Broadcast Address
      Address Class: A
      Is Valid Host: No
    } else if(entire address = xxxxxxxx.11111111.11111111.11111111){
      Address Is: 0xxxxxxxx.11111111.11111111.11111111
      Address Type: Network Address
      Address Class: A
      Is Valid Host: No
    } else if(address = xxxxxxxx.00000000.00000000.00000001){
      Address Is: 0xxxxxxxx.00000000.00000000.00000001
      Address Type: Router Address
      Address Class: A
      Is Valid Host: No (Router Only)
    } else{ Address is definitely Class A Valid Host Address
      Address Is: 0xxxxxxxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx
      Address Type: Host Address
      Address Class: A
      Is Valid Host: Yes
    }
  }
} else if(octet 1 starts with '10'){ Address is definitely Class B NNNNNNNN.NNNNNNNN.HHHHHHHH.HHHHHHHH - 10xxxxxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx
  address is in format = 10xxxxxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx - Individual types identified as with Class A addresses.
} else if(octet 1 starts with '110'){ Address is definitely Class C NNNNNNNN.NNNNNNNN.NNNNNNNN.HHHHHHHH - 110xxxxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx
  address is in format = 110xxxxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx - Individual types identified as with Class A&B addresses
} else { Address is definitely 111xxxx, so is invalid
  If(octet 1 starts '1110'){ Address is definitely Class D - 111xxxxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx
    Address Is: 111xxxxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx
    Address Type: Invalid
    Address Class: D
    Is Valid Host: No
  } else if(Address = 11111111.11111111.11111111.11111111){ Address is broadcast
    Address Is: 11111111.11111111.11111111.11111111
    Address Type: Broadcast
    Address Class: X
    Is Valid Host: No
  } else { Address is definitely Class E
    Address Is: 11110xxx.xxxxxxxxx.xxxxxxxxx.xxxxxxxx
    Address Type: Invalid
    Address Class: E
    Is Valid Host: No
  }
}

```

The resultant IP Addresses and their validation criteria is summarised in the Appendix – Allowed IP Addresses

Summary String: This method returns the String representation of the IP Address. e.g. "79.155.134.214"

Detailed String: This method returns an HTML formatted String summary of the IP Address.

e.g. **IP Address**

Decimal Value: 79.155.134.214

Address Class: A [N.N.N.H]

Binary Value: 01001111.10011011.100000110.11010110

Address Type: Host

Address Valid: Yes

Router Port IP:

The final facility provided by the IP Address class is used by the model to generate a router port's IP Address. The model, the router port IP Addresses should be the 'Router Address' of whichever Host IP Address they are connected to off that port. A method has been implemented that generates and returns the Router IP Address

Object of which ever IP Address that Object is. This method functions by changing the Host portion octets of the IP Address into all 0's with a 1 at the end.

e.g. Class A (N.H.H.H) - 79.155.134.214 (01001111.10011011.**100000110.11010110**) Host Address will return 79.0.0.1 (01001111.**00000000.00000000.00000001**) Router Address.

MAC Address:

MAC Addresses are processed in a similar way to IP Addresses, but have simpler rules governing their ranges. The MAC Address Object requires a text-representation of the MAC Address as a String parameter, consisting of 6 groups of 2 hexadecimal numbers, separated by a delimiter, e.g. "6E:C9:7F:B5:F0:D4". Any MAC Address that has the correct syntax is a valid address, except 00:00:00:00:00:00, which is a Loopback MAC Address and FF:FF:FF:FF:FF:FF which is a broadcast MAC Address.

If Statement:

As there are only 4 types of MAC Address, Loopback, Broadcast, Valid and Invalid, a simple 'if statement' was implemented to calculate the MAC Address type. This if statement contained 3 regular expressions to recognise Broadcast, Loopback and other valid MAC Addresses in turn. If the String was not recognised by the third regular expression, it was an invalid MAC Address.

All MAC Address regular expressions accept both upper and lower case hexadecimal letters and can use either ':' or '-' symbols as delimiters. This was implemented following the HCI issue found with leading zeros on for the IP Address recognition. This allows MAC Addresses to be valid with colon or hyphen delimiters, or upper/lowercase letters as both formats are commonly used.

Broadcast Regular Expression: $\Sigma:\{f, F, -, : \} "[fF][fF]([- | :][fF][fF]){5}"$

This expression matches only six groups of 2x f's where the f's are upper or lower case and the groups are separated by a ':' or '-'. E.g. FF:FF:FF:FF:FF:FF, ff:ff:ff:ff:ff:ff.

Loopback Regular Expression: $\Sigma:\{0, -, : \} "[00][00]([- | :][00][00]){5}"$

This expression only matches MAC Addresses consisting of all zeros and delimiters. E.g. 00:00:00:00:00:00 or 00-00-00-00-00-00.

Host MAC Address Regular Expression:

$\Sigma:\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, d, D, e, E, f, F, -, : \}$
 $"[0-9a-fA-F][0-9a-fA-F]([- | :][0-9a-fA-F][0-9a-fA-F]){5}"$

This matches all other valid MAC Addresses after the Broadcast and Loopback addresses have been identified.

Summary String: This method returns the String representation of the MAC Address. e.g. "37:7E:05:7F:6E:23"

Detailed String: This method returns an HTML formatted String summary of the MAC Address.

e.g. **MAC Address**

Hexadecimal Value: 37:7E:05:7F:6E:23

Is Broadcast Address: No

Is Loopback Address: No

Address Valid: Yes

Port Number:

The Port Number Object is a simple class that represents Transport Layer Port Numbers. It takes an integer parameter, which is the requester port number. Port numbers can be in the range of 0-65535. They are split into 3 ranges:

- 0-1023 – These are IANA Privileged Ports and are reserved for specific applications, so are not allowed by other software.
- 1024-49151 – These are IANA Registered Ports, however that can be used by any application.
- 49152- 65,535 – These are unregistered Ports and can be used by any application.

The acceptance criteria for Port Numbers is checked by a simple 'if statement', conducting range-checks on the number supplied.

```
if(Supplied Number >= 1 & <= 1023){
    Port Type = IANA Privileged
    Port Valid = No
} else if(Supplied Number >=1024 & <= 49151){
    Port Type = "IANA Registered";
    Port Valid = Yes;
} else if(Supplied Number >= 49152& <= 65535){
    Port Type = "Un-Registered";
    Port Valid = Yes;
} else {
    Port Type = "Out Of Range";
    Port Valid = No;
}
```

PDU Classes

The PDU classes are objects used by the 'model'. Similarly to PDUs, they do no processing beyond of internally stored data. Utilising Java's abstract class facility, all PDU classes extend a single master PDU class. This class contains abstract methods to:

- `getString()` - returns a String representation of the PDU
- `getDetailedString()` - returns a Detailed String representation of the PDU and its internal variables
- `getType()` - returns an integer, representing what type of PDU it is.
 - 0 = ARP
 - 1 = Frame
 - 2 = Packet
 - 3 = Segment
 - 4 = Application Layer PDU
 - 5 = Software Layer PDU

The first two abstract methods allow the view to call for information used on-screen, without having to individually identify what layer the PDU's source is. When any PDU is created, it internally generates the String & Detailed String (HTML-formatted) representations of itself that are used by the view.

The third method allows model layers to query what type of PDU it has received. This is specifically implemented at the Physical Layer (discussed later).

Each Layer's PDU contains data used to process that PDU at that specific Layer. Depending on the layer, PDU's encapsulate PDU's from higher layers, as explained in the research section of this project.

Software Layer PDU:

The Software Layer PDU is sent between the simulated Software and the Application Layer. This data the PDU encapsulates is context sensitive. This is because the meaning of the encapsulated data for the receiving Layer varies depending on the current simulated states of the Software and Application Layers.

Every Software Layer PDU contains a String variable to represent the message sent to or received by the Application Layer, in addition to variables only use some of the time. For example, IP Address, which is used when the Client Software Layer wishes to send a message to that IP address and when the Server Application Layer receives a message from that IP Address. Both of these IP Addresses are held in the same IP Address variable. This saves memory and reduces processing requirements to run the simulation.

The PDU identifies the context with which the receiving stack layer should read the PDU via a PDU Type variable. This signifies what context each Software Layer PDU should be read in. The variable is an integer, passed to the PDU Object constructor as a parameter when it is created. The PDU constructor is 'overloaded' to allow the all the data required for that PDU context to be supplied to the Object without the need for additional set methods.

To ensure the simulation is executed at an acceptable speed, integers were chosen to identify PDU type, rather than a human-friendly representation (eg: '8' instead of 'PDU LISTEN INCOMING DATA RESPONSE' as computers can evaluate a single integer far quicker than evaluating a String, which involves comparisons on a character-by-character basis. However, in the event the source code is read by programmers other than the original developer, the code must be easily understood. An arbitrary number would have no meaning to the programmer, but a String representation would. To allow easy code maintenance, specific use of Java's Object Orientated naming conventions was adopted with the Software Layer PDU class. This involved specifying integers with names for each of the possible PDU as final variables with the class source code. The keyword 'final' was used to signify to the constructor that these variables value would not alter between specific PDU Objects. Example variables were:

```
private int pduType;
private final int PDU_CREATE_SOCKET = 1;
private final int PDU_CREATE_SOCKET_RESPONSE = 2;
private final int PDU_SOCKET_SEND = 3;
```

Then whenever the PDU type was referenced within the source code, the human-friendly String version could be used instead of an integer. For example (pseudo-code):

SoftwareLayerPdu = PDU_LISTEN_CLOSE_SOCKET, not SoftwareLayerPdu = 9

This allows source code to be easily understood by humans, but when the code is compiled and executed, the processor is conducting integer comparisons, not String comparisons, so the application is executed much faster.

The software Layer PDU also contains a Boolean variable, indicating whether or not a task was successful.

E.g. `PDU_CREATE_SOCKET` (Software Layer→Application Layer) is responded by `PDU_CREATE_SOCKET_RESPONSE` True/False(Application Layer→ Software Layer) indicating success or failure of the request.

As with all PDUs, the Software Layer generates String summaries of itself, containing a user-readable representation of its variables. Example Software Layer PDU Representations:

- String = "Soft(CreateConnection)"
- Detailed String =
Application Layer PDU
Task: Create Active Connection
Server IP: 49.208.112.243
Server Port: 4323

Application Layer PDU:

The Application Layer PDU is of a similar nature to the Software Layer PDU. It contains context-sensitive variables and utilises final integer representations to signify its function. This PDU class creates Objects passed between the Application and Transport Layers.

Example Application Layer PDU Representations:

- String = "Soft(CreateSession)"
- Detailed String =
Software Layer PDU
Task: Create Session on Ephemeral Port
Socket #: 17
Server IP: 49.208.112.243
Server Port: 4323

Transport Layer PDU (Segment):

The Segment Object contains all a source and destination Port Address Object and all variables required for two Transport Layers to communicate via a Session, with the exception of a checksum variable. This has been excluded as error correction is only to be simulated at the Data-Link Layer (see Problem Analysis). These are specified in the background research section of this project. The Segment class uses Java's facility to 'overload' the constructor. This allows the Transport Layer to create Segments, passing in only the data that needs to be set.

Example Segment Representation:

- String = "Seg(ActiveOpen)"
- Detailed String =
Segment 0
Source Port: 1072
Destination Port: 4323
Encapsulated Data: ActiveOpen
PSH: True **SYN:** True **Fin:** False **Win:** True
Window Size: 4

Logical layout of real Segment:

Source Port				Destination Port			
Sequence Number							
Acknowledgement Number							
URG	ACK	PSH	RST	SYN	FIN	Window Size	
Checksum				Urgent Pointer			
Data							

Segments contain get & set methods for all of its variables. This allows the Transport Layer model to read and configure each Segment it has to process. *Segments extend the abstract PDU Object.*

Network Layer PDU (Packet):

The Packet Object's task is to encapsulate the Transport Layer Segment into a Network Layer Packet. An actual Packet contains many variables identified as out of the scope of this simulation. These have been selected as they relate to specific implementations of IP, not the general implementation this project is to simulate. Packet Object contains the encapsulated Segment Object, a Source & destination IP Addresses, the Packet Number, and a Time to Live variable.

As Packets are generated for individual Segments, they are not context-sensitive in the same way Segments are (e.g. an individual Segment will fit in a specific place in a series of multiple Segments). Because of this, unlike Segment Objects, there is no need to overload the constructor to accept specific groups of variables.

Packets contain get & set methods for all of its variables. This allows the Network Layer model to read and configure each Packet it has to process. *Packets extend the abstract PDU Object.*

Example Packet Representation:

- String = "Pakt(ActiveOpen)"
- Detailed String =
Packet
Source IP: 179.1.51.41
Destination IP: 49.208.112.243
Time to Live: 20
Encapsulated Segment: Seg(ActiveOpen)

Data-Link Layer PDU (Frame):

Unlike other PDUs, the Frame Object is used by both the Data-Link and Physical Layers. This design decision was made as the Physical Layer Signal contains exactly the same data as a Frame, but instead of being in a computerised digital format, it is in a signal format. Frames are structured in a similar way to other PDU Objects. It contains a Source & Destination MAC Address and the Packet Object it encapsulates. It contains the standard get methods for the String & Detailed String summaries for the Object. Example Frame Representation:

- String = "Frame(ActiveOpen)"
- Detailed String =
Frame
Source MAC: EF:59:86:5F:FD:B7
Destination MAC: 61:06:80:AC:5E:B2
Encapsulated Packet: Pakt(ActiveOpen)

In addition to these standard variables, two additional variables had to be generated to hold the String and Detailed String summaries of the Frame when it is to be represented at the Physical Layer. In order to illustrate to the user the way a computerised data structure such as a Frame is turned into a signal of binary digits for transmission across the network media, it was decided to create a binary representation of the Frame.

Initially implemented methodologies utilised Java functionality to serialize the Frame Object and convert it into a binary format. It was discovered that this representation created an extremely long binary representation that was infeasible to display within the space allotted on the simulation view. It was also processor consuming serialize and process these objects, slowing the simulation speed down un-necessarily, reducing the user quality of experience.

After evaluating the problem further, it was concluded that the binary representation did not have to accurately reflect the contents of the Frame. This is because the user would not be reading-off the binary version and converting it back into a Frame themselves. All the user needs to see is a Frame object being represented in a series of 0's and 1's and that the binary representation was different for Frames containing different data.

Then resultant methodology to generate the binary representation involved the Frame Object calling a Java method called hashCode(), inherited from Java.Object. This method is a way of quickly computing an Integer object representation of any object. Importantly, "whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same Integer."¹⁴ This ensured that the representation would create an identical output for identical inputs. The Integer representation was then converted into a binary format using the Integer Object's toBinaryString() method. This implementation was significantly quicker than serializing the object and produced a binary representation of an appropriate length for display by the view.

The Physical layer required similar methods for displaying the String and Detailed String representations of the Object for the view. These were implemented within the Frame class along side the standard get methods inherited from the abstract PDU class. Example Physical-Layer Signal Representation:

- String = "1000111101101000101100110"
- Detailed String =
Frame Signal
Representing: Frame(Active Open)
Binary Representation: 1000111101101000101100110

The Frame class requires one further facility not implemented by other PDU classes. This is to simulate errors. It was decided to only simulate errors at the Data-Link/Physical Layer during the Problem Analysis section. Within the simulation, a Frame can have an error applied to it whilst it is being transmitted through the network medium. This is to simulate outside interferences that may damage the signal as it is transmitted. As it has been decided not to use advanced algorithms for error detection within the TCP/IP stack, a simple Boolean variable is used in each Frame Object. This variable signifies whether a Frame/Signal has an error or not and is set to true when the user causes an error from a view on-screen control. This is passed to the network cable model which changes the Boolean variable to reflect the error. This variable is read by the Data-Link Layer, which evaluates whether or not a Frame contains an error.

When a Frame contains an error, this is illustrated to the user by altering the Detailed String representations used by the view. This allows the user to more easily track Objects of specific of interest:

- | | |
|--|---|
| <ul style="list-style-type: none"> ➤ Physical Layer Detailed String =
 Frame Signal (HAS ERROR)
 Representing: Frame(Active Open)
 Binary Representation: 100011110111000101100110 | <ul style="list-style-type: none"> ➤ Data-Link Layer Detailed String =
 Frame (HAS ERROR)
 Source MAC: EF:59:86:5F:FD:B7
 Destination IP: 61:06:80:AC:5E:B2
 Encapsulated Packet: Pakt(ActiveOpen) |
|--|---|

ARP Request/Response PDU:

This PDU is a specialised implementation of a Frame, as such has the same functionality to be processed and displayed by both the Data-Link and Physical Layers within one class. ARP requests/responses are used by the Data-Link Layer to find MAC Addresses associated with a known IP Address, as such these Objects contain the variables required to complete this task.

Routers intercept ARP Requests & Responses and send proxy-ARP requests & proxy-ARP responses in turn. To identify these differences to the model, the ARP PDU contains an overloaded constructor. One for normal ARP signals, the other for proxy-ARP signals. These constructors initialise the internal variables to reflect this.

¹⁴ Java API Website, <http://java.sun.com/j2se/1.3/docs/api/java/lang/Object.html>, [Accessed 29th March 2008]

Communication Classes

The communication classes create objects responsible for storing PDUs in. PDUs are written to the object by one model TCP/IP layer and removed by the recipient TCP/IP layer for processing.

Data Bus:

The Data Bus class has the capacity to hold any two PDU's (one for each direction of communication). A Data Bus Object joins each 2 neighbouring stack layers together, allowing them to communicate. Each Data Bus model has an associated View class to represent the data stored in it.

Network Cable:

The Network Cable model provides a similar function to the Data Bus classes, except that instead of connecting two vertical layers together, it connects two Physical Layers together. Unlike Data Buses which always accept a specific layer of PDU, the Network Cable can accept both Frames and ARP Request/Responses. In order to read and process the PUDU Object correctly, it uses the abstract method that returns an integer identifying what PDU type it is. This is found in all PDU classes as it is inherited from the abstract PDU class.

Unlike data buses which are always connected to their neighbouring stack layers, a Network Cable can be connected to one computer and the other end connected to nothing. If a signal was sent down such a cable on a real network, that electrical pulse would be lost. To simulate this, the Network Cable class automatically destroys PDU's that have been stored in the cable for more than a specified period of time. This only occurs when the cable is not connected to a second computer to read-off the PDU's.

Network Cables are a point of user interaction. When a PDU is contained within the network cable, the user can trigger the simulation to destroy or damage the PDU. This is achieved via controls integrated into the user interface view class for the Network Cable. If the user triggers the PDU to be destroyed, the class simply destroys the PDU Object. If damage is triggered, the class changes the PDU's error Boolean field to true. Both these events will be detected by the view through the Observer: Observable communication structure and the interface will be updated to bring it in line with the model.

Stack Layer Classes

The layers of the stack can be split into two primary types. The Software, Application and Transport Layers all process PDUs based on the state they are currently in. This is defined by the previous PDUs they have sent and received. The lower three layers, Network, Data-Link and Physical, all process PDU's on a case-by case basis. They do not have the same contextual-awareness that is required by the higher layers in order to perform their functional tasks.

The top three layers can be summarised in a State Machine. This is because at any given time, a state will have been directed by the previous inputs it has received and outputs it has made. The Layer can only accept contextually correct PDU inputs to process. E.g.: the Transport Layer shouldn't receive an Active Open from a Host that it is currently already connected to. The entire functionality of a piece of software and all the possible states it can be in can be summarised in a State Transition Diagram. The starting state is called S₀. All processing steps (inputs or outputs the software makes) are transitions to other states. Transitions are marked by lines on the State Transition Diagram. Some transitions will keep the layer in the same state as the next step it needs to take will be of a similar nature to the previous one.

The top three classes were designed to operate as state machines, processing received PDU's based on their current state, and changing states after a correct sequence of events has occurred. The state machines were written using switch statements that accepted integers, defining their current state. As with PDUs, these integers had a String-representation to make the code human-readable.

The most important factor when two State Machines are interacting with one another (i.e.: Client & Server) is that they have to avoid deadlock. Deadlock is "A situation wherein two or more processes are unable to proceed because each is waiting for one of the others to do something."¹⁵ An example of this is where a Socket on a Client cannot send any more Segments until it has some free space in its Sending Sliding Window, but the Server Session will not send an Acknowledgement (whose receipt would free up space in the Clients Sending Sliding Window) until it has received one more Segment for the Client. In this situation, no Session could progress any further and both would be deadlocked until the application was terminated manually. The state machines were designed to avoid this eventuality and tested thoroughly.

As a TCP/IP stack layers have been capable of acting as either a Client or Server, both these requirements functionalities have been implemented within a single class for each stack layer. The state machines have been split into two diagrams per layer to identify the different functionality provided by different sections of the same class. Both S₀ nodes represent the same start point and the state machine will progress depending on whether the user has requested an active (Client) connection or passive (Server) connection (based on input that stack layer receives).

¹⁵ Institute of Direct Marketing. (2006), *IDM Marketing Guide*, <http://www.theidm.com/index.cfm?fuseaction=contentDisplay.&chn=3&tpc=23&stp=0&glos=D> [Accessed 16th January 2008].

Each layer has publicly accessible variables allowing their associated view classes to display each PDU received, what processing has been performed on each PDU and what PDU's have been generated as a result of this.

Model Host Computer:

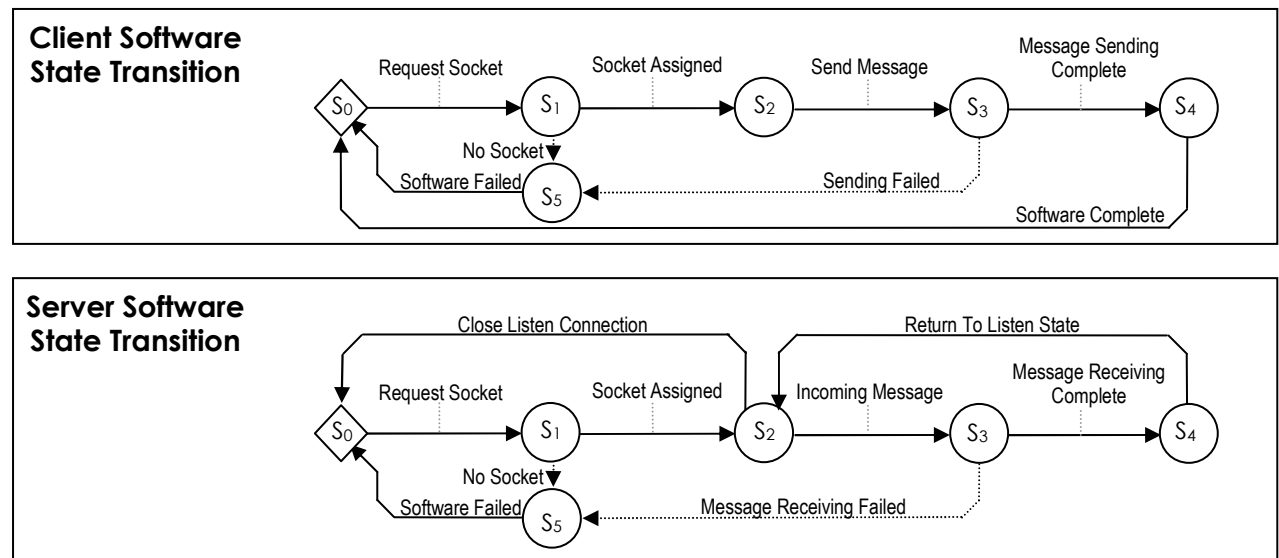
All Model components of a stack are contained within a parent class, the Model Host Computer. This class is responsible for assigning each component its initial variables. It also passes on each 'tick' from the control to its child-classes, triggering them to perform their processing responsibilities. Each Model Host Computer can be turned 'on and off' via controls accessible from the view classes. When a host is turned on, it triggers each of its child classes to reset to their default configurations. This is represented by the view classes accordingly. When the host is off, no processing is conducted by its child classes. The view will detect that the components are off and return them to a 'greyed-out' state to signify this to the user.

Software Layer:

The Software Layer is the primary interface between the user and the simulation. The class is initially configured as a Client or Server based on a Boolean flag provided by the control class. This initialises the class to perform in that role. The view of the software layer must also configure itself based on whether it is a client or server. This data is retrieved from the Software Layer model as the view initialises. The Software Layer class can perform either as a Client or Server and is only initialised for a specific task to allow the view to correctly format itself to illustrate each layer's functionality more clearly to the user.

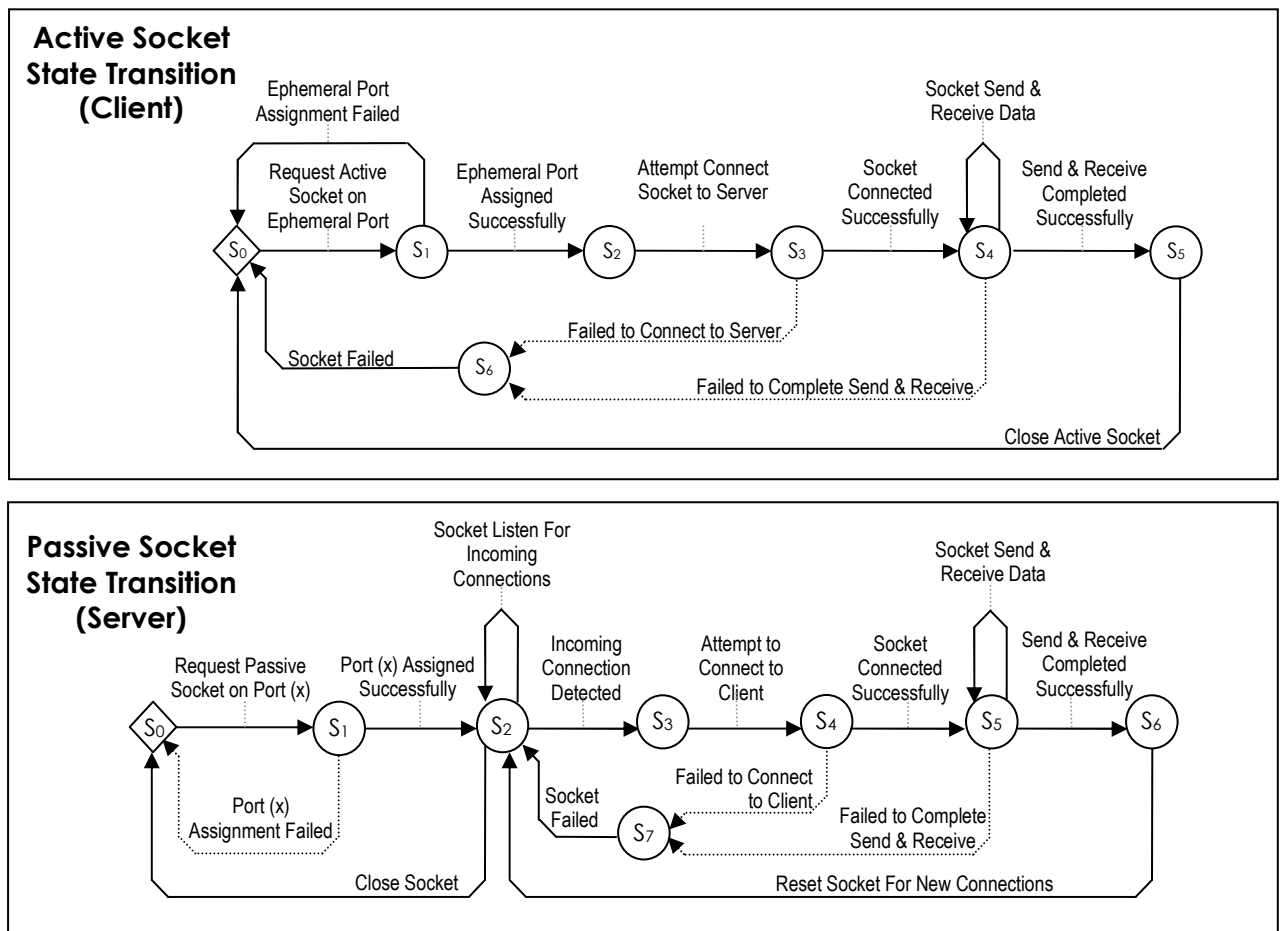
The Client Software Layer accepts a target IP Address, Port Number and message to send to that address. These variables inputted from the view class by the user. The addresses are checked for range and formatting errors to ensure they are valid. If either is invalid, an error will be reported to the user via the view.

The Server Software Layer simply accepts a local Port Number to listen for connections on from the user. As the simulation progresses, messages are passed between the Software Layer and Application layer in order to send the Client message to the Server. Progress and success or failure of this task is fed back to the user via the view at every stage via a status variable, which alters based on the current state of the Software Layer's state machine.



Application Layer:

The Application Layer acts as an interface between the Software and Transport Layers. For each piece of simulated Software trying to communicate via the stack, the Application Layer spawns a Socket Object, identified by a Socket ID. Each Socket is individually responsible for managing the interactions between the Software and Transport Layers. This allows the application model to simulate multiple pieces of Client & Server Software on each host. Although this can only be illustrated if an appropriate view is created to represent this. The specification for this project only requires one piece of software to be interacting with the stack. The model functionality has been implemented to allow multiple Sockets so simple modifications can be conducted for the simulation to provide additional functionality if required at a later date. Each Socket provides public variables which the view can read-off to identify the Socket's current state and present to the user. Messages received via the Data Bus are passed to the correct Socket based on their target Socket ID.



Transport Layer:

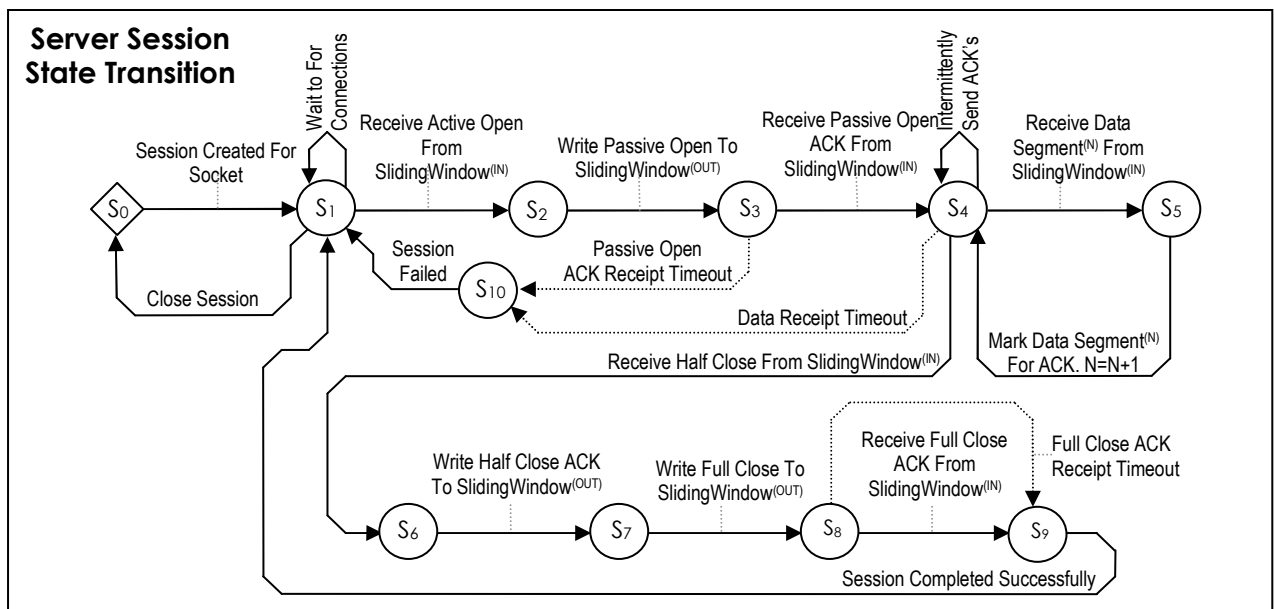
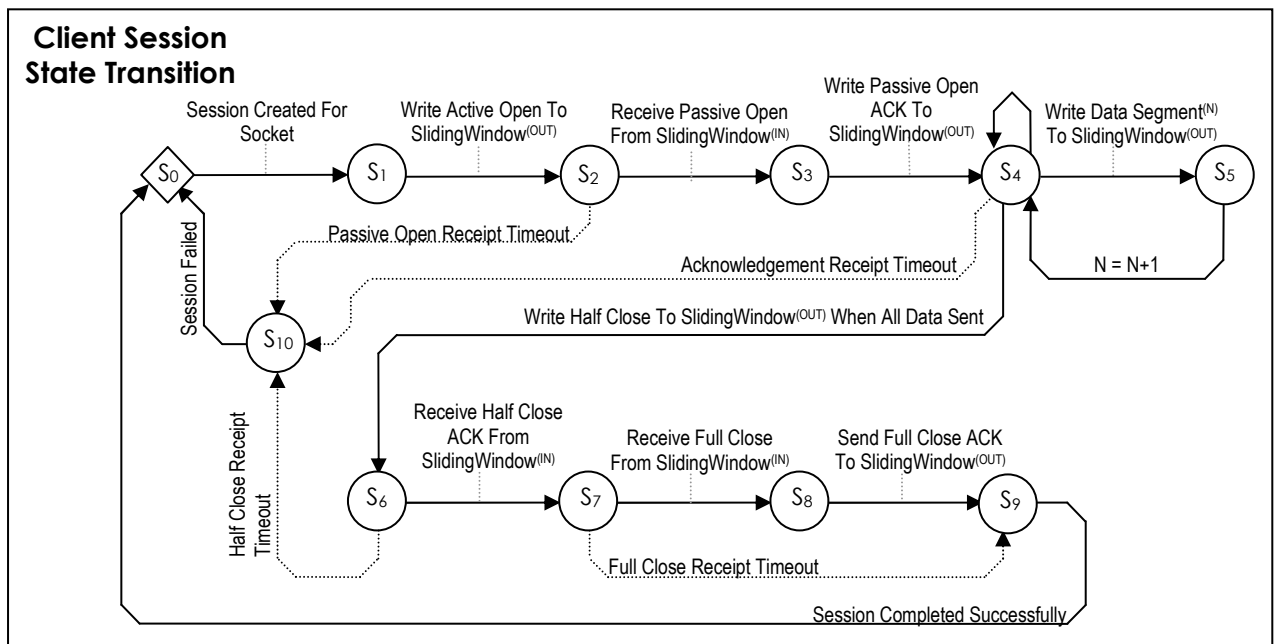
The Transport Layer is the most complex out of all the simulated stack layers as it is responsible for monitoring the progress and re-transmitting un-acknowledged Segments. As with the Application Layer, a Session Object is created to interact with each Socket Object, allowing for multiple Sessions on the same host. Sessions are identified by their local Port Number.

Server (Passive) Sessions are assigned the Port Number requested by the user at the Software layer if available, else the Session is denied as the Port is already in use. This message is relayed back to the user via the Software Layer. Client (Active) Sessions are placed on a random, Ephemeral port. This is generated randomly with the valid range of Port Numbers and is checked to ensure another Session is not currently using that Port.

Each Session is responsible for monitoring its own state and reporting this to the view. Unlike other Layers, the Sessions do not interact directly with the Data Bus connected to the layer below. Instead, each Session has two Object called Sliding Window^(In) and Sliding Window^(Out). As described in the Background Research section, these act as buffers to facilitate the efficient transfer of Segments between the interaction Session Objects (via the Network Layer).

The Sliding Window^(In) is provided with incoming Segments, read by the Transport Layer from its lower Data Bus. The correct target Sliding Window^(In) is identified according to the target Port Number of received Segments, as this relates to an individual Session. Segments to send are written to the Sliding Window^(Out), which in turn passes them to the Transport Layer for sending to the Network Layer.

When a Session intends to send a message to the Application Layer, it is passed to its parent Transport Layer which sends it to the Data Bus up the stack.



Sliding Window^(In):

As a Session is only capable of processing the next Segment it expects to receive in the transmission series, the Sliding Window^(In) is responsible for only allowing the Session to read-off and process this exact Segment. The window acts as a buffer for received Segments. It is implemented as an Array List in Java. The size of the list is kept at the window size specified by its parent Session. The window contains a variable indicating the next segment number expected. The first array position is always reserved for this Segment, the second for the next Segment and so on. When a Segment with the correct expected Segment number is received, a Boolean variable is flagged as true to the parent Session identifying that a Segment is ready to be processed. This Segment is removed from the array window and sent to the Session; the variable is flagged to false and all other Segments in the window shuffle to one space earlier in the array. The next Segment expected number is increased by 1*. If there now the next expected Segment in the first array position, the Session can read off another Segment to continue the cycle.

If a Segment is received who's Segment Number is not identical to Segment Expected (to be put in the first array position), its target position is calculated by:

$$\text{'Index Position for Segment'} = \text{'Segment Recieved.getSegmentNumber()'} - \text{'segmentNumberExpected'}$$

If the resulting index position is negative, then that Segment has been received previously and has been re-transmitted by the corresponding Session These Segments are then discarded as they have no further use. If the index position is greater than the size of the Sliding Window, the Segment cannot be stored as it exceeds the range of the buffer and is discarded.

Both these occurrences are a possibility during a TCP/IP session and occur primarily as the result of acknowledgement timeouts being triggered due to communication delay. They are side-effects of the transmission latency of the connection or the result of Segments being lost in transmission.

**The Segment expected number is only increased when Segments are received that contain PSH data. Acknowledgement Segments do not increase the Segment expected number as under TCP as these Segments do not cause the Segment number to be increased. This allows for multiple Acknowledgements to be sent and only some of them to be received successfully, without causing an error in the Session's counting mechanism. This is implemented as each ACK sent supersedes all previous ones, as it identifies the Segment number received and all previous Segments to that. If an ACK is lost, and a newer one subsequently received, it does not matter that the previous was lost, so the Session does not need to receive it before it can progress. This is implemented within the TCP/IP specification.*

Sliding Window^(Out):

The Sliding Window^(Out)'s function is to manage re-transmission of Segments whose delivery has not been successfully acknowledged. These provide the crucial role of creating a reliable delivery system over an unreliable network. The Sliding Window^(Out) is implemented as an Array List of Sliding Window Objects. Each Object contains a Segment and the variables required to monitor its delivery state. These are the number of times it has been sent, its age and whether it has been acknowledged.

A Session can only send Segments if there is room in this window (its size is specified by the Session). When a Segment is added to the window it is sent. From this point on, its age is increased by 1 for each time period the simulation progresses through. When a Segment is acknowledged, its ACK flag is changed from false to true. This segment is removed from the window during the next time period simulated. Acknowledgment signals are sent to the Sliding Window^(Out) from the Session, who received the acknowledgment from the Sliding Window^(In). Segments remain in the window once acknowledged for one time period so the user can see the Segment in has been acknowledged in the view. This is implemented to make the process clearer to the user, where as it would be immediately removed in an actual TCP/IP Session to create sending space as soon as possible.

If a Segment's age gets to the maximum age (specified when the window is created), the Segment is re-transmitted and its send count increased by 1. This allows Segments lost in transmission the opportunity to be received by their destination Session, creating a reliable delivery of data. Segments can only be re-transmitted up to a maximum threshold. This is implemented to stop Sessions getting stuck in deadlock, whereby they repeatedly re-send Segments to a computer that is unreachable, typically due to network media failure, and get stuck in a loop. Once a Segment has been re-transmitted up to the maximum threshold without being acknowledged, an error is reported to the parent Session, which in turn shuts itself down, reporting the error to the above layer (in turn this is reported to the user via the software). Server Sessions that experience communication failure are reset to their initial state, waiting for new connections, rather than closing entirely.

Network Layer:

Unlike the top three-layers, the functionality provided by the Network Layer is simple. Once a Session is created at the Transport Layer, it sends a message via the Data Bus, indicating the Session's local Port Number and its associated target IP Address. These two variables are stored in the Address Table, an array containing Address Table Objects, each with the Port and IP Address. Each Segment sent received by the Network Layer has its Source Port Number read; this is used to encapsulate the Segment into a Packet. The Packet's destination IP address will be found by searching the Address Table for the Segment's source Port Number. Its source IP Address is the IP Address assigned to the Network Layer by the control class.

Packets received are read by the Network Layer. If their destination IP Address is a broadcast address, Loopback address or an address equal to the Network Layer's own IP Address, the Packet is processed, if not it is discarded as its destination is not that computer. To process a Packet, the Network Layer demultiplexes it by removing the encapsulated Segment. This is then passed to the Transport Layer for processing by its associated Session. If no Session exists for the target Port Number, the Segment will be discarded by the Transport Layer.

Data-Link Layer:

The Data Link Layer behaves in a similar way to the Network Layer. It receives Packets from the Network Layer and encapsulates them into Frames, using a source and destination MAC Address. Destination MAC Addresses are stored with their corresponding IP Addresses in the ARP Table, an array containing ARP Objects. Each of these Objects contains a Destination IP Address and its associated MAC Address.

Unlike the Network Layer's IP Addresses, destinations MAC Addresses are not known prior to connection. When a Packet is to be encapsulated into a Frame and the destination MAC Address has not been identified and stored in the ARP Table, the Packet is cached. An ARP Request is sent to a broadcast MAC Address. When an ARP response is received, the MAC Address is added to the ARP table and the Packet can be encapsulated into a Frame. It is then sent to the Physical Layer via the Data Bus. As with real TCP/IP, the simulation automatically removes entries in the ARP Table that have not been utilised after a specified amount of time.

Any PDU's the Data-Link Layer receives from the Physical Layer first have to be identified. This is because unlike all higher stack Layers, which only accept one type of PDU from the layer below, the Data-Link Layer can receive Frames or ARP Requests/Responses. The Data-Link layer initially treats all received PDU's as abstract PDU Objects, it then calls the abstract PDU's `getType()` method, returning an integer identifying the PDU type. The abstract PDU can then be cast into either a Frame or ARP PDU based on the result and processed accordingly. The Data-Link Layer only accepts PDU's with a destination MAC address equal to the Data-Link Layer's MAC address or a

broadcast address. If the Data-Link layer receives an ARP request intended for that computer, it is responsible for generating and returning an ARP response. This is done automatically.

The Data-Link Layer is also responsible for error detection. All received PDU's Boolean error flags are checked to see if they are true. If false, the PDU is demultiplexed into a Packet and sent to the Network Layer. If true, the PDU contains an error. Status fields are updated to indicate this to the user via the view. The PDU is then discarded and it cannot be read.

Physical Layer:

The Physical layer's processing requirements are extremely limited. This is because they operating using the existing Frame objects. No encapsulation or demultiplexing needs to be simulated as it is not required. This Layer reads and writes Frames to the Communication Cable. It also provides the view class with a space to represent the Frame in its binary (Physical-Layer) format before it is transmitted or once it is received from the Communication Cable.

Intermediate Network Device Classes

Hub:

The Hub is a simple Physical-Layer device that sits between two other devices; in this case, between the Client and Server's Physical Layers. It reads off Frame/ARP PDU Objects from one Communication Cable and re-transmits them down the other.

Switch:

Switch Objects also sit between the two Communication Cables. They operate at the Data-Link layer. Each switch has its own Switching Table. This is an Array containing MAC Address entries and the physical port on the Switch that each address is accessible from. This is used to decide which port to transmit received PDU's from. Although the Switch in this simulation has only 2 ports, the Class is designed to operate with any number of ports, similarly to a real network Switch.

Switching tables start empty; MAC Addresses are added as PDUs are received by the Switch. As the Switch initially does not have a record of both computer MAC Addresses, it assumes a 'default port' for unknown destination MAC Addresses. This will be the opposite port from the port through which the PDU was received.

Router:

Routers operate at the Network Layer of the stack, they also conduct the Data-Link layer processing to encapsulate and demultiplex Packets to this layer. They operate similarly to Switches as they only forward PDU's out of the correct physical port to reach the destination, except they decide this using IP Addresses and the PDUs they send are Packets. IP Addresses and MAC Addresses are stored in a Routing Table array, along with the associated physical port on the router. The important difference between a switch and router is that the router receives one Data-Link Layer PDU, removes the encapsulate Packet and creates a new Data-Link Layer PDU for the next part of the Packet's journey, as it does during a real TCP/IP connection.

As Router's physical ports have their own IP and MAC Addresses, a Frame is sent to its destination via the Router will have their destination MAC Address equal to the Router port's MAC Address. The Router will then demultiplex the Frame and create a new Frame with the original Packet, but the correct destination MAC Address for the next leg of its journey.

Routers are also responsible for relaying ARP requests and responses. Computers sending one via the Router will receive the Routers MAC address instead of the MAC address of the target PC in their ARP response. Before the Router can respond however, it must generate its own ARP Request for the target computer. Once the Router has received a response, so knows the target computer's MAC Address, it will send a response to the original computer.

View Classes

Overview

As previously explained, every model class for the stack layers and communication buses has an associated view class. These are responsible for displaying processing conducted and PDUs contained within each model class. These views are implemented as Java JPanels within the simulation. Each JPanel implements Observer and monitors its associated model class. Whenever a model class's state changes, it is responsible for calling an update method. This notifies the view, triggering it to refresh its displayed data by retrieving the public variables of the model class.

View classes were developed using an evolutionary design methodology. They proved much simpler to implement than model classes. This was because all the computations are processed by the model, the view simply had to display the variables in an appropriate fashion. All the methods and functionality to access these variables was implemented in the model classes or by the Java Observer:Observable feature.

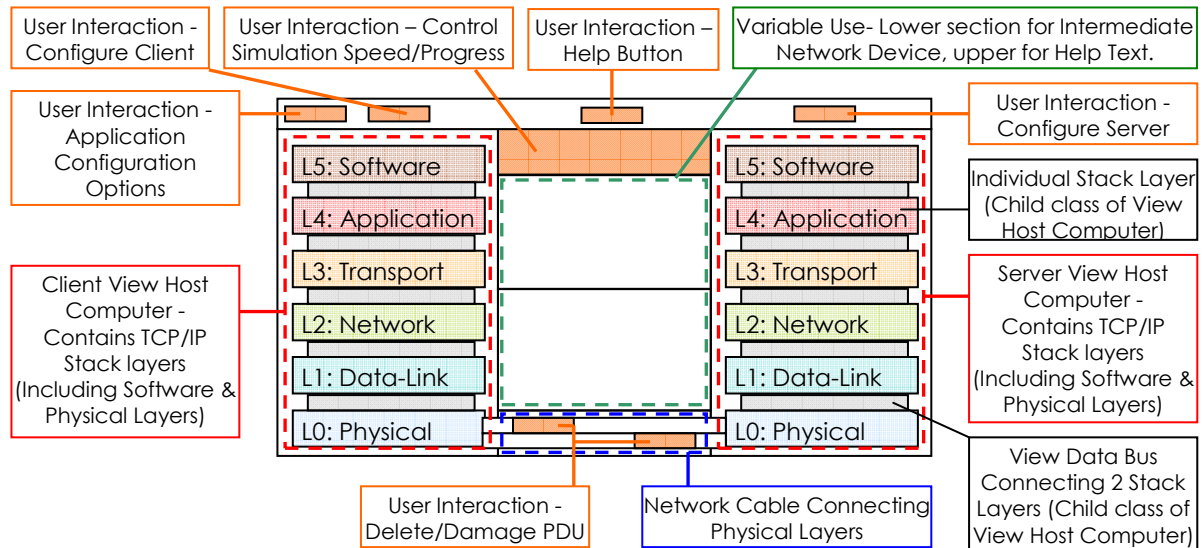
- The first stage was to create a master frame. This provided a structural framework which could then be populated by the individual view panels, representing the simulation components. When each model class was completed, a view class could be implemented for it. Initially each stack layer was assigned an identical amount of screen space to represent its model. As layers were completed in turn, the variables to be displayed by individual layers became clearer and the layout and layer sizes were refined.

This process of continual refinement was conducted over the lifespan of the implementation phase. The result of this process led to most stack layers being of a similar size, except the Transport Layer, as it has to display the contents of both of a Session's Sliding Windows. These designs were implemented, reviewed and updated with a 'trial and error' evolutionary approach. As such, the reasoning for individual decisions is difficult to quantify as it was a fluid process.

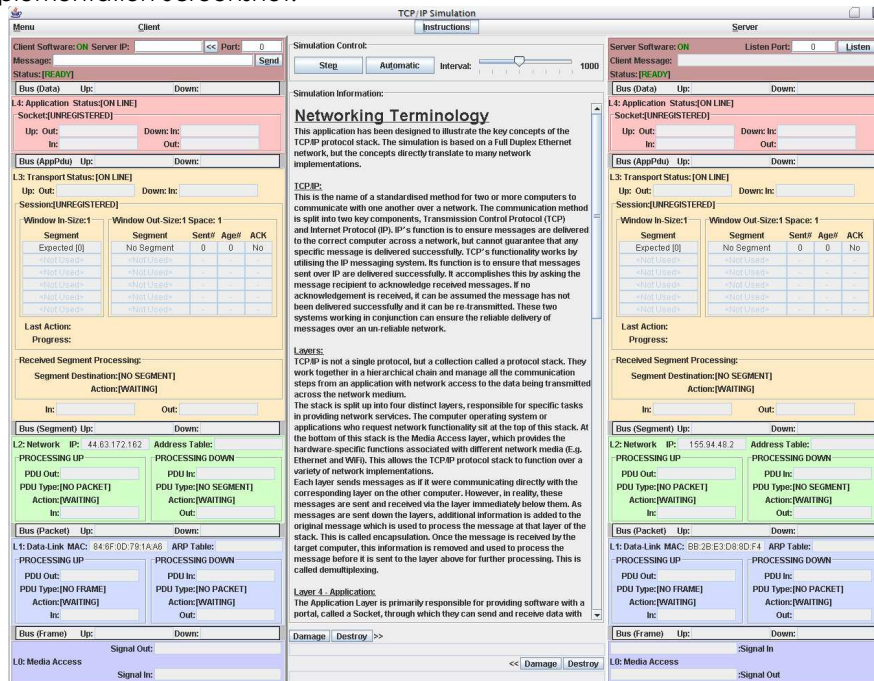
- The overall design layout was designed to portray the hierarchical structure of two stack layers. This was achieved by placing each stack's layers on top of one another, separated by their corresponding Data Buses. The colour-scheme utilised within this project was also implemented as the background colour for each layer. This was to more clearly identify individual layers from one another and illustrate how layers correspond to each other on both computers and any Intermediate Network Devices.
- The Client stack layer was placed on the right of the screen, the Server's on the left. This was done to portray the physical separation of the two computers from one another.
- Physical Layers were connected by the view class for the Network Cable. This identifies that the only connection between the independent computers is this communication medium.
- The views of any Intermediate Network Devices were placed between the two computers stacks, but a margin was always left between each computer and the middle device. This was to identify that these devices are only connected by the network cable.
- Depending on the device, Intermediate Network Devices operate between the Physical Layer (Hub) and the Network Layer (Router). The portion of the screen above this, running parallel with the Transport and Application Layers was reserved for help text, providing a summary of the key details associated with each layer and their PDUs.
- The central portion of the screen in parallel with the Software Layer was reserved for the simulation controls. These are used to progress the simulation through time periods, or set it to run automatically.
- A traditional 'menu bar' runs along the top of the window. This provides 3 menus.
 - The primary menu is located in the standard left-most corner of the screen. This was decided to mimic the interface positions adopted by the majority of application windows to make the application more intuitive to use. This menu provides access to selecting the type of simulation (Client↔Server, Client↔Hub↔Server, Client↔Switch↔Server, Client↔Router↔Server), provides access to a help window, explaining how to configure and run a simulation and an exit application dialog window.
 - The other two menus are located directly above the Client and Server stacks in order to visually tie them to their associated computers. They provide access to configuration windows for each computer, in addition to controls for turning the simulated computers on and off.

Overall User Interface Structure:

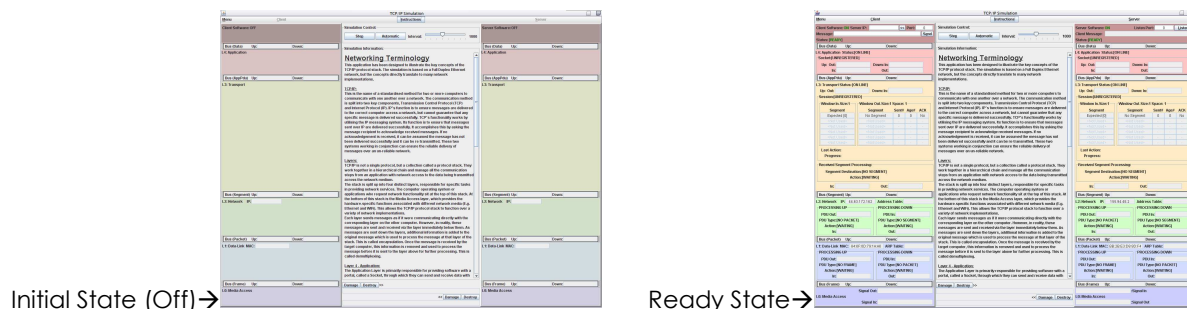
- Summary of the logical structure of the application window:



- Actual Implementation Screenshot:

**Interface functionality:**

In addition to illustrating the internal progression of the simulation on-screen, the design of the user interface has a strong influence on the user's ability to operate the application without help. In order to help the user identify areas of interaction, the application initialises with very little on-screen. The stack layers are still coloured, but have a washed-out tinge to them. All controls that are only accessible once variables have been configured by the user are greyed-out. These visual themes were implemented to illustrate to the user that in the simulation's current state, these areas are un-usable, illustrated below-



Controls & Shortcuts:

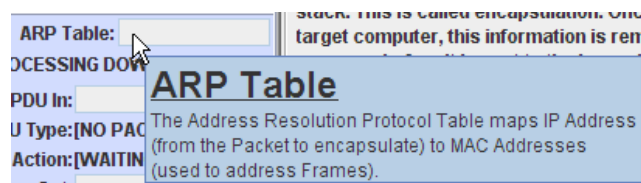
To conform to the Usability Requirement 1.1, all user interface functionality must be controllable from either the keyboard or mouse. To conform to this requirement, these functions were implemented:

- Every button and menu control was implemented with keyboard shortcut key. This allowed for example, the 'Menu' to be accessed by pressing the keyboard combination Alt+m. The key to press is identified by the underlined character on the interface component. Smaller components can be accessed by pressing the 'tab key', cycling through each interactive component, then pressing the enter key.
- The simulation can be progressed to the next time period by pressing the spacebar when looking at the main application window. This triggers the 'tick' event in the control class via a Java keyboard listener.
- To enter a computer's IP & Mac Addresses without a keyboard, a 'random' button was implemented on the configuration window. This generates a random, valid address for each field. The requirement for this feature was identified early during the development phase. This is because the simulation needs to be accessible to users who do not wish to learn about the formalities of IP and MAC addresses so they will not want to input them manually. Additionally, by having a facility to automatically generate these addresses, users who wish to get to the simulation phase as quickly as possible to illustrate a specific point do not have to waste time configuring fine details of the program.
- To save the user typing the Server's IP Address into the Client Software Layer (a required fields to create a connection), a button was added to the Client Software Layer View. If the Server's IP Address has been configured, this button would be enabled (signified by it no longer being 'greyed-out'). Once pressed, it automatically copies the Server's IP Address into the required field of the Client Software Layer. This was added after a review of the software highlighted that some user became frustrated that they had to type-out the Server's IP Address manually each time they wanted to simulate a connection. This feature was implemented purely as a time/effort saving feature and does not degrade the validity of the Software Layer's simulated functionality.

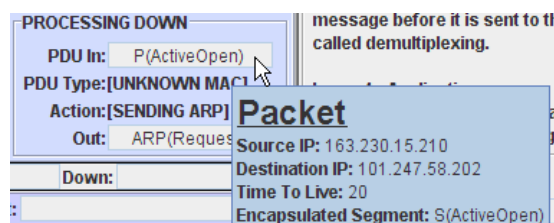
Help Functionality:

As the simulation is complex, it may not be immediately apparent to a user how to configure the required settings to run the simulation. As first opinions count, if the user is unable to get the application to a state where the simulation can actually be run within a few seconds, they may abandon it in favour of alternative sources of information. To counteract this possibility, a help-page can be brought up, explaining step-by-step actions the user can take to start the simulation. This is presented in a separate widow that 'hovers' on top of the main window. It is accessible either from the main 'Menu', or by clicking a clearly identified 'Help' button located at the top-centre of the main window.

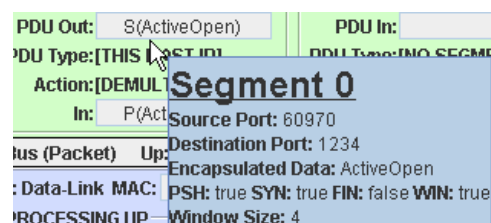
Each on-screen item (e.g. an entire Stack Layer, Address Table or Sliding Window) has an associated tool tip text. These are written into the view classes and displayed when the user hovers their mouse cursor over any item for a few seconds. This allows them to quickly and easily identify and understand the function of any item displayed on screen. Writing and formatting these comments accounts for a large portion of view classes' source code. HTML was used on the tool tip texts to format their contents to make them easier to read. Example tool tip text for ARP Table-



Tool Tip Texts are also implemented to display the detailed summaries of PDUs. These are retrieved by the view from the PDU Objects contained within the model by calling their getDetailedSummary() method. An Examples of PDU tool tip texts-



Example of Packet Tool-Tip-Text

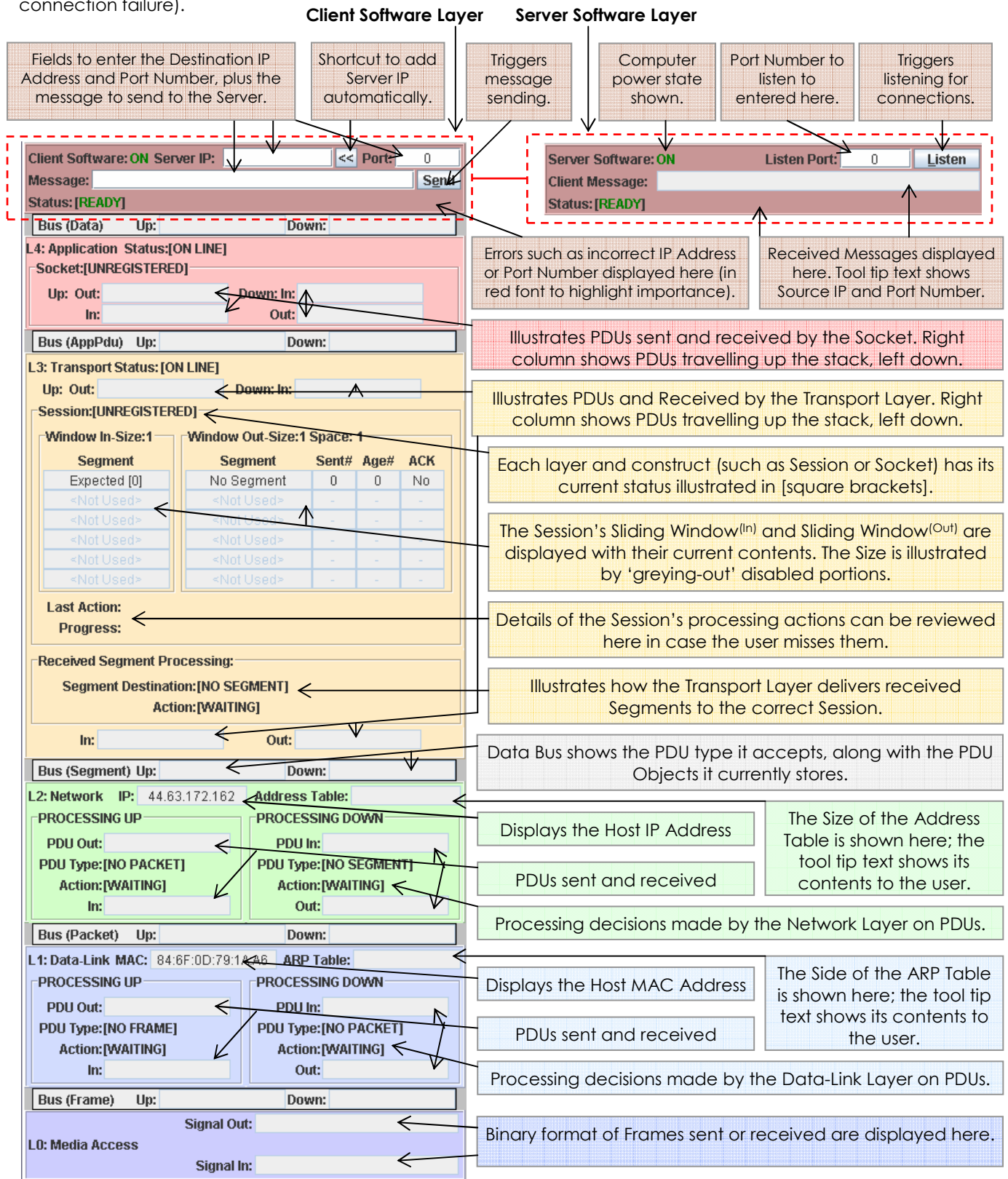


Example of Segment Tool-Tip-Text

Single Host Layout:

This section illustrates the overall layout of a single View Host Computer, with its child views representing stack components. Two versions of this view are displayed by the user interface: the Client and Server stacks. The TCP/IP stacks are implemented identically, whichever computer they are on. Only the software interacting with it varies. This is illustrated by the view classes by all stack layers being identical, with only different Software Layers to represent the Client and Server software.

During the course of a TCP/IP communication, the displayed states and variables for each stack layer will alter, eventually returning to their default states once communication has been completed (or reset due to a connection failure).



Centre-Screen Components:

These components occupy the space between the Client and Server computers. They include the simulation progress control, Network Cable(s), any Intermediate Network Device and Summary Help text for the user.

The screenshot displays the simulation interface with several key components and their functions explained by callouts:

- Instructions:** A button to bring up a help window, explaining how to use the simulation.
- Simulation Control:** This section allows the user to control the progress of the simulation. It includes a **Step** button (progresses the simulation one time portion), an **Automatic** button (progresses the simulation one time segment every x-milliseconds), and an **Interval** slider (can be changed at will to dynamically alter the execution speed).
- Simulation Information:** A section explaining key concepts and terminology, such as TCP/IP, to help new users understand aspects of the simulation. The size of this window varies depending on the presence of simulated Intermediate Network Devices.
- Intermediate Network Device's views:** Displays views for each section operating in parallel, using identical colour schemes to the computer stack layers. The Router device is shown here, with details for its L2 (Network Layer) and L1 (Data-Link Layer).
- Network Cable View:** Positioned here, showing a single cable connecting two Hosts. It illustrates PDU objects within the cable and includes controls to simulate Damage or Destruction of PDUs on the Network Cable.
- Host Configuration Window (Right):** A window used to configure each Host computer, including IP Address, MAC Address, and Advanced Settings.

Host Configuration Window (Right):

This is the window used to configure each Host computer. The top portion configures the IP Address and the middle is for the MAC Address. The lower section is used to fine-tune settings used by that computer's Session. This portion is not displayed by default, but expands out of the smaller window when an 'Advanced' button is pressed. This is to hide more complex controls to make it easier for first time users.

Details of entered addresses are displayed to the user. To avoid the user manually entering an address in, the 'Random' buttons generate random valid addresses.

There is a help button for each section which displays a window containing an explanation of each section, along with allowed address ranges, etc. The user can only apply the settings when each contains a valid input. An error will be displayed explaining invalid inputs.

The Host Configuration Window (Right) is titled "Client" and "Define Address Configuration for Client:". It contains the following sections:

- IP Address:**
 - IP Address: 192.114.149.10 (Set)
 - Binary Value: 11000000 . 01110010 . 10010101 . 00001010
 - Address Class: C [N.N.N.H] (Random)
 - Address Type: Host
 - Address Valid: YES (Help)
- MAC Address:**
 - Mac Address: E8:C1:CF:CA:B3:0F (Set)
 - Decimal Value: 232:193:207:202:179:15
 - Address Type: Host (Random)
 - Address Valid: YES (Help)
- Advanced Settings:**
 - Sliding Window Size: 4
 - Segment Timeout: 50
 - Segment Re-Transmit: 3
 - Segment Size: 4
 - (Help)

Buttons at the bottom: Apply, Cancel, ^ Hide ^

Design Alterations

Application Timing:

Initially each computer's model was 'ticked' by its own thread Object, which would then sleep for a random amount of time (within a defined threshold) before ticking the model again. This was implemented to simulate that computers communicating over a network are not synchronized, because each computer has its own processing capacities and timing schemes. The result of this made the simulation difficult to follow for the user as there was no uniform timing scheme controlling the entire application; variables would change on the Client and Server at different times. This was confusing to the user as the relationship between cause and effect was not clear, making the process hard to follow.

A decision was made to change the simulation from the more accurate thread-based timing mechanism to a simpler unified call structure. Each time a 'tick' call is made by the control, it is passed to all model objects at the same time. This forces both computers to perform read then write tasks at identical times. This made the simulation far easier to follow for users as the simulation progressed at a uniform speed. Each 'tick' is generated by the user pressing a button within the view, allowing the user to progress the simulation at their own pace. An automated system has also been implemented, whereby 'ticks' are generated automatically every x milliseconds, where x is controlled by the user from a view control mechanism. This allows the user to simulate progress over time without having to manually press the next button repeatedly.

Help Text:

Over the course of the initial testing, it became clear that users who had less experience of the TCP/IP stack did not easily understand the processing tasks performed by individual components. Additional tool tip texts were added to better explain the responsibilities of the components.

A test subject noted that "Initial configuration of the server and client is not very intuitive," so a help-page was added (accessible via the help button or main menu). This explains the required configuration steps that must be taken to initialise a simulation. They are presented in simple bullet-point fashion to make the steps easier to follow in sequence. This improved usability and made the simulation easier to use without documentation. The central explanation of key concepts was rewritten to cover a wider range of information. This removed the need for users to reference external material whilst using the simulation.

Simulating More Than Required:

After testing some information was removed from the view as it was felt there was too much information on-screen for the user to easily understand. Some data had been displayed which was not relevant to the scope of the simulation as defined by the requirements. This information was removed from the view classes, but is still contained within the model classes. By implementing the changes in this fashion future revisions to the simulation can still utilise this data.

The simulation also processes variables which are not fully utilised in the current implementation, such as the Packet's Time to Live value. These variables will be required if the scope of the simulation is increased (e.g. to include multiple routers). These were originally implemented so that in the event of increased simulation scope, changes could be implemented easily. The computational power required to process the additional information was not found to have a noticeable effect on execution speed, so they were left in the final source code. They can then be used during future upgrades to the simulation.

Address Invariants:

Initial testing identified aspects of the simulation that had been overlooked during the design phase. The most important of these was that there was no system to ensure that multiple devices did not have the same IP or MAC Addresses. In its initial state, any number of Host computers or Intermediate Network Devices could have identical addresses. Although it was not specified in the requirements, TCP/IP networks do not allow identical addresses.

Once identified a log system was implemented within the control class. Whenever a device registers a new IP or MAC Address, it is first checked against a list of currently used addresses. If any are used, the device will not be allocated the identical address. Instead, an error message will be reported to the user informing them to pick a different address. In the case of addresses being automatically generated, a new one will be picked.

Layer State Feedback:

Initially, the view provided little feedback as to the current state of a stack layer of component (such as Socket or Session). To ensure the user understands why specific processing decisions were made by the simulation, a more comprehensive feedback system was implemented. This involved specifying String comments for each state a component can be in. These are then displayed by the view, clearly identifying the processing logic of the simulation. Error states are formatted to provide their output in a red font colour. This clearly identifies unusual states to the user, drawing their attention to important points. These comments are displayed in square brackets '[]'.

Implementation Problems

- Initially user interface was not considered as the highest priority. The bulk of the schedule had been focused on simulating the TCP/IP as stack accurately as possible. As the application came closer to completion, it became clear that the design of the user interface would have a significant impact on the usefulness of the simulation. Because of this, the schedule was altered to provide a greater weighting of development time for the user interface. This included several testing stages where potential users were invited to comment on the interface design and features. This information was then used to further evolve the view classes to provide an enhanced user experience. In several places this meant removing the output of some data altogether as it was not found to be beneficial to presenting the stack's functionality. If a similar project was attempted again, further research into Human Computer Interaction (HCI) issues would be recommended.
- An implementation problem identified due to poor design was found in the algorithms that log addresses (e.g. Network Layer Address Table, where IP Addresses are mapped onto Segments). The initial implementation logged records using one address. To check to see if a record existed for an address, a search would be conducted using the address as the key. This worked under standard conditions. A flaw was discovered where one of the addresses alters, but the other stays the same, e.g. the computer's IP Address remains the same, but the Port Number the Transport Layer is listening to changes. Under these conditions the address would be found to exist, and the old value would be used because the new associated Port Number had not been updated. The algorithms were altered to update existing records to new variables as well as add unrecorded new records to the tables.
- A simple design issue was identified during the development of the user interface view classes. In order to conform to the requirements every control in the user interface had to be accessible using a keyboard. The primary implementation device for this was to assign buttons with a keyboard shortcut e.g. Menu was accessible using the shortcut Alt+m. As the number of display components increased it became increasingly difficult to assign components meaningful shortcuts (primarily by using the first letter of the word as the shortcut key) because many of the components labels started with identical letters. The solution implemented involved using letters found later in the label as shortcuts, e.g. Automatic (Alt+t). This solution worked, but was less elegant than using the first letter. It added a small amount of unwanted complexity for users identifying shortcuts, but overall proved to be a successful solution.
- Initially tool tip texts were implemented using standard the Java functionality of applying a String comment to a screen component. These Strings were unformatted and all appeared on a single line. This made large comments difficult to read. More comprehensive formatting was required in order for this method of illustrating data to be successful. Research was conducted and it was found that Strings containing HTML formatting tags would be rendered correctly by the Java Virtual Machine. This allowed new lines, bold, underlining and font colours to be used within the tool tip texts. Careful formatting of this data made the texts significantly easier to understand, enhancing the user experience.
- An unexpected problem was identified with the implementation of tool tip texts to provide information on view components. If the component was located on the extreme right or bottom of the screen, the outside portion of large tool tip texts appeared off-screen. This made the information unreadable thus useless. Each tool tip text had to be individually tested, re-worded and formatted to ensure that its contents would be rendered entirely on-screen.
- A serious problem was identified with the implementation of the sliding window algorithm written for the Sliding Window^(ln). The algorithm itself was correct, but the Java constructs used for the implementation were not fully understood during the implementation process.

To store received Segments an Array List was used. This provided the standard facilities of the array data structure, but automated the process of moving pointers when array elements were removed from the start or middle. This was selected as it removed the requirement to manually change pointers when Segments were removed from the window. What was not known was that if the Array List size is n , an object can only be added to positions $0 - (n-1)$. The Sliding Window^(ln) has to deal with Segments with high sequence numbers arriving before earlier sequence numbers (e.g. Segment# 1 arrives, then 4 arrives before 2 and 3). When this occurs the Array List size is 1 (as it is populated by Segment 1). The software then attempts to place Segment 4 into the 4th position and an array out of bounds exception is thrown. This is because the Array List size was not big enough to accept an Object into that position. This was a critical error as it created instability within the application and meant the simulation could not be implemented correctly. It is not possible to enforce a minimum size for Array lists in Java.

Due to time constraints it was decided to program a 'work-around' to the Array list problem, rather than researching or creating a different implementation from scratch. The solution implemented involved running a loop at the creation of the Window and each time a Segment was removed from it. This loop populated empty array sections with a 'null' Object up to the intended size of the window. Whenever a Segment was added this null Object was removed and replaced with the new Segment. This ensured the window was always large enough to accept a Segment into any valid position without causing an error. The view class responsible for illustrating the Window had to be programmed not to illustrate Sliding Window positions containing null Objects.

This implementation proved successful but was inelegant. In future implementations could examine alternative techniques to implement the Sliding Window algorithm more efficiently.

- The final problem found that was of critical importance related to Java's Swing and AWT visual component libraries. Both these libraries were used to format on-screen components within view classes. The main window had a highly complex structure which included many sub-frames, each responsible for rendering a specific portion of the simulation. These frames had to be drawn in the intended layout so they could represent the hierarchical structure of the TCP/IP stack. This had been identified as being of critical importance during requirements elicitation.

When the main window was resized the layout of components altered and no longer fulfilled the design requirements. Various alternatives were tested, including dynamically altering the size of frames in order to maintain their overall layout positions on-screen. This was unsuccessful as frames often became too small to successfully render all their internal components. This led to data not being represented at all which was unacceptable. The final solution implemented was simply to enforce a fixed window resolution. This ensures components were always rendered correctly and that no data would be missed.

The Java layout manager GridBagLayout was used to arrange components within the main window. This was selected as it created a grid-type foundation to which individual frames could be 'dropped into'. This was selected over similar layout managers as it meant variable sized grid sections and components could span a varying number of columns and rows. This was critical as the implementation of the centre portion of the view required that components such as the Network Cable could span the entire middle, or just certain portions of it. This allowed for Intermediate Network Devices to be slotted between the two host computers. It was found that when the fixed-size main window was displayed on a screen that did not have a large enough resolution to render the entire window at once, the window manager would force the application window to a reduced size. The GridBagLayout would then not have sufficient room to render its sub-components at their 'preferred size'. This caused the layout manager to fail and all the on-screen components would collapse into the centre of the screen in an unreadable mess.

The solution to this problem was found by changing the commands used to set the size of the main window. Originally, the `setPreferredSize`, `JFrame.setSize` and `JFrame.setResizable` methods were used to define the size of the main window. By changing the size command used to `JFrame.setBounds`, a more rigid implementation, the window would no longer be forced to render at a reduced size. This in turn avoided the problem of the GridBagLayout manager not having sufficient space to render its components and fail.

Testing

It was identified that the application developed for this project would be highly complex, and that its source code would be of a large size. When selecting appropriate project management techniques it was decided that testing should be conducted throughout the implementation of the project. The primary reason for this was to reduce the requirement of reserving several weeks of the timetable for a specific testing phase. Instead, testing was carried out over the course of the development lifecycle. This allowed for a far smaller dedicated time slot for testing of the application, freeing up time for development. Additionally, as errors could be identified earlier, it would be less likely that functionality would be implemented that uses the incorrect code. This reduces the overall impact of individual errors. Reducing the time requirements for the testing phase created slack in the project timetable. This was utilized for a user interface review session with a group of potential users. The results of which were then used to further improve the user interface, creating a higher quality product. A Software Test Plan was produced and encompassed the following sections:

Development Testing:

Although the software development lifecycles selected specified the order of class development, how the source code was implemented with each class was a reflection of the programmer's individual style and technique. Java is an object orientated programming language and each Object's functionality is delivered by sub-sections of code called methods, each responsible for performing a specific task. As each of these methods was written its functionality was tested using facilities provided by the Netbeans IDE. These facilities are provided by the debugging section of the IDE, and allowed the developer to monitor the value of each Object's variables over the course of execution. Dummy data was passed to each method and the tool was used to confirm it was processing data as expected. This was repeated for each method written throughout the development lifecycle and represents the lowest level of structured testing conducted.

Testing of individual portions of code was conducted as required to verify its functionality. As these were not conducted in a structured fashion they will not be explained in detail.

Unit Testing:

Unit testing encompasses the functionality provided by each software unit, in this case class. The heuristic plan followed for unit testing was to test each model class with dummy inputs, while reviewing outputs generated and displayed in Java's 'System.out' development console. Then, conducting the same tests, the model's outputs were checked on the associated view class in the user interface. These were used to verify that the class was processing data in accordance with the software design. The accuracy of the user interface was verified in parallel with the second portion of these tests. This was conducted when each class was finished, and repeated if modifications were made to a class for any reason (Progression Testing).

In addition to standard unit testing, areas of code identified as being of a high level of complexity were tested separately. This in-depth testing was conducted as these areas of the application are more difficult to test comprehensively, and more prone to errors due to their technical complexity. Depending on the context a black or white box testing methodology was used. Examples of such tests include:

White Box Testing:

State Machines implemented on high-level stack layers were tested by supplying each state with all the possible inputs it could receive at that state, and checking that the correct action or transition was performed. These were calculated by reviewing the state transition diagrams (included in the Architecture section).

Black Box Testing:

The regular expressions used to validate IP and MAC Addresses are highly complex. It is impractical to test each regular expression for every possible valid and invalid input, so more time was set aside during their initial development to ensure the expressions were correct. To test the MAC Address regular expression, an existing section of the code was utilised. This was the facility implemented to generate random MAC Addresses. This was used 50 times and the resultant MAC Addresses were checked for validity. 50 invalid MAC Addresses were manually inputted for testing those of invalid ranges. Deliberate mistakes were added to these addresses, including using too few or too many characters, using invalid symbols and a mix of upper and lower case characters.

IP Addresses are governed by numerous rules regarding valid ranges for address class and type. To test all these ranges a comprehensive table was drawn up identifying all possible IP Address types, their valid ranges and types. *This is included in the Appendix.* Using this table the regular expression was tested with manually selected IP Addresses. These focused on the ranges of addresses close to the boundaries between address classes and types. This was to ensure that range checks had been implemented correctly. A range of random addresses was tested to check that addresses that fall between these boundaries are also correctly identified.

Both the MAC and IP Addresses were tested using the Host Configuration Window, as shown in the Class Design section, and required no direct interaction with the source code. This was selected as it allowed manual input of addresses and access to the software's capacity to randomly generate addresses. The generated output illustrates all the variables for verification. This testing approach allowed this portion of the user interface to be tested to ensure it outputs and formats data correctly. This testing was conducted at the same time as regular expression testing.

Integration Testing:

Once each stack layer had been completed it was tested by having it communicate with its corresponding stack layer on the other computer directly through the peer-to-peer interface. Then, as outlined in the Software Engineering section of Application Architecture, because of the layered characteristics of the TCP/IP stack, rigid SAPs and PDUs, integration testing could be conducted progressively throughout the development lifecycle. This was possible as layers were implemented from the bottom up, starting with the Physical Layer. Communication between each new layer and the one previously completed could be tested during the development of the new layer by allowing it to communicate via the older stack layer below it.

This tested both the integration of classes and further tested the functionality of the new and previously developed layers. This process was developed exclusively for use on this project and is only applicable due to the compliance of the simulation's structure to that of the TCP/IP stack.

Beta Testing:

The five potential end-users identified in the Requirements Elicitation section of the Project Specification were consulted a second time following completion of the first version of the application. This group consisted of 2 current students, 2 students who had already graduated and a network technical trainer. As the model functionality had been thoroughly tested prior to this, the Beta-testing focused on the design of the user interface supplied by the view classes. As with the requirements elicitation this process followed an informal structure. The participants were invited to use the application for 30 minutes and then provide feedback. The results of this process identified several areas to improve:

1. The application was not intuitive during the simulation configuration because the tasks that had to be completed to initialise the simulation were not clear to the user.
2. The functionality provided by each layer was not summarised in a simple text form that could be viewed while the simulation was running.
3. The speed of the automatic progression of the simulation was not controllable.

Following this phase the following improvements were implemented:

1. A help window containing simple bullet point instructions on how to configure the simulation was added. This was made accessible via the 'Menu' or by selecting the 'Instructions' button located top-middle of the application window.
2. The explanation of key concepts displayed in the central column of the application window was revised to include more detail on the processing responsibilities and data structures used by each layer.
3. A 'slider' was placed next to the 'automatic' button controlling simulation progression. This slider can be used to dynamically alter the speed of the simulation when in automatic mode.

Release Testing:

The week prior to the project presentation was reserved for final testing in preparation for demonstration of the product to the customer (examiner). All aspects of the application were checked for execution and formatting errors. During this stage the content and formatting of several tool-tip-texts were revised to be clearer. During this week requirement validation and verification was conducted. Please refer to the Project Specification section for a detailed explanation of individual requirements.

User Requirements:

- 1.1 The product fulfils all scope and simulation requirements outlined in section 1.1. These have been individually identified and checked off.
- 1.2 The user can configure all the initial variables outlined in section 1.2.
- 1.3 The application allows the user to damage or destroy signals in the network media.
- 1.4 The simulation progresses in a step-by-step format and can be prompted to do so automatically.
- 1.5 The simulation conducts variable range and validation checks to ensure invalid data is not used.
- 2.1 The simulation is represented on screen to the user.
- 2.2 The simulation displays all relevant variables on a single screen.
- 2.3 Each stack layer and component is identified on screen using a colour-coded key and includes an explanation of its function in the form of a tool-tip-text.
- 3.1 The application simulates all defined network hardware components.

All User Requirements were verified successfully.

Non-Functional Requirements:

Usability – The application can be controlled entirely by a mouse or keyboard. In order to damage or destroy signals contained in a Network Cable when using the keyboard, the user must 'tab' through multiple controls and then press return when they have selected the correct one. This is not the most effective control mechanism, but does fulfil the requirement.

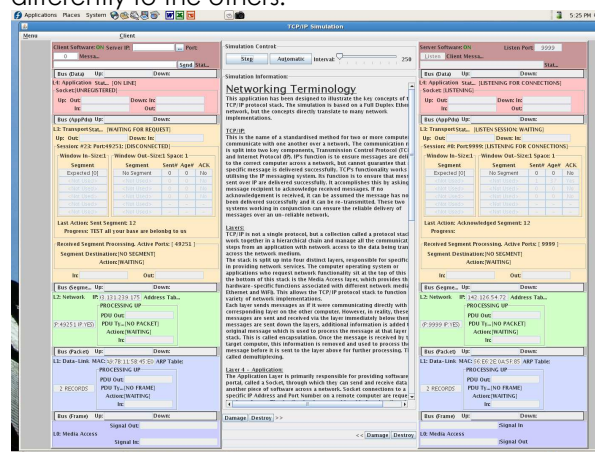
The simulation identifies and explains invalid data inputs from the user.

Performance – The application can be executed and be ready to use in under 10 seconds on a computer that meets the minimum specifications for the Java Virtual Machine. This is quicker than required by the specification. Users are able to configure the simulation and make it ready to run within 10 seconds of the application loading, although this is only possible by using the 'generate random address' function provided to configure each host. This fulfils the requirements.

Space – The compiled JAR file for the application is 369Kb, this is smaller than the maximum 1.44MB specified.

Reliability – As the Java Virtual Machine is run in the user-space of the Kernel, it is extremely unlikely that the application could destabilise the system it is running on in the event of an error. This requirement is difficult to quantify beyond this point.

Portability - A requirement for this project is that the application can be executed on a variety of operating system platforms (Non-functional requirement 2.4.1). To confirm that it has fulfilled this requirement four commonly used platforms were selected: Microsoft Windows XP-Professional Edition, Microsoft Window 2000, MAC OSX, and the Linux Distribution Fedora. All computers fulfilled the minimum system requirements to run the Java Virtual Machine and had a screen resolution of at least 1280x1024. The simulation was run on all four computers. The same configuration settings were used and the same message was sent from Client to Server. The simulation was run in 'automatic mode' with the timer set to 500ms. All simulations ran successfully. The only difference noticed between each was that the application run on the Fedora platform was rendered slightly differently to the others.



Fedora Screenshot (Left)

Grey boarders were visible on both the left and right edges of the application; however they did not have any further effects on the display. Other than this, the simulation was rendered and executed correctly.

It can be assumed that the difference between this and the other three tests is due to the specific implementation of the Java Virtual Machine used on the Fedora platform or its interaction with the kernel's window management system.

As the error did not have a negative affect on the applications usability or function, the decision was made to take no action to resolve the problem due to time constraints.

Delivery – The application and these tests were completed a week before the specified deadline of 17th March.

Implementation – The application could be installed on the School of Computer Science's systems and its execution far exceeded the defined minimum performance requirements.

Standards – According to the research conducted, the simulation accurately portrays the processing of a TCP/IP stack within the defined scope.

All source code is commented and included JavaDoc descriptions.

Interoperability – No requirements to fulfil.

Ethical - No requirements to fulfil.

All Non-Functional requirements were validated successfully.

Improvements:

During the testing phase errors and deficiencies in functionality were identified. Solutions were designed and implemented. The altered classes and those who interact with them were retested. This was to check that the problems were fixed and that no new errors had been created as a result. These are identified and explained in the Design Section of this report under the headings Design Alteration and Implementation Problems.

Evaluation

Appraisal:

This project has been highly successful. It was initially envisaged as a simulation of TCP/IP communication, but the opportunity was identified to utilise this simulation for teaching purposes. The software produced is a competent teaching tool that is equally functional for use in group teaching or individual work. It fulfils all the requirements laid out in the specification and in many cases exceeds them.

The initial priority for the project was to accurately simulate the TCP/IP stack. This required a significant amount of research into the technical aspects of TCP/IP and the implementation of complex processing functionality in the source code. The process of implementing the TCP/IP stack presented many coding difficulties which had to be overcome. The developer learnt valuable information about efficient processing techniques, which were utilised to ensure the simulation executed smoothly without the need for a high specification computer. This was illustrated most clearly when the comparisons used by state machines at high layers of the TCP/IP stack were changed from String to integer evaluation. This yielded a significant increase in execution speed with very little redevelopment time required to implement the changes. This solution then created further difficulties, as the comparisons were no longer between human meaningful words but numbers, the source code became more difficult to understand. To combat this, variable names were reworked to act as more meaningful identifiers of their function.

Once the simulation was complete the focus shifted from a technical based project to how to appropriately display this information in an HCI-orientated development process. The priority throughout this process was to only present the information the user needed to know, and to display this in a meaningful format. Examples of this include the consistent use of colour schemes throughout the application and its documentation. This scheme clearly identifies stack layers at a glance, reducing the need for users to read text based identifiers. Other examples include the context sensitive formatting of information, such as identifying network and host octets of IP Addresses and using appropriate colours to identify the separate portions. This formatting was then carried over to the binary representation of the address, clearly identifying how the type of address is influenced by its binary values.

The user interface draws upon well understood colour identifiers used in real life situations to provide feedback to the user, focusing their attention to points of interest. The implementation of this concept focused on the premise that simple black text would be used to present normal information, bold formatted red text to report errors and bold green text to report successes. The inspiration for this was taken from the formatting used by signs in public places, such as green identifiers for fire exits and red danger signs. These colour associations are so commonly implemented in the real world that their meaning is second nature to users and requires no further explanation.

None of Java's native networking capabilities were used in the simulation. The decision to exclude these was taken after an evaluation of packages providing that functionality was conducted. These classes were designed as automated interfaces to provide network functionality transparently to the classes that use them. As the goal of this project was to show the functionality not hide it, these packages were not suitable. As execution speed had been identified as being of prime importance, every data structure and class in the simulation was constructed from scratch. This ensured they stored only the required variables and performed no unnecessary processing. The most advanced package imported that was not written by the developer was a regular expression analyser used during address verification. This was done purely to remove unnecessary development tasks. Other than this, only simple native constructs such as Strings and Array Lists were used.

Successes of the application include the fact that users can use and learn from only the aspects of TCP/IP that are of interest them. For example, the configuration window for IP Addresses on its own is an extremely powerful tool, validating and identifying information on any IP Address a user inputs. The application also implements a basic form of intelligence used to identify the different ways people may input data into a computer. This includes recognising the presence of leading zeros on IP Address octets (5 is the same as 005) and formatting conventions, such as different delimiters and the use of upper or lower case letters in a hexadecimal number. These were used during MAC Address validation. This system can also overcome simple human errors such as changing the case of letters used mid-variable during address input.

If a user wishes to learn about the TCP/IP stack but not about address structures, they can opt to use the application's facility to automatically generate these, allowing the user to focus on the aspects that interest them. The terminology used in user feedback by the simulation was phrased specifically so it does not require prior knowledge of these concepts.

The simulation reproduces the TCP/IP stack as close to the specification as available research material would allow. A failure that must be identified is that the research was unable to clearly identify the means by which the

Network Layer is able to identify the destination IP Address of a Segment it is to encapsulate into a Packet. The means by which the Network Layer provides this facility could not be clearly identified, as it varies between individual vendor implementations of TCP/IP. Logical deduction was used to identify how this process is achieved in its simplest form. This was then implemented in the simulation. As layers can only communicate through SAPs, each Session must identify its associated target IP Address to the Network Layer at its initiation. This was achieved via a specialised initialisation Segment being passed between the two layers as the Session is instantiated. Address mapping techniques used by all other layers were identified and implemented in accordance with the specification.

The primary drawback of the user interface implemented is that it is not resizable and requires a minimum display resolution of 1280x1024. Until recently, this resolution would exceed that found on many LCD panels used in laptops and projectors, so there may be some difficulties in using this software in classroom environments that are equipped with old computer facilities. This may reduce its use in shared facilities such as lectures. However, because the TCP/IP stack is a defined protocol standard, the high-level implementation simulated by this application will not become incorrect over time, so even if it can not be used in some faculties now, as they are updated it may become more useful in the future. Typical desktop computers' displays have exceeded this resolution for many years so the minimum resolution will have less impact for users on individual systems.

The processes used throughout the lifecycle of this project proved highly successful. Appropriate project management techniques were identified and implemented very early. These ensured that the process stayed on schedule. Initial time assessments were weighted towards overestimation to factor in float, allowing time to compensate for overruns. The float left at the end of the development phase provided an additional week with no scheduled activities. This was utilised to conduct an extra round of user testing; a group of 5 potential end-users were invited to test and comment on the application. This feedback proved very useful in identifying areas that did not perform to their full potential. The result of the review prompted an overhaul of the design of some user interface components. Additional help texts were incorporated into the application to more clearly explain how it is operated. The changes resulted in a noteworthy improvement of user feedback received relating to the applications usability.

The application proved to be very robust. This was due to the continual testing of stack layers during the development of subsequent layers. It was not possible to cause the application to behave in an unexpected fashion or generate errors following the first round of testing. The application was considered stable over a month before the final deadline.

The design and software engineering processes selected for this project proved effective. The phased implementation of the simulation model was successful. It made progress easy to monitor and its structure correlated with the software design. As predicted, the requirements for the user interface were difficult to quantify. Having reviewed alternatives, the evolutionary model was still the most suitable choice as it allowed testing and improvement cycles – each revision improving the user interface.

The simulation's design was very effective; it fulfilled its primary goal of being structured and performing identically to the TCP/IP stack, whilst still providing the functionality required by the user interface. Because each class' requirements were identified during the research phase, the simulation's design was completed before code implementation began. The designs created fulfilled the requirements with very little modification required during the coding stage.

A side effect of the unknown user interface requirements was that in several places the simulation processes data which is never displayed to the user. This data was not removed before delivery because there was no identifiable performance degradation from processing it, and the variables may be used in future revisions to the application. Because of the thorough design process, there were very few errors to identify during the testing phase. Various testing techniques were used throughout the application development. The testing plan constructed was effective as it provided a thorough testing strategy; this identified errors early on, minimising their potential impact.

The application successfully fulfils all of its initial requirements, and in some cases exceeds it. This is because it provides functionality that was not originally specified. This was included as it was identified as being beneficial to providing an effective learning experience to the user. This conclusion is reinforced by the user feedback gained during the testing phase. This was very positive; the consensus being that the application fulfilled or exceeded its initial requirements. It proved to be a useful tool for learning or enhancing the user's knowledge about not only TCP/IP, but addressing schemes and other network devices.

This report successfully identifies each stage of the development process. Due to the scope of what was achieved it was not possible to fully explain design decisions taken, so only specific areas of complexity or difficulty were explored in detail.

Further Work:

The biggest challenge of this project was its scope. It is possible to implement the stack to increasingly complex levels and simulate additional devices. The priority was to clearly identify what would be simulated and what level of detail was required to portray this to the user effectively without unnecessary complexity. It was always envisaged that the simulation produced could be expanded further in future revisions to increase its scope. Because the TCP/IP structure was rigidly adhered to, the implementation can easily be expanded to simulate different configurations with very little additional work. For example: the final product only simulates two computers but the model is capable of simulating multiple connections between multiple computers. The difficulty is designing a way of illustrating all this information to the user in an understandable fashion.

Future revisions may simulate additional computers, specifically how a single server application can spawn multiple threads to cope with each connection. This simulation may be of benefit to students studying topics related to real-time system implementations, in addition to networking subjects.

Scope could be increased to encompass multiple Routers between the Client and Server. Protocols such as Routing Information Protocol and Border Gateway Protocol could be implemented for the Routers. These could be used to illustrate how they share their address tables in a distributed fashion and how Packets are dynamically routed between multiple networks. Controls on the Routers could be used to disable them or simulate heavy traffic or latency over specific connections which would impact on how Packets are forwarded.

As the stack is modular, additional Physical Layer implementations could be simulated and made user-selectable. These may include wireless or satellite transmission technologies. Similarly, the UDP protocol system could be added to the Transport Layer with no need to rework other stack layers. The simulation could then be used to compare the respective benefits of each system to the other under different conditions.

A suggestion made during the presentation of the application would be to allow the user to view specific layers of the stack only. This would be implemented by simple changes of the view classes. No changes would be needed on the model classes due to the model-view-control architecture.

Conclusion

The aim of this project was to design and implement a teaching tool that simulates the processing and interactions involved at all stages of a TCP/IP communication. The identified users of this application are university students, lecturers and engineers studying the networking field.

The project succeeded in creating a TCP/IP simulation by simulating Client and Server computers, each with their own TCP/IP stack. A message is entered into simulated software on the Client computer so that the user can then view the processing and message passing conducted in order for TCP/IP to deliver the message to the Server software. The application displays the entire process within a single window. The structure of the display identifies the two computers and the individual TCP/IP stack layers on each computer. The application can also simulate the presence of network hardware such as Hubs, Switches and Routers between the Client and Server computers. How these devices process the messages and interact with the other computers is also simulated. The structure of the simulated TCP/IP stack and the processing conducted by each layer was implemented in an identical fashion to that of a real TCP/IP stack.

The interface was designed to portray the logical hierarchical structure of real TCP/IP stacks to the user. A colour scheme was implemented to highlight how communicating stack layers correspond to one another. Individual portions of data were formatted to illustrate how their structure is implemented in TCP/IP and how that structure effects how that data is processed by TCP/IP. The simulation proved to be an effective teaching tool and received positive feedback from users. It fulfilled all the requirements that were specified during the elicitation process. It was implemented in a modular structure that mimics that of the TCP/IP specification. This modular structure and defined interfaces were developed to allow further units to be added to simulate additional TCP/IP functionality if required at a later date. The application was implemented using model-view-control architecture. This separated the simulation's processing from the user interface. This allows the interface to be customised to display specific information with no need to alter the model's code to implement it.

In order to efficiently manage the development and implementation of the software, various project management and software engineering techniques were reviewed. The most suitable were selected and utilised for this project. These techniques proved successful in ensuring a robust final product that was delivered by the deadline that fulfilled all its requirements.

Appendix

Bibliography:Websites:

- Cisco Systems Company Website (2007), www.cisco.com, San Jose, California, USA, Cisco Systems Inc.
- O'Reilly – Safari Books Online. (2007), <http://www.oreilly.com/>, Sebastopol, California, USA, O'Reilly Media, Inc.
- Peking University (2007), *TCP/IP for Internet Demonstrators*, <http://en.pku.edu.cn/>, Beijing, China.
- Mc Quaid, Jim (2007), *Network Performance Daily: Whiteboard Series*, <http://www.networkperformancedaily.com/2007/08/>, Austin, Texas, USA, NetQoS Inc.
- The Institute of Direct Marketing Company Website (2008), <http://www.theidm.com>, Teddington, Middlesex, UK, The Institute of Direct Marketing Training Ltd.
- Nexion Data Systems Online (2008), *Frequently Asked Questions*, <http://www.nexiondata.com>, Coopers Plains Qld Australia, Nexion Data Systems.
- Bart Guetti, Environmental Systems Research Institute Website, <http://www.esri.com/index.html> Redlands, California, USA, Environmental Systems Research Institute, Inc.
- American Dynamics Company Website (2008), <http://www.americandynamics.net>, San Diego, USA, Tyco International Ltd.
- About.com Networking Website, <http://compnetworking.about.com/>, About Inc, The New York Times Company, New York City, USA.
- Request For Comment (RFC) Database, <http://www.iana.org/protocols/>, The Internet Assigned Numbers Authority (IANA), USA.
- RAD Data Communications Website (2008), <http://www.raduniversity.com/>, RAD Data Communications Ltd. RAD Data Communications, Tel Aviv, Israel.
- Sun Microsystems Java.com Website, <http://www.java.com/en/>, Sun Microsystems, Inc (2005), Sun Microsystems, Santa Clara, California, USA.
- Java™ 2 Platform API Std. Ed. v1.3.1 Website, <http://java.sun.com/j2se/1.3/docs/api/>, Sun Microsystems, Inc (2005), Sun Microsystems, Santa Clara, California, USA

Publications:

- Stevens, W. Richard (1993), *TCP/IP Illustrated, Volume 1 The Protocols*, Massachusetts, USA, Addison-Wesley Publishing Company, ISBN:0-201-63346-9.
- Casad, Joe (2003), *SAMS Teach Yourself TCP/IP in 24 Hours*, Indiana, USA, Sams Publishing, ISBN:0-672-32565-9.
- Tanenbaum, Andrew S. (2003), *Computer Networks Fourth Edition*, New Jersey, USA, Pearson Education International, ISBN:0-13-038488-7.
- Peterson, Larry L. & Davie, Bruce S. (2003), *Computer Networks: A Systems Approach Third Edition*, San Francisco, USA, Morgan Kaufmann Publishers, ISBN:81-8147-206-3.
- Joan Jackson, External Lecturer for the University of Birmingham (2004), *Software Engineering A – Project Genesis*, University module reference material.
- Joan Jackson, External Lecturer for the University of Birmingham (2004), *Software Engineering A – Planning*, University module reference material.
- Mike Wooldridge, Lecturer for the University of Liverpool (Unknown Date), *Software Project Management, Lecture 5 handout*, University module reference material.
- Joan Jackson, External Lecturer for the University of Birmingham (2004), *An Introduction to Project Management – Managing Risk*, University module reference material.
- Mike Field & Laurie Keller (1998), *Project Management 2002 Edition*, The Open University, Thompson Learning, London, UK, ISBN:1-86152-274-6.
- Ian Sommerville, *Software Engineering - 7th Edition*, International Computer Science Series (2004), Addison Wesley Publishing, ISBN: 978-0321210265.

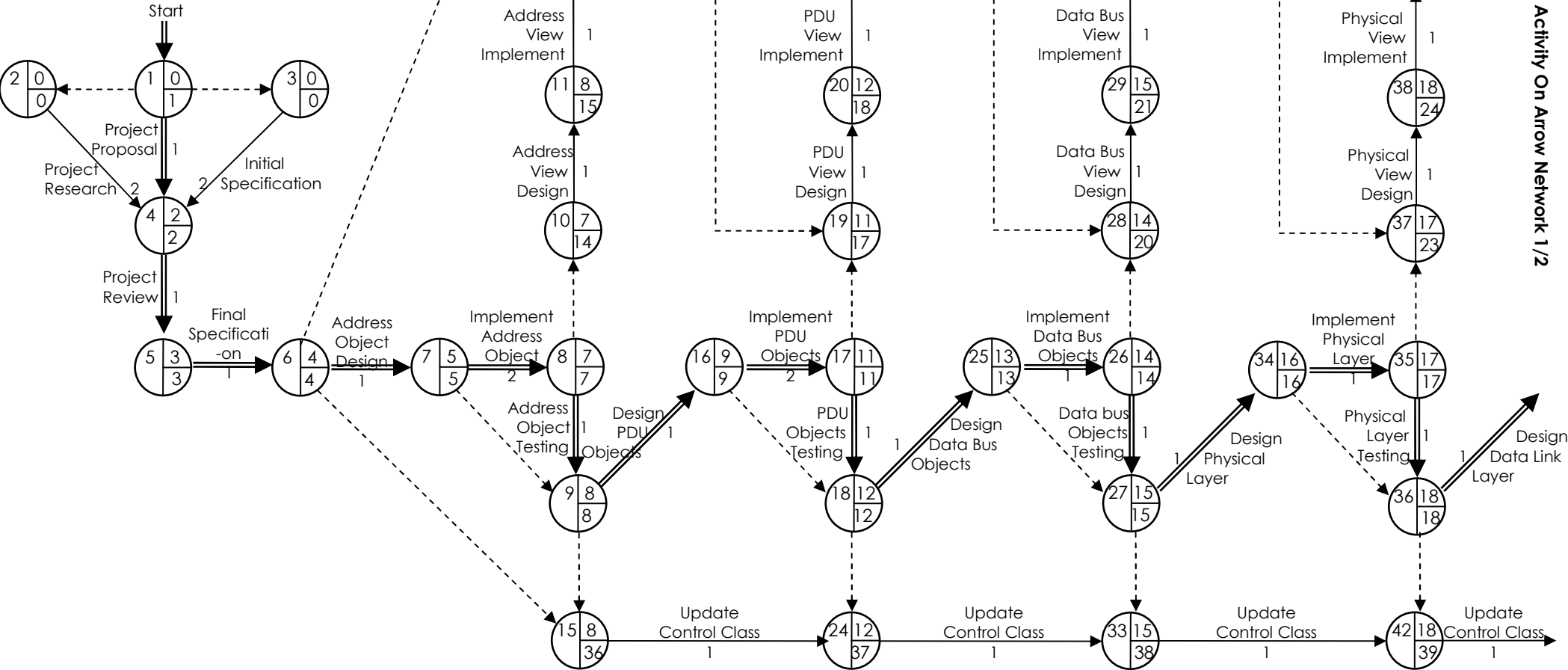
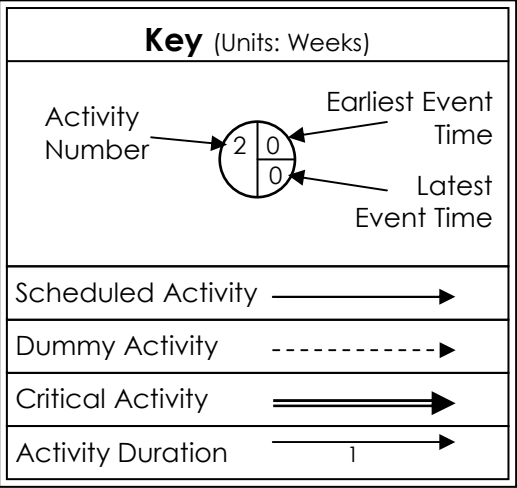
CD Contents:

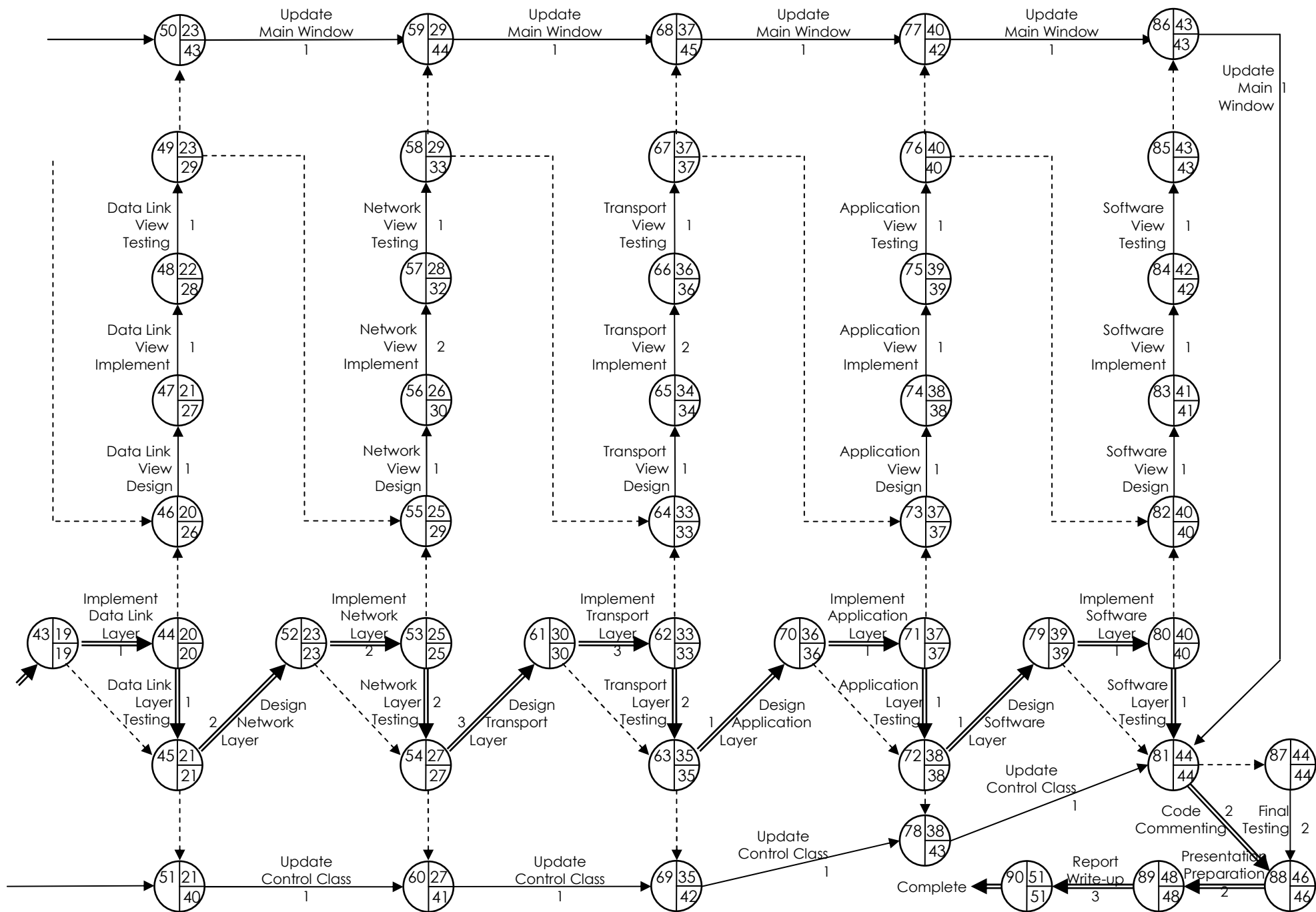
This report was submitted in conjunction with a CD. This disc contains a digital copy of this report, the slide show used during the project demonstration and the application's source code.

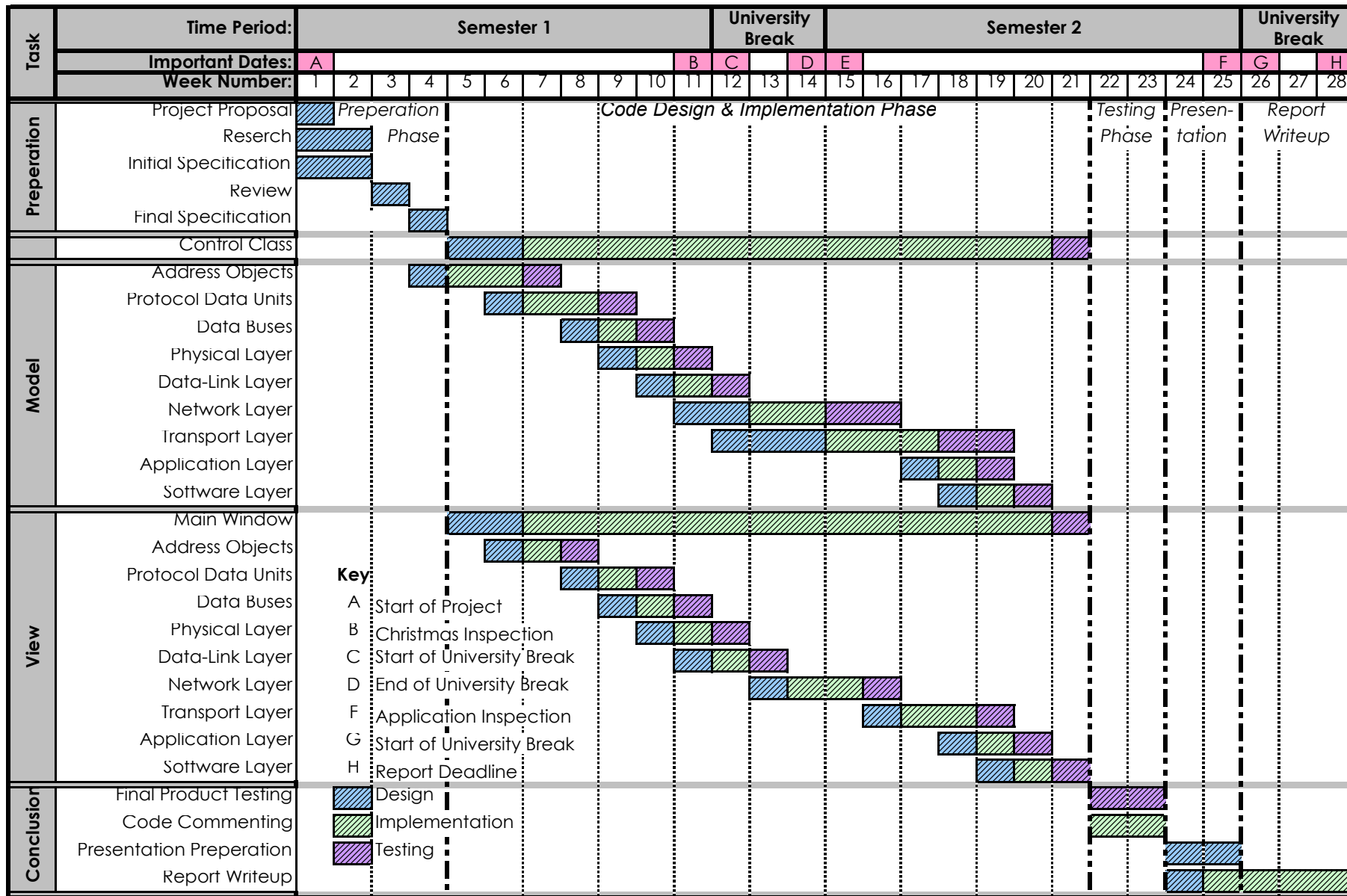
The produced application has been compiled into the JAR file "TcpIpsimulation.jar". This file can be executed from the command line or user interface of any computer with the Java Virtual Machine installed.

The source code has been included. This is in the file structure used by the Netbeans IDE. The source files can be viewed in a normal text editor or the entire project can be mounted in Netbeans. This is achieved by running Netbeans, selecting File → Open Project → locate the project folder "TestBed" and select Open Project Folder. The application can now be viewed or run from within the Netbeans IDE.

The Application's generated JavaDoc is viewable by loading the javadoc\index.html file in a web browser.







Appendix - Gantt Chart

Appendix – Allowed IP Addresses

Class	Octet 0									.	Octet 1									.	Octet 2									.	Octet 3									Address Class	Address Type	Address Allowed
	Dec	128	64	32	16	8	4	2	1		Dec	128	64	32	16	8	4	2	1		Dec	128	64	32	16	8	4	2	1		Dec	128	64	32	16	8	4	2	1			
NA	0	0	0	0	0	0	0	0	0	.	0	0	0	0	0	0	0	0	.	0	0	0	0	0	0	0	0	.	0	0	0	0	0	0	0	0	None	This host	No			
	255	1	1	1	1	1	1	1	1	.	255	1	1	1	1	1	1	1	1	.	255	1	1	1	1	1	1	1	.	255	1	1	1	1	1	1	1	Local	Broadcast Local	No		
	127	0	1	1	1	1	1	1	1	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	None	Loopback Address	No		
A	1-126	0	x	x	x	x	x	x	x	.	0	0	0	0	0	0	0	0	.	0	0	0	0	0	0	0	0	.	0	0	0	0	0	0	0	0	Class A	Network Address	No			
	1-126	0	x	x	x	x	x	x	x	.	255	1	1	1	1	1	1	1	1	.	255	1	1	1	1	1	1	1	.	255	1	1	1	1	1	1	1	Class A	Broadcast Address	No		
	1-126	0	x	x	x	x	x	x	x	.	0	0	0	0	0	0	0	0	0	.	0	0	0	0	0	0	0	0	.	1	0	0	0	0	0	0	1	Class A	Router Port Address	Yes		
	1-126	0	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	Class A	Host Address	Yes		
B	128-191	1	0	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0	0	0	0	0	0	0	0	.	0	0	0	0	0	0	0	0	Class B	Network Address	No		
	128-191	1	0	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	255	1	1	1	1	1	1	1	.	255	1	1	1	1	1	1	1	Class B	Broadcast Address	No		
	128-191	1	0	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0	0	0	0	0	0	0	0	.	1	0	0	0	0	0	0	1	Class B	Router Port Address	Yes		
	128-191	1	0	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	Class B	Host Address	Yes		
C	192-223	1	1	0	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	0	0	0	0	0	0	0	0	Class C	Network Address	No		
	192-223	1	1	0	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	255	1	1	1	1	1	1	1	Class C	Broadcast Address	No		
	192-223	1	1	0	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	1	0	0	0	0	0	0	1	Class C	Router Port Address	Yes		
	192-223	1	1	0	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	2-254	x	x	x	x	x	x	x	Class C	Host Address	Yes		
D	224-239	1	1	1	0	x	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	Class D	Network Address	No		
E	240-247	1	1	1	1	0	x	x	x	.	0-255	x	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	.	0-255	x	x	x	x	x	x	x	Class E	Network Address	No		

* Where X signifies either a 0 or a 1