George Grasham

Hessle High school and sixthform  Candidate Number 3365

NSHG: A Network Simulation Hacking Game

Table of Contents

[Analysis 3](#_Toc8988086)

[1.1, 1.2 Introduction (What is networking and why is it important) 3](#_Toc8988087)

[Interview Summary 6](#_Toc8988088)

[Teacher Interview 6](#_Toc8988089)

[Student Interviews 7](#_Toc8988090)

[The Current system 7](#_Toc8988091)

[What’s the problem? 8](#_Toc8988092)

[1.4 Success Criteria 8](#_Toc8988093)

[Limitations of my system 9](#_Toc8988094)

[Possible Solutions 9](#_Toc8988095)

[1 Revision Game 9](#_Toc8988096)

[2 Simulation game 9](#_Toc8988097)

[3 Story game 10](#_Toc8988098)

[Chosen solution (and why) 10](#_Toc8988099)

[Research 11](#_Toc8988100)

[The OSI model 11](#_Toc8988101)

[Headers 11](#_Toc8988102)

[Routing Tables 20](#_Toc8988103)

[Design 22](#_Toc8988104)

[Overall Summary 22](#_Toc8988105)

[Packet protocol 22](#_Toc8988106)

[File Design 22](#_Toc8988107)

[Record Design 23](#_Toc8988108)

[Information Transfer 23](#_Toc8988109)

[Hierarchy charts 23](#_Toc8988110)

[Server 24](#_Toc8988111)

[Client 25](#_Toc8988112)

[PCL 26](#_Toc8988113)

[IPSO Charts 28](#_Toc8988114)

[Server 28](#_Toc8988115)

[Client 28](#_Toc8988116)

[Data Dictionaries 29](#_Toc8988117)

[OOP Class Design 30](#_Toc8988118)

[Applications 30](#_Toc8988119)

[Protocols 34](#_Toc8988120)

[Systems 37](#_Toc8988121)

[IP 38](#_Toc8988122)

[MAC 39](#_Toc8988123)

[Network 39](#_Toc8988124)

[NetworkInterfaces 40](#_Toc8988125)

[PacketProtocol 42](#_Toc8988126)

[Interface Design 42](#_Toc8988127)

[Client 42](#_Toc8988128)

[Server 44](#_Toc8988129)

[Algorithms 45](#_Toc8988130)

[Network 45](#_Toc8988131)

[Testing 46](#_Toc8988132)

[Testing Strategy 46](#_Toc8988133)

[Testing Plan 47](#_Toc8988134)

[Evidence 47](#_Toc8988135)

[Evaluation 47](#_Toc8988136)

[Performance against the objectives 47](#_Toc8988137)

[Independent feedback 47](#_Toc8988138)

[Improvements 47](#_Toc8988139)

# Analysis

## 1.1, 1.2 Introduction (What is networking and why is it important)

The fundamentals of communication and networking is a massively important topic in the computer science A-level, with the growing connectivity of the world through the internet of things it is growing ever important that any computer scientists worth their weight will need a strong understanding of what these concepts are and how they ate implemented in the real world.

The topic in the A-Level is split up into 4 topics which are each split into further sub topics, these main and sub topics are as follows.

1. Communication
   1. Communication methods
      1. Serial/parallel
      2. Synchronous/Asynchronous
      3. Start and Stop bits
   2. Communication basics
      1. Baud rate, Bit rate, Bandwidth, Latency, Protocol
      2. Differentiate baud rate and bit rate
      3. Understand the relationship between bit rate, baud rate and bandwidth.
2. Networking
   1. Network Topology
      1. Difference between and how star and bus topologies work
   2. Types of networking between hosts
      1. Explain when peer to peer may be used
      2. Explain when client-server may be used
   3. Wireless networking
      1. Explain the purpose of Wi-Fi
      2. Be familiar with the components required for wireless networking
      3. Be familiar with how wireless networks are secured
      4. Explain the wireless protocol Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with and without Request to Send/ Clear to Send (RTS/CTS).
      5. Be familiar with the purpose of Service Set Identifiers (SSID)
3. The Internet
   1. The Internet and how it works
      1. Understand the structure of the internet
      2. Understand the role of packet switching and routers
      3. Know the main components of a packet
      4. Define Router and gateway and consider where they are used
      5. Explain how routing is achieved across the internet
      6. Describe the term uniform resource locator in the context of internetworking
      7. Explain the terms fully qualified domain name (FQDM), Domain name and IP address
      8. Describe how domain names are organised
      9. Understand the purpose and function of the domain name service and its reliance on the domain name server (DNS) system
      10. Explain the service provided by internet registries and why they are needed
   2. Internet Security
      1. Understand how a firewall works (packet filtering, proxy server, stateful inspection)
      2. Explain symmetric and asymmetric encryption and key exchange
      3. Explain how digital certificates and digital signatures are obtained and used
      4. Discuss worms, Trojans, and viruses , and the vulnerability’s they exploit
      5. Discuss how improved code quality, monitoring, and projection can be used to address worms, Trojans and viruses.
4. The TCP/IP protocol stack
   1. TCP/IP
      1. Describe the role of the four layers of the TCP/IP stack (application, transport, network and link)
      2. Describe the role of sockets in the TCP/IP stack
      3. Be familiar with the role of MAC addresses
      4. Explain what well known ports and client ports are used for and the differences between them
   2. Standard application layer protocols
      1. Be familiar with the following protocols
         1. FTP
         2. HTTP
         3. HTTPS
         4. POP3
         5. SMTP
         6. SSH
      2. Be familiar with FTP client software and an FTP server, with regard to transferring files using anonymous and non-anonymous access
      3. Be familiar with how SSH is used for remote management
      4. Know how an SSH client is used to make a TCP connection to a remote port for the purpose of sending commands to this port using application level protocols such as GET for HTTP, SMTP commands for sending email and POP3 for retrieving email
      5. Be familiar with using SSH to log in securely to a remote computer and execute commands
      6. Explain the role of an email server in retrieving and sending email
      7. Explain the role of a web server in serving up web pages in text form
      8. Understand the role of a web browser in retrieving web pages and web page resources and rendering these accordingly
   3. IP address structure
      1. Know that an IP address is split into a network identifier part and a host identifier part
   4. Subnet masking
      1. Know that networks can be divided into subnets and know how a subnet mask is used to identify the network identifier part of the IP address
   5. IP standards
      1. Know that there are currently two standards of IP address, v4 and v6.
      2. Know why v6 was introduced
   6. Public and private IP addresses
      1. Distinguish between routable and non-routable IP addresses.
   7. DHCP
      1. Understand the purpose and function of the DHCP system
   8. Network Address Translation (NAT)
      1. Explain the basic concept of NAT and why it is used
   9. Port Forwarding
      1. Explain the basic concept of port forwarding and why it is used
   10. Client server model
       1. Be familiar with the client server model
       2. Be familiar with the Web socket protocol and know why it is used and where it is used.
       3. Be familiar with the principles of Web CRUD Applications and REST:
          1. CRUD is an acronym for:
             1. C – Create
             2. R – Retrieve
             3. U – Update
             4. D – Delete.
          2. REST enables CRUD to be mapped to database functions (SQL) as follows:
             1. GET → SELECT
             2. POST → INSERT
             3. DELETE → DELETE
             4. PUT → UPDATE.
       4. Compare JSON (Java script object notation) with XML.
   11. Thin vs Thick client computing
       1. Compare and contrast thin-client computing with thick-client computing.

This is a vast specification and in order to answer the questions fully the student must understand all of these points and be able to apply them and evaluate aspects of them. For example discuss how the TCP/IP stack works, what information is needed my each layer and what information is passed between them.

To begin my project I am going to interview my end users, whom are teachers and students, to see some of the problems they have with teaching and learning networking concepts, engaging students with the material and the understanding of the abstract concepts covered in networking and the internet.

The interview will be semi-structured having a list of required questions, written before the interview, and asking more questions based of the end users answers. This will allow for more in-depth information to be gathered reducing the risk of a possible re-design when showing the end user the “alpha” version of the project.

During the interview I will be taking notes

In the interviews I will discuss the problems and then possible solutions to those problems. I will focus on managing the expectations of the end user as to not have them expect more than I can produce in the given time.

## Interview Summary

### Teacher Interview

After having an interview with the teacher he has given me a list of lessons/concepts that are covered in networking and the internet when teaching the subject that are the hardest to grasp, this is TCP/UDP DHCP and ICMP. These are the topics that will need to be assessed for my solution for this project as implementing the whole solution would take a vast amount of time

The students find it very difficult when being talked at for the full lesion blocks and then being given a work sheet to complete. He was unsure why this was the case. However he would like to keep the idea of having constant feedback from the students in the form of task results.

At the end of the solution there must be an end of unit test which assesses everything my solution has implemented. This should output the results in a csv file so it can be processed externally. As my system cannot be trusted in a school setting to be secure.



### Student Interviews

The work sheets are boring



I don’t understand why we need to know about this.



When would I ever need to use DHCP in real life?



I want to actually do something to do with networking rather than just looking at PowerPoints and answering questions.



## The Current system

The subject of networking is usually taught in 4 weeks with 5 hours a week. The lessons 1 hour long with 1 block of 3 lessons and 1 block of 2 lessons each week. This results in a total of 20 hours to teach the specification.

The class size currently is 7 people thought this varies from year to year (with a maximum of 15)

The lessons consist of a power point with several task sheets to provoke questions from the students and assess their ability while going through the lesson. These tasks are fast and easy to complete.

At the end of the subject there is a 2 hour subject test which allows the teacher to make sure that the teaching has been successful and that the students have taken in the information this can then be processed to find what parts of the subject were misunderstood and what was received well which is used for revision/ recaps for a re-test if it is deemed necessary by the teacher.

### What’s the problem?

The current problem with the system is the large amount of time and complexity seen when implementing the system, the worksheets are dry and boring, and the long periods of teaching can send the students into a daze.

## 1.4 Success Criteria

Know the main components of a packet

Describe the role of the four layers of the TCP/IP stack (application, transport, network and link)

Be familiar with the role of MAC addresses

Explain what well known ports and client ports are used for and the differences between them

Know that an IP address is split into a network identifier part and a host identifier part

|  |  |
| --- | --- |
| Success Criteria | Operationalised |
| The solution must implement and test the specification points shown above | Have a check list of all teaching/information points that need to be taught & check they are all covered. Further check that they someone who hasn’t completed the specification before can use the program, with the test in front of them, and answer all of the questions fully. Given time |
| The solution must be engaging to students to make them want to learn about networking. | Test with the students to make sure that they enjoy the solution and engage with it. Found through observation and a self-report after they finish using the solution |
| The solution must have an “End of unit test” in order for the teacher to be sure all the students fully understand the topic | Have the end of unit test implemented as well as the one in the system to compare results to make sure the system is working, this ensures reliability of the new test. |
| The end of unit test would be self-marked | Have a teacher and the system mark the test to make sure it is marked correctly (interrater reliability) |
| Results should be output as a csv file making it easy to analyse | Have the results of each of the students be output as a CSV value (or another file type that can be imported into google sheets/excel) |
| The solution should be written asynchronously so the code doesn’t block the UI | Uses begin and end call-backs in code, can be observed by UI not stopping responding when input is provided |
| A student should be able to work easily with less, but not no, input from the teacher | When testing with students the students shouldn’t need help on average any more then in a normal lesson, this will be judged by me and the teacher who are both familiar with the amount of questions usually asked |
| Should allow for automatic collection of results from clients meaning client server code must be implemented using Sockets and my own protocol | Uses socket based programming |

## Limitations of my system

With this system being used in a school environment there are many technical limitations to my solution. Firstly the school network has a large firewall that will blocks lots of network traffic to make sure the school network says secure. This means that any data will have to be hosted within the school system so it can be accessed while in school. However this has the problem that the data would only be available while in school. A possible (though unlikely) solution to this is allowing an exception through the school firewall to allow the data to be accessed from outside the system.

Another limitation is that I will not be able to store information about students as otherwise I wouldn’t be able to be able to ensure its security (as I am not qualified). My solution to this is not storing any personal information about the student meaning it cannot be stolen from me. This does mean that I can’t do any data analysis on the students and instead have that handled by the current data handling system, which has been recently updated.

## Possible Solutions

I brainstormed some ideas with my end users that I thought could solve the problems my end user was having from there I will show those ideas to the teacher and the students and see what idea they will find most engaging and will be most representative of what my end user had in mind

### 1 Revision Game

My first idea was to create a revision game where students would answer questions and be told if their answers were right or wrong. When the student got a question wrong it would be added to a Challenge bank to be used later. The challenge bank would allow students to continue with questions then revisit what they found hard to see if they had learned what they got wrong after a period of time, this is to make sure it goes into their long term memory instead of short term memory.

When a student gets a question correct they are given points in order to associate correct answers with a positive outcome. This is a great idea however many implementations like this already exist online and with the scope of my project I wouldn’t be able to implement anything near that in time. This also fails to make the teaching much more interesting and arguably would be harder to keep their interest as it would just be reading text of a screen, worse than a teacher talking. Lastly this isn’t suitable as it wouldn’t be able to actually teach the student

### 2 Simulation game

This idea would be a simulation of a network that would have the users “hack” into different systems using the required knowledge for the course. They would be hacking to find a flag that would be hex value. The flag would then be sent in as their answers, for each flag they send they would get a mark and a time stamp. Each flag would be attributed to a different specification point.

This solution would be much more interesting for the students as hacking would be much more interesting then filling out a work sheet. Lastly this will have the ability to teach and assess with multiple different networks. For this alpha project I could have 2, one to teach the concepts required and another to assess. It also would have the ability to output to a csv file with the username, mark, and time they gained it, if they gained it. This however would be a lot of work as I would have to implement each of the specification points that would need to be assessed.

### 3 Story game

This would be a story game where the student plays as a packet on a network roaming around and learning about different concepts. There would be different levels where the user would learn about different concepts. This would involve an overarching story and some simple combat/ challenges when traveling over the network. This would keep the students interested with a story and challenges.

This solution is very good for keeping the students interested however it would be difficult to stop the students from skipping text and just doing the challenges. And there would be no way to assess the students abilities in a story game so the problem of the boring task sheets would still be there, however this wouldn’t be a deal breaker.

## Chosen solution (and why)

I proposed my three solutions to some students who are potential end users. And received their feedback about each of the ideas which is included below and asked them which of the three they would prefer to use in the classroom.

“I have the attention span of a fish so the ‘hacking’ game would keep me occupied and engaged for longer. Especially when compared to the first solution and story games are boring”

“Defiantly the hacking game, I’ve always wanted to do something like that in the real world”

“The Hacking game is a good idea, but story games really interesting and I love getting stuck into them”

I conducted a survey of the students in my class to see what the overall results were

|  |  |
| --- | --- |
| Revision Game | 0 |
| Hacking Game | 4 |
| Story Game | 1 |

For me this is good enough evidence that the solution I should implement is the Hacking game.

## Research

Now I will need to conduct some deep research into each of the concepts I will be implementing and how I am going to do so. This is what follows.

### The OSI model

The Open Systems Interconnection model (OSI model) is a way to standardise the communication between systems it is broken down into 7 layers;

1. The physical layer
2. Data Link
3. Network
4. Transport
5. Session
6. Presentation
7. Application

I will go over each of the layers in detail now

#### Layer 1: Physical

This defines the physical, electrical connections between devices e.g. electrical cable or optical fibre. This includes: voltage levels, timing of voltage changes, physical data rates, maximum transmission distances, and physical connectors.

#### Layer 2: Data Link

This layer describes the connections between nodes and doesn’t deal with the physical layers. This layer has basic error correction that will occur on the physical layer. This layer doesn’t deal with connecting multiple nodes together (networking) it is only concerned with the connection between two nodes and terminating it.

This layer can sometimes be divided into two sub layers

##### The Medium access control (Mac) Layer

This is the sub layer responsible for how devices gain access to data and permission to transfer

##### The Logic Link Control (LLC) layer

This layer controls the encapsulation of protocols and error correction

#### Layer 3: Network

#### Layer 4: Transport

#### Layer 5: Session

#### Layer 6: Presentation

#### Layer 7: Application

### Headers

How does data travel across the internet?

I purchased the book The Networking Bible in which I found a reference to something called an RFC so I decided to investigate, the RFC it pointed to was RFC 791 which is the official standard for how the IP Header works and how IP addresses are structured I found that the RFC’s are a great source of info. RFC stands for Request for comments and they are used to show the implementations of systems for example the IP RFC shows how an IP header is formatted and how each field is used.

#### Internet Layer

##### IP [[1]](#footnote-1)

This is the basic framework for the internet, in the A level we are required to know that the IP header contains the source and destination IP addresses. In reality the IP header contains a large amount more than just the source and destination addresses which only take up two fifths of the entire header. 8 of the 20 bytes (minimum) of the header. For my research into how the IP specification and header is formatted I used Wire shark, a piece of software on my pc to intercept all packets coming to and from my system and then using this information along with the RFC 791 which allowed me to see, in the real world, how the IP header Is used and formatted, and what everything meant along with a hex array to see, bitwise, the same thing.

The IP Datagram is formatted in 20 bytes minimum with an optional options datagram at the end of the header. This has been depreciated and isn’t used anymore for this reason I will not be implementing the options and that data will always be length zero. The header is formatted as seen below [[2]](#footnote-2)

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|Version| IHL |Type of Service| Total Length |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Identification |Flags| Fragment Offset |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Time to Live | Protocol | Header Checksum |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Source Address |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Destination Address |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Options | Padding |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

This shows byte by byte how the header is formatted. This is a good time to point out that there are a few values that are over 1 byte in size. E.g. The header checksum which is 2 bytes large, a unsigned 16 bit integer. This means that I will take into account Evidence when programing my solution. I will most likely use the systems default endianness instead of writing my own code to reverse it to the standard as this will not make a difference to the end solution.

Now I will briefly cover what each of the fields in the header represent.

###### Version (4 bits)

This is the version of the IP Header. For example 4 as this uses IPv4. This only changes when using IPv6. Which my solution will not us.

###### IHL (4 bits)

This is the “Internet Header Length” Measured in 4 byte words. For my purposes this will always be 5 as the header will always be 20 bytes in length

###### Type of Service (8 bits)

This is how the packet is treated. For example high or low latency This is formatted as shown below but will not be implemented in my solution

Bits 0-2: Precedence.

Bit 3: 0 = Normal Delay, 1 = Low Delay.

Bits 4: 0 = Normal Throughput, 1 = High Throughput.

Bits 5: 0 = Normal Relibility, 1 = High Relibility.

Bit 6-7: Reserved for Future Use.

Precedence

111 - Network Control

110 - Internetwork Control

101 - CRITIC/ECP

100 - Flash Override

011 - Flash

010 - Immediate

001 - Priority

000 – Routine

###### Total Length. (16 bits)

This is the total length of the datagram. Including the IP Header This is measured in octets (bytes). The minimum length that a host must be willing to accept is 576 bytes. For a IP Header with no options and no datagram would have a total length of 20.

###### Identification (16 bits)

This is the ID of a datagram and is used for when a data gram is fragmented to put all the pieces back together.

###### Flags (3 bits)

These are 3 flags. The first is reserved, then if the second bit is on it specifies that the datagram cannot be fragmented. And off means it may be fragmented. And the Last bit specifies if there are more fragments to come. A value of 1 specifies that there are more fragments.

###### Fragment Offset (13 bits)

This is the offset of the data of the packet. The first fragment has offset of zero. If this held the first 200 bytes the offset of the second packet would be 200.

###### Time to Live (TTL) (8 bits)

This is the time (in seconds) that the packet can exist before being destroyed. If a packet has TTL of 0 then it is discarded and an ICMP packet will be sent to the source IP Address of the packet specifying that this happened.

This is usually implemented as hops instead of seconds as a packet will be forwarded in a matter of milliseconds and a TTL of 30 seconds would be too long and redundant and is instead decremented by 1 every Hop. So a TTL of 30 Hops would be more appropriate in terms of speeds of the internet.

###### Protocol (8 bits)

The Protocol of the header in the datagram of the packet e.g. 1 – ICMP, 6 – TCP.

###### Header Checksum (16 bits)

The ones compliment of the ones compliment sum for each 16 bit word in the header, not including the datagram.

###### Source Address (32 bits) Destination Address (32 Bits)

These are the source and destination addresses of the systems on the network. For example if a system with the IP address 192.168.1.1 wanted to send a packet to the system with IP address 192.168.2.9 The source field would be the senders IP address and the destination field would be the destination systems IP address.

#### Transport Layer

##### UDP

UDP is a transport layer protocol, similar to TCP but much simpler. This means it will take much less time to implement hence I am only using UDP for my sample project.

UDP is a port oriented protocol that is used to transfer data from one application on a source computer to another application on the destination computer. For example HTTP requests would be sent to the port 80 on the http server as this is where a UDP listener would be listening for packets arriving addressed to that port so it can respond to request. UDP isn’t a connection oriented protocol meaning that it, data is sent in a packet by packet manner and there is no way of making sure all data was received by the destination system. Compare to TCP where a connection is first established and held for the duration of the connection, until one end closes it, or it is closed gracefully by either end. UDP has no system like this and just sends data. This is also mirrored in the simple, small header as shown and labelled below.

0 7 8 15 16 23 24 31

+--------+--------+--------+--------+

| Source | Destination |

| Port | Port |

+--------+--------+--------+--------+

| | |

| Length | Checksum |

+--------+--------+--------+--------+

|

| data octets ...

+---------------- ...

###### Source Port

This is port that the information is being sent from, for example when taking to a host on the port 80 the source port would be chosen as any, random, port that isn’t in use or reserved.

###### Destination Port

This is the port on the destination that the data will be passed to, for example when accessing a web page using HHTP the destination port 80 would be used. There is a list of reserved ports in the Assigned Numbers RFC 762.

###### Length

The Length of the Header and Datagram measured in octets (Bytes), minimum value of 8 for a packet with no datagram.

###### Checksum

The ones compliment of the ones compliment sum of each 16 bit word in the header and data. Calculated the same way as the IP Checksum. If the length of the header isn’t a multiple of 2 it is padded with one octet of zeros.

###### Data octets

This is the data of the underlying packet. For example a DHCP Packet.

#### Application Layer

##### DHCP

Now I have the Headers sorted I need to think about what happens when a system first connects to a network. The first thing a system does when it connects to a network is obtain an IP address and other information through DHCP. This means I will have to implement a version of the DHCP protocol which uses UDP. The basic premise behind DHCP is 4 packets, 1 a broadcast from a DHCP client onto their local network to discover a DHCP server if there is one, 2 A DHCP server Responds multicast with an offer to lease an IP address, 3 the client responds (unicast) with a request to the DHCP server requesting the offered IP Address, 4 The DHCP server responds with a DHCPAck packet, which contains confirmation of the IP address, the subnet mask, Default gateway and server[[3]](#footnote-3) [[4]](#footnote-4) [[5]](#footnote-5)

I will use the RFC1 to form the layout of my packets however with the scope of this project and to reduce the processing overhead I will have to simplify it and it will not be fully featured, only the barebones will be there

This is a good time to note that DHCP is an extension of the older BOOTP Protocol so some of the fields in the datagram are now depreciated.

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| op (1) | htype (1) | hlen (1) | hops (1) |

+---------------+---------------+---------------+---------------+

| xid (4) |

+-------------------------------+-------------------------------+

| secs (2) | flags (2) |

+-------------------------------+-------------------------------+

| ciaddr (4) |

+---------------------------------------------------------------+

| yiaddr (4) |

+---------------------------------------------------------------+

| siaddr (4) |

+---------------------------------------------------------------+

| giaddr (4) |

+---------------------------------------------------------------+

| |

| chaddr (16) |

| |

| |

+---------------------------------------------------------------+

| |

| sname (64) |

+---------------------------------------------------------------+

| |

| file (128) |

+---------------------------------------------------------------+

| |

| options (variable) |

+---------------------------------------------------------------+

The DHCP Packet will be structured as seen above (Excerpt from RFC 2131 Page 34).

The options field is defined with a tag (1 byte) and possibly followed by an option length and/or data.[[6]](#footnote-6)

In the DHCP RFC 2131 Page 10 there is a table of the different fields in the DHCP Datagram seen above. I have copied the table here and will comment in more detail what each of the fields are used for and examples.

###### Field (Length in Octets)

RFC Definition

My Expansion

###### op (1)

Message op code / message type.  
1 = BOOTREQUEST, 2 = BOOTREPLY

Used to specify if a message is going to/from the DHCP server

###### htype (1)

Hardware address type, see ARP section in "Assigned" RFC; e.g., '1' = 10mb Ethernet.

Will only have a value of 1 for my implementation as I am not implementing down to the MAC layer.

###### Hlen (1)

Hardware address length (e.g. '6' for 10mb ethernet).

Length, in bytes, of the hardware (MAC) address.

###### hops (1)

Client sets to zero, optionally used by relay agents when booting via a relay agent.

Will be set to zero when sending DHCP packets, used to track how many, if any, relays were used when sending the packet.

###### xid (4)

Transaction ID, a random number chosen by the client, used by the client and server to associate messages and responses between a client and a server.

###### secs (2)

Filled in by client, seconds elapsed since client began address acquisition or renewal process.

###### flags (2)

Flags

###### ciaddr (4)

Client IP address; only filled in if client is in BOUND, RENEW or REBINDING state and can respond to ARP requests.

###### yiaddr (4)

'your' (client) IP address.

###### siaddr(4)

IP address of next server to use returned in DHCPOFFER, DHCPACK by server.

###### giaddr(4)

Relay agent IP address, used in booting via a relay agent.

###### chaddr(16)

Client hardware address.

###### Sname (64)

Optional server host name, null terminated string.

###### File(128)

Boot file name, null terminated string; "generic" name or null in DHCPDISCOVER, fully qualified directory-path name in DHCPOFFER.

###### Options (var)

Optional parameters field. See the options for a list of defined options.

#### DHCP State Transition Diagram

-------- -------

| | +-------------------------->| |<-------------------+

| INIT- | | +-------------------->| INIT | |

| REBOOT |DHCPNAK/ +---------->| |<---+ |

| |Restart| | ------- | |

-------- | DHCPNAK/ | | |

| Discard offer | -/Send DHCPDISCOVER |

-/Send DHCPREQUEST | | |

| | | DHCPACK v | |

----------- | (not accept.)/ ----------- | |

| | | Send DHCPDECLINE | | |

| REBOOTING | | | | SELECTING |<----+ |

| | | / | | |DHCPOFFER/ |

----------- | / ----------- | |Collect |

| | / | | | replies |

DHCPACK/ | / +----------------+ +-------+ |

Record lease, set| | v Select offer/ |

timers T1, T2 ------------ send DHCPREQUEST | |

| +----->| | DHCPNAK, Lease expired/ |

| | | REQUESTING | Halt network |

DHCPOFFER/ | | | |

Discard ------------ | |

| | | | ----------- |

| +--------+ DHCPACK/ | | |

| Record lease, set -----| REBINDING | |

| timers T1, T2 / | | |

| | DHCPACK/ ----------- |

| v Record lease, set ^ |

+----------------> ------- /timers T1,T2 | |

+----->| |<---+ | |

| | BOUND |<---+ | |

DHCPOFFER, DHCPACK, | | | T2 expires/ DHCPNAK/

DHCPNAK/Discard ------- | Broadcast Halt network

| | | | DHCPREQUEST |

+-------+ | DHCPACK/ | |

T1 expires/ Record lease, set | |

Send DHCPREQUEST timers T1, T2 | |

to leasing server | | |

| ---------- | |

| | |------------+ |

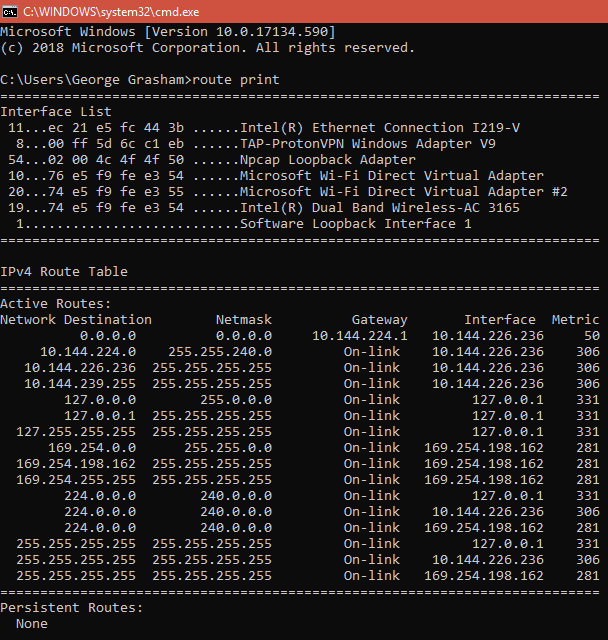
+->| RENEWING | |

| |----------------------------+

----------

### Routing Tables

I will now have to look at how a system decides what adapter to send a packet to for it to forward on. After a brief google search I found that Routing tables are responsible for looking at a packets IP header and deciding what physical connection to send it down. You can access your own routing table in windows by using the “route print” command, I ran this command to see what I could understand from the beginning and what I needed to research more about. The result of my route print command is included below



From here I can see a few familiar things, 1 at the top there is an interface list. This seems to be a list of network adapters that the system has. For me I have a few that stand out, The Inter(R) Ethernet Connection I219-V which is my Lan Ethernet adapter so If I plugged an Ethernet cable into my laptop that adapter would be used, The Intel(R) Dual Band Wireless-AC 3165 which is my current adapter as I am connected to my current network wirelessly. There is also the software loopback which is the loopback adapter for my system, for example if I wanted to ping myself I would use the loopback adapter so I don’t need to be connected to a network.

Next the IPv4 Route Table. This is the routeing table for my system, this confuses me a little but here are my first ideas about each of the columns before research. The destination is the destination address that a packet would have to be addressed to be forwarded to a specific adapter, the Net mask also play a role in this to as the net mask would be applied in the packet that needs to be routed to see if it fits the destination address once it is also had the net mask applied to it. For example for my system if I sent a packet with the address 127.6.1.1 it would be accepted by the 127.0.0.0 entry as if you apply the mask 255.0.0.0 to 127.6.1.1 it yields 127.0.0.0 meaning that is the route it would take. The first entry in these addresses has the 0.0.0.0 address and the 0.0.0.0 mask meaning that any address would be mapped to this entry successfully. This leads me to believe when a system is using the table it doesn’t use a top down approach and instead likely has something to do with the last column “Metric”. The next column is the Gateway. I think this is the address that the packet needs to be forwarded to in order to reach its destination. As of right now I am unsure about on-link but I would like to note that each entry with on-link is a local address. I believe that Interface is the specific adapter that the packet needs to be routed down to reach its destination. The reason it uses the IP address of the interface instead of the mac address is unclear to me but could have something to do with setting the source address of the packet. I will now research to see what I was correct about

From my research I know that what I thought was basically correct, when sending a packet, the routing table is used to know what interface to use. The system will look up the table in a decrementing fashion looking at the highest metric first and getting smaller, the destination and netmask are used to see if the sending destination address is matched by the rule, if the address is a match then the packet is sent on the appropriate interface. The Gateway specifies that the destination isn’t on the local network and needs to be forwarded to that gateway for NAT to take place and then be forwarded from there onto the larger network.[[7]](#footnote-7),[[8]](#footnote-8)

My next question was how Microsoft populates this table. I have decided that I will not be using any wans and will instead just use the data provided by DHCP to populate the routing table.

# Design

## Overall Summary

My Project is a Network Simulation Hacking Game or “NSHG” that will aim to interest students in learning networking concepts in networking and the internet in a fun, easy but still academic way. There should be 2 main pieces of software in my package. The client and the server.

The server will be ran by one of the teachers possibly on one of their pc’s for this reason it must be light and easy to run. The server will let the student’s login using their client, run a simulation of the network for the students to “hack” and learn through, and lastly a results collection system will be implemented to collect the results of the tests done by the students. Success will be gauged by which “flags” they were able to collect with each flag requiring a different skill in order to acquire.

The client will be run on the student’s desktop computers and will be used to login to and communicate with the server. The client will be a virtual command line interface to their “system” in the game and will provide them with the material needed in order to progress through the learning levels and then hints/tips for the tests and the correct direction to go.

There will also be a portable class library which will allow for shared code between the server, client and test projects. The PCL will contain the simulation code such as the System code which will act as a pc in my simulation. It will also include most of the server side coding allowing for the implementation of the server to be easily changed, for example adapting the front end to instead be a web interface wouldn’t be difficult. This would make my system much more adaptable.

### Packet protocol

From my previous experience working with networking in C# I know that it can be very difficult and buggy when sending packets, for example if two packets are sent from the server to the client one after another very fast the client may receive that as one large packet. This is especially bad if the client will need to process these two responses separately. Previously I used some 3rd party code written by Stephen Cleary[[9]](#footnote-9) which is admittedly very old but to my knowledge still works. I have elected to use this code as it is time consuming and a pain to write my own packet protocol code especially with the extremely large workload that will be coming with implementing other internet standard code for my simulation and to me this is a known solution that is well documented and easy to implement.

It would also be a good time to show my reference for C# socket programming which is in the way of a GitHub repo and I have reprehended this a lot when writing my own net code however it is vastly simpler then my net code will be.[[10]](#footnote-10)

### File Design

A file will be stored in xml format. XML works with having one root node then child nodes within that. When loading an XML file I will look at the name of each of those XML child Nodes and load the xml inside that node based of the name, for example if a node was named flag, this would prompt the loader to look inside that node for a mark and a flag. An example of a flag xml node would be

<Flag>

<id>192.168.1.1</id>

<mark>Parsing DHCP Datagram for DHCP Server Address</mark>

</Flag>

The same would go for; any they of system, scenario, connection ect

This makes the file pretty easy to generate and load.

### Record Design

Records will be exported to a csv file, the first row will be the username of each client, the first column will be each flag’s mark text. Each user will have the time of their flag in each of the corresponding rows below their name. This is so the teacher can make sure the flags weren’t collected too fast. E.g. if one student got all the flags within one minuet in a 5 minuet scenario this may pull up some red flags (haha) for the teacher indicating they managed to cheat.

### Information Transfer

Information can be transferred from client to server using this set of rules.

* the first word of the packet will

## Hierarchy charts

I will now show the hierarchy charts for each of the pieces of software along with a description of what each modules function will be.

### Server

|  |  |
| --- | --- |
| Module | Function |
| **Simulation** | Running the simulation that the clients will interact with, this involve ticking the network over and over after a specific interval of time to ensure that communications are sent & applications run. Also provides an interface to run commands on any system in the network and interface with the network |
| Tick | This procedure will iterate through each system in the network and tick their AI, and add one to the amount of ticks. This will also stop any other instance of this command being ran at the same time so only one tick can happen at a time. |
| Tick Timer | This is a ticking clock that when running will send a pulse every cycle, this will be a delegate which is subscribed to an event that is raised every cycle. The cycle time is variable. There will be a system in place to make sure that the previous tick has finished completing before the next tick starts running. This will run the tick commands for each system in the pseudo network. |
| Command | This is a command line interface that will allow the server operator to start, stop and change the interval of the tick timer. It will also enable the operator to edit each & run commands on each system in the network. This will allow the server operator to assist students who are struggling with what commands to run. Will also allow the user to load networks and generate reports. |
| **User Interface** | This is a very simple UI for the user and will be a WPF XAML interface which will basically just be a command line however instead of typing into the same window that the log appears you will type into a different input text box and either press return or a command button. |
| Log | This will be large text box which is a log of all command any users are sending, packets being sent between systems. |
| Command entered | Small text box that can be typed into, will be responsible for cleaning any input, removing whitespace, and logging the commands onto the command log. This text box should accept a return |

### Client

|  |  |
| --- | --- |
| Module | Function |
| **Connect Screen** | Used to establish a connection to the server |
| Request connection | Will begin attempting to connect to the IP address supplied by the user |
| Return connection status | Will tell the user how many attempts have been made to connect to the server (max 10). If connected will move to the login portal |
| **Login Portal** | This module will only be used once when connecting to the server. You will need a user name and, if that username already exists the password, otherwise the server will generate one and send it to the user. |
| Submit new user request | Will send a packet to the server prefixed with “new” then the username the user provided |
| Submit login request | Will send a packet to the server prefixed with ”login” then the username followed by a space then the password supplied by the user. |
| Display login results | Will tell the user if they were successful logging in/creating a new account. In the case of a new account will also tell them their password. Will then allow the user to move on to the play screen with the press of a button |
| **Play Screen** | The UI for this package will be more complex than any others and will have a main screen for logging in, then progress to a main screen where there will be an interface for their system along with a results entry page |
| Command Log | There will be a main command log for their system. Which will be a large text box that cannot be typed into |
| Command Input | The command input is a text box that can be typed into, that when the return key is pressed will send an appropriate command to the server |
| Scenario Log | There will be a command log and input screen for each application slot on the system, even those without any applications running this will allow the user to start running new applications. |
| **Flag Management** | This screen will be used to submit flags to the server, then to see what the results are of sending that flag, e.g. successful or unsuccessful |
| Submit flag | This will submit a flag to the server through the server connection, will be prefixed with ‘flag’ followed by the flag itself |
| Flag results | Will receive any responses prefixed with ‘flag’ then display the rest of the data on the packet screen. |
| **Server Connection** | This is responsible for all communication, this will be a socket base connection running TCP and the packet protocol, which prefixes all packets with the length of bytes. |
| Receive data | This will pass the data to the packet protocol (seen in PCL) where it will parse the data and wait for a full packet to be received then the packet will be processed by the current window |
| Send data | This will wrap the data given with the prefix then send the data on the TCP socket. |

### PCL

|  |  |
| --- | --- |
| Module | Function |
| **Network** | This is the bulk of the simulation, provides methods to ready and write xml files and generate the networks linking all the system objects together, this part of the code will also contain the player management keeping a list of flags |
| System List | This is a list of all the systems in the network |
| Connection List | List of all the network connections between the pseudo systems |
| Player Slot Management | Manages all of the player slots that are free for a user to login and claim, also keeps track of any claimed systems. And tracks flags |
| XML Load | This will load an xml file and parse the child nodes of the root node, looking at the nodes names and passing that child node to a class to be instantiated then added to the network. |
| XML Save | This is only used for testing purposes to make sure that systems will generate from xml, will not generate applications |
| **Systems** | This is a set of systems which are abstract implementations of a computer system for my simulation. Will all run separately and, without the tick, will not run any code. |
| XML Load | Will be capable of creating a system object from an xml file for network loading |
| XML Save | Will have to be capable of saving to xml for testing purposes. |
| AI | AI will be implemented using a tick system which will tick all subsequent network adapters and applications contained within the system |
| Application Slots | This is a list of slots that can be filled with applications. If no application is loaded the slot will be null |
| Network adapters | These are another layer of abstraction that are used to communicate with other systems. When a packet is received is then passed to an internal packet processing method inside the system where it is distributed to other applications |
| **Applications** | This is the logic that will run on a system, an example of what an application might be is a routing table or packet sniffer |
| Routing Table | The routing table is used to route packets from the current system, any system that is capable of sending data over a network will need a routing table in order to decide what adapter to send data across |
| Packet Sniffer | A packet sniffer will read all packets that are sent to a system and, if enabled, show their content to the user. This will be used to make sure students understand how data is sent |
| DHCP Client | This will be used to request and confirm an IP address and will also gain other knowledge about the network such as the subnet mask and the default gateway |
| DHCP Server | The DHCP server is responsible for handing out IP addresses and distributing data around the network |
| **Headers** | These are headers for different protocols and will all have implementations to turn them to bytes. |
| **Types** | These are underlying types I will define to make handling the data easier |
| IP Address | 4 bytes long will include incrementation, bit wise And and or, and equality operators |
| MAC Address | 6 bytes long will include incrementation, bit wise And and or, and equality operators |
| **Client/Server Communication** | This aspect of the program will be responsible for communication with the client and accepting connections, this will most likely be a part of the network module to make reprehending easier and for code efficiency |
| Packet Header | The packet protocol will be used to prefix each packet with a 4 byte integer that is its length, same for receiving |
| Packet Type | The type of packet will be determined by the first work when the data is processed into an asci string e.g flag 12345678 would be treated as a flag packet |
| Constants | Constants such as the max send/receive size and the Port. |

## IPSO Charts

These charts show what modules deal with Input Processing Storage and Output in each program.

### Server

|  |  |  |
| --- | --- | --- |
| IPSO | Program Section | Item |
| Input | Command Entered | When a command is entered into the text box and confirmed by either the enter button or return key being pressed, will strip the command and empty the command text box, deals with adding the command to the list of used commands . Will then call command for processing and storage. |
| Processing | Tick | Calls the tick command on the network and increments the tick variable, makes sure only one tick can occur at once |
| Tick Timer | Will count time and make sure ticks are started on a regular basis |
| Command | Processes the command to see what the user is trying to start/stop, see algorithm for details. |
| Storage | Command | Can be used to generate a report to a csv file. |
| Output | Log | Outputs whatever is passed to it to the main log box of the server. |

### Client

Client IPSO

|  |  |  |
| --- | --- | --- |
| IPSO | Program Section | Item |
| Input | Connect | IP address of server |
| Login | Username  Password  New User or Login |
| Command Input | Command (string) |
| Flag Management | Flag to submit |
| Receive Packet | How many bytes received  remove x many bytes from receive buffer  pass that many bytes to the packet protocol |
| Processing | Packet Protocol | Check if correct amount of bytes received from the preceding length prefix  if all bytes received process Packet  else wait for more data |
| Process Packet | Encode bytes of data to a string  split string on every space  switch on first word in string and push data to correct text box |
| Storage | Command Log | When the client sends a command add command to command log |
| Scenario Log | When the client receives a message add message to scenario log |
| Output | Scenario Box | When a scenario message is received it is displayed in the scenario box |
| System Box | When a system log message is received it is displayed in the scenario box |
| Flag Response | Once a flag has been submitted it is output here if the flag was correct or incorrect |

## Data Dictionaries

These are dictionaries to show all variables that will be used in my project

|  |  |  |
| --- | --- | --- |
| Data Item | Data Type | Description |
| **Server** |  |  |
| Command Log | List of strings | Log of all commands ran by the user |
| System Log | List of strings | Main Log of the network |
| Tick Timer | Timer | Timer when elapsed runs the Tick procedure |
| Ticking | Boolean | Bool that tells the timer whether the previous tick is still in progress |
| Tick | uint | How many ticks have elapsed overall |
| Filepath | String | Current opened File path of network |
| Network | Network | The current Network that is being simulated |
| NetworkLoaded | Boolean | Bool that tells the server whether or not a network is already loaded. |
| **Client** | | |
| Attempts | Int | Count of how many times a connection attempt has been made to the current IP |
| Buffer | Byte array | Receive buffer for the socket, when receiving data it will be written to this buffer before being copied elsewhere |
| Client Socket | Socket | Socket that is connected to the server |
| System Log | List of strings | List of responses from the system and commands that the user has used. |
| Log | Action(string) | Action used when logging a system action, so it is pushed to the log box and to the system log |
| Scenario Log | List of strings | List of messages from the scenario |
| Command Log | List of strings | List of commands the user has executed |
| PacketProtocol | PacketProtocol | Deals with parsing each packet seperatly |

## OOP Class Design

### Applications

#### Application

|  |  |  |
| --- | --- | --- |
| Virtual Application : Application | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Closed: bool | Public | Whether the application has been closed internally |
| Log: Action<string> | Public | Action used when something is added to the local application log |
| Log: List<string> | Public | List of the log of the application, can be used as a list of what has occurred and will be shown to the user |
| **Constructors** | | |
| Application : Action<string> |  | Base constructor, initialises the list of logs and the Log action to add to that list |
| **Methods** | | |
| Command: void (string) | Virtual | Used to pass a command to the application to tell it what to do, e.g start |
| OnTick: void (uint) | Virtual | All applications must be able to be ticked as part of the AI |

#### DHCPServer

|  |  |  |
| --- | --- | --- |
| DHCP Server : Application | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Leases: SortedList<IP, Lease> | Public | A list of all the leased IPAddresses |
| NetInterface: NetworkInterface | Public | The network interface this server is running on, will assign IP Addresses to anything on the other side of the interface |
| Reserved: SortedList<IP,Lease> | Public | A list of all the reserved IPAddresses |
| Gateway: IP | Private | The IPaddress of the network gateway |
| IP: IP | Private | Ip of the DHCPServer, cannot be leased |
| Subnetmask: IP | Private | The subnet mask of the current subnet |
| DNS: IP | Private | The IP of the DNS server, will be defaulted to a specific value |
| **Constructors** | | |
| DHCPServer: (System, NetworkInterface, IP, Action <string>) | Public | Will call the base constructor |
| **Methods** | | |
| Command: void (string) | Override | The command interface for the DHCP, to set up static leases, will not be implemented for my project |
| isAvailable: isAvailable(IP,MAC) | Public | Checks if an Ipaddress is available for a specific mac address, e.g. if the IP is reserved for that mac address, or if the IP is free |
| NewAddress: IP (MAC) | Public | Generates a new IP address to be offered to a specific mac address, should also reserve the Address |
| OnTick: void (uint) | Override | The Tick logic, checks to see if DHCP Leases are up |
| Packet void (IPv4Header, UDPHeader, Uint16,NetworkInterface) | Public | Called when a UDP Packet is received, subscribed to OnUDPPacket. |
| **DHCPServer.Lease** | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| MACaddress: MAC | Public | Mac address leased to |
| IP: IP | Public | IP Leased |
| Length : uint | Public | Length in ticks of lease |
| StartTick: uint | Public | tick that lease started |
| EndTIck: uint | Public | Tick that lease ends |
| **Constructors** | | |
| Lease (ip,mac,uint,uint) | Public | Creates a new lease of the IP to MAC starting at uint1 lasting for uint2 ticks |
| **Methods** | | |

#### DHCPClient

|  |  |  |
| --- | --- | --- |
| DHCP Client : Application | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Sessions: List<session> | Public | List of sessions running, one for each adapter |
| **Constructors** | | |
| DHCPClient: (System, Action <string>) | Public | Will call the base constructor |
| **Methods** | | |
| AddAdapter: void (NetworkInterface) | Public | Adds a new adapter for the client to start a new session on |
| Command: void (string) | Override | Commands to see current info |
| OnTick: void (uint) | Override | Ticks the AI, ticks each session |
| Packet: void(IPv4Header, UDPHeader, uint16, NetworkInterface) | Public | Subscribed to OnUDPPacket |
| **DHCPClient.Session** | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| xID | Public | The xID that this instance of communication will be using |
| State |  | The current state of the system, taken from the DHCP Finite state machine seen in research. |
| **Constructors** | | |
| Session: (Uint32) | Public | Initalises a new session with specified xID |
| **Methods** | | |
| OnTick: void (uint) | Public | Checks if timers have ticked down & if new attempts/renews need to be made |
| Packet: void (DHCPDatagram) | Public | Called if a DHCP Datagram is received addressed to the sessions xID |
| **DHCPClient.State : Enum** |
| **Item** |
| **Bound** |
| **Init** |
| **InitReboot** |
| **ReBinding** |
| **ReBooting** |
| **Renewing** |
| **Requesting** |
| **Selecting** |

#### PacketSniffer

|  |  |  |
| --- | --- | --- |
| Packet Sniffer | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| on: bool | Private | Whether or not to echo packet data |
| ip: bool | Private | Whether or not to echo IP data |
| udp: bool | Private | Whether or not to echo UDP data |
| dhcp: bool | Private | Whether or not to echo DHCP data |
| **Constructors** | | |
| PacketSniffer (Action<string>) | Public | Calls Base constructor |
| **Methods** | | |
| Command: void (string) | Override | Command interface to change switches |
| OnTick: void (uint) | Override | Tick, no AI as this is event based, however still needs to be implemented |
| Packet: void (byte[]) | Public | Subscribed to onPacket |

#### RoutingTable

|  |  |  |
| --- | --- | --- |
| Routing Table | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Entries: SortedList<uint,Entry> | public | List of routing entries sorted on their metric |
| StaticEntries: SortedList<uint,Entry> | Public | List of static, user created entries sorted on their metric |
| System: System | public | The Parent system running on |
| **Constructors** | | |
| RoutingTable (System, Action<string>) | Public | Calls base constructor |
| **Methods** | | |
| Command: void (string) | Override | Command interface to add a static route |
| OnTick: void (uint) | Override | Tick to update generated entry list |
| Route: void (IPv4header) | Public | Routs a packet down the best, appropriate entry with the highest metric |
| AddStaticEntry: void (entry) | Public | Adds a static entry to the list of static entries |
| **RoutingTable:Entry** | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Destination: IP | public |  |
| Gateway: IP | Public | IP address of the gateway the packet might have to be sent via, may be null |
| Interface: MAC | public |  |
| Metric: uint | Public |  |
| SubnetMask: IP | Public |  |
| **Constructors** | | |
| Entry (IP,IP,MAC,Uint,IP) | Public |  |

### Protocols

#### Protocol

#### DHCPDatagram

|  |  |  |
| --- | --- | --- |
| DHCPDatagram: Protocol | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Op: byte | Public | Fields in a DHCP Header, view DHCP Research into header for explanation of each attribute |
| Htype: byte | Public |
| Hlen: byte | Public |
| Hops: byte | Public |
| Xid: uint32 | Public |
| Secs: uint16 | Public |
| Flags: uint16 | Private |
| Broadcast: bool | Public |
| Ciaddr: IP | Public |
| Yiaddr: IP | Public |
| Siaddr: IP | Public |
| Giaddr: IP | Public |
| Chaddr: MAC | Public |
| Sname byte[] | Public |
| File byte[] | Public |
| Options List<option> | Public |
| **Constructors** | | |
| DHCPDatagram(byte, uint32, byte, byte, byte, uint16, bool, ip, ip, ip, ip, MAC, List<Option>) | Public | Generates a DHCPDatagram from the fields |
| DHCPDatagram(byte[]) | Public | Generates a datagram from a byte array |
| **Methods** | | |
| ToBytes: Byte[] () | Public | Makes the datagram a byte array. Formatted like a standard DHCP Datagram |

#### ICMP

|  |  |  |
| --- | --- | --- |
| ICMPHeader: Protocol | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Checksum: uint16 | Public | Uses the inherited method from Protocol to calculate the checksum of the packet |
| Code: byte | Public |  |
| ICMPType: byte | Public |  |
| **Constructors** | | |
| **Methods** | | |
| ToBytes: Byte[] () | Virtual | Makes the datagram a byte array. Formatted like a standard ICMP Datagram |

#### ICMPEchoRequestReply

|  |  |  |
| --- | --- | --- |
| ICMPEchoRequestReply: Protocol | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Identifier: uint16 | Public |  |
| Sequence Number: Uint16 | Public |  |
| **Constructors** | | |
| ICMPEchoRequestReply (byte, uint16, uint16) | Public | Takes in the type, identifier and sequence number |
| ICMPEchoRequestReply (byte[]) | Public | From bytes |
| **Methods** | | |
| ToBytes: Byte[] () | Override | Makes the datagram a byte array. Formatted like an ICMPEchoRequestReply Datagram |

#### IPV4Header

|  |  |  |
| --- | --- | --- |
| IPv4Header: Protocol | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Version: byte |  | Always 4 |
| IHL: byte |  | Always 5 unless options are included |
| TOS: byte |  |  |
| Length: uint16 |  | Calculated automatically |
| Identification: uint16 |  | Max value of 8191 |
| RES: bool |  | Will be combined with fragment offset, will be a property |
| DF: bool |  | ---||--- |
| MF: bool |  | ---||--- |
| Fragmentoffset: Uint16 |  | Default 0 only changed when fragmenting a packet |
| TTL: byte |  | Max 225, default 30 |
| Protocol: byte |  |  |
| Checksum Uint16 |  | Calculated automatically using protocol.calculatechecksum |
| SourceAdddress: IP |  |  |
| DestinationAddress: IP |  |  |
| Options: byte[] |  | Will always be size zero, options depreciated |
| Datagram: byte[] |  |  |
| **Constructors** | | |
| IPv4Header: (byte, uint16, bool, bool, bool, uint16, byte, ProtocolType, IP, IP, Byte[], byte[]) |  | Takes in all parameters, even those that would otherwise have a default value |
| IPv4Header: (Uint16, bool, bool, byte, ProtocolType, IP, IP, byte[]) |  | Takes in parameters that don’t have a default value |
| IPv4Header: (byte[]) |  | From bytes |
| **Methods** | | |
| ToBytes: byte[] () |  |  |
| **IPv4Header.ProtocolType : Enum** | |
| **Value** | **byte** |
| ICMP | 1 |
| TCP | 6 |
| UCP | 17 |

#### UDPHeader

|  |  |  |
| --- | --- | --- |
| UDPHeader: Protocol | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| SourcePort: uint16 | Public | Source port of the data in the datagram being sent |
| DestinationPort: uint16 | Public | Destination port the packet will be forwarded to |
| Length: uint16 | Public | Automatically generated |
| Checksum: uint16 | Public | Automatically generated |
| Datagram: byte[] | Public | Internal datagram. |
| **Constructors** | | |
| UDPHeader: (uint16, uint16, byte[]) | Public | Source, destination, datagram |
| UDPHeader: (byte[]) | Public | From bytes |
| **Methods** | | |
| ToBytes: Byte[] () | Override | Sends to bytes in the same format as a real UDP Header |

### Systems

#### System

|  |  |  |
| --- | --- | --- |
| System | | |
| Item | **Accessibility** | **Description** |
| **Attributes** | | |
| Apps: Application[] | public | List of applications, fixed length defined on instantiation |
| ID: Uint | public | The unique identifier for the system |
| LocalLog: Action<string> | public | Action to add to the local log, will be applications etc |
| Locallog: List<string> | public | The list of things that have been locally logged |
| Log: Action<string> | public | Will log in a broader sense externally, passed in on instantiation |
| NetworkInterfaces: Dictionary<Mac, NetworkInterface> | public | A dictionary of all network interfaces sorted on their unique mac address |
| RespondToEcho: Boolean | public | Whether or not the ICMP should |
| RoutingTable: Application | public | A pointer to the routing table in the list of applications |
| **Events** | | |
| OnCouruptPacket: Byte[], NetworkInterface | public | Called when a processed packet’s checksum is wrong |
| OnICMPPacket: IPv4Header, ICMPHeader, byte, NetworkInterface | public | Called when an ICMP Packet is received |
| OnNotForMe: IPv4Header, NetworkInterface | public | Called when a packet is received that is not addressed to this host |
| OnRecievePacket: Byte[], NetworkInterface | public | Called when a packet is received |
| OnTCPPacket: IPv4Header, TCPHeader, UInt16, NetworkInterface | public | Called when a TCP Packet is received |
| OnTick: uint |  | Called when the application is ticked |
| OnUDPPacket: IPv4Header, UDPHeader, UInt16, NetworkInterface | public | Called when a udp packet is received |
| **Constructors** | | |
| System: uint, Dictionary<MAC, NetworkInterface>, bool, int, bool, Action<string> | Public |  |
| FromXML: XMLNode, Action<string> | Public |  |
| **Methods** | | |
| AdapterTick: void(uint) | Public | Ticks each of the network adapters |
| AddApp: bool (application, int) | Public | Adds an app in a safe way, no exceptions |
| ApplicationTick: void (uint) | Public | Ticks all apps |
| AppsInit: void () | Public | Initialises the standard array of applications |
| Command: void (string) | Public | Runs a command on the system, used to start apps and interact with them, as well as to look at information about the system |
| Equals: bool (object) | Public | Finds if this system is equal to another |
| GetConectableAdapter bool ,networkinterface () | Public | Gets an adapter that is available to be connecte to |
| GetconnectedUnassociatedAdapter: bool, networkinterfaces (uint) | Public | Gets an adapter that is connected but not associated to a specific system |
| HandleICMPPacket: void (ipv4Header, NetworkInterface) | Public | Processes a received ICMP Packet |
| Packet: void (byte[], NetworkInterface) | Public | Handles a received packet and get the ipv4 data |
| SendPacket: bool (MAC, IPv4Header) | Public | Sends a packet via the routing table |
| Tick: void (int) | Public | Ticks the system, in turn calling adapterstick and apptick |
| ToXML: XMLNode () | Public | Turns the system to an xml node |

#### Router

|  |  |  |
| --- | --- | --- |
| Router: System | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| **Constructors** | | |
| Router(uint, Dictonary<MAC,NetworkInterface>, int, bool, bool, Action<string>) |  | Calls base constructor |
| FromXML: Router (XMLNode) |  | Initialises a new router from xml |
| **Methods** | | |
| AppsInit | Override | Starts a different array of apps |

### IP

|  |  |  |
| --- | --- | --- |
| IP : IClonable, IComparable | | |
| **Item** | **Accessibility** | **Description** |
| **Static Attributes** | | |
| Broadcast: IP | Public | Static ip for broadcast |
| Loopback: IP | Public | Static ip for looping back to the same system |
| Zero: IP | Public | Static ip for an all zero address |
| **Attributes** | | |
| ip: Byte[] | Private | The bytes of the ip |
| **Constructors** | | |
| IP (byte[]) |  | Construcs from a byte array |
| IP (byte[], int) |  | Int – start index in array |
| **Static Methods** | | |
| != : bool(IP,IP) |  | Tests if an IP is not equal to another |
| & : IP(IP,IP) |  | Performs a bitwise and between two IP’s |
| | : IP(IP,IP) |  | Peforms and exclusive or with each of the bits |
| ~ : IP(IP) |  | Gets the ones compliment of each byte |
| ++ : IP(IP) |  | Increments the Ip by 1. E.g. 192.168.1.1 would go to 192.168.1.2 and 192.168.1.255 to 192.168.2.0 |
| == : bool(IP,IP) |  | Tests if two ip’s are equal |
| isNull: bool(IP) |  | Static method tests if an ip is null |
| Parse: IP (string) |  | Parses and ip from a string, will throw an exeption |
| tryParse: bool,IP (string) |  | Parses an Ip from a string will not throw an exeption, bool returned to show success.failure |
| **Methods** | | |
| Clone: IP(IP) |  | Clones an IP address |
| Compare int(object) |  | Compares an Ip, shows if larger or smaller |
| Equals: bool(object) | Override | Tests if an ip is equal to another object |
| ToString: string() | Override | Turns the Ip to a string |
| ToBytes: byte[]() |  | Turns the ip into a byte array |

### MAC

|  |  |  |
| --- | --- | --- |
| MAC | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Mac: Byte[] | Private | The bytes of the mac |
| **Constructors** | | |
| MAC (byte[]) | Public | creates a mac address object from a byte array |
| MAC (byte[], int) | Public | Int – start index in array |
| **Static Methods** | | |
| Parse: MAC (string) | Public | Parses a mac address from a string, will throw errors |
| tryParse: bool,MAC (string) | Public | Parses a mac address from a string, won’t throw errors, returns a bool to show success |
| Random: MAC () | Public | Generates a random mac address |
| **Methods** | | |
| Equals: bool(object) | Override | Tests I the mac address is equal to another object |
| ToString: string() | Override | Turns the mac address to a string |
| ToBytes: byte[]() | Public | Returns the mac address byte array |

### Network

|  |  |  |
| --- | --- | --- |
| Network | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Systems: Dictonary<uint,System> | Public | Dictionary of all system in the network by their uint ID’s |
| Connections: List<uint, uint> | Public | List of all connections between the systems |
| Flags; List<string ,string> | Public | List of all the flags in the system |
| Starttime: Time | Public | The time that the system was loaded |
| UnallocatedPlayers: List<uint> | Public | List of all id’s of unallocated player systems |
| TakenMacAddresses: List<MAC> | Public | List of all mac addresses that are used in the system, to avoid duplicates |
| ServerSocket: Socket | Public | The server socket where connections will be accepted |
| Buffer: byte[] | Public | The buffer for the server socket |
| Users: List<user> | Public | List of all user objects that have been instantiated |
| Scenario: sortedList<uint,string> | Public | List of all messages that need to be sent and the time to send them |
| Log: Action<string> | Public | Action to log to the main log |
| **Constructors** | | |
| Network(Action<string>) | Public | Instantiates an empty network that will log to given action |
| LoadNetwork: Network (string, Action<string>) | Public | Instantiates a new network from the xml file path given |
| **Methods** | | |
| Setupserver: void () | Public | Sets up the server and starts receiving connections |
| AcceptConnectionCallBack: void () | Public | Called when a connection is accepted |
| PlayerLoginCallBack: void () | Public | Called when a player sends bytes to the server after connecting, will process it as a login request |
| ProcessPacket(byte[], User) | Public | Processes a packet once a client has logged in |
| SendCallback: void () | Public | Call back for sending data |
| asSystem: void (string) | Public | Runs a command as a given system in the network |
| Message: void (string) | Public | Sends a message as a scenario to all connected clients |
| Report: void (string) | Public | Generates a report at the given file path |
| Tick: void (uint) | Public | Ticks all systems in the network |
| Connect: void (uint,uint) | Public | Connecs two systems by their id’s |
| Connect: void (XMLNode) | Public | Connects two systems from a connection xml node |
| SaveNetwork: bool (string) | Public | Saves a network at given filepath, only used for testing purposes |
| Exit: void () | Public | Exits the client sockets on the users nicely |
| **Network.User** | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Username: string | Public Readonly | Username of a user, cannot be changed once instanciated |
| Password: uint | Public Readonly | Password of a user, generated for them |
| sysID: uint | Public Readonly | The sys id that the user controls |
| Socket: Socket | Public | The socket that the client communicates through |
| RecieveBuffer: byte[] | Public | The buffer for receiving bytes from the client |
| PacketsNotSent: List<byte[]> | Public | List of all packets that failed to send |
| Flags: List<string,Time> | Public | List of all flags the user has collected |
| Log: Action<string> | Public | The action to log to the network on a packet received ect. |
| Packetprotocol: PacketProtocol | Public | An instance of the packet protocol to wrap packets |
| Parent: Network | Public | The network this user is a part of |
| **Constructors** | | |
| User(string, uint, uint, Network, Action<string>) | Public | Instanciates a user with the given username, passoword, sys id, parent network and log function |
| **Methods** | | |
| PlayerRecieveCallback: void () | Public | Called when the client sends data to the server |
| SendCallback: void () | Public | Called when the server finishes sending data to the client |
| Send: void (string) | Public | Called to send a packet to the client. Given to the system local log |
| Connect: bool (socket) | Public | Called to connect a client socket to the user |

### 

### NetworkInterfaces

#### NetworkInterface

|  |  |  |
| --- | --- | --- |
| NetworkInterface -> Abstract | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| Name: string | Public | The name of the adapter |
| SysID: uint | Public | The parent system’s id |
| MyMacAddress: MAC | Public | The unique mac address of the interface |
| LocalIP: IP | Public |  |
| SubnetMask: IP | Public |  |
| Connected: Bool | Public | If the interface is connected |
| Associated: Bool | Public | If the interface holds a reference to the other interface it is connected to |
| SendQueue: Queue<byte[]> | Public | A queue of packets for the interface to send, will send one per tick |
| RecieveQueue: Queue<byte[]> | Public | A queue of packets for the interface to receive, will receive one per tick |
| Log: Action<string> | Public | Action to log to |
| **Events** | | |
| OnRecievePacket: (byte[], NetworkInterface) | Public | Called when a packet is recieved |
| **Constructors** | | |
| NetworkInterface (MAC, uint, Action<string>) | Public | Instantiates a new interface given a Mac address, parent system id an log action |
| **Methods** | | |
| isConnectedTo(uint); | Public | If the interface is connected to a specific system |
| CallOnRecievePacket(byte[], NetworkInterface) | Public |  |
| Reset: void () | Public | Resets the adapter |
| sendPacket: void (byte[]) | Public | Sends a packet to the other adapter only adds to queue |
| RecievePacket: void (byte[]) | Public | Receives a packet from another router. Only adds to queue |
| Tick: void (uint) | Public | Takes one packet of the send and receive queue and dose just that with each of them |
| ToXML: XMLNode () | Public | Turns the adapter to xml, testing purposes only |
| Connect: bool(NetworkInterface) | Virtual | Connects the adapter to another interface |

#### Adapter

|  |  |  |
| --- | --- | --- |
| Adapter: NetworkInterface | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| DefaultGateway: IP | Public |  |
| DNS: IP | Public |  |
| OtherEnd: NetworkInterface | Public | The other interface the adapter is connected to |
| OtherEndID: uint | Public | The id of the other system |
| **Constructors** | | |
| Adapter(MAC, uint, string, IP, IP, IP, IP, uint, bool, Action<string>) | Public | Calls base constructor |
| FromXML: void (XMLNode) | Public | Generates an adapter from an xml node |
| **Methods** | | |
| isConnectedTo(uint); | Override | See overridden method description |
| Reset: void () | Override | See overridden method description |
| Tick: void (uint) | Override | See overridden method description |
| ToXML: XMLNode () | Override | See overridden method description |
| Connect: bool (NetworkInterface) | Override | See overridden method description |
| Equals: bool (object) | Override | Tests to see if the adapter is equal to another object |

#### GroupAdapter

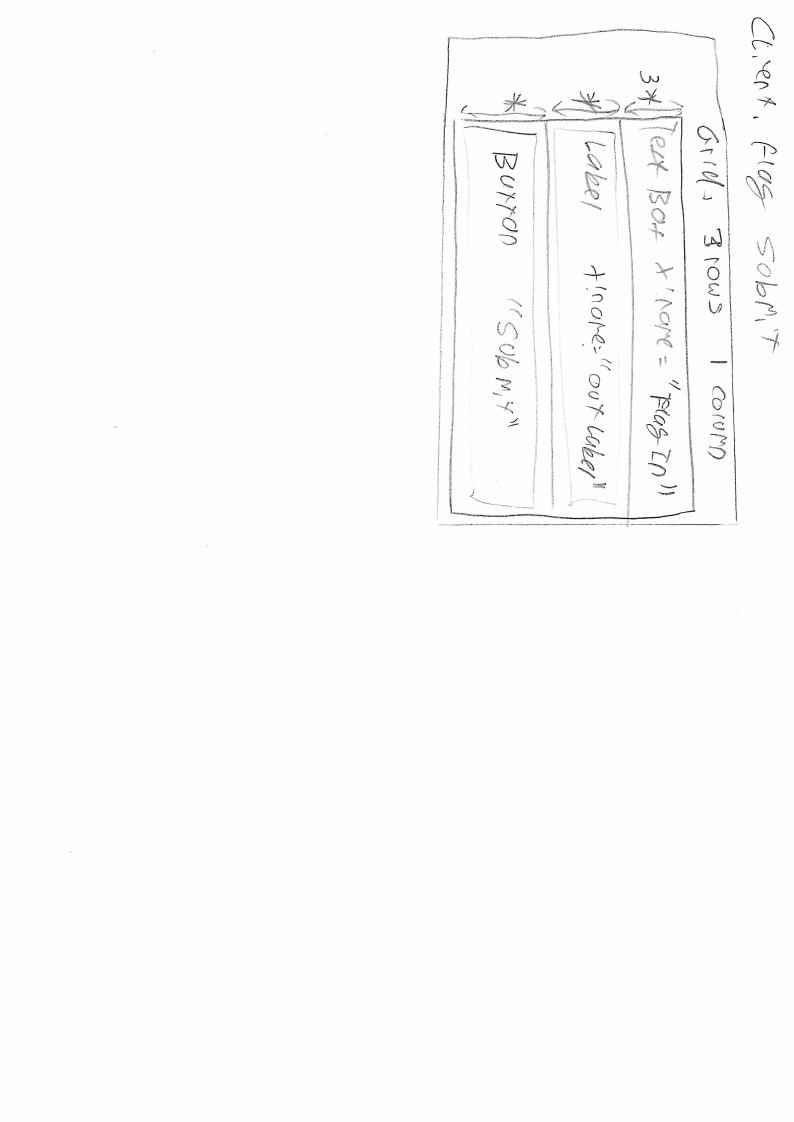
|  |  |  |
| --- | --- | --- |
| GroupAdapter: NetworkInterface | | |
| **Item** | **Accessibility** | **Description** |
| **Attributes** | | |
| OtherEnds: List<uint, NetworkInterface> | Public | List of connected adapters |
| OtherEndIDs: List<uint> | Public | Lis of the id of the connected systems |
| **Constructors** | | |
| GroupAdapter(MAC, uint, string, IP, IP, List<uint>, bool, Action<string>) | Public |  |
| FromXML: void (XMLNode) | Public | Generates a group adapter from an xml node |
| **Methods** | | |
| isConnectedTo(uint); | Override | See overridden method description |
| Reset: void () | Override | See overridden method description |
| Tick: void (uint) | Override | See overridden method description |
| ToXML: XMLNode () | Override | See overridden method description |
| Connect: bool (NetworkInterface) | Override | See overridden method description |

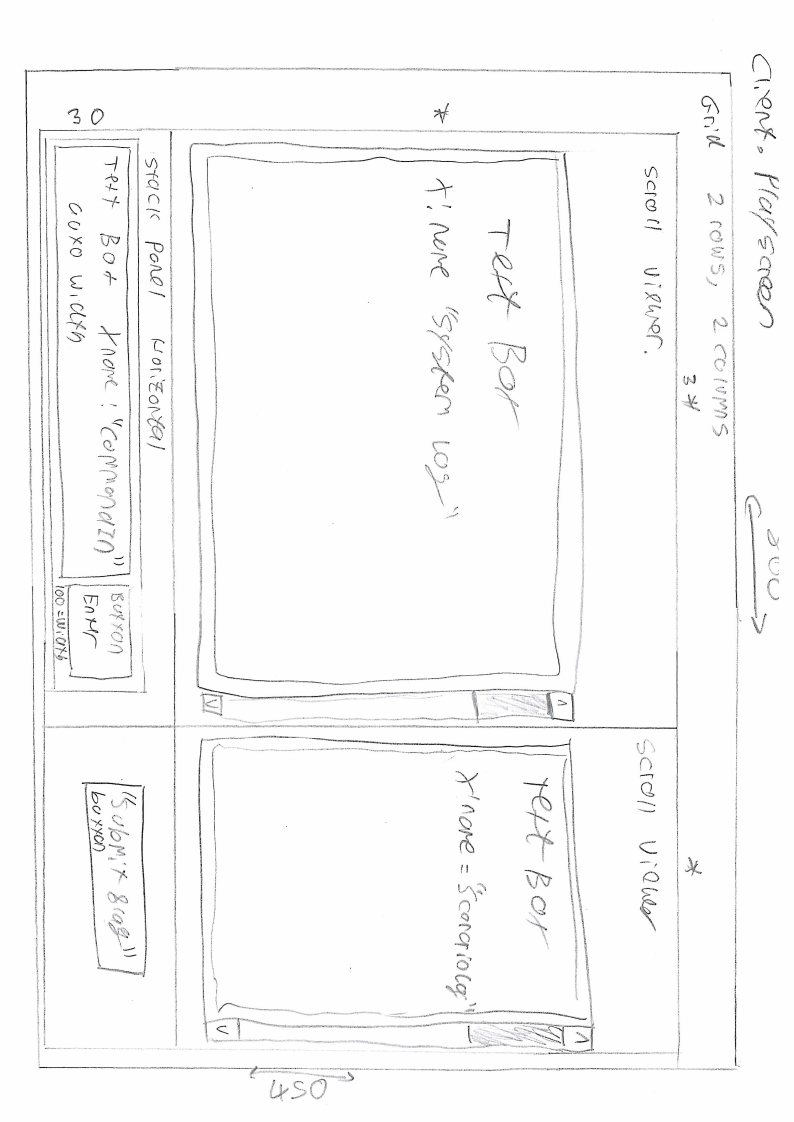
### PacketProtocol

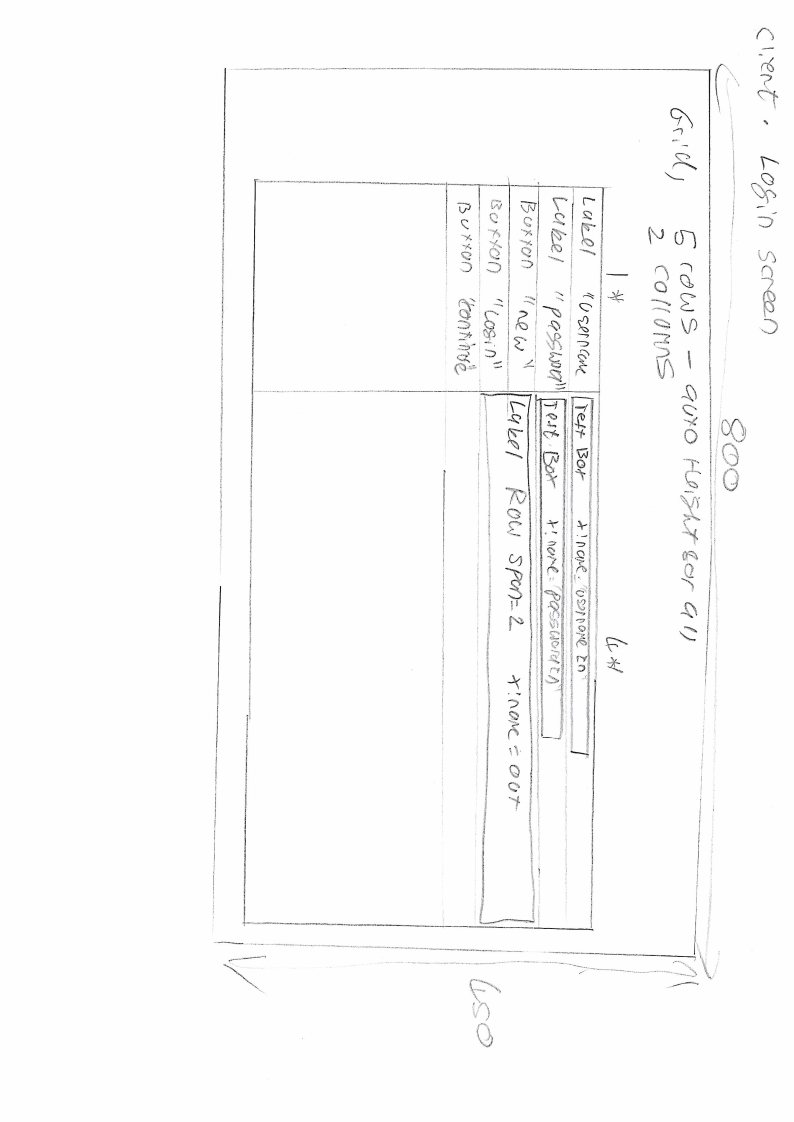
The packet Protocol class will not be of my design as it is someone else’s code as mentioned in the design overview, hence no design for it.

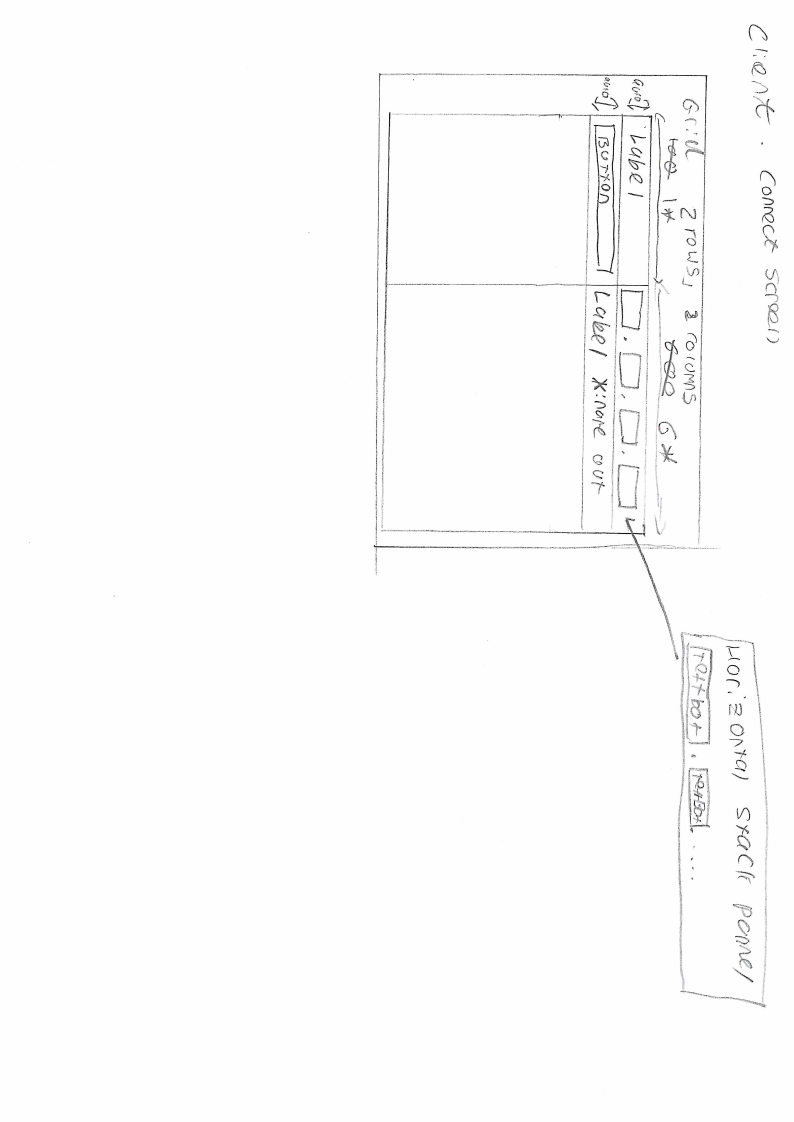
## Interface Design

### Client

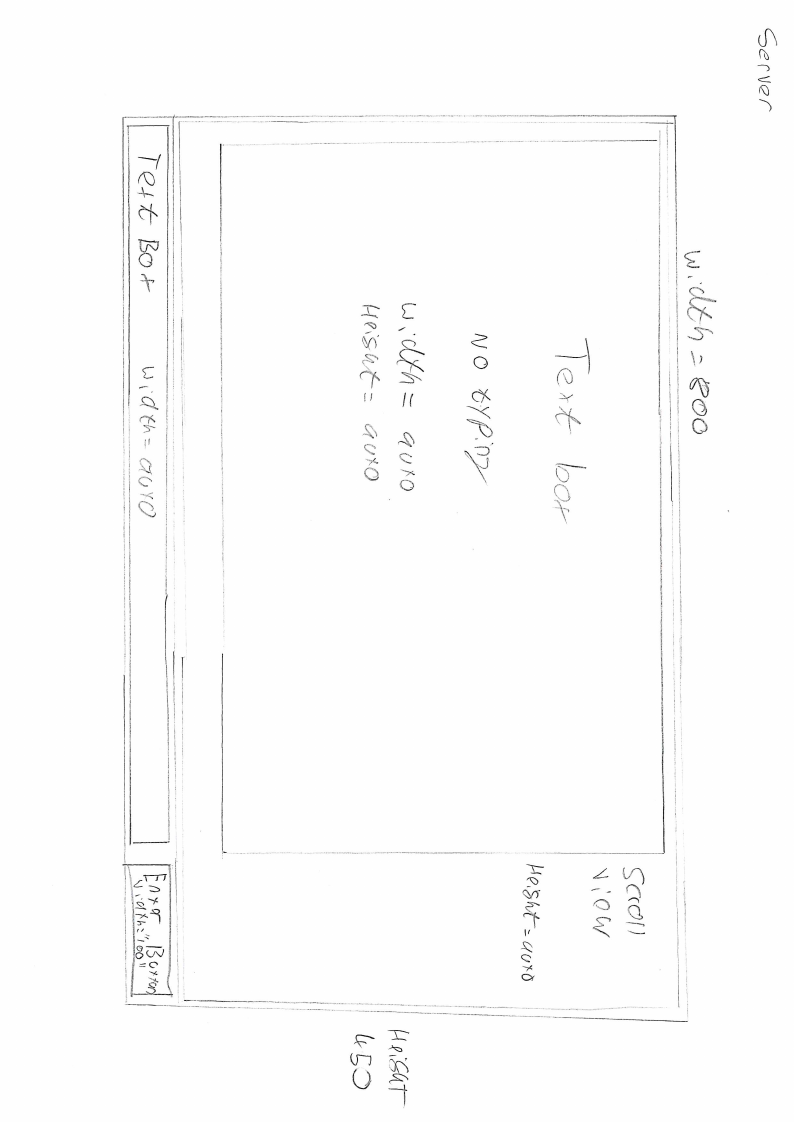








### Server



## Algorithms

This will be the pseudo code for my main algorithms that will be complex, I am going to omit things such as to bytes methods and to and from XML as these methods are very self-explanatory and therefor pointless

### Network

Sub ProcessPacket (data, user)

Text <- asciiEncoding(data)

Call Log(“Processing Packet”)

Command <- text.split(“ ”)

Switch command(0)

Case “command”

text.remove(7)

asSystem(user.sysid + text)

Endcase

Case “flag”

Text.remove(5)

If flags.contains(text) and not(user.flags.contains(text))

User.flags.add(text,TimeNow-DateTime)

endif

Endcase

Case “logout”

User.socket.close()

Endcase

EndSwitch

EndSub

SUB Report(filepath)

Initialise report to new 2d array

IF not(filepath.endswith(“.csv”))

Filepath.append(“.csv”)

ENDIF

I <- 0

WHILE I < users.length

Report(0,i+1) <- users(i).username

I = I+1

ENDWHILE

I <- 0

WHILE I < flags.length

Report(I+1,0) <- flags(i).mark

J <- 0

WHILE J < users.length

Foreach flag in users(j).flags

If flag.mark == flags(i).flag

Report(1+i,1+j) = flag.time

ENDIF

J = J+1

ENDWHILE

I = I+1

ENDWHILE

OPEN FILE filepath

Filepath.write report

EndSub

Sub LoginRecieveCallBack()

Current = socket

Received = current.endrecieve()

Data = buffer.copy(0,received)

Text = asci.decode(data)

Command = text.split(“ “)

Switch Command(0)

Case “new”

Username = command[1]

If not(users.contains(username))

Password = random.random()

Senddata = asci.encode(“success, password is ” + password)

Current.send(senddata)

User <- new

user(username,password,unallocatedplayers[0])

User.connect(current)

Current.beginrecieve(users.recievecallback)

ELSE

Senddata = asci.encode(“error username already exists”)

Current.send(senddata)

Current.recievedata(LoginRecievecallback)

ENDIF

EndCase

Case “login”

Username = command[1]

If users.contains(username)

TRY

Password = Int.parse(command[2])

Currentuser = user.select(username)

If Currentuser.password == Password

Senddata = asci.encode(“success”)

Current.send(senddata)

Current.recievedata( currentuser.recievecallback)

currentuser.connect(current)

Else

Senddata = asci.encode(“error incorrect password”)

Current.send(senddata)

Current.recievedata(LoginRecievecallback)

EndIF

ENDTRY

ELSE

Senddata = asci.encode(“error username doesn’t exist”)

Current.send(senddata)

Current.recievedata(LoginRecievecallback)

ENDIF

EndCase

EndSwitch

endsub

# Testing

As a reminder here is a copy of my success criteria

Know the main components of a packet

Describe the role of the four layers of the TCP/IP stack (application, transport, network and link)

Be familiar with the role of MAC addresses

Explain what well known ports and client ports are used for and the differences between them

Know that an IP address is split into a network identifier part and a host identifier part

|  |  |  |
| --- | --- | --- |
| No | Success Criteria | Operationalised |
| A | The solution must implement and test the specification points shown above | Have a check list of all teaching/information points that need to be taught & check they are all covered. Further check that they someone who hasn’t completed the specification before can use the program, with the test in front of them, and answer all of the questions fully. Given time |
| B | The solution must be engaging to students to make them want to learn about networking. | Test with the students to make sure that they enjoy the solution and engage with it. Found through observation and a self-report after they finish using the solution |
| C | The solution must have an “End of unit test” in order for the teacher to be sure all the students fully understand the topic | Have the end of unit test implemented as well as the one in the system to compare results to make sure the system is working, this ensures reliability of the new test. |
| D | The end of unit test would be self-marked | Have a teacher and the system mark the test to make sure it is marked correctly (interrater reliability) |
| E | Results should be output as a csv file making it easy to analyse | Have the results of each of the students be output as a CSV value (or another file type that can be imported into google sheets/excel) |
| F | The solution should be written asynchronously so the code doesn’t block the UI | Uses begin and end call-backs in code, can be observed by UI not stopping responding when input is provided |
| G | A student should be able to work easily with less, but not no, input from the teacher | When testing with students the students shouldn’t need help on average any more then in a normal lesson, this will be judged by me and the teacher who are both familiar with the amount of questions usually asked |
| H | Should allow for automatic collection of results from clients meaning client server code must be implemented using Sockets and my own protocol | Uses socket based programming |

## Testing Strategy

The testing strategy I will be using in this project is black box testing meaning I will be testing interaction between the user and the code rather than specifically testing each method/procedure etc.

## Testing Plan

|  |  |
| --- | --- |
| Test No | 1 |
| Testing Objective | A |
| Purpose of test | To test if we test for knowledge of each specification point stated in success criteria |
| Description of Test | By going through the specification and showing proof of scenario’s and flag’s for each spec point to be shown. |
| Test Data | - Know the main components of a packet  - Describe the role of the four layers of the TCP/IP stack (application, transport, network and link)  - Be familiar with the role of MAC addresses  - Explain what well known ports and client ports are used for and the differences between them  - Know that an IP address is split into a network identifier part and a host identifier part  - Know that networks can be divided into subnets and know how a subnet mask is used to identify the network identifier part of the IP address |
| Expected Result | Each specification point would be taught and tested in a lesson |
| Actual Result | Each point Is taught and tested in LessonAndTest1.xml evidence can be seen in the testing video. Has been confirmed with teacher that |

|  |  |
| --- | --- |
| Test No | 2 |
| Testing Objective | B |
| Purpose of test | To see if students enjoy the system |
| Description of Test | Will have students run through the tutorial and Lesson 1 and take a rating of their engagement from their teacher (my end user) compared to a normal lesson. The data will be collected win a qualitative manner. |
| Test Data | Tutorial.xml  LessonAndTest1.xml |
| Expected Result | The engagement in the lesson should be greater than or equal to the normal engagement in a networking lesson |
| Actual Result |  |

|  |  |
| --- | --- |
| Test No | 3 |
| Testing Objective | D,E,H |
| Purpose of test | To show the report function generating a report |
| Description of Test | Will run 3 clients simultaneously getting some marks and all marks on one to show the server generating a report  To show objective H is true server will be hosted on a different machine, my laptop. |
| Test Data | One client will get every other flag another will get every third flag and the last will get every flag, Will then generate a report  Will generate the xml file Test3.xml for this test |
| Expected Result | The data stated above will be represented in the report.csv file   |  |  |  |  | | --- | --- | --- | --- | |  | User1 | User2 | User3 | | Flag1 | Time | Time | Time | | Flag2 | Time |  |  | | Flag3 | Time | Time |  | | Flag4 | Time |  | Time | | Flag5 | Time | Time |  | | Flag6 | Time |  |  | |
| Actual Result |  |

|  |  |
| --- | --- |
| Test No | 4 |
| Testing Objective | E |
| Purpose of test | Show UI isn’t blocked due to async code |
| Description of Test | Will show that on the connect button press the UI stays responsive |
| Test Data | Connect to a wrong IP Address |
| Expected Result | When connecting the UI should stay responsive, should be able to move the window around, press buttons, and type in text boxes. |
| Actual Result | Expected result was seen as shown in video evidence |

|  |  |
| --- | --- |
| Test No | 5 |
| Testing Objective | G |
| Purpose of test | To make sure the system doesn’t crash when being used |
| Description of Test | Will run 2 clients at once and run through the tutorial to show my system working and not crashing |
| Test Data | Will connect to a not connected IP address and a connected IP address  Will login to an account that doesn’t exist, try making a username that already exists and finally make a new user  Will run each command with no, wrong and correct parameters |
| Expected Result | The program shouldn’t crash and if an error would occur it would be dealt with gracefully. |
| Actual Result |  |

## Evidence

Video & stuff

# Evaluation

## Performance against the objectives

## Independent feedback

## Improvements

Evaluation points

* Persistence on logging in and out
* input sanitisation
* web interface

1. <https://tools.ietf.org/html/rfc791> [↑](#footnote-ref-1)
2. <https://tools.ietf.org/html/rfc791> [↑](#footnote-ref-2)
3. <https://tools.ietf.org/html/rfc2131> [↑](#footnote-ref-3)
4. <https://www.youtube.com/watch?v=S43CFcpOZSI> [↑](#footnote-ref-4)
5. <https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol#Operation> [↑](#footnote-ref-5)
6. <https://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xhtml> [↑](#footnote-ref-6)
7. http://techgenix.com/making-sense-windows-routing-tables/ [↑](#footnote-ref-7)
8. http://library.mobrien.com/Manuals/MPRM\_Group/images/rightframe\_files/routing.htm [↑](#footnote-ref-8)
9. <http://blog.stephencleary.com/2009/04/sample-code-length-prefix-message.html> [↑](#footnote-ref-9)
10. <https://github.com/AbleOpus/NetworkingSamples> [↑](#footnote-ref-10)