

Nathan Morgan Write up

by Nathan Morgan

FILE	11994_NATHAN_MORGAN_NATHAN_MORGAN_WRITE_UP_349477_711805111.DOCX (52.57K)		
TIME SUBMITTED	12-MAY-2021 12:12AM (UTC+0100)	WORD COUNT	4222
SUBMISSION ID	152786174	CHARACTER COUNT	23004

Socket programming and concurrency

NATHAN MORGAN 1801338

Contents

1 Transmission Control Protocol	2
1.1 Three-way handshake	2
1.1.1 SYN	2
1.1.2 SYN-ACK	2
1.1.3 ACK	3
1.2 Error handling	3
2 Maximum transmission unit	4
3 Synchronicity and asynchronicity	6
3.1 The different types of dispatch models.	6
3.1.1 Blocking Input output model	6
3.1.2 Nonblocking input output model	6
3.1.3 Multiplexing input output model	6
3.1.4 Signal-driven input output model	6
3.1.5 Asynchronous input output model	7
4 Interoperability	8
5 Report on implementation	9
References	11

1 Transmission Control Protocol

TCP packets are encapsulated by IP packets. TCP can be broken down into 3 stages, establishing connection, data transfer and then release of resources [1]. The stages are managed using states [2]. Each packet contains a 1-bit state flag that will be used to identify the purpose of the packet before further interaction is committed [3].

1.1 Three-way handshake

Throughout communication via transmission control protocol a three-way handshake is performed this is a series of interactions that require confirmations of receipt to ensure accurate transmission.

The handshake can be compared with the two General's problem. In the two General's problem the issue stands that how to get two armies to attack at the same time, because if both armies do not work together they will surely fail.

The Generals must communicate over a distance with considerations to issues that could arise when using a messenger to travel from one camp to another through an unsecure path. The issue could be that the messenger gets captured, or his message tampered with leading to the Generals making a mistake due to the failure in communication. To ensure the message is received a confirmation is sent back.

The question is how does the General know that the confirmation message is received without another message being sent? This could quickly lead to an infinite number of messages being required. TCP aims to resolve this by adding more information to each message to track if a message were received with essentially tracking ID's known as sequence numbers. This does not guarantee success, but it accepts that there is a chance of failure and attempts to mitigate the loss. This thought experiment is widely used when discussing fault tolerance in communication.

1.1.1 SYN

When establishing a connection, the three-way handshake starts with the client sending a request to the server to open a connection, this uses the SYN flag to request synchronization and a sequence number to track the interaction [4].

The initial request can be compared to General Dave sending General John a message with the randomly generated message ID of 200 and a message saying, "Tonight I request you attack at 10pm!".

1.1.2 SYN-ACK

Then the server will respond to the SYN request with a SYN-ACK packet that contains the SYN flag, the received sequence number incremented by one as the ACK number with the ACK flag and the servers own sequence number as the packets sequence number. The SYN flag here asks the client to open a connection to the server and confirming the receipt of the original SYN request.

In this interaction General John receives General Dave's request and responds with an acknowledgement containing General Dave's message ID of 200 and returns 201 to show that it was received as expected and then General John adds his own message back to General Dave saying, "I Received your message, and I too request that you attack at 10PM" with General Johns randomly generated message ID attached of 400.

1.1.3 ACK

The SYN-ACK request is then acknowledged by the client by sending the server an ACK packet that has the SYN flag set to 0 and the servers own sequence number incremented by one as the acknowledgement number parameter and ACK flag and the clients sequence number incremented by one as the packets sequence number. This is then used to complete the handshake and completely establish the connection and set the states of both points to established.

Finally General Dave receives General Johns message request General Dave to also attack at 10PM and the acknowledgement is verified by checking the ID. General Dave Agrees to attack at 10PM so he signs the acknowledgement message by returning General Johns message ID of 400 and returns it as 401. This will be used to verify the acknowledgment by General John and complete the three-way handshake [5].

1.2 Error handling

Transmission control protocol uses a handshake system to receive a confirmation of delivery as a means of error-detection. This is managed by the recipient sending a packet back containing an acknowledgement flag, and upon receiving the acknowledgment flag it is verified by the expected ID. This ensures that the recipient has received the packet and if the packet fails to be received another can be sent to ensure delivery [6] .

Failure can be measured by using time gates to measure if a response should have been received within an allotted time frame and if not, the package will likely be counted as lost and then retransmission can commence.

With this system the packets (protocol data unit) can be analysed to ensure that all the packets are received [7]. Using flags that contain the sequence number and the number is used to order them in the correct sequence and ensure that all numbers are present before defragmenting the data stream [6].

2 Maximum transmission unit

Maximum transmission unit is the maximum size of a PDU (protocol data unit) that can be transmitted in a single interaction over the network [8]. An MTU is essentially specifying how large the transmitted data will be. MTU's can be problematic in the sense of when communicating between two interfaces, the size of the MTU can be different on each interface leading to issues such as an MTU mismatch. An MTU mismatch will typically occur when attempting to send data that is larger than the allocated MTU on the receiving end, this issue can be countered by adhering to the MTU of the recipient, but to do this the MTU must be known. This will typically be transmitted on the initial handshake.

When transmitting data larger than the recipients MTU the data can be sent over several packets to distribute the load and then be reassembled by the recipient once all the packets are received, this is referred to as IP fragmentation [9].

IP fragmentation is the solution to working with different size MTU's as different end points will have different allocated MTU's. By reducing the packets into smaller packets will ensure they fit through the MTU. IP fragmentation can be beneficial for the security of routers to prevent fragmentation attacks, as with larger MTU sizes allowing for more to be sent at once oversaturating a socket or being refused as the packet does not conform to the MTU. When working with IP fragmentation MTU limitations but ultimately the larger the number of fragments, the more opportunities for failure present themselves. A larger MTU can be used to improve throughput efficiency by reducing the amount of analysis the recipient will have to perform such as defragmenting the packets, and less data transmitted when the number of packets sent is reduced there will be less packet headers [10].

This poses the issue of when a large PDU is sent over the network if the transmission was to fail the entire packet would fail to be received when compared to a fragmented version of the same packet being fragmented and sent in for example, 100 smaller packets, if one was to fail 99% of the data will have been transmitted and only 1% would have to be sent again, providing a greater fault tolerance and a more resilient approach rather than an all or nothing approach.

A PDU is seen at each layer of the open systems interconnection model and it can be referred to by its utility in the given layer. At each stage of the OSI model the data is encapsulated and de-capsulated depending on if it the data is being sent or received [11]. The process is very similar each way, except de-capsulation is essentially the reverse [12].

Layer 7, the application layer. This is where the data is transmitted by a software application.

Layer 6, the presentation layer. This is where the data is formatted for presentation.

Layer 5, the session layer. This is where the data is passed to the network connection.

Layer 4 transport layer PDU is the TCP segment or the UDP datagram, as TCP is the most common use, it is often referred to as a segment. At the fourth layer this is where the first level of encapsulation is performed where TCP/UDP headers are added [13].

Layer 3 network layer PDU is the packet or an IP packet. At the third layer the encapsulation is furthered by adding an IP header [14].

Layer 2 data link layer PDU is typically where the frame is built by encapsulating a frame header and a tail. The frame header contains data such as the destination and source mac address and the

maximum transmission unit of the data or the payload capability and further control information, the tail signifies the end of the frame and some error checking.

There are different types of frame such as ethernet frames, token ring frames, Wi-Fi frames to name a few. There is a wide variety of frames at this layer, even different implementations of the same type of frames such as gigabit ethernet frames are different to standard ethernet frames as they use Jumbo frames. For two devices to communicate the same frames must be transmitted.

A typical ethernet frame will be comprised of 7 parts, the preamble that contains information for the receiving network card to inform the recipient of that there is a frame coming and the starting sequence ID, recipient mac, sender mac, type of data that is being transmitted such as ipv4 or ipv6, data field that contains the payload containing the IP packet from the network layer, padding to ensure the frame meets the minimum size to beat collision filters, and the frame check sequence for error handling [15].

Layer 1 physical layer PDU is where the packet is broken down into a series of binary bits by the network card to be transmitted [16].

3 Synchronicity and asynchronicity

There are 5 types of Input output dispatch models under UNIX, they are blocking, nonblocking, multiplexing, signal driven and asynchronous. Regarding these dispatch models 4 out of 5 are synchronous as the input output operation blocks the process until complete, only the asynchronous model does not block the process [17].

3.1 The different types of dispatch models.

The 5 types of I/O dispatch models.

3.1.1 Blocking Input output model

Blocking Input output model is the most common blocking model, and it works based on the principle that no other task can be complete until the datagram has been received in its entirety.

The blocking input and output model is often seen in a client server application, when the client interacts with the server a connection is created, and the client and the server become linked over a socket. At each socket, a thread is launched for handling the connection, listening, and waiting for communication and that thread is blocked until a request is received or executed. Threading is utilized to deal with multiple clients.

This approach is adequate when working with a limited number of concurrent connections but as the active number of users increases the number of threads launched will increase exponentially and will require more memory and processing power [18].

3.1.2 Nonblocking input output model

Nonblocking input output model is typically used in a system that is polling to see if an operation is ready. if an operation cannot be completed without sleeping the process return an error [19].

An example of nonblocking i/o would be when the CPU is working on a task and it reaches a point where it requires resources that are being downloaded by the network card, instead of waiting for those resources to finish downloading, to suspend the process and work on something else while the CPU waits. This is seen implemented in Node.js [20].

3.1.3 Multiplexing input output model

Multiplexing input output model is a more intricate approach to the problem that uses two system calls to handle the I/O, calling select on multiple file descriptors waiting for a descriptor to be ready [21]. Once ready calling to the system to copy and process the data. This approach is typically used in a scenario such as a web client or when the server will handle multiple services or protocols [22].

3.1.4 Signal-driven input output model

Signal-driven input output works where the kernel will raise a SIGIO signal when a descriptor is ready. A signal handler is created with relevant information such as the socket owner and destination [23]. Then on the socket a signal is sent to inform the kernel when to read from the socket instead of constantly trying to read the data, only attempting to read and process the data when the signal has been received. This approach uses a signal handler system call to return to the kernel. The advantage of this is that it does not block while waiting for the data and the main process can continue while waiting and be informed when it is time to process the data by the signal handler [24]. Signal driven i/o is used with network time protocol.

3.1.5 Asynchronous input output model

Asynchronous input output model and is often used in real-time functions where the operation is performed, and the kernel will inform the system when the operation has completed. The key difference between the asynchronous I/O model and the signal-driven I/O model is that the kernel tells the signal-driven model when it has started an operation, but with asynchronous it tells the system when it has completed.

An Asynchronous input/output model is defined by the POSIX specification. The POSIX specification is an IEEE standard for application portability with the ambition to maintain compatibility between operating systems [25].

POSIX also defines that an asynchronous input/output operation does not cause the requesting process to block, unlike a synchronous input/output operation that causes a block until that input/output operation completes [26].

4 Interoperability

Interoperability is the way communication between machines is implemented in an unambiguous way. This term in the context of networked applications is typically referring to the way data is interfaced between applications, devices, information systems, and databases in a compatible manner and will ideally be considered at all stages of the design.

Interoperability is not limited to the format, syntax or communication of the data and it can be applied to the practices and policies implemented to ensure compatibility. The compatibility within interoperability may also refer to the operating system that is bound too, and the dependencies required to run it. Interoperability can also refer to the way different programming languages can interact as part of the same system [27]. A good implementation of interoperability in networked applications will follow a standardized approach that will allow for the transmission of data between departments in a raw format that can be received without additional changes on the recipient's infrastructure. Utilizing a uniform structured approach across systems allows for the transmission of data efficiently. There are many ways interoperability can be implemented with standardized approaches such as XML [28], HTTP [29], JSON [30] and YAML [31].

A standardized approach is ideal when communicating across a wide area network as it will allow for the development of different applications utilizing the data without time spent interpreting and processing. This will lead to an overall efficiency increase.

A great example of this type of communication is the implementation of REST API's. In a REST API an application will interact with the end points using standardized HTTP methods such as GET requests to request a return of data in a standardized format, typically but not limited to JSON. An API may be used by the public so taking a uniform and standardized approach allows the developers interacting with the API to build around the response of the API.

An example of interoperable programming language could be JavaScript. JavaScript interacts with HTML and CSS natively and come together to work as a harmonised system. As almost every modern browser has the means of running JavaScript code due to the embedded engine provided by the browser, different browsers have different compilers. In the early days of the language the different browsers that supported had unique features that were not cross compatible leading to developers writing code to specifically detect and confirm the browser the page is viewed on. In more recent times the different engines have collaborated to resolve these issues as much as possible to create a more interoperable platform for the language [32].

5 Report on implementation

In the implementation of a socket based digital form of a boardgame the various resources developed in class were used, such as the format of the messages that are sent between the client and the server and the wrapper of Winsock2tcpip to implement a TCP server and client. The application developed follows a structured approach derived from the Tic Tac Toe implementation by Dr Tim Bashford.

The implemented application is a head-to-head classic battle ships game where two plays position their ships on the board and then take turns attempting to sink the opponents ships by firing at them. Each ship has a different length ultimately the longer the ship the more hits it takes to be sunk. Once all sections of a ship are damaged the ship sinks. Once all the players ships have sunk they have lost the game, and the person with ships surviving wins. The game operates over a TCP/IP architecture due to the nature of the game having a set format where each communication about the activities by each player are required for the continuation of the game. Alternatively, to TCP UDP could have been used but contingencies would have been required to ensure delivery.

An algorithm was implemented to randomly place ships into the game grid, checking if they are an appropriate size to the game grid with considerations to the other ships that are already placed, and the ships are placed within the borders of the map. This also allows for the placement of ships in various directions, such as north, east, south, and west that rotate around the starting point. This allows the algorithm to randomly generate a set of coordinates and then attempt to position the ship on them in 4 different directions.

Manually place ships with visual feedback of positioning was implemented alternatively to randomly placing the ships on the board. The player can opt to set the positions manually, after selecting the class of the ship and the starting coordinates the user will be taken to an interface that gives a visual representation of the playing grid and where their ship will be positioned, it is here that they will be presented with the option to rotate their ship around the provided coordinates and if the user is happy with the position, the user is presented with the option to place the ship or alternatively break out of the positioning and chose a new set of coordinates to begin the process again. This process will repeat until all classes of ship are present on the playing field.

When developing the solution, a consideration was made to the best practices when building an object-oriented program, there are a few paradigms to follow to ensure best practices. An approach with the mnemonic SOLID was taken with a focus to try and ensure reusability and reliability of code while implementing polymorphism. Any code that is repeated is broken off into its own function to ensure minimal repetition following the single responsibility principle and any opportunity to avoid hard coding was taken. An attempt to ensure that all functions have a single purpose with a good use of naming convention to ensure all classes, functions and variables are named appropriately to their purpose and that their usage can be easily determined by their naming convention. With the open/closed principle in mind classes are expanded to accommodate additional functionality rather than existing code modified.

Throughout development documentation of the code was written as it is implemented, this can be found as comments alongside the code. The comments are intended to highlight the purpose of the code and give a brief description of the functionality.

A feature of the implementation is there is on screen feedback giving an instant notification if your attack was a success, or as an opponent if your ships were attacked.

For each message received on the server error handling is implemented to ensure that the state is updated in the eventuality of a player disconnecting and if an incorrect message is received that does not follow the design paradigm a rejection message can be sent.

The game grids are held server side for the purpose of reducing the opportunity to cheat, as well as the functionality of detecting a win, so whenever a move is sent to the server the servers held playing field is updated and the win is calculated based on the grids held on the server rather than the grids held locally. Originally each client had exclusive control of their own grids, but this provided a greater opportunity to cheat and required honesty from the players for a fair game.

5.1 Suggested improvements and further development.

The goal was to provide the player with an option to play head-to-head, or against the computer held server side. The choice to hold the AI server side was made as the intention was to implement a search algorithm on the AI's turns to try and find the best next turn and failing that a backtracking algorithm could be used to find the next best option. The idea was to implement a depth first search to achieve this as this would suit the dynamic of the ship placement.

A further goal would be to implement a messaging system to detect if a ship had been completely sunk, and in the instance of a ship being sunk a relevant message being sent from the server to the appropriate client informing them of the event. This would allow for more precise communication between the clients and allow the client to be informed of when they sink the opponent's ships. This would have been utilized by the AI as a cue to stop searching for more of the same ship.

A further improvement could have been the implementation of features to create a more competitive environment such as a leader board that tracks statistics such as turns to win the match, a waiting room with a chat room for the players to chat while waiting. This would utilize more of the same network code implemented and provided a more engaging experience.

References

- [1] Y. D. C. S. Vinton Cerf, "PROGRAM, SPECIFICATION OF INTERNET TRANSMISSION CONTROL," December 1974 Version.
- [2] "TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION," September 1981, p. page 21.
- [3] "TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION," September 1981, p. page 15.
- [4] U. o. S. California, "3.3 Sequence numbers," in *TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION*, September 1981, p. 24.
- [5] A. M. C. J. S. M. Paxson V, "rfc6298 Computing TCP's Retransmission Timer," in *Computing TCP's Retransmission Timer*, June 2011, p. section 2 page 2.
- [6] D. E. Comer, "Internetworking with TCP/IP: principles, protocols and architecture.," 2006.
- [7] University of Southern California, "INTERNET PROTOCOL DARPA INTERNET PROGRAM," September 1981, pp. section 2.2 page 5-10.
- [8] U. o. S. California, "INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION," 1981, p. 25.
- [9] U. o. S. California, "fragmentation," in *INTERNET PROTOCOL*, 1981, p. page 8 section 2.3.
- [10] D. D. Clark, "IP DATAGRAM REASSEMBLY ALGORITHMS," in *RFC: 815*.
- [11] [Online]. Available:
http://www.tcpipguide.com/free/t_DataEncapsulationProtocolDataUnitsPDUsandServiceData.htm.
- [12] T. - T. T. T. C. Dictionary. [Online]. Available:
<https://techterms.com/definition/pdu#:~:text=Stands%20for%20%22Protocol%20Data%20Unit,OSI%20model%20is%20listed%20below..>
- [13] Internet Engineering Task Force, "TRANSPORT PROTOCOLS," in *Requirements for Internet Hosts -- Communication Layers*, October 1989, p. page 77 .
- [14] CloudFlare, [Online]. Available: <https://www.cloudflare.com/en-gb/learning/network-layer/what-is-the-network-layer/#:~:text=The%20network%20layer%20is%20layer,for%20connections%20between%20different%20networks..>
- [15] "The TCP/IP Guide," 2005-09-20. [Online]. Available:
http://www.tcpipguide.com/free/t_DataLinkLayerLayer2.htm.

- [16] [Online]. Available: <https://erg.abdn.ac.uk/users/gorry/course/intro-pages/encapsulation.html> .
- [17] B. F. A. R. W. Richard Stevens, "UNIX Network Programming," vol. 3, p. 6.2 i/o models.
- [18] R. Athapathu, "Blocking I/O and non-blocking I/O," medium, [Online]. Available: <https://medium.com/coderscorner/tale-of-client-server-and-socket-a6ef54a74763#:~:text=With%20blocking%20I%2FO%2C%20when,do%20nothing%20else%20but%20wait.>
- [19] F. Rosner, 04/05/19. [Online]. Available: <https://blog.codecentric.de/en/2019/04/explain-non-blocking-i-o-like-im-five/>.
- [20] N. js. [Online]. Available: <https://nodejs.org/en/> .
- [21] the daily programmer, [Online]. Available: <https://www.thedailyprogrammer.com/2016/09/network-programming-io-multiplexing.html>.
- [22] U. N. Programming, vol. The Third Edition, pp. 6.2 I/O Models, I/O Multiplexing Model.
- [23] J. Benson, "signal driven i/o presentation," 2016. [Online]. Available: <https://slideplayer.com/slide/7415072/>.
- [24] U. N. Programming, "Signal-Driven I/O Model".
- [25] [Online]. Available: http://www.opengroup.org/austin/papers/posix_faq.html.
- [26] "Asynchronous I/O Model," in *UNIX Network Programming*.
- [27] T. Malone, "Interoperability in Programming Languages".
- [28] [Online]. Available: https://www.w3schools.com/xml/xml_what.asp .
- [29] R. T. Fielding, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content Section 4," Internet Engineering Task Force , June 2014. [Online].
- [30] D. Crockford, " "the JavaScript Object Noation", " Internet Engineering Task Force (IETF) , december 2017. [Online]. Available: <https://tools.ietf.org/html/rfc8259#page-16>.
- [31] [Online]. Available: <https://yaml.org/spec/history/2001-12-10.html>.
- [32] J. Looper, "A guide to JavaScript Engines for Idiots," Telerik Developer Network, 2015. [Online]. Available: <https://web.archive.org/web/20181208123231/http://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/>.

<https://javascript.info/intro>

<https://en.wikipedia.org/wiki/Interoperability>

<https://notes.shichao.io/unp/ch6/#chapter-6-io-multiplexing-the-select-and-poll-functions>

https://en.wikipedia.org/wiki/POSIX#cite_note-1

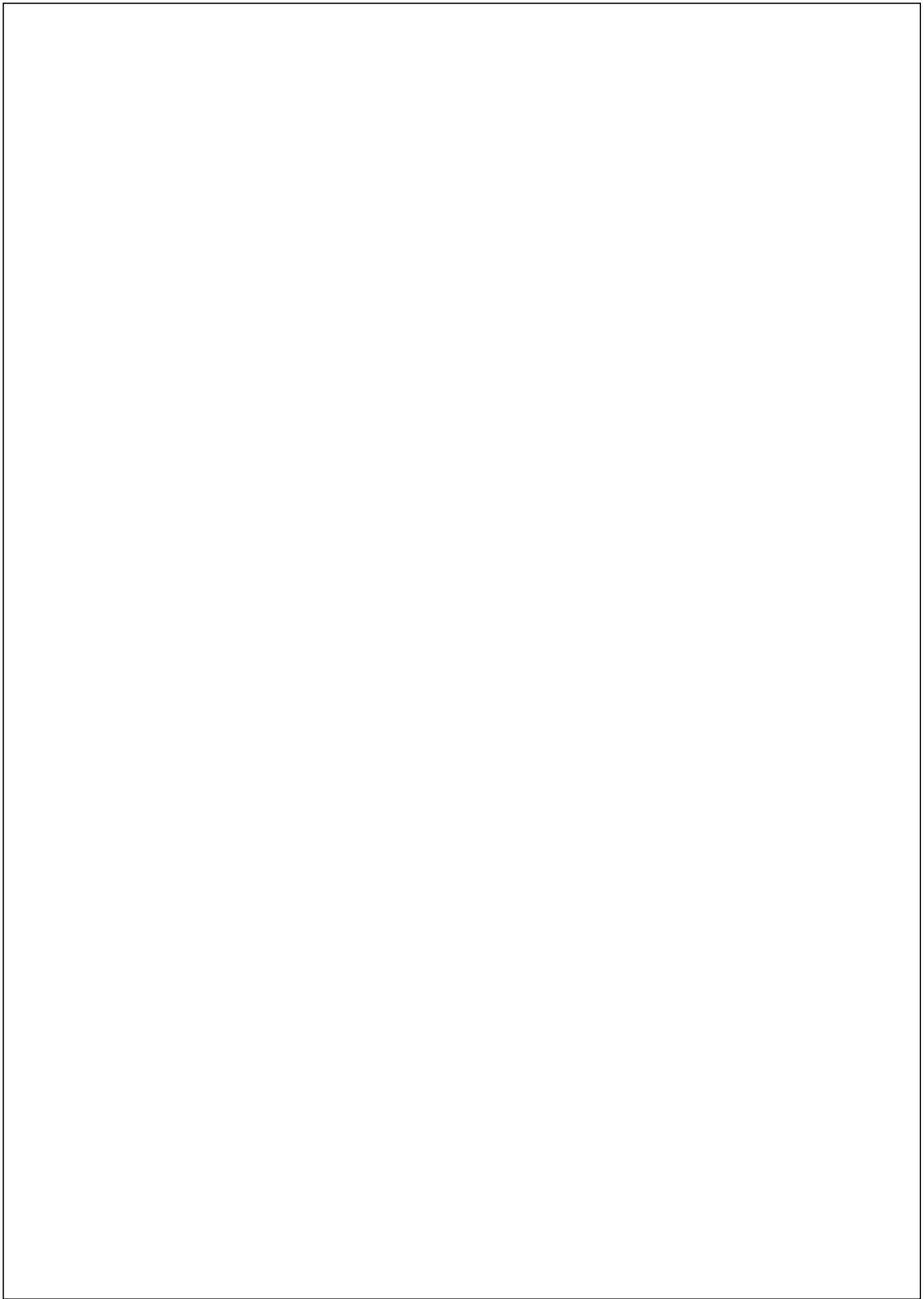
https://www.youtube.com/watch?v=xMtP5ZB3wSk&ab_channel=SunnyClassroom

https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Vulnerabilities

https://en.wikipedia.org/wiki/Maximum_transmission_unit#Tradeoffs

https://en.wikipedia.org/wiki/Protocol_data_unit

https://en.wikipedia.org/wiki/IP_fragmentation



Nathan Morgan Write up

GRADEMARK REPORT

FINAL GRADE

84/100

GENERAL COMMENTS

Instructor

A really good piece of work Nathan, in all respects. You've clearly gotten your head around this difficult topic.

The report is well written, structured and sourced. Some of the sources maybe trend a little too heavily towards non-authoritative web sources, but granted it's not the easiest topic to find modern academic material for.

The implementation is a really good effort - clearly a fair bit of work went into it. As you acknowledge, it's not 100% of the way there, but it's a really good piece of work nonetheless.

Report (100): 86

Implementation (100): 82

Module average: 84

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15
