

Assignment 1 and 2

by Sarann Elizabeth Strange

FILE	1268_SARANN_ELIZABETH_STRANGE_ASSIGNMENT_1_AND_2_349477_427632272.DOCX (1.89M)		
TIME SUBMITTED	18-MAY-2021 05:53PM (UTC+0100)	WORD COUNT	4897
SUBMISSION ID	153786881	CHARACTER COUNT	26638

Socket Programming and Concurrency

Name: Sarann Strange

Student Number:
1701763

University of wales Trinity
saint David

Stream: B.Sc. Software
engineering

Assignment code:
ACCA6028

Lecturer: Dr Tim
Bashford

1 Table of Contents

1	Table of Contents	1
2	Table of figures	2
3	Introduction	3
4	Investigation	4
4.1	Transmission Control Protocol (TCP) / Internet Protocol (IP)	4
4.2	TCP checksum	5
4.3	The two general's problem	6
4.4	The three-way handshake	7
4.5	Transfer of Data	8
4.5.1	Maximum Transmission Unit (MTU) Size	9
4.5.2	Protocol Data Unit (PDU)	10
4.6	Synchronicity & Asynchronicity	11
4.7	Interoperability for Networked Applications	14
5	Implementation Assignment 2	15
5.1	Recommendations.	19
6	Conclusion	22
7	References	23

2 Table of figures

Figure 1 TCP Header Format	4
Figure 2 Example of the Three-way-Handshake	7
Figure 3 synchronous data transfer example	11
Figure 4 Asynchronous data transmission example	11
Figure 5 Synchronous and asynchronous example of transmission	12
Figure 6 Solution Structure.....	15
Figure 7 if to check if client list has reached its limit.....	16
Figure 8 chatClient main	17
Figure 9 Server and two clients.....	18
Figure 10 SendMessages function.....	19
Figure 11 ClientInfo	20
Figure 12 RunServer loop	20

3 Introduction

This report includes the discussion of the Transmission Control Protocol (TCP) / Internet Protocol (IP) what it is how it functions and its relevance within networking. The four layers of TCP are touched upon along with packets and how they are sent and constructed. TCP checksum is also discussed along with its function within TCP.

The two general's problem and the three-way handshake are discussed at length explaining what they are and what they represent.

An explanation on the transfer of Data is provided along with the Maximum Transmission Unit (MTU) Size and its impacts on data when it is sent across the network. Protocol Data Unit (PDU) is covered explaining what it is and how it represents data throughout the OSI model.

Synchronicity & Asynchronicity are compared along with an explanation on their uses and why each would be chosen over the other.

Interoperability for Networked Applications is explored explaining both syntactic and semantic interoperability their uses and how it effects the transfer of data across a network.

4 Investigation

4.1 Transmission Control Protocol (TCP) / Internet Protocol (IP)

TCP/IP are protocols that transmit/exchange/transport data across the internet through sessions which are temporary links to exchange data between two devices or multiple devices. If sessions are not sending data, they time out. But they do not need to be constantly sending data to prevent timeout, for long periods of time when data is not being sent the session will occasionally send a small packet of data to keep the link open [1].

The TCP/IP is used within certain instances as it is highly reliable offering end-to-end byte stream. TCP defines the communication channels that are created for applications and servers. It manages the deconstruction of messages into packages so they can take the fastest route across the internet to their desired destination. Headers are put onto each packet that instructs each packets order during reassembly along with other relevant and required information such as the destination and the source port and error checking, that ensures that the packets arrive as intended [2][1].

TCP has four layers:

1. The **application layer** standardises the data exchange through protocols such as HTTP, SMTP.
2. The **transport layer** involves the maintenance of end-to-end communication over networks through ports.
3. The **internet / network layer** uses IP to connect the origin and destination IP address effectively connecting networks to transport packets.
4. The **network interface layer** deals with MAC addressing to ensure that packets arrive at the correct physical device.

At each layer of the TCP/IP protocol the data is encapsulated. Each layer encapsulates the previous layers header and data then adds its own header and data along with its protocol information, this persists until the data is finally sent and unwrapped at its destination. The image within figure 1 displays an example of the format of a TCP header [1].

TCP packets

[3]

Source port			Destination port		
Sequence number					
Acknowledgement number					
Data offset	Reserved	Flags		Window	
Checksum			Urgent pointer		
Options (+padding)					
Data					

Figure 1 TCP Header Format

4.2 TCP checksum

TCP checksum is the error detection that checks for corruption in the data that was sent over a network. Checksum is calculated and stored in the header of the packet before being sent, a basic example can be seen within table 1.

Table 1 Example of First checksum calculated.

Sent data	Checksum
11010	26

The checksum of the data that has arrived at its destination is recalculated as seen within table 2 then checked against the checksum within the header (table 1). The checksum within table 2 is different to the checksum within table 1 this indicates that the data has been corrupted.

Table 2 Example of Second checksum calculated.

Sent data	Checksum
11000	24

If data has been corrupted or lost during transit this check will recognise that the values do not match, and the corrupted packet is requested again, and it will continue to be requested until it arrives at the destination intact.

TCP/IP protocols are connection-orientated, this means that a connection is not only established but maintained until the message has completed its journey. This established connection along with the instructions within the header and error checking along with checksum is how the protocol endeavours to guarantee the delivery of the data.

4.3 The two general's problem

The two general's problem is a thought experiment that explores the issues that arises when two remote entities are trying to synchronise their actions. The thought experiment asks the participant to imagine two generals trying to synchronise an attack on their enemy [4].

1. General 1 wants to organise an attack at 14:00 and needs to inform general 2 of the plans.
2. General 1 sends a messenger to general 2 with the details of the plan.
3. When general 2 receives the message, they need to respond to confirm that they received the message from general 1 so general 1 knows that the message has successfully been delivered.
4. General 2 sends their own messenger with the confirmation of the first message.
5. General 1 receives the confirmation message from general 2 and needs to confirm the confirmation, so that general 2 knows that general 1 has successfully received the confirmation.
6. General 1 sends another messenger to confirm the confirmation.

The issue with this scenario is there is no end to the back and forth of confirming the receipt of messages from one general to another [5][6].

A similar issue occurs in networking when a client and a server on a network are starting or ending a session. This issue is solved by the three-way-handshake.

4.4 The three-way handshake

The three-way handshake is a part of the TCP/IP protocol, it is an algorithm that establishes and terminates a connection between a client and a server through a series of messages.

The message types and the order that are shared between the client and the server during the three-way handshake are:

1. **Synchronize Sequence Number (SYN)** – This initiates the connection between the client and server and establishes the connection by sending an initial sequence number.
2. **Synchronize Sequence Number-Acknowledgement (SYN-ACK)** – Once the server receives the initial SYN the server responds by sending its own initial sequence number, and the clients initial sequence number incremented by 1, as an acknowledgement that it received the initial SYN.
3. **Acknowledgement (ACK)** – The client acknowledges the connection effectively confirming that it has received the SYN-ACK by sending the servers initial sequence number incremented by 1 back to the server.
4. **Finish (FIN)** – this ends the connection between the client and server by sending a termination request. Another ACK is sent as a recognition response to the FIN both client and server must terminate their connection.

The image within figure 2 displays a visual representation of the three-way-handshake between host P and host Q [7].

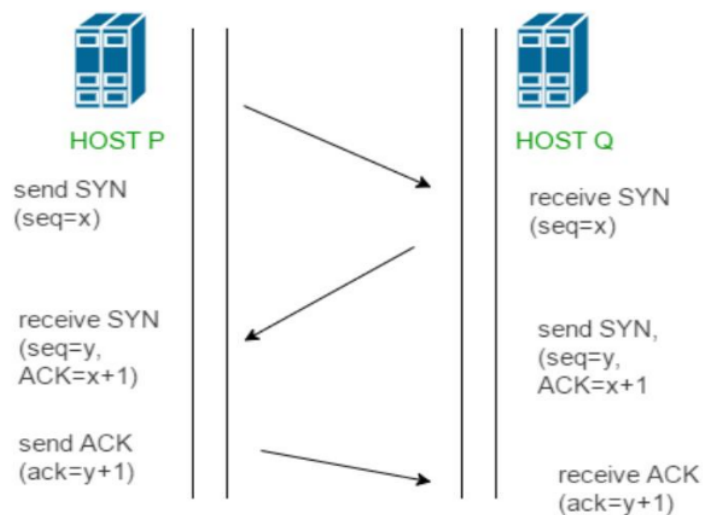


Figure 2 Example of the Three-way-Handshake

[8]

4.5 Transfer of Data

Data transfer/data transmission refers to the transmission or transfer of data / information between two digital devices or multiple devices.

Data is transferred through two methods parallel transfer or serial transmission.

Serial transmission uses a single channel to send data bits serially. This method is used when data is small or is being transferred long distance. This method of sending data ensures the integrity of data as it is sent in order so when the data arrives at its destination it is in-sync.

Parallel transmission uses multiple channels at the same time sending multiple data bits simultaneously. This method is faster than serial as it uses more channels. It is good to use if the data is large, time sensitive or needs to be sent quickly [9].

Data transfer is relevant to developers as considerations must be made on how the data is sent due to necessary security, the complexity of data, the tracking of data and the performance of the transfer of data depending on its size and the processing time required to send and receive the data [9].

4.5.1 Maximum Transmission Unit (MTU) Size

Before a data packet/frame can successfully travel over a network the size of the packet/frame is checked and measured against the size of the MTU. The packet size is checked against the size of the MTU. This check occurs in layer three of the open system interconnection (OSI) model, while the frame size check is done in level two of the OSI model, the data link layer [10][11].

If the packet/frame is equal to or less than the MTU it can pass through the network to its destination. The MTU calculates its measurements in bytes, the standardised maximum size of the MTU is 1500 bytes. If a packet/frame is larger than the MTU it can go through fragmentation (dividing packets into smaller packets) so they can continue across the network. Fragmentation of packets occurs not only between the individual devices that are communicating with each other, but across the network at servers, switches, and routers that all have their own MTU sizes.

Defragmentation/reassembly occurs once the packets arrive at their destination [12].

If fragmentation does not occur and a packet is too large for an MTU it is dropped, and internet control message protocol (ICMP) information is sent to the host informing them that the packet has been dropped due to MTU violations [12].

The drawbacks of MTU and fragmentation is the increase in the overhead of the packages. Each time a packet is fragmented it is given its own header resulting in slightly more data being sent over a network, this can impact the bandwidth available to a user. If a single packet from the fragmentation is lost all the other packets even if they arrived at their destination must be resent, where they will be fragmented once again risking the loss again. Fragmented packets may arrive at their destination out of order, so not only do the fragments need reassembly they may require reordering before it is delivered to a higher protocol layer, but this also uses additional resources of the CPU [13].

The size of the MTU can be manipulated and increased beyond the standard 1500 bytes to 9000 bytes. Increasing the size of the MTU allows more data to be sent increasing the rates of transmitted data, fewer packets are processed, and the overhead of the packets is the same making less load on the network. "Storage management software providers" and local area networks (LAN) usually increase the size of their MTU. A drawback of increasing the size of the MTU is when errors occur the larger packet has more data that is destroyed and is re-transmitted in comparison to smaller packets with errors, less data is destroyed and re-transmitted [13].

MTU impacts the design choices of a developers for various reasons such as the amount of data that needs to be sent across a network along with MTU checks along the network such as routers, switches and other things that have their own MTU check. The developer can choose to stop a packet from fragmenting if they need to or increase the MTU size to send more data at once. The developer must decide on the needs of the company they work for and the availability of resources such as bandwidth, and the needs of clients requesting larger packets of data.

4.5.2 Protocol Data Unit (PDU)

PDU is used to describe and define the name and state of the data unit used at any given protocol layer of the OSI model. Table 3 displays the OSI model, TCP/IP and the PDU. The Information that is sent through the network is the same, but the actual data and packets are altered at different layers so that it can be sent successfully through the network [13].

Table 3 OSI, PDU TCP/IP table with explanations on what they are and what they do.

OSI	PDU	TCP/IP	Explanation
Application	Data This represents the originally data from an application.	HTTP, SMTP, SSH	This layer is where software applications interact and send data or receive data through websites, outlook, or other applications.
Presentation			This layer deals with syntax processing. It converts data from one format into another. For instance, it translates the data from the application format to the network format or the other way around.
Session			This layer constructs the connection between devices, it authenticates the connection and if a network interruption occurs it reconnects the devices. [13]
Transport	Segment This contains the original data with the added TCP headers. The information added by the headers includes the receivers and senders port numbers.	TCP, UDP	This layer is responsible for the transmission of data. It quadrantes the speed of the data, the route the data takes through the network, the amount of data being sent and if data needs to be sent again. At this layer, the data is examined for errors. [13]
Network	Packet This includes the data along with the previously added TCP headers and the network layer header (this contains the IP address of the receiver and the sender) [14]	IP, ICMP, DHCP	This layer manages the mapping of physical and logical addresses through address resolution protocol (ARP). It contains the address for where the information came from and the destination address.
Data Link	Frames This layer is comprised of the data with TCP headers, and the network headers with the final addition of a trailer and the data link header that holds the MAC addresses.	Ethernet, XDSL	The two subdivisions of the data link layer are: 1. Media access control (MAC) 2. Logical Link Control (LLC) It creates links between the physical networks. It evaluates the presence of transmission errors before packaging the bits into data frames, it manages physical addressing methods of MAC or LLC. [13]
Physical	Bit This is the (1s or 0s) that transmits the data.		This layer is the electrical and physical layer. The data unit in the physical layer is represented in bits and sent across electrical wires. It represents raw bits, cables wires, radio frequencies and pulses of light.

4.6 Synchronicity & Asynchronicity

Synchronous and asynchronous are two different types of data transmission within networking [15].

Synchronous transmission involves a permanent synchronisation of the sender and receiver through streaming data signals and timing signals. The timing signal that is used by both the sender and receiver that enables the transfer of data is generated by an internal clock, which can be a replica of the clock signal or the same clock signal from the sender or the receiver. The image within figure 3 displays a timing signal being sent to the receiver along with the relevant data [16][17][18].

Synchronous data transmission



[19]  Timing Signal  Data

Figure 3 synchronous data transfer example

Asynchronous unlike synchronous sends data by using “start bit” and a “stop bit” it is not permanently synced. When transmitting data both the receiver and the sender must be synced and the amount of data being sent is pre-agreed, so the receiver knows where the “start bit” starts. The sender and receiver sync through timing the distance of the “start bit” to the bit of data after the “start bit”. The image within figure 4 displays the “start bit” followed by the relevant data and an “end bit” [15].

Asynchronous data transmission



[19]  Start  End  Data

Figure 4 Asynchronous data transmission example

The image displayed within figure 5 displays the differences between both synchronous and asynchronous data transmission.

Synchronous and Asynchronous Transmission

Point of Comparison	Synchronous Transmission	Asynchronous Transmission
Definition	Transmits data in the form of chunks or frames	Transmits 1 byte or character at a time
Speed of Transmission	Quick	Slow
Cost	Expensive	Cost-effective
Time Interval	Constant	Random
Gaps between the data?	Does not exist	Exist
Examples	Chat Rooms, Telephonic Conversations, Video Conferencing	Email, Forums, Letters

[20] *Figure 5 Synchronous and asynchronous example of transmission*

Synchronous has a faster transition than asynchronous as it transfers blocks/frames of data and can send large amounts of data at one time, it does not have gaps between the data, it is fast and allows connected devices real-time communication. A disadvantage of synchronous transmission is that the accuracy of the data relies on the bits being accurately counted by the receiver. Another drawback is the reliance of the receiver and transmitter operating at the same time using the same clock signal. Applications that use synchronous data transmission include multiplayer video-games, chat-rooms and other applications that allow users to instantly communicate and interact with one another [16][17][18].

Asynchronous works by sending one byte/character at a time and is proficient at sending small amounts of data at a time with a gap between the data, unlike synchronous it is slow. A disadvantage of asynchronous is that timing errors can occur due to issues identifying synchronicity and the necessary addition of data due to the stop and start bits. Applications that use asynchronous data transmission include email, forums, and other applications that users can interact with that does not require instant real-time communication [15].

When developing a system developer will need to consider if the system requires synchronous or asynchronous data transmission this choice relies on the requirement of the application/system. For instance, if the system requires real time communication between different users, they will require synchronicity transmission of data as this would allow for separate devices to make changes and updates to data simultaneously [16][18].

If data is being stored and does not require instant communication between users, then the asynchronous is the best option as it does not require a constant synchronization to carryout tasks making it cost effective as it can send the required data by synchronizing as and when it is needed.

4.7 Interoperability for Networked Applications

Interoperability in the context of networked applications, refers to the ability of different systems, software, and applications on the same network to not only exchange data in-real-time but receive data at a constant unrestricted rate. Interoperability allows the shared data to be used and understood in a meaningful way, during busy periods and normal periods, for the network [21].

When different systems send and receive data, they must follow the same data structure protocols and data format for that data to be interpreted and understood, across all the different devices. The two types of interoperability include syntactic and semantic.

Syntactic interoperability is the process that permits the different devices on a network to work together, this includes interfaces that use different programming languages to one another [22][21].

Semantic interoperability interprets the data that is sent or received through the network by a system. It provides metadata to the data and uses an information exchange reference model to interpret the data. Effectively allowing the data to be accessed in a meaningful way by all the different systems [21].

Interoperability is useful as it provides a standard across all devices within a network reducing errors and costs and overall productivity. Interoperability allows collaboration between individuals, governing bodies, or businesses. For instance, healthcare organisations such as hospitals using different devices and systems can access relevant data when it is needed. Public organization's use a multitude of devices such as schools when they need to access records or information, they would use Interoperability [22].

To achieve Interoperability metadata is used, it is included onto all data within the system that links that data to a common vocabulary that is used throughout the network between all the devices and systems. Interoperability effectively future proofs new devices that are included to the network as a pre-established vocabulary exists. Developers must ensure that they follow the structure of the pre-established protocols to allow new data and systems to successfully translate data and connect and interact with other devices on a network successfully [22].

5 Implementation Assignment 2

A chat application was designed and created for the implementation section of this report. The solution is written in C++ and uses the Windows Winsock API for networking, and various C++ Standard Libraries for general programming and multi-threading it was created in Visual Studio. A number of resources were used to aid in the development of the solution they include MicrosoftDocumentation [23], ysSocketChat [24], Bits and Pieces of Code [25], VBForums [26].

The following text and images will explain the implementation of code and the functionality of the chat application along with desired improvements that the student of this report wished to carry out if given more time to do so.

It contains two projects: A Chat Client and a Chat Server. The Chat Server needs to be run first since it handles all client connections. Once it is run it will wait forever for a client to connect.

The Chat Client can then be run, which will automatically connect to the server. If the Chat Client is run first, it will immediately fail and close as it does not detect a server to connect to.

Figure 6 shows the file solution structure for both Chat Client and a Chat Server.

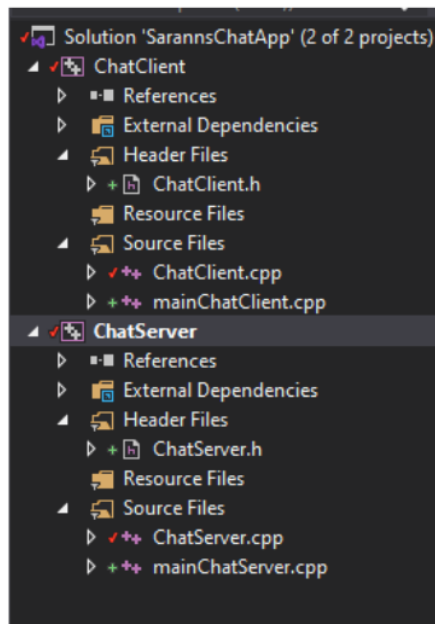


Figure 6 Solution Structure

The chat app is setup to run locally with the Server running on port "27015", and a client is hardcoded to connect to localhost ("127.0.0.1"). However, the program has been tested on a local network and successfully ran a client on a laptop that connected to the ChatServer on a local desktop computer.

However, the program was not tested across the internet. The home network is configured with a firewall and other protections that stop users from connecting to the ChatServer on the local computer the student of this report was unable to work around them.

The programs use TCP for communication as it is more reliable for sending information across a network compared to UDP. The windows network stack is capable of thousands of concurrent connections to the server, however it is not something that will be suitable for this app. Instead, the ChatServer has a hardcoded limit of 16 concurrent client connections. If more users attempt to connect, they are rejected by the server and told that it is full, the server then closes their connection socket as seen within figure 7

```
// Check connection limit
// If too many users have joined, newest user shown messaged, and kicked
if (serverClientCount >= serverClientLimit)
{
    SendMsg(tempSocket, "Server is full.");
    closesocket(tempSocket);
    continue;
}
serverClientCount++;
```

Figure 7 if to check if client list has reached its limit

The server stores the client's information into a vector when they connect. This is used for:

- Keep track of all the active connections.
- A convenient way to broadcast messages to all other clients.
- Form a "username" by extracting the client's hostname and port from their connection.

A vector of threads is used to handle each client connection. The thread runs in a loop, receiving messages from a client, then sending that message onto all other connected clients. Once the connection has ended the client's details are removed from the vector and the thread ends. The threads are joined in the ChatServers destructor on program end.

The ChatClient is not as complex as the Server. When the program is run it automatically attempts to connect to the server and a user can start typing messages once its connected.

It uses one thread to receive messages from the server. The thread runs in a loop constantly receiving messages and printing them to the screen. While the main thread is used for sending messages to the server by running in a loop, accepting std::cin input from the keyboard and then sending the message through winsock, as seen within figure 8.

```
int main()
{
    // Create ChatClient class with local ip address and random port
    ChatClient client("127.0.0.1", "27015");

    // Set up the client
    client.Connect();

    // Create a thread to receive messages from the server
    client.ReceiveMessages();

    // Send keyboard messages to server
    client.SendMessages();
}
```

Figure 8 chatClient main

The image within figure 9 displays the chatServer along with two clients who can communicate with one another freely.

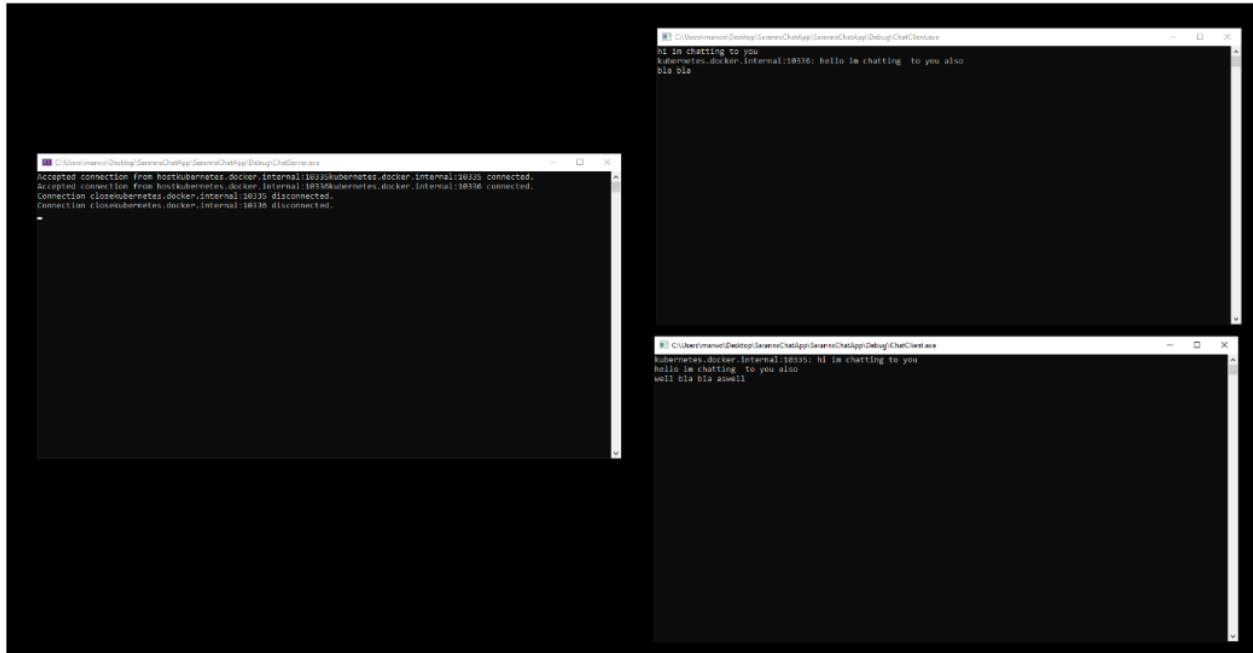


Figure 9 Server and two clients

5.1 Recommendations.

There are several recommendations to improve the code and the application as a whole due to time restraints the student was unable to complete and fix the desired code.

The image within figure 10 displays the ChatClients SendMessages function. The improvement that needed to be made to this function involves creating a way for the user to quit the program successfully and safely. Currently it runs forever, and the user has no way to cleanly exit the chat application. This could be fixed by capturing a written command, checking it then breaking the while loop. Or capturing CTRL+C the standard way to break from a console.

```
void ChatClient::SendMessages()
{
    // Main thread stays here forever getting input from the keyboard and sending it to the chat server
    // When user wants to quit we break the while loop then go to shutdown the connection socket

    // TODO: Add quitting functionality to leave the while loop so we can shutdown the socket cleanly
    // Capture consoles CTRL+C to quit or the user can write "quit"
    while (true)
    {
        // Get a line of text from the users keyboard
        std::string msgToSend;
        std::getline(std::cin, msgToSend);

        //send string message to user
        int iResult = send(connectionSocket, msgToSend.c_str(), msgToSend.size(), 0);
        if (iResult == SOCKET_ERROR)
        {
            std::cout << "Send failed with error: ", WSAGetLastError();
            break;
        }
    }

    //Shutdown the connection since no more data will be sent
    int iResult = shutdown(connectionSocket, SD_SEND);
    if (iResult == SOCKET_ERROR)
    {
        std::cout << "shutdown failed with error: ", WSAGetLastError();
    }
}
```

Figure 10 SendMessages function

The image within figure 11 contains a clients information structure. The changes that needed to be made to this include a client sending a username of their choice once they have connected to the server. Currently the client hostname and client port are combined and used as a username instead.

```
//keeping a record of the clients details.
// We get the connection details (ip, port) from the sockaddr
// TODO: The client needs to send their username after they have connected
struct ClientInfo
{
    sockaddr addr;
    SOCKET clientSocket;
    std::string clientHostName;
    std::string clientPort;
    std::string userName;
    int closed = 0;
};
```

Figure 11 ClientInfo

the image within figure 12 displays the main loop within RunServer for accepting new clients and handling each client in their own thread. The issue with this code is like figure 10 there is no way to cleanly exit the loop. The fix for this code is the same.

```
//loop forever accepting clients and placing each client in its own thread
// TODO: Add quitting functionality to leave the while loop so we can shutdown the server cleanly
// Capture consoles CTRL+C to quit?
while (true)
{
    sockaddr temp_addr;
    socklen_t temp_len = sizeof(temp_addr);

    //accept a client into tempSocket
    SOCKET tempSocket = accept(listenSocket, &temp_addr, &temp_len);
    if (tempSocket == INVALID_SOCKET)
    {
        std::cout << "accept failed with error: " << WSAGetLastError();
        closesocket(listenSocket);
        return;
    }

    // Check connection limit
    // If too many users have joined, newest user shown messaged, and kicked
    if (serverClientCount >= serverClientLimit)
    {
        SendMsg(tempSocket, "Server is full.");
        closesocket(tempSocket);
        continue;
    }
    serverClientCount++;
}
```

Figure 12 RunServer loop

Currently there is no serialisation being used in the programs, the application directly sends simple null-terminated strings between the client and server. As a future improvement more complex data structures will be designed and implemented, such as the use of a class that not only contains the message string, but also contains things like a username, message timestamps, a user's choice of colour for text messages. To achieve a higher level of functionality with this chat application the popular serialisation libraries such as `boost.serialization` [27], `JsonCPP` [28] or `Cereal` would have been used.

6 Conclusion

The report successfully identified many of the protocols and procedures that manipulate data and change it and its structure as it travels through a network to its desired destination. The report talks in depth about the OSI model and the various protocols that use its structure to identify data at various stages such as Protocol Data Unit (PDU). Or break the data down into packets so that they can travel across a network such as the fragmentation that can occur due to the Maximum Transmission Unit (MTU) Size.

The report successfully identified how data travels across networks through synchronicity and asynchronicity and how interoperability for networked applications can enable different devices within a network to transfer, receive, and understand data using protocols.

Finally the implementation of a chat application was created to allow users to send and receive messages.

7 References

- [1] Microsoft, "TCP/IP and Data Communications Administration Guide," *Microsoft*, 2020. .
- [2] IBM, "TCP/IP protocols - IBM Documentation," *IBM*, 2020. <https://www.ibm.com/docs/en/aix/7.2?topic=protocol-tcpip-protocols> (accessed May 11, 2021).
- [3] Dr Tim Bashford, "Course: L6 - Socket Programming and Concurrency (ACCA6028) HTTP over TCP/IP," 2020. <https://moodle.uwtsd.ac.uk/course/view.php?id=6605> (accessed Apr. 06, 2021).
- [4] Microsoft, "The three-way handshake via TCP/IP - Windows Server | Microsoft Docs," *Microsoft Build*, 2021. <https://docs.microsoft.com/en-us/troubleshoot/windows-server/networking/three-way-handshake-via-tcpip> (accessed May 11, 2021).
- [5] O. V. V. Ekaterina Pshekhotskaya, "The Simplified Two Generals Problem for Communication Immutability and Security Over Multiple Heterogeneous Channels," *asp*, 2018. .
- [6] J. F. Eric Conrad, "Three-Way Handshake - an overview | ScienceDirect Topics," *science Direct*, 2012. <https://www.sciencedirect.com/topics/computer-science/three-way-handshake> (accessed May 11, 2021).
- [7] MWAMI MUGALA, "SECURITY VULNERABILITIES AND COUNTERMEASURES IN TC," 2014. .
- [8] GeeksforGeeks, "TCP 3-Way Handshake Process ," *GeeksforGeeks*, Sep. 06, 2019. <https://www.geeksforgeeks.org/tcp-3-way-handshake-process/> (accessed Apr. 06, 2021).
- [9] T. Chowdhury and R. Mustafa, "Parallel data transmission: A proposed multi-layered reference model," in *Novel Algorithms and Techniques in Telecommunications and Networking*, 2010, pp. 139–142, doi: 10.1007/978-90-481-3662-9_23.
- [10] S. V. Ahamed and V. B. Lawrence, "The Role of the OSI Model," in *Intelligent Broadband Multimedia Networks*, Springer US, 1997, pp. 94–123.
- [11] M. Gregg, "Layer 3: The Network Layer," in *Hack the Stack*, Elsevier, 2006, pp. 103–150.
- [12] cisco, "Resolve IPv4 Fragmentation, MTU, MSS, and PMTUD Issues with GRE and IPsec - Cisco," *cisco*, Jan. 08, 2019. <https://www.cisco.com/c/en/us/support/docs/ip/generic-routing-encapsulation-gre/25885-pmtud-ipfrag.html> (accessed May 11, 2021).
- [13] C. M. Stefan Rommer, "Protocol Data Unit - an overview | ScienceDirect Topics," *scienceDirect*, 2020. <https://www.sciencedirect.com/topics/computer-science/protocol-data-unit> (accessed May 11, 2021).

- [14] digital tut, "OSPF router ID selection," *digital tut*, 2021.
<https://www.digitaltut.com/?s=eigrp> (accessed May 18, 2021).
- [15] C. Wang, S. Li, B. Li, and Q. Chen, "An Asynchronous Data Transmission Method in Heterogeneous Wireless Networks," *Complexity*, vol. 2020, 2020, doi: 10.1155/2020/8828794.
- [16] N. bandOleksandrP. M. NikolaosBardis, "Synchronization Error Correction for Asynchronous Channels Data Transmissi," 2017.
- [17] Barry Dwyer, "Synchronous Communication - an overview | ScienceDirect Topics," *scienceDirect*, 2016. <https://www.sciencedirect.com/topics/computer-science/synchronous-communication> (accessed May 11, 2021).
- [18] IBM, "Synchronous transmission - IBM Documentation," *IBM*, 2020.
<https://www.ibm.com/docs/en/aix/7.2?topic=synchronization-synchronous-transmission> (accessed May 11, 2021).
- [19] Computer Science GCSE GURU, "Synchronous and Asynchronous ,"
Computer Science GCSE GURU, 2021.
<https://www.computerscience.gcse.guru/theory/synchronous-and-asynchronous> (accessed Apr. 13, 2021).
- [20] tech computer science, "Synchronous & Asynchronous Data Transmission | Computer Science," *tech computer science*, 2021.
<https://teachcomputerscience.com/synchronous-and-asynchronous/> (accessed Apr. 13, 2021).
- [21] C. Jovellanos, "Semantic and syntactic interoperability," 2003, p. 266, doi: 10.1145/779928.779991.
- [22] P. S. Teixeira de Sousa, "TELECOMMUNICATION NETWORK INTEROPERABILI," *TELECOMMUNICATION SYSTEMS AND TECHNOLOGIES* -, 2009. .
- [23] MicrosoftDocumentation, "Getting Started with Winsock - Win32 apps | Microsoft Docs," *MicrosoftDocumentation* , 2021.
<https://docs.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock> (accessed May 12, 2021).
- [24] ysSocketChat, "C++ Cross Platform Multi Thread Chat Application | ysSocketChat," *ysSocketChat*, 2021.
<https://www.yusufsezer.com/ysSocketChat/> (accessed May 12, 2021).
- [25] Karim Oumghar, "Client-Server chat in C++ using sockets – Bits and Pieces of Code," *Bits and Pieces of Code*.
<https://simpledevcode.wordpress.com/2016/06/16/client-server-chat-in-c-using-sockets/> (accessed May 12, 2021).
- [26] Jacob Roman, "C++ Simple Chat Program In a Console Window (Winsock & Multithreading)-VBForums," *VBForums*.
[https://www.vbforums.com/showthread.php?781947-C-Simple-Chat-Program-In-a-Console-Window-\(Winsock-amp-Multithreading\)](https://www.vbforums.com/showthread.php?781947-C-Simple-Chat-Program-In-a-Console-Window-(Winsock-amp-Multithreading)) (accessed May 12, 2021).
- [27] Boost, "Serialization," 2004.

- [28] open-source-parsers and /jsoncpp, "GitHub - open-source-parsers/jsoncpp: A C++ library for interacting with JSON.," *GitHub*. <https://github.com/open-source-parsers/jsoncpp> (accessed May 12, 2021).

Assignment 1 and 2

GRADEMARK REPORT

FINAL GRADE

74/100

GENERAL COMMENTS

Instructor

A very good report with an appropriate level of detail. A nice implementation too - a little basic, but I realise there were time constraints. Nonetheless, a good effort!

Report (100): 82

Implementation (100): 66

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

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26
