

1910716 ACCA6028 Final Submission

by Ollie Bird

Submission date: 02-May-2023 09:03PM (UTC+0100)

Submission ID: 204395308

File name: 18246_Ollie_Bird_1910716_ACCA6028_Final_Submission_499749_218986986.docx (216.69K)

Word count: 5089

Character count: 27055

ACCA6028 Task 1 – Research Driven Essay into Application Networking Concepts

Oliver Bird

Apprentice Software Engineer – BT, Data Services

Applicable word count - 3480

1910716@student.uwtsd.ac.uk

1. Transport Control Protocol (TCP)

a) Overview - TCP is a networking protocol contained in layer 4 of both the OSI and TCP/IP models. Its use covers a large number of networking requests globally due to it being a reliable method of transmitting data. The protocol was generally defined in RFC 793, but over time has developed many further standards for more specific use cases and abstractions (such as RFC 1323 for high performance) and is most recently documented in RFC 9293 [1]. The finer intricacies of TCP greatly transcend this report but are covered generally instead. A more verbose description can be found in the above standards, or texts such as [2]. TCP provides a large set of functionality to bridge the gap between the network and application layers, as detailed below:

- Addressing – handles the addressing of many different applications in use on one system above the TCP layers.
- Connection Management – TCP provides a set of logical operations for devices on a network to follow to maintain, re-establish and terminate a connection.
- Packing – TCP provides a mechanism to package data in a standard format which can be deconstructed on arrival.
- Transfer – TCP passes data down the model to the network layer for transport.
- Flow Control – TCP can handle the flow rate of data when sending data to not overload the network whilst also handling network congestion in the case of bandwidth issues.

b) Limitations – before discussing TCP's defining quality of delivery *guarantee*, some limitations should be noted:

- TCP is merely the transport protocol; it requires the applications to *know* how to use TCP to handle the data transfer.
- It guarantees no security, privacy, or authenticity to the data it transmits.
- TCP sends data as a continuous stream; therefore, applications must define the boundaries between pieces of data.
- TCP cannot handle every kind of disconnect between two systems. It can simply retry to send messages which weren't acknowledged. Furthermore, it can control flow of data, but cannot handle every possible scenario where it may need to.

c) The Two Generals Problem – TCP is an *attempt* to remediate the two generals' problem when applied to networking. The conundrum states that two generals which are both trying to attack a castle at a synchronized time have no method of guaranteeing the other general received their attack plans, as delivering them would cause a messenger to cross hostile territory between the generals. The initial solution is for the general **receiving** the attack message to send a messenger back across the hostile plain to confirm receipt of the attack plans. This however does not solve the problem, as the general who initially called the attack may receive the confirmation of the attack plans delivery, but then the second general would not know if his confirmation had been received, meaning that the first general may not attack. This process has an infinite number of sending confirmations and has no exact solution using this method.

An accepted solution to this problem is not to directly solve it, but to mitigate as much as possible the chance of the original message and confirmation not being received. TCP provides a solution to this problem by including an identifier on each segment that it sends and receives a confirmation of each of these. Should a confirmation not be sent for a given segment, it is resent. Furthermore, at least in the case of the generals, reducing the number of messengers needing to be sent an estimation of the time for the message to be sent and responded to is found (Round trip time RTT). If a confirmation is not sent within the expected RTT, the message is sent again.

D) Delivery Guaranteed! – TCP defines rigorous rules for data delivery and retransmission in the case of data not being received (or received and not yet acknowledged!):

Segmentation is the process of splitting out a data stream from the application layer into a smaller collection of groups of bytes (octets). Upon the connection being established, both devices negotiate a beginning sequence number for the stream of bytes, before the data transfer. This is discussed further later. Each byte within the overall data stream is then given a sequence value as an identifier. A segment contains a set of these bytes (up to a maximum length defined above), which is then given a header and passed on to the IP protocol to be transferred.

This sequence of bytes can then be identified into:

- 1) Bytes sent and acknowledged
- 2) Bytes sent but not yet acknowledged

- 3) Bytes not yet sent but the recipient is ready to receive
- 4) Bytes not yet sent and are not ready to be received.

The above categories are key to identifying which bytes have not actually been received. The receiver sends an acknowledgement for each byte sequence range (not each byte sequence number to save performance), any sequences not received are not acknowledged. Any subsequent segments that are received are held on the receiver and not acknowledged, as the order of the sequence matters because the acknowledgement is cumulative. The sending device must be configured in a way to decide on how and when the bytes should be retransmitted to allow the acknowledgements to continue from the receiver. This is the example described in the two generals' problem, as it can either wait a fixed amount of time before an automatic retransmission, send immediately, or send a smaller segment etc. These are complex and out of scope of this report, but ultimately TCP will attempt to retransmit any byte sequences not acknowledged by the receiver.

The entire TCP connection has the following finite state machine from first to final connection:

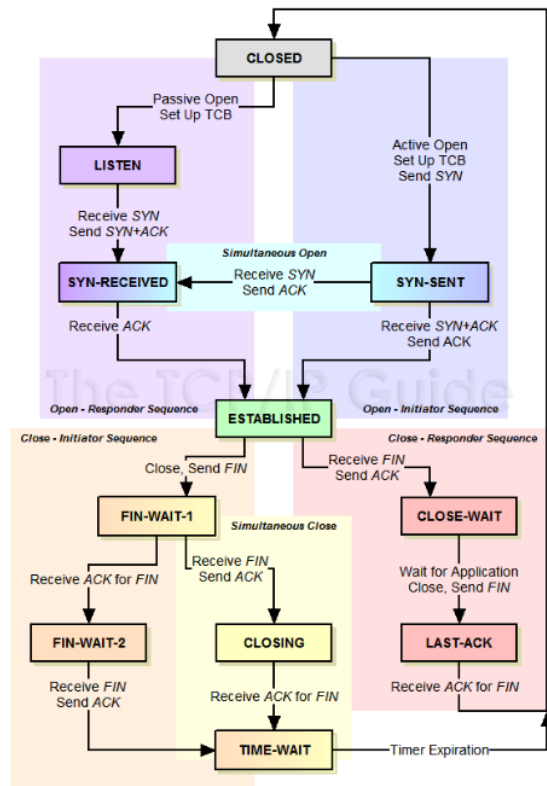


Fig. 1 TCP Finite State Machine [3]

The above state machine describes all possible states for a TCP connection to be in, along with the requests sent to trigger the

states. The top section of Fig.1 details what is known as the “3-way handshake”.

The 3-way handshake is how a TCP connection is established between two parties. The initiator first sends a SYN (synchronization message) request to a provider in a listening state. The SYN request contains information about the connection requirements from the sender in an attempt to synchronise the connection. This includes an initial sequence number (in modern devices this is randomised); and can include a window scale factor in the case for wanting a different number of bytes allowed in the send window (this is the number of bytes contained in categories 2 & 3 in the byte sequence list on page 1), the SACK parameter (to allow a receiver to acknowledge non-consecutive data) or an alternative checksum method.

The receiver then responds with a SYN-ACK request to the original sender to firstly acknowledge receipt of the SYN (this is the ACK) and to confirm the above described parameters. It also sends its own SYN request as part of this.

Finally, the client responds with a final acknowledgement ACK of the receivers SYN-ACK to fully establish the connection. Once this is done, data transfer can proceed following the segmentation process above.

As is mentioned in b, TCP does not guarantee the authenticity of a message, but it does attempt to provide some level of proof of the original message being transferred. This is done via a checksum, in which **some** of the headers and the entire TCP body are summed in their binary formats. This value is then added to the checksum field in the TCP header. These values are checked on the recipient and matched against the header field. If they do not match, it is assumed the data has changed in transit and hence an acknowledgement of that segment is **not** sent.

e) Developer notes – For developers it is then important to consider the use cases for TCP connectivity (such as application type, e.g., if files transferred require acknowledgement), the exact implementation of TCP (e.g. the use of SACK or the order and timing of the retransmission queue) and any potential security / application risks for the abuse of TCP (such as *cheats* in games where client data can be modified at or below the TCP layer, or the classic SYN flood denial of service attack [4]).

2. The Maximum Transmission Unit

a) Overview - The maximum transmission unit for a given entity on a network is the maximum size of a data packet (including its headers) which the device can process at one time. For Ethernet 1, the MTU is 1520 bytes including the IP header. This means, for the majority of the internet the MTU is also 1520, meaning a maximum SDU (of layer 2) being 1500 bytes in size.

Given that the internet contains many possible nodes and routes from a given point A to point B, it is **not** safe to assume that the above transmission size is concurrent and observed by all devices across a potential route. A solution to this is to undertake a process called MTU Path Discovery. When transmitting the data across a set of links, ideally the most efficient method of passing the data should be used (i.e., not fragmenting the data into sizes too small where the

ratio of data to header is smaller than it could be). This means that the MTU of the entire path needs to be discovered, in order to maximise the sizes of the packets (and furthermore PDUs) sent across. This process is done by sending large packets across the network with the 'do not fragment' flag set to true. Packets that are dropped or returned with a MTU too large error indicate that a smaller MTU should be used. This process is repeated until the packets are successfully transmitted.

b) IP Fragmentation - [5] In a similar vein to this, there is a process for IP fragmentation, in which packets across a network can be fragmented (similar to that of TCP) in order to abide by the MTU of a given link.

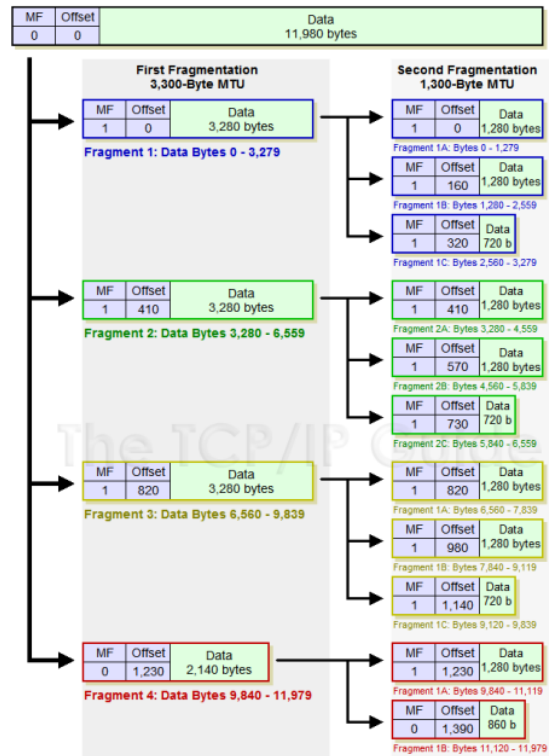


Fig. 2 IP Fragmentation for Different Device MTUs [6]

Fig.2 shows the fragmentation process which can be enacted during transmission of the packet on their receipt by a linking device. This process can only be done if the original packets are not flagged with 'do not fragment'. In that case, a smaller MTU should be chosen.

It should be noted further that for IPv4, packet fragmentation in transit is permitted (given the above acceptable flag), but for IPv6 this is not the case. IPv6 networks must at least conform to a minimum MTU of 1280 bytes [7]. If the MTU is exceeded across IPv6, the link node will return a ICMPv6 packet to the source indicating that the MTU has been exceeded. The onus is then on the sender to fragment the packets further to pass the MTU size limits.

c) Protocol Data Units – Each layer of the OSI model contains different protocol data units. These are the units of data (such as packet for the network layer, or segment for the TCP layer) which each layer transmits. The SDU (service data unit) is the actual data passed between each layer of the model. This makes the PDU of a layer, the SDU of the layer below it, prior to the addition of that layer's header. When relating this information with the concept of the MTU, it is clear that ultimately, in the data link layer of the OSI model, the frame (PDU for layer 2) cannot exceed the MTU of the overall network which it is trying to traverse.

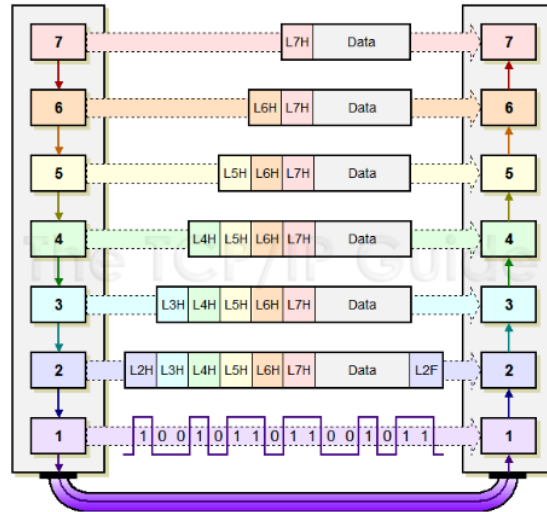


Fig. 3 Layer Encapsulation

One way this can be handled is during the 3-way handshake described in 1d. The TCP layer can negotiate the maximum segment size (MSS) through the SYN request stage of the handshake. By manipulating this value to be :

the maximum MTU of the network - IP header - TCP header

Eq. 1 MSS of Network Equation

By using the above, it is possible to configure the most efficient point-to-point TCP connection in terms of MTU sizes.

It should be noted that other protocols can add further headers, more than the simple cases discussed above. For example, IPsec [8] will add additional headers and followers to packets, which contain authentication and encryption information used by the receiver to validate the state of the packets. This in turn consumes some of the available bytes for the MTU.

Therefore, in general, the maximum MTU for a p2p data transfer is:

$$\text{Possible MTU} = \sum(a) + \sum b + c \leq \text{Known Network MTU}$$

Eq. 2 Possible MTU for a given network.

Where a is the number of bytes of the headers in each layer, b is the number of bytes of followers in each layer, and c is the bytes

of original application data. Should this value exceed the known MTU for a network, then the original data must be fragmented at the source to allow the above to be true. For TCP segments over the internet, in most cases this results in a maximum segment of 1380 bytes, as the layer 4 and 3 headers are a maximum of 60 and layer 2 20, adding up to the maximum total for an ethernet frame of 1520 bytes.

d) Considerations – For developers, there are a number of considerations to make with regards to MTUs and hence applications and devices:

- Network type – For some networks, certainly LANs and high-performance networks, there may be a larger MTU than the rest of the internet to make use of. Jumbo Frames [8] are an example of this, in which some devices can handle network frames of 9000 bytes. This increase increases the payload size per packet and hence reduces the processing costs of handling them.
- For secure applications, the application layer PDU will likely have to accommodate additional headers which could result in dropped/sent back packets.
- Most generally, consider the infrastructure being used to send packets, and **all** of the associated header and follower sizes.

3) Synchronicity

Synchronicity is a description of how messages are passed between two machines, in which data is able to be transmitted and received at the same time. This is accomplished via synchronization between the two machines through a mesochronic system clock. This means that the clocks measure time at the same frequency, but do not necessarily state the same time. These clocks are calibrated using a primary master clock which dictates the time over the internet, or in some institutions a direct cable.

The synchronous flow of data is a more efficient way to transfer data than asynchronous (as is discussed below) as data is free to move in both directions, (full duplex) and in larger blocks as the entire transmission can be sent without any breaks in the data stream.

Asynchronous transmission differs in that the systems do not require any sort of synchronization for the message to be transmitted. The sender is able to send a stream of bits or characters, each encapsulated by a start and end bit to delineate each message. This method is suited to applications where time is more available to the system (for example in cases of minimal user interaction).

Each system has benefits and drawbacks depending on the use case:

a) Synchronous – This type of transmission is fast as data can be exchanged at the same time, for example video transmission between two users.

There are no periods in which the data link is not in use, as messages are constantly being sent with no gaps between them (as is found in asynchronous models). The gaps are shown below

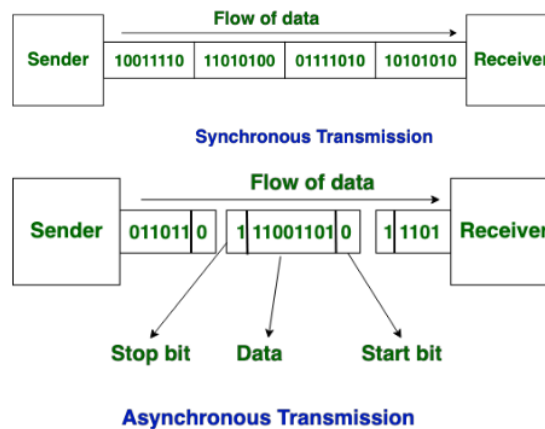


Fig. 4 Synchronous Vs Asynchronous Transmission [9]

A higher efficiency is found in this transmission type as there are no parts of the transmitted message which are not the actual underlying transmission content, i.e., the start, end, error, or gap sections of the transmission.

b) Asynchronous (AS) – This type of transmission and by extension style of operation dispatch can be more economical than synchronous by allowing the sender to vary its send rate, should it need to do so. This allows for dynamic quality scaling on video streaming sites. It does not require synchronicity with the receiver and can therefore be more flexible.

AS can also handle lower quality transmission media, such as older communication lines which may be more prone to noise. Error handling can be encoded directly into the data stream to assist the receiver in determining further action.

For data transfer that does not require immediate interaction or confirmation throughout the sending process (such as sending an email from a client to the SMTP server, sending a chat message on various apps or submitting a form via a webpage), the data can be encapsulated in a less costly format in which the receiver does not require any or little response action.

The asynchronous concept can have very useful applications in other areas of computing, aside from networks. In programming spaces, asynchronous functions allow for methods to return values once they have loaded, for example an external API, without slowing down the overall application. Furthermore, asynchronous processors *can* solve issues with traditional synchronous computing [10], given that synchronous clock systems are limited by their overall clock cycle being forced to be synchronous.

The High Level Data Link Control protocol [11] is an example of a synchronous data link layer protocol, derivatives of which power many of the most common data link protocols in use today (such as the point to point protocol PPP and the IEEE 802.2 protocol [12])

c) Considerations – Thought should be taken by application and network developers regarding the types of communication required by

their apps. The fundamentals of their application and certain specific communications may require different transmission types, perhaps even to different locations or via a different media. For games, there may be an element of synchronicity between the client and server, but then the server to an additional system (perhaps a chat function) could be asynchronous. Furthermore, a single player game may have largely asynchronous connectivity to the game server, however, to receive updates through TCP, this is synchronous [13].

4. Interoperability

Interoperability is the ability for a given system to work on multiple different platforms, organisations, or users, without the need to configure them differently for each. This can include all manner of actions from data transfer to policy application [14]. Generally, these come in the form of:

- Organizational Interoperability [15] – The ability for organizations to conform to common standards, processes and policies between them. This may be in the form of healthcare [16] in commonality between different areas and treatment types or organizations collaborating with each other or even extending that into the public sector.
- Syntactic Interoperability [17] – The case of language syntax and structure being uniform across different systems. This can bridge a logical gap between communicating machines, e.g., two computers with different operating system types and network cards, being able to both handle the data. Furthermore, a language like SQL or format like JSON are syntactically interoperable, as they can be used by many different systems.
- Semantic Interoperability – Similarly to Syntactic Interoperability, semantic interoperability allows for applications to exchange data between each other as part of a larger system. The difference between them is that the structure and use of the data throughout the system is uniform.

For software developers, interoperability takes form in a number of ways:

- Application Communication – Can an application being developed handle communications and send communications to **any** kind of operating system? An interoperable application is able to talk to a wide range of different system configurations.
- Data formatting – Using syntactically interoperable formats such as XML or JSON to pass information between languages.
- The use of communication frameworks that exist independently of platform or application, such as REST [18]
- Endianness – The order of byte storage in memory between different operating systems, architectures and device types can cause inaccuracies and confusion when transferring data across a network and accessing it on a different system. Big Endianness being the usage for network protocols and Little Endianness for many common processor architectures [19].
- Language choice – application development in languages which are interoperable between systems mean they can be easily run on any architecture. Languages such as Java can

do this as the java compiler provides bytecode which can be run on any platform which has the Java Virtual Machine (the version of which is of course platform dependant).

In the modern world of the Internet of Things, more and more devices are becoming available for a variety of uses. A large issue noted in the development of these devices is the lack of a standardised protocol for these devices to communicate. The Connectivity Standards Alliance [20] is an example of a modern collection of organisations driving to create a new interoperable standard for all IoT devices, irrelevant of who created them. The new protocol is called Matter, which allows any 'Matter Certified' device to communicate using it [21].

One of the biggest cases for interoperability is within telecommunications, and how the large swathe of organisations and technical systems must be able to communicate. In the UK, this is governed largely by Ofcom [22], the governmental regulator for communication systems. The direct interoperability of telecommunications providers is through NICC Standards LTD [23], which mediate between organizations the technical specifications for their systems to be interoperable. The extension of systems like these extends to the entire planet, through the internet in the abstractions of the OSI model to various connectivity types (cable, microwave, satellite etc).

5. References

- [W. Eddy, "RFC 9293," MITI Systems, West Springfield, 2022.
1
]
- [K. C, THE TCP/IP GUIDE, San Francisco: No Starch Press,
2 2005.
]
- [C. Kozierok, "TCP Connection Termination Procedure," in *THE
3 TCP/IP GUIDE*, San Francisco, No Starch Press, 2005, p. 764.
]
- [CompTIA, "Identifying Denial of Service and Distributed Denial
4 of Service," in *CompTIA Security+ Study Guide*, Indianapolis,
] Wiley, 2018, pp. 297-298.
- [CloudFlare, "What is MTU (maximum transmission unit)?,"
5 [Online]. Available: [https://www.cloudflare.com/en-
\] gb/learning/network-layer/what-is-mtu/](https://www.cloudflare.com/en-gb/learning/network-layer/what-is-mtu/). [Accessed 25 03 2023].
- [C. Kozierok, "IP Message Fragmentation Process," in *THE
6 TCP/IP Guide*, San Francisco, No Starch Press, 2005, p. 344.
]
- [R. Hinden and D. S, "RFC 2460," The Internet Society, 1998.
7
]

[Amazon Web Services , "Network maximum transmission unit
8 (MTU) for your EC2 instance," 2023. [Online]. Available:
] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/network_mtu.html. [Accessed 24 03 2023].

[MKS075, "Difference between Synchronous and Asynchronous
9 Transmission," GeeksForGeeks, [Online]. Available:
] <https://www.geeksforgeeks.org/difference-between-synchronous-and-asynchronous-transmission/>. [Accessed 26 03 2023].

[N. Paver, "The Design and Implementation of an Asynchronous
1 Microprocessor," University of Manchester Department of
0 Computer Science, Manchester, 1994.
]

[C. Kozierok, "Point to Point Overview and Fundamentals," in
1 *THE TCP/IP Guide*, San Francisco, No Starch Press, 2005, p.
1 144.
]

[T. Lammle, "The Open Systems Interconnection Specification,"
1 in *CompTIA N+ Study Guide*, Indianapolis, Wiley, 2015, p. 44.
2
]

[T. Kelly, "Synchronous or Asynchronous Gameplay
1 [Definitions]," [Online]. Available:
3 [https://www.whatgamesare.com/2011/08/synchronous-or-](https://www.whatgamesare.com/2011/08/synchronous-or-asynchronous-definitions.html)
] [asynchronous-definitions.html](https://www.whatgamesare.com/2011/08/synchronous-or-asynchronous-definitions.html). [Accessed 26 03 2023].

[TechTarget, "interoperability," TechTarget, [Online]. Available:
1 [https://12ft.io/proxy?q=https%3A%2F%2Fwww.techtarget.com](https://12ft.io/proxy?q=https%3A%2F%2Fwww.techtarget.com%2Fsearcharchitecture%2Fdefinition%2Finteroperability)
4 [%2Fsearcharchitecture%2Fdefinition%2Finteroperability](https://12ft.io/proxy?q=https%3A%2F%2Fwww.techtarget.com%2Fsearcharchitecture%2Fdefinition%2Finteroperability).
] [Accessed 29 03 2023].

[V. Margariti and T. Stamati, "A holistic model for assessing
1 organizational interoperability in public administration,"
5 *Government Information Quarterly*, vol. 39, no. 3, 2022.
]

[NHS England, "FHIR (Fast Healthcare Interoperability
1 Resources)," 11 03 2022. [Online]. Available:
6 [https://digital.nhs.uk/services/fhir-](https://digital.nhs.uk/services/fhir-api#:~:text=Fast%20Healthcare%20Interoperability%20Resources%20(FHIR)%20is%20the%20global%20industry%20standard,information%20can%20be%20shared%20quickly..)
] [apis#:~:text=Fast%20Healthcare%20Interoperability%20Resources%20\(FHIR\)%20is%20the%20global%20industry%20standard,information%20can%20be%20shared%20quickly..](https://digital.nhs.uk/services/fhir-api#:~:text=Fast%20Healthcare%20Interoperability%20Resources%20(FHIR)%20is%20the%20global%20industry%20standard,information%20can%20be%20shared%20quickly..) [Accessed 28
03 2023].

[M. Hosseini and B. Dixon, "Chapter 8 - Syntactic Interoperability
1 and the Role of Standards," in *Health Information Exchange*,
7 Elsevier , 2016, pp. 123-136.
]

[R. Fielding, "Architectural Styles and the Design of Network-
1 based Software Architectures," University of California, Los
8 Angeles, 2000.
]

[Baeldung, "Big Endian Vs. Little Endian," Baeldung, 10 12 2022.
1 [Online]. Available: [https://www.baeldung.com/cs/big-endian-vs-](https://www.baeldung.com/cs/big-endian-vs-little-endian)
9 [little-endian](https://www.baeldung.com/cs/big-endian-vs-little-endian). [Accessed 29 03 2023].
]

[CSA, "The Alliance is the Foundation and Future of the IoT,"
2 2023. [Online]. Available: <https://csa-iot.org/about/>. [Accessed 29
0 03 2023].
]

[O. Source, "connectedhomeip," [Online]. Available:
2 <https://github.com/project-chip/connectedhomeip>. [Accessed 29
1 03 2023].
]

[Ofcom, "Making Communications Work for Everyone," Ofcom,
2 [Online]. Available: <https://www.ofcom.org.uk/home>. [Accessed 29
2 01 04 2023].
]

[NICC Standards LTD, "The Home of telecommunications
2 network and service interoperability standards," 2023. [Online].
3 Available: <https://niccstandards.org.uk/>. [Accessed 01 04 2023].
]

ACCA6028 Task 2 – Implementation of a networked multiplayer game (BattleDinghys)

Oliver Bird

Apprentice Software Engineer – BT, Data Services

Applicable word count - 1030

1910716@student.uwtsd.ac.uk

1) Game Features

The game (dubbed BattleDinghys) is a crude implementation of the classic childhood game *Battleships*. This is a turn-based strategy* game in which two opposing admirals command their ships to fire on an enemy fleet until all of the opponents ships are sunk. The developed application comprises of a game client to be used via a player and a game server. The game is configured to run on LAN, however, connectivity permitting, could be hosted, and connected to over WAN.

The game functionality comprises of 2 players per game. The game server is currently limited to 1 game at a time, however, can be easily adapted to have more games concurrently via dynamically creating a new game each time a client connects, unless there is a non-full game already. Each connecting client is threaded however, and each references the same game instantiated class concurrently. This design is what would make it *easy* to implement many concurrent games.

The implementation has simplified some elements of the original game (hence the name BattleDinghys) as all vessels within the game comprise a single square, not multiple as in the original game. Modifications to the original were designed (such as variable weapon types, fighter planes from an aircraft carrier and ship protection methods), however were not able to be implemented.

Connectivity between the client and server is implemented using the Win32 POSIX library with each connection being made used the TCP protocol. TCP was chosen as to ensure that no positional data was lost whilst the game is being played. Furthermore, the contents of messages sent between the systems drives certain conditions for the game, for example deciding if the game is won. If these messages were to be lost or modified, it could result in incorrect game processes firing, therefore ruining the game.

The game has been created using knowledge learned within the ACCA6028 module, and the examples provided by lecturer Dr Tim Bashford. Cases in which code was not written by the author, or has been modified / used as a base, have been noted in the .cpp files showing the method implementations. Some have also been included at the class level should an entire class have been used. The network core uses the implementation from the example provided in class.

The game uses a generic messaging system to communicate between the client and server, with various types of messages

controlling various game actions. The most common of these are to send strings (the AcceptMessage class has been used to handle this, as a recreation of this class with a different declaration but identical content felt unnecessary), send the boards from server to their clients, and to send coordinates of attacks from client to server.

Effort has been made to reduce the chance of foul play by players. Only information directly relating to each player has been sent, and the boards / views have been separated and are not sent to players in any capacity in which they shouldn't be.

2) Design Choices

Certain design choices have been made to both imitate a real-life naval conflict and to simplify the development process.

The game has been designed in a 2 stage turn based format:

- The first stage is the overall turn of the game, which exists on the server. This means that both players firing choices are enacted synchronously, as they would be in real life. Ships are able to fire on each other in real time (at least from this stages perspective), ultimately meaning that there is a chance that ships sink each other at the same time, resulting in a draw.

Each cycle of the server contains two player turns, the states of which are evaluated successively, but evaluated as one overall turn.

- The second cycle is the players individual actions. These are split into a firing player and an awaiting player. One player has the option to fire at a coordinate while the other waits, which is then reversed to complete the first stage.

This implementation choice results in the feeling of the original *Battleships* experience whilst having the advantage of a more accurate real-world conflict.

The majority of the application is built using pointers to reduce the overall memory usage, in which many function calls are made using references to pointers.

3) Best Practice

Best practice is an extremely large field, much of which is beyond the scope of this application (such as the complete notes provided by Microsoft [1]). In general, however, the following practices have been used:

- Polymorphism – this has been used to construct common classes, in this case messages / communications between the client and server at the application layer.
- Documentation – The application is heavily documented throughout following the Javadoc syntax [2]. This provides a large amount of readability at the function level. Furthermore, the code is commented where necessary to further the readability of the application.
- Error handling – The application does provide some level of data authenticity and error protection, certainly in the handling of user information. Users are restricted from attempting to access coordinates outside of the board at the client layer, meaning no damaging requests are sent to the server.
- Memory management – Many of the created objects are deleted from memory once they have served their purpose, as C++ does not provide a default garbage collector.

4) Limitations

While not explicitly required, the author believes some limitations should be noted:

The error handling within the application does not handle changes to network states (such as disconnects), being client initiated or accidental. To remediate a disconnect, a new game must be started via a server restart.

The application is limited by the front-end console, meaning there are a plethora of potential issues when interacting with it. The most common of these is regarding the input buffer. Care must be taken when typing values into the console as it will constantly read lines in, even when not prompted to do so.

The client applications will not boot without the server being available, as there is no logic to handle this. Furthermore, should the server become unstable or shutdown, the client applications will crash and not terminate cleanly.

Only a single game can be played on the server at one time. A more concurrent implementation was not found within an appropriate timeframe.

6) References

- [1] Microsoft, "C++ language documentation," Microsoft, Seattle, 2023.
- [2] Oracle, "How to Write Doc Comments for the Javadoc Tool," 2023. [Online]. Available: <https://www.oracle.com/uk/technical-resources/articles/java/javadoc-tool.html>. [Accessed 25 April 2023].

1910716 ACCA6028 Final Submission

GRADEMARK REPORT

FINAL GRADE

79/100

GENERAL COMMENTS

Instructor

A very good effort, Ollie. A nicely written report covering the key concepts, well-sourced and presented.

An excellent effort at the implementation, showing a clear understanding both of your work and its limitations. Really well done.

Part 1 (report): 78

Part 2 (implementation): 80

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8
