Student Name: Cameron Dawes

Student Number: 22019386

Critical Review

GDEV50037 Multiplayer Games Programming

# Introduction

This networking project contains a lot of different components that allow it to function as intended. Some of the most important components are the transfer protocol, packets and encryption. The intriguing thing with networking is it has such a wide range of applications and things within the field that are interlinked, it has an almost endless amount of complexity as you can create better protocols and encryption methods and more secure methods of setting up secure lanes of communication along with that you can always compress the data smaller using different compression algorithms. As we go through this critical review, I hope to explain the implemented networking features in an understandable manor and discuss my implementation and why or why not it was the most suitable for my use case.

# Critical Review

In the project we use the TCP protocol which allows us to send data over the network using packets. This protocol allows for reliable and ordered communication between two computers, this method of communication allows for the developer to communicate as if reading and writing to a file. UDP on the other hand breaks the data we intend to send into datagrams which are small chunks of a larger piece of data and directs it at a single IP but sends it through multiple different paths and is not guaranteed to arrive in order, at the same time or at all. The receiving device then rebuilds the sent data using the datagrams, it checks that these received datagrams are indeed correct by using checksums which verifies the integrity of the received data and allows the receiving device to request the data again whereas TCP must have the receiver realise they haven’t received a packet then send an acknowledgment to the server where it sends it again. On top of that, the speeds of the protocols vary drastically as TCP uses a single network route and can sometimes buffer the sent data when the packets are too small as it waits for more packets to be sent so it can clump it together, on the other hand UDP sends all the small datagrams to one IP address via multiple routes meaning it will arrive faster. So, when considering what protocol to use for a game that requires synchronisation UDP is the better choice as it not only allows for faster data transfer as it does not contain the possibility to delay the packets for efficiency.  [Glenn Fiedler](https://gafferongames.com/)(2008) states that “TCP was simply not designed with this in mind.” as he mentions many times in his “UDP vs TCP” article that TCP is very slow compared to UDP. article which in addition he states, “TCP abstracts everything so it looks like you are reading and writing to a file, hiding all complexities of packets and unreliability from you.” this complexity he mentions will also affect the time taken before the packets further supporting my conclusion.

The project uses asymmetric encryption and uses the commonly found RSA algorithm to encrypt it. Asymmetric encryption works by both the devices having their own public and private key, they send their public key to the other user like a handshake of sorts. They use the other devices public key to encrypt the data they want to send to them. This data then can not be decrypted without the others private key. This system is very secure as it is resistant to man in the middle attacks as the attacker does not have the private keys so even if they get a hold of the data, they will have to brute force it which takes a lot of resources. Asymmetric does have its weaknesses though and Stretch (2010) describes them as such ‘Compared to symmetric encryption, asymmetric encryption imposes a high computational burden, and tends to be much slower. Thus, it isn't typically employed to protect payload data. Instead, its major strength is its ability to establish a secure channel over a nonsecure medium (for example, the Internet).’ so even with the glaring negatives to Asymmetric encryption it has its uses. An alternative to asymmetric encryption is symmetric encryption, this encryption method works by sharing a single key between the two devices which is used to both encrypt and decrypt the sent data. This method of encryption is very vulnerable to man in the middle attacks as if they get the key they can encrypt and decrypt whatever information they want without either party knowing what’s happening. There is one benefit to symmetric encryption and that’s when both devices have the encryption key preset on the system this allows for no transmission of the key eliminating the encryption methods main weakness, but this will only really work on private networks which already have less risk on by default. For our use our method of encryption is adequate though our initial exchange of keys is not encrypted which is a weak point of our system but there is a way around it by using a verified middle server to send the keys through so they cannot be connected to both devices wanting to send data to one another.

Synchronization is a large part of multiplayer games and that’s why it must be done properly so it doesn’t become out of sync. I approached the synchronization of paddles slightly differently then on the tutorials as I send over the players paddle key presses instead of the paddles position this allows me to send less packets overall with the same experience. Initially I struggled with synchronizing them as the more I moved the more out of sync they got which took me quite a while to realise was a problem. This issue is very well put into words by Paul Bettner and Mark Terrano which describe how they perceive finding out of sync errors ‘The difficulty with finding out-of-sync errors is that very subtle differences would multiply over time. A deer slightly out of alignment when the random map was created would forage slightly differently - and minutes later a villager would path a tiny bit off or miss with his spear and take home no meat.’. Furthermore, the more complex the data you transfer the higher the likely hood there will be data out of sync, this may not even be directly caused by you as a rounding error can easily cause data to become out of sync. Even worse is that a missing packet could cause issues 100s of packets down the line and the nature of networking it isn’t always possible to get the missing packet before the following one arrives stopping you from resolving the missing packet in your system. I faced issues with synchronization but solved them by implementing a dirty flag which gives time for the paddle position to be updated correctly as it forces it to stop moving to allow the system to translate its position before it continues its path. As well as that, I updated the position of the paddle every time a movement key was pushed down or let go which meant that it started in the position it should be and after moving even if it overshoots it would be brought back to the correct finishing position allowing for it to continue moving from the correct location.

# Conclusion

In conclusion, my experience with the project was very erratic and unpredictable as there was a lot of times where I was struggling to figure out certain issues simple or complex or overlooked simple fixes and/or implementations of features and alternatively there were times where everything worked first try. I believe I would benefit from taking time to plan the implementation of a feature before I start adding it as that would allow for me to iron out any misunderstandings about the feature which would allow for a much smoother programming experience and would probably allow for me to familiarise myself with the features and content more so than I have preplanning the implementation would also mean the chance of forgetting or overlooking something simple and causing wasted time spent trying to debug. I believe the features implemented to the project were the right fit except for the data transfer protocol as for our use case we would be better off using UDP. Looking at the quality of the code in the program there is a clear difference between the networking code and the game code, the game code is a lot less refined than the networking code and lacks any complexity as I have felt overwhelmed trying to get the game working. In retrospect I would like to have 3 instances of the game where the server has its own instance, and the players send and receive data exclusively from the server’s game instance and not receiving data forwarded on by the server from the other player. Not only would this allow for better synchronization for the players I believe after setup it would be a better programming experience then as it is now. The reason I believe this is looking at my problems with synchronization with starting the game and moving the ball if the server had its own instance, it could access real object making more complex features possible. Having the server be where the actual game is running it would allow for implementation of a predictive algorithm for the game objects as we have direct access.

# Bibliography

Glenn Fielder

1st October 2008

<https://gafferongames.com/post/udp_vs_tcp/>

Stretch

23rd November 2010

<https://packetlife.net/blog/2010/nov/23/symmetric-asymmetric-encryption-hashing/>

Paul Bettner, Mark Terrano

22nd March 2001

1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond

Joshua Glazer, Sanjay Madhav

30th October 2015

Multiplayer Game Programming: Architecting Networked Games