Lab 9: Networking Challenges

During tackling any kind of networking application, the initial question will be which type of connection to use: TCP or UDP? While UDP is often more flexible and fats, TCP is more secure as packets can be lost when using UDP. Ultimately though, in the context of creating a game, UDP is the better option. Speed is everything when it comes to communication in games, and oftentimes games will be sending information back and forth so often that it should more than make up for any loss in data from packets not being sent properly.

The next question will be the type of communication between game instances: will we use dedicated servers or peer-to-peer(P2P)? P2P is far less costly to implement as it fully relies on the connection between players, either by having one player act as a host, are the shared connection between players works non-stop to keep the game going. On the other hand, the use of a dedicated server is far more secure as P2P being player based means the connection can be interfered with, allowing for rampant cheating, or even safety risks. Additionally, P2P requires a decent connection between all connected players to function properly if a host isn’t being used, while a server allows for more wiggle room. Servers also allow for much higher player numbers in games than P2P. In the end, it depends on the type of game being made. Fighting games can get by using P2P for example, as there will be a limited player number at any given time, and the simplicity of only having two characters on screen making decisions will lessen the amount of information being passed. On the other hand, military sim games like ARMA all but require servers to function on account of the large maps and high player numbers.

Let’s say that a dedicated server is being used, another challenge that can be encountered is the simple maintenance of the server. Should anything occur and the server is rendered inoperable, our game is dead in the water as no players can interact with the networking features of the game. This means that regular checks and maintenance are essential to our game’s health, as well as backup servers if possible, preferably offsite in case something like a power outage occurs. But this backup server will also require maintenance as a result. Not to mention that all of this maintenance will require manpower to keep running. This means recruiting team members to exclusively work on the servers.

On that note, there is also the issue of regional servers. The further distance that information must travel, the slower the response and update on screen during gameplay. For example, to ensure a steady connection and a smooth gaming experience, a player in Ireland wouldn’t want to connect with a player in Japan. To that end, locking servers to particular regions is necessary. Having a server in Europe will ensure that players in Europe will be able to connect with each other and have a smooth gameplay experience. But this creates its own set of problems. For starters, there is the cost of either setting up or renting servers in other regions, as well as hiring workers in these regions to then maintain these servers. Furthermore, let’s say the company I work at is based in an English-speaking country, if I were to try and set up a server in Asia, I’d have to find and get in contact with someone in that region, whose primary language may not be English, making communication between the company and the server worker overseas difficult.

As well as that, circling back to an earlier point, there are the security risks involved with networking. In order to ensure the safety of both the company’s servers as well as the players themselves, making sure that all connections to with servers and players are secure is essential. In cases were the connections are unsecure, at best cheating will be repent in the game, and at worst the information of players or the company may be accessible and as a result at great risk. A hacker may be able to discern the IP addresses of other players, or render the servers inoperable.

One last thing I would have to consider is the netcode used by the game: delay-based or rollback netcode. Netcode is a blanket term largely associated with the syncing of information between clients and servers. This is an issue in all online games, especially games that require fast reactions like FPS games or fighting games. Delay-based netcode refers to a solution when the inputs of a player arrive late the game delays the inputs of the local player at the same time to synchronise the two inputs simultaneously. While a simple solution, this delay can be infuriating to the player base, especially in games with high action a s any delay could very well result in a loss. The modern alternative that’s being adopted more and more is rollback netcode. This system immediately runs the inputs of the local player (so that they are not delayed as with delay-based netcode), as if it were an offline game, and predicts the inputs of the remote player or players instead of waiting for them (assuming they will make the same input as the one in the previous tick). Once these remote inputs arrive, the game can act in two ways: if the prediction is correct, the game continues as-is, in a totally continuous way; if the prediction was incorrect, the game state is reverted and gameplay continues from the corrected state, seen as a "jump" to the other player. The best solution tends to be a hybrid of these two solutions, to ensure that the game doesn’t “jump” constantly in games with a bad connection, but still largely feels like an in-person gameplay session.

Not every challenge in networking tends to have an easy solution, many of the costs of using a server just can’t be ignored and come down to the games budget. But in this essay, I’ve tried to provide solutions to what problems I could, such as the challenge of netcode usage. At the end of the day, this all comes down to what kind of game the company is trying to make, and their budget.