Tom Chambers

# Investigation into Net-code Solutions of Multiplayer Games

## The Problem

Latency in video games is a significant issue that developers must tackle to make competent, networked games. However, over any network a perfect solution, one that eradicates latency, is impossible. Current methods exist to mitigate or hide the effects of latency, but they all come with their own drawbacks. My artefact investigates these impacts on user experience and their effectiveness at reducing the effects of latency.

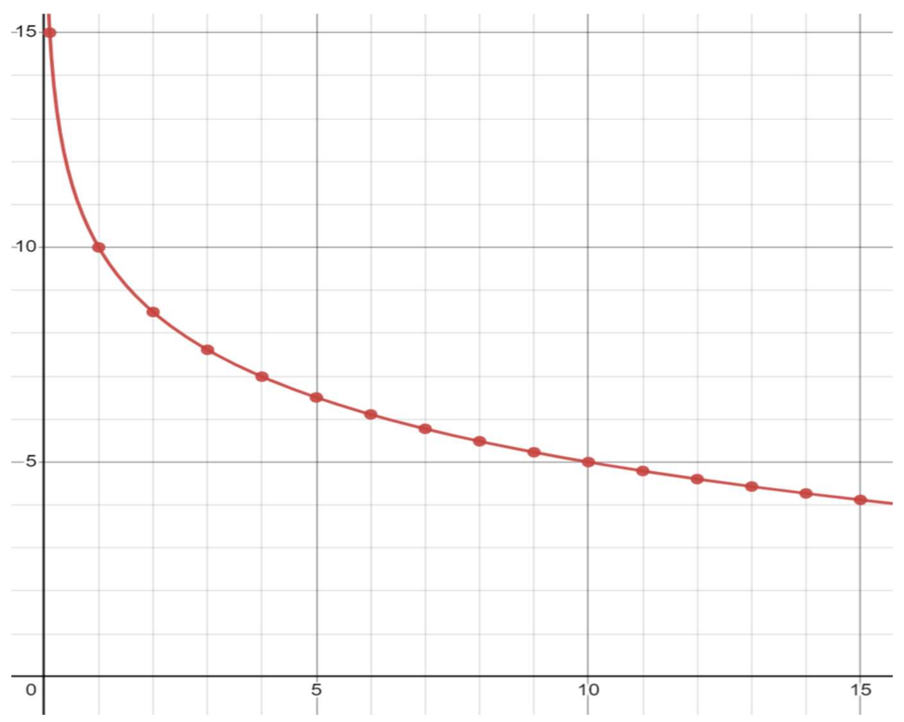
## Solution

My research is encapsulated by a top-down 2D Unity game using Unity's Networking package, where the objective for players is to gain points from eating blobs and then eating players once they are larger. There are two different networking solutions implemented, Client-Authoritative Basic Lockstep and Sever Authoritative Deterministic Lockstep.

## Game Design

Using Unity's networking package, Remote Procedure Calls (RPCs) are used to pass data to the server. Every object in the game is a Networking Object, allowing them to interact with net code.

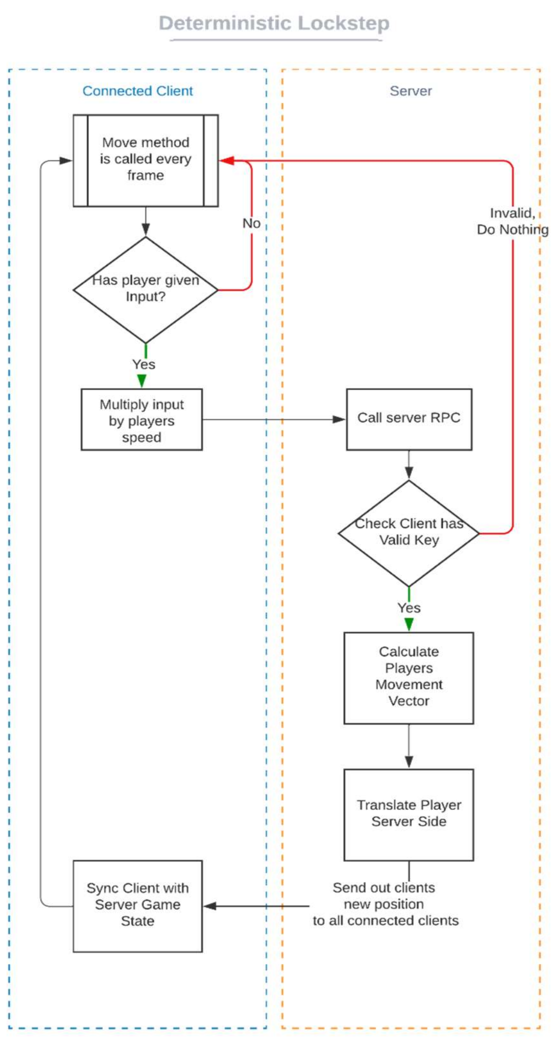
Points blobs are dynamically spawned around the environment, giving players a way to increase points before attacking each other. The effective points gain of these goes down the more you eat, encouraging PvP. The speed of players decreases as their points increase, due to the logarithmic function, y = -5 log(x) + 10 with the speed being bounded to <= 15.

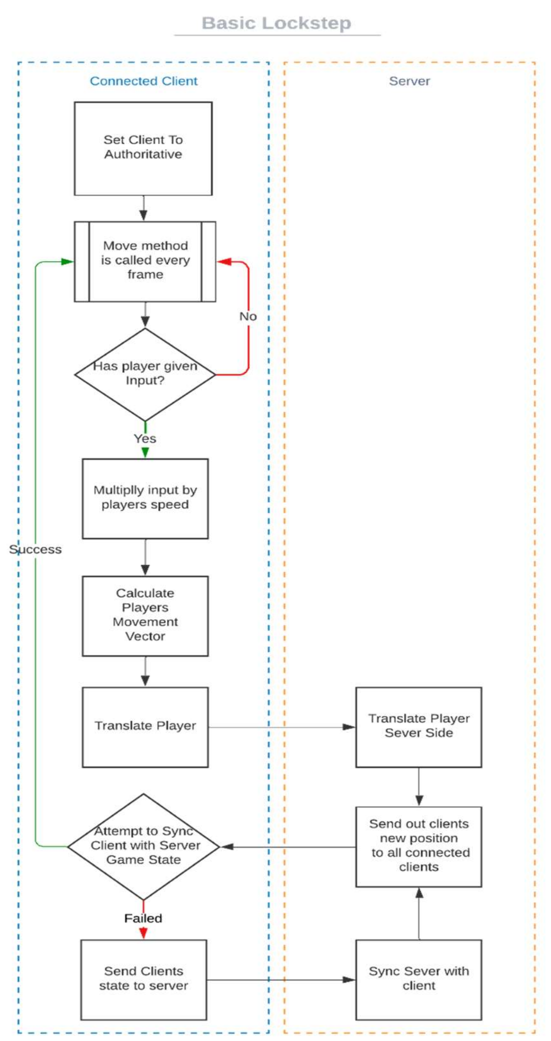


## Client-Authoritative Basic Lockstep

This method allows the client to send their game state to the player, by overriding the server's authoritative flag. Each client can write to their transform, which is then synced with the server and all other clients.

This approach benefits the client, who can instantly receive and act upon any input. However, other players might experience delays and larger amounts of data sent to the server, especially on poor bandwidth connections. The biggest issue is that this method violates the game's security, as the client can manipulate their data before synchronization and cheat in the game.

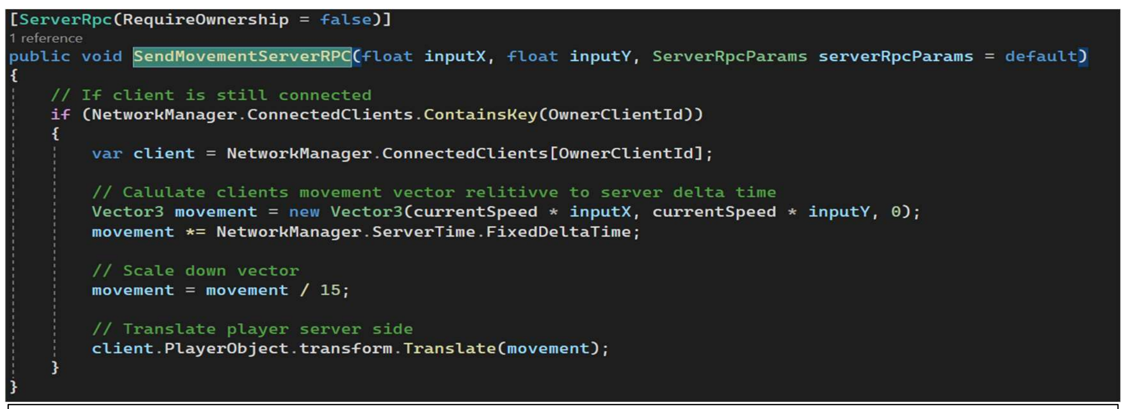

Deterministic Lockstep
Connected Client
Server
Move method is called every frame
No
Invalid, Do Nothing
Has player given Input?
Yes
Multiply input by players speed
Call server RPC
Check Client has Valid Key
Yes
Calculate Players Movement Vector
Translate Player Server Side
Send out clients new position to all connected clients
Sync Client with Server Game State


Basic Lockstep
Connected Client
Server
Set Client To Authoritative
Move method is called every frame
No
Has player given Input?
Yes
Multiplly input by players speed
Success
Calculate Players Movement Vector
Translate Player
Translate Player Sever Side
Send out clients new position to all connected clients
Attempt to Sync Client with Server Game State
Failed
Send Clients state to server
Sync Sever with client

## Server-Authoritative Deterministic Lockstep

This method relies on the server receiving the client's movement intent by RPC, rather than their transformation. There are 4 Server RPCs written. SendMovementSeverRPC() syncs the transformation of the players. To save bandwidth, only two floats are sent.

This method has noticeable latency for all connected players but does not let the client write directly to their transform, protecting the legitimacy of the game. It also has a small bandwidth footprint, so a smaller Round Trip Time (RTT) total is experienced for the client.



```
[ServerRpc(RequireOwnership = false)]
1 reference
public void SendMovementServerRPC(float inputX, float inputY, ServerRpcParams serverRpcParams = default)
{
    // If client is still connected
    if (NetworkManager.ConnectedClients.ContainsKey(OwnerClientId))
    {
        var client = NetworkManager.ConnectedClients[OwnerClientId];

        // Calulate clients movement vector relitivve to server delta time
        Vector3 movement = new Vector3(currentSpeed * inputX, currentSpeed * inputY, 0);
        movement *= NetworkManager.ServerTime.FixedDeltaTime;

        // Scale down vector
        movement = movement / 15;

        // Translate player server side
        client.PlayerObject.transform.Translate(movement);
    }
}
```

## Outcomes, Issues and Next Steps

Out of the two implementations, Basic Lockstep is the most responsive. As soon as there is input the client moves on their screen. However, there is still a delay for the other connected clients, meaning that the player moves on their machine first, before the others, leading to a synchronisation issue. This would be best suited for a game where players move in turns, such as a strategy game.

With Deterministic Lockstep, there is a delay for all clients when input is given, however, all clients move at the same time. Since the server changes the client's state through RPCs, clients avoid the synchronisation issue. This would be best suited for games where all players take action at the same time as each other.

## Outcomes, Issues and Next Steps

I can conclude that netcode architecture is vital to the feel of multiplayer games when clients are distributed across a network. Developers should be conscious of gameplay and security issues when deciding on net-coding methods.

If I expanded the project, testing with multiple connected clients would have allowed investigation into the synchronisation issue in greater depth, possibly leading to new solutions. Unity's networking package is also still lacking features, blockading the development of prediction and rollback. With a new project, I would move away from the package, allowing more control over packet data and timings, then implementing rollback features.