CS179I Final Report - Expanding upon Host-Client Topology

Justin Dang
jdang065@ucr.edu
University of California, Riverside

Video Demo: https://youtu.be/V47U0kG6qjo

1 Introduction

The Host-Client topology lowers costs of networking in multiplayer games by making the host-client topology more flexible. Servers are expensive, an average of $\$80/_{month}$ for 500 concurrent users(CCU) leads certain genres of multiplayer games to settle for host-client topology and its risks[1]. Host-Client topologies risk clients cheating and accessing another clients information(IP, Location, misc).

With my implementation, Host-Client topology become more accessible to other genres of games(RTS, Co-op, MMO instances) and provide a semi-authoritative Host-Client topology through a "majority vote".

The majority vote is inspired by the Byzantine Problem(as suggested by Professor Tan), giving the majority of player authority over the game. This solution relies on the majority being those of good faith and helps with data integrity. Scripts and user-side cheats/bots are unaffected since they follow the rules of the game.

While host-client topology already has a similar structure, the new system explores having multiple hosts/simulations for a single instance, communicating and validating. Normally there would be a single host simulating the game for all clients.

2 Overview

This project is an hands-on demo of the "majority vote" in action. The majority vote is used in this case as which state(An integer value) players should adopt. The majority vote is handled by a dataManager, which stores the synchronized states. Simply counting each state is how the majority vote is calculated for this instance.

The system works on a single machine using Unity's Netcode(this project may work on a local network, but I can not to test). RPC's sync player positions and NetworkVariables synchronize data across players. To improve UI, I use shaders and objects to signify each different player. I make the data accessible by having it displayed above each respective player using a dataManager.

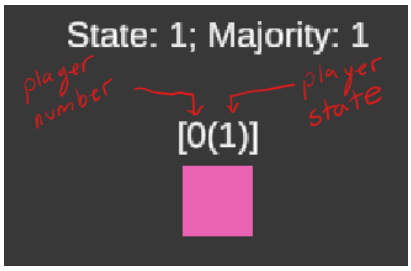

Figure 1. Information display for each player

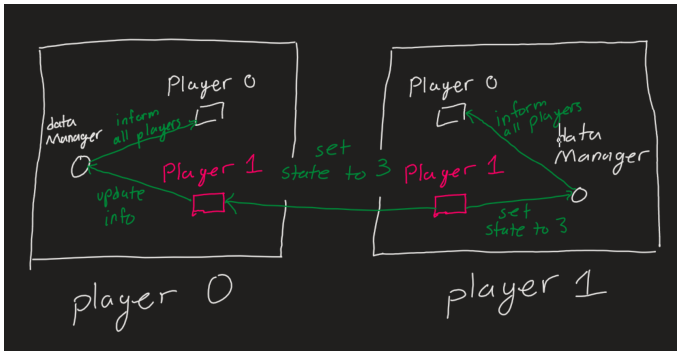

Figure 2. How data is synchronized and Handled

In Figure 1, the brackets represents the array of clients currently connected and their respective data. As more players connect the array will increase in order of connection, respective to each client.

In Figure 2, this example shows how data is handled and synchronized for two players, this example is the same for more players. Each player here has their own copy of the simulation. They control their own state, which will reflect in every other simulation this player is a part of. Due to the nature of Unity Netcode, players can only modify their own variables. So player 0 can not modify player 1's state in their own session and inform player 1's session of this change.

I initially wanted to have multiple hosts connect together, but I opted to have a single host with multiple clients. This was because it would not affect how this system would operate and would save me development time. I offset this decision by making each player connected simply run their own simulation(like a shared Topology[2]).

CS204 final project, Fall, 2023, UCR

3 Detailed Implementation

I developed this project using Unity and the Unity Networking libraries. I use the libraries to develop the network I use. Everything else was made by me.

Each player contains a NetworkVariable, a data container for synchronizing primitive data across players from Unity's Netcode. It is a state(integer number) that synchronizes with all other copies of their player that are in another player's session. Refering to Figure 2, when player 1 connects to the game, a copy of them appears in player 0's session with their information synchronized through the NetworkVariable, vice versa for player 0 appearing in player 1's session.

The data manager is a dictionary containing a pair <player number(in order of connection), state>. Referring to Figure 2, it distributes the data to every player in the same session as the manager whenever there is an a state changes on any player. When a player changes their state, the NetworkVariable detects this and informs all instances(including self) of the player to inform their respective dataManager. Then the UI is then updated for all players in the same session as the dataManager.

As for UI, I use a location manager to move the players in the session. I simplify synchronization by using the host to control the locations of all players. When a player connects, it sends an RPC to the host to request it to give the player a position.

The color for each player done during runtime by using the xy position to pick a color in a color gradient.

4 Evaluation

This system works better as more players join. For games with 2 or less players, this system is irrelevant.

In the real world, this system would be used to allow primitive data(numbers and strings) to maintain integrity on the majority of players. It isolates the cheater's session and allows all players to detect if another player is cheating by checking their state/data against the majority state/data.

Applications outside of games consist of situations where users come together to agree on a particular set of data. If a user in this group decides to act in bad faith and sabotage the data, the majority will not be effected.

In conclusion, this system will help reduce costs in games/genres that do not need to maintain data persistently. But when data is required to be stored persistently, this system can act a mediator between the player's session and the server data. This means server wont have to dedicate itself to hosting the game and remain as a data storage.

5 Major Roadblocks and Lessons Learned

This was my first time working with Unity Netcode. SO I struggled to learn the interactions between NetworkVariable and RPC. Synchronizing data was a key struggle as well as understanding how code operates on different clients.

I learned that unity RPC work by calling the function on the copies of the player and within each session. I initially had each player clone track their own data in each session. I had difficulties getting data to be properly stored and created a dataManager that holds all the data. Each player clone modifies the values in dataManager.

Since each client had their own logic for where they should be placed, they would stack on top of each other. I fixed this by having the host track the position of all clients. I then ended up opting for a simpler solution that is non-scalable for which I use the playerID(0, 1, 2, ...).

A key takeaway from this project I learned is that there are many ways to go about solving a problem. Understanding core concepts and carefully planning helps reduce the difficulties faced. But it won't mean that a solution is impossible.

6 Artifacts

The code consists of 4 scripts(locationManager, dataManager, ClientNetwork, NetworkManagerUI). A shader is used to dynamically color each player as well.

a. locationManager: Tracks all possible locations and assigns one to each client, up to 6.
b. dataManager: Tracks all states and clients. Calculates the majority vote.
c. ClientNetwork: Controls input, UI, sending data to other clients, and setting a location for itself.
d. NetworkManagerUI: Shows the initial buttons for start and start host.

To run the project:

1. launch the executable once for every player you want to test(Up to 6 total windows).
2. On one window press start. This will create a host.
3. Press create host for every other window.
   - The player array won't be synchronized properly since the array updates as players join. New players won't have the data of previously joined players.
   - This will not cause any problems with the majority vote.
4. Freely change the state of each player using integers, no other value will be accepted.
5. When done just close the windows.

7 Conclusion and Future Work

This project is meant as a proof of concept. It can be expanded to handle more than just an integer and worthwhile benefits for minimal cost if any. It allows for the host-client topology to be more flexible. While not completely solving the issues for every situation, a large majority of games would find this topology useful and cost-effective.

The applications for this system are simple in hindsight. Although this means that large complex systems can integrate my host-client changes into their topologies to save costs where possible without giving up data integrity.

Future developments for this topology would be seem to expand more into other applications for this system and other ways to implement security for Host-Client topology other than a majority vote. Since majority vote only improve which genres of games can take advantage of Host-Client topology.

8 References

[1] https://support.unity.com/hc/en-us/articles/6821475035412-Billing-FAQ

[2] https://www.youtube.com/watch?v=IWi6iv7_sT4