RTS “Cloudstepping”

Matthias Harden
CSC 466 University of Victoria

Overview

Real Time Strategy (RTS) games haven’t had a ton of networking innovation since the creation of lockstepping. For the most part lockstepped RTS games haven’t required new networking strategies. This is still true today. However with the recent blow up of the battle royale genre, it left me thinking a massive player count RTS game would be really neat. There has never been a massively multiplayer RTS game such as starcraft or company of heroes. (At least that I am aware of.) While this may be in part due to design challenges, another important factor is the scalability of a lockstepped system. RTS games require hundreds to thousands of units for each player. You can imagine how adding a new player then leads to syncing massive amounts of information across players devices. For this paper it has two key points of focus. The first is my newly proposed state based streaming via a cloud gaming solution. The second topic of focus is the application of my state based streaming cloud gaming solution to the RTS genre which I have called cloudstepping.

Cloud Gaming

Before getting into my state based cloud gaming solution I’ll be giving a light overview of the history and general concepts within cloud gaming. This history and overview should make it much easier to understand my own project and proposition.

History of Cloud Gaming

When the idea of cloud gaming was first entering the gaming scene it was really exciting. Its main focus was to have a game running in the cloud on a powerful device. That way a user could play a graphically intensive game on their chromebook, or even a phone. The initial strategy for this was to create a video stream from the device in the cloud. Then the user's client would receive this stream and render the video frames. This meant that the user needed a gpu capable of rendering streamed video frames and no game to be downloaded on their device.

There were quite a few companies that started getting into this market however they all faced similar issues. The player quality of experience (QOE) was generally quite low. Streaming the game to the client is relatively expensive on the network. This led to high delay when playing games. High delay in a video game is disastrous for the QOE as the player loses control of their character. There has been a lot of effort to resolve this issue.[1] New articles continue to be published every year. One of the earliest solutions that was implemented was to use stream quality control mechanisms. This did improve the delay issues however it often came at the cost of the games visuals. Which again, lowered the player QOE.

Hybrid Cloud Gaming

The issues that cloud gaming faced created a sub branch of research. This was research focused on using resources on the client side. With the traditional cloud gaming solution the client was only responsible for rendering the streamed video frames. However this puts a tremendous amount of stress on the network. With Hybrid Cloud Gaming solutions they attempt to alleviate the network's workload by making use of the client's device resource power.

Asset Based HCG Solution

There is a specific solution that was recently proposed[2] that I wanted to cover specific aspects of. This is because my state based streaming solution uses a few similar mechanisms and concepts.

The asset based hybrid cloud gaming solution still has a device running the game in the cloud. However rather than streaming a sequence of video frames, the cloud streams assets to the user's device. The client device then renders the assets using their device’s graphical pipeline. The constructed frame is then displayed on the client screen. If the user's device is not able to construct the frame in time the solution still has the capability to stream video frames just like in a standard cloud gaming setup.

State Based Hybrid Cloud Gaming Solution

Now we come to my proposition for a state based hybrid cloud gaming solution. For this solution the game is simulated and running in the cloud. There are a few major assumptions that my solution makes that other HCG/CG solutions do not assume. First, the user's device is expected to be a relatively robust client. They will be rendering one hundred percent of the frames on their own device so they must have a device that can handle this. The second assumption is that the user has all assets on their device locally. In other words the game has been downloaded on the local device. While my solution uses many of the concepts from hybrid cloud gaming it is not trying to solve the same problems. My solution focuses on using the power of the cloud to network a game between players.

Game State

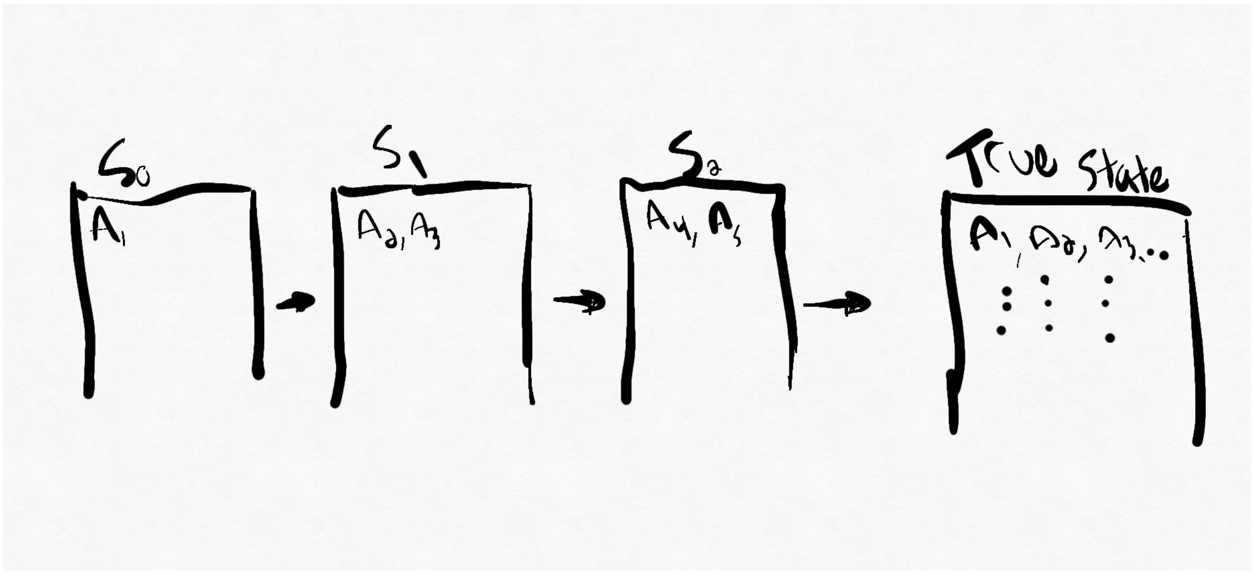
This solution relies on the concept of the game state. The game state tracks information of all objects within the game. For a standard RTS this could be units, buildings, resource nodes etc. My solution uses the cloud to maintain the game's state. This state is then streamed to the connected players in the form of state updates.

True State

The cloud stores a state which I have called the “true state” . This true state is the state that the game's logical operations occur in. For example if a new unit is built, it would be added to the true state. Any update to the game's state occurs in the true state. This true state can be thought of as the clouds state.

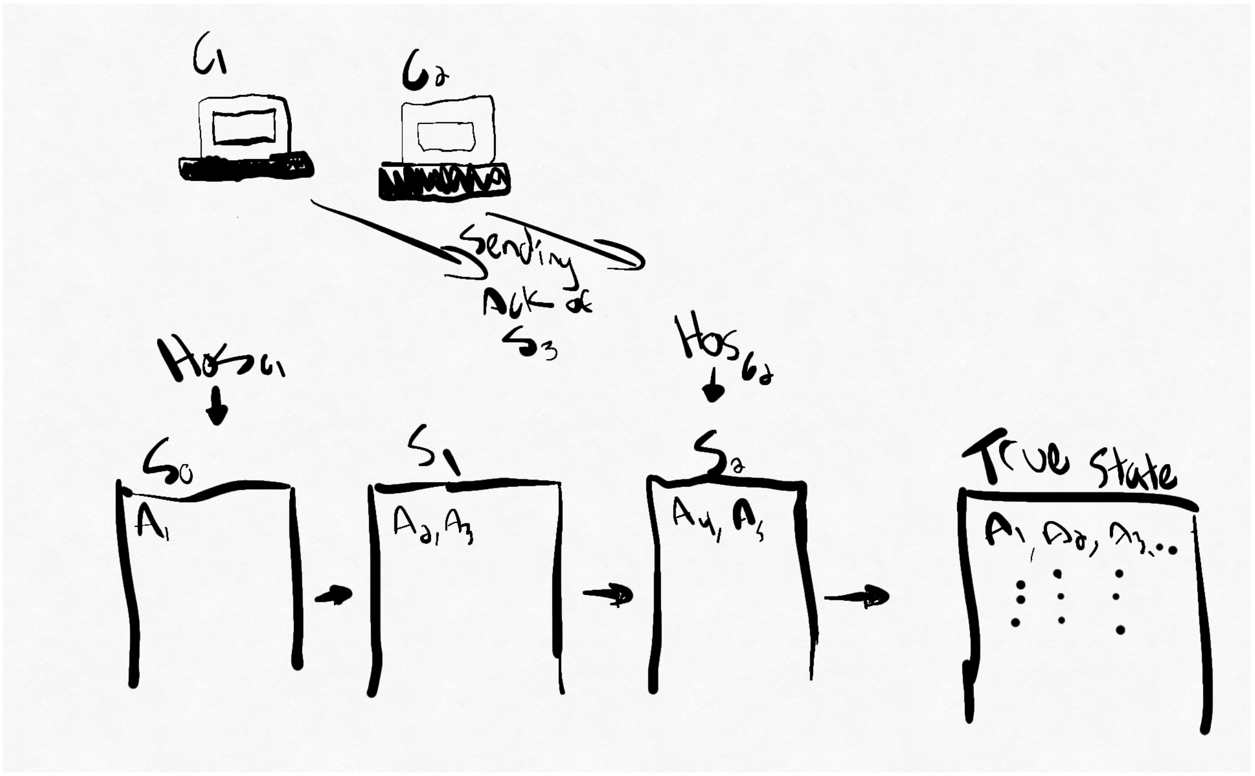
State Updates

The cloud maintains more states than the true state. Each time a lockstepped turn ends all actions are triggered. Once every action has affected the state we create a state with all changes made to the true state. This way we can track what changes need to be made to reach the true state. This leaves us with a state update representing the actions that occurred in a given turn. The state update is constructed from the true state during the execution of all actions that occurred for the turn. A state timeline is the reference to a sequence of these state updates that eventually lead to the true state.



Head of State

The cloud then also maintains a head of state or HOS for each connected client. The HOS points to the clients last known state update on the state timeline. When state updates are sent out to the clients from the cloud they are constructed by beginning at the HOS for that client. The state updates are then constructed by navigating across the state timeline from the clients HOS until the true state is reached. When the client receives a state update they will send back an acknowledgement of the received state to the cloud. (Which should, at any given time, be the true state.) When the cloud receives the clients acknowledgement that clients HOS is progressed up the state timeline to the acknowledged state. This allows us to decrease the amount of data that needs to be sent over the network by decreasing the state size and only sending the differences in state.



Client Side Rendering

Since the cloud is streaming the game state to the client the client has full responsibility for rendering frames. As opposed to a more typical cloud gaming solution where the cloud streams video frames and the client only needs to display them. With my state based streaming solution the client uses its own graphical pipeline to render each and every frame that is required. The client caches the state that is received from the cloud. This cache state is then used with the local assets by the graphics pipeline to render each frame. A frame would be composed of all state data such as where units stand, what their action is and even where the client's camera is located.

Syncing Clients

My proposed approach has a few major benefits for syncing the game between users. Compared to a standard lockstepped strategy where each user must simulate and maintain their own state, my solution takes advantage of the cloud's computational power to maintain a large shared state between all users. This state is then simply streamed to all users for rendering.

Shared States

As mentioned a standard lockstepped solution requires each client to maintain and simulate their own game state. This allowed inputs to be synced across devices. However with this strategy a very real issue occurs, the possibility of users states being out of sync. With my strategy all clients share a single state within the cloud. This means that any change made from one client happens in the true state within the cloud. This state is then streamed to each client. In this way it is not possible for the state to ever be out of sync for a client. Instead we can have a few other issues.

Disruptions / Sync Issues

If a user's cached state is behind the true state they may send inputs to the cloud that would make sense on their local device, but not make sense on the cloud. A concrete example would be if the user wanted to tell a soldier to attack another player's soldier. On the user's local device they move their mouse over the soldier unit and issue the attack command. These inputs are then sent to the cloud where they will be processed at the end of a turn. If the user's cached state was behind, it is possible that the units they are attempting to attack have moved. Meaning that their mouse move and attack command will now not be selecting the unit. This can be solved using the HOS and the state timeline. We can parse through the state timeline to find the moment where they would have selected and attacked the enemy unit. The game would likely need to detect when this scenario occurs. This could be easily resolved by sending the games cached state ID along with their inputs.

Another common scenario would be a temporary loss of connection. For the client this would appear very similarly as a poor connection in a standard client server model. If they were disconnected for a longer period of time, when they arrived there would be a larger state update for their client to render.

Real Time Strategy Applications

Real time strategy games have for the most part used a strategy referred to as lockstepping. It’s an effective strategy for managing massive data sets between multiple clients. However at large player counts the network cannot handle syncing data and the users local devices do not have the computational power required to handle the games logic. My proposed solution takes advantage of the cloud's ability to compute on large datasets while also taking advantage of robust clients to render frames.

With lockstepping all clients must simulate and compute their own version of the game. This means that each client has their own game state that is being updated and altered. Each client then needs to process every other client's inputs and calculate the outcome on their local version of the game. Not only do local clients need to compute all game logic for every client's inputs but they need to manage and ensure that all clients are in sync.

Lockstepping

Lockstepping is the technique where clients send one another their own inputs. Then each client is responsible for executing and simulating corresponding results of the received inputs. This is done using a concept of turns. A sequence of turns is constructed where each turn has some interval length, say for example two hundred milliseconds. Any inputs that arrive to a client will be assigned to a turn. When the end of a turn arrives all actions assigned to that turn will be executed. This ensures that all clients actions happen on a relatively fair basis. It is possible that a client with a slow connection will have their actions happening on later turns, but they will still execute. It’s worth noting that if this strategy is used there can be a delay up to the length of the turn interval size. For our example the turn interval was two hundred milliseconds and so any input could have a delay of that time on the local client. It was generally found that the delay did not decrease the QOE for RTS games as long as the delay was consistent. For this reason most lockstepping solutions implement it in such a way that commands always have a minimum delay of the turn interval length.

The Application of State Based Streaming to a Lockstepping Solution

My project is based around the two key concepts of a lockstepped game and a state based streaming solution as described earlier in the paper. The combination of these ideas allows for a large player count to operate on a true game state within the cloud.

However there are some key differences in the implementation of a lockstepping solution and in what I will refer to as a cloudstepping solution.

In a cloudstepping solution each client does not need to maintain and simulate their own game state. Rather, the cloud simulates the true state and handles all game logic. The cloud is the only device that can alter the true state. Client devices still send their inputs like in a standard lockstepping solution, however, rather than sending their inputs to all other clients they just send them to the cloud. As mentioned prior in this paper the cloud uses the concept of turns to schedule these inputs. The cloud then streams the game state to each connected client and the client uses local assets to render game frames based on the cached stream state.

Conclusion

I think this idea has a lot of promise. Generally most previous work around cloud gaming solutions has focused on the concept of a thin client. This paper instead proposes a new networking strategy using the computational power of the cloud along with storing a true state of the game there. Rather than streaming video frames to a thin client, or streaming assets for it to render its own frames, I proposed the concept of a state based streaming solution. This solution uses the cloud's computational power to simulate and handle all game logic while also sending state updates to all connected clients. Unlike many other cloud gaming solutions the focus is on robust clients that can render their own frames by the use of a local graphics pipeline.

Future Work

There is quite a lot of potential future work. It was out of scope for this project to implement the actual rendering of frames using the graphics pipeline. The implementation discussed here was designed specifically to allow large player counts in RTS games with massive unit counts per player. However I think this concept of a cloud managed game for robust clients could be extended to many other game genres. The ability to have a true state within the cloud that is streamed to all players is exciting. I could see interesting implications for MMO games, however the strategies would likely need tweaking especially to move away from a lockstepped solution.

References

[1] *Illahi, Gazi Karam, et al. "Foveated streaming of real-time graphics." Proceedings of the 12th ACM Multimedia Systems Conference. 2021.*

[2] *Mohammadi, I.S., Ghanbari, M. & Hashemi, M.R. A hybrid graphics/video rate control method based on graphical assets for cloud gaming. J Real-Time Image Proc 19, 41–59 (2022). https://doi.org/10.1007/s11554-021-01159-y*

[3] *F. Metzger et al., "An Introduction to Online Video Game QoS and QoE Influencing Factors," in IEEE Communications Surveys & Tutorials, vol. 24, no. 3, pp. 1894-1925, thirdquarter 2022, doi: 10.1109/COMST.2022.3177251.*

[4] Bettner, Paul and Mark Terrano. "1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond." (2004).