**Final Project Report**

**Overview**

While most modern games follow a server-authoritative model, RTS games traditionally use a form of lockstep peer-to-peer (P2P). In this model each client runs a full simulation of a game, and any inputs are sent to all other clients, validated, then used in a simulation. Most multiplayer frameworks for large game engines do not support this form of multiplayer, making the genre largely inaccessible for indie developers.

In this project I attempt to present a solution to this problem by designing and implementing a P2P protocol for RTS games in Unreal Engine based on the networking technology of Age of Empires.

**Protocol Design**

The protocol involves sending any actions performed in the game between clients. Once all clients have received actions the actions, the game is simulated. Communication is split into turns, where each turn consists of sending receiving, and simulating a turn, followed by a turn done message.

**Packets**

Each packet sent between the network has the following data in its header:

1. Flags – Flags indicating the contents of the packet. This could indicate an ACK, a turn done message, an action, or a keep-alive packet.
2. Owner - The index of the player who sent the packet. For an ACK this indicates the player whose packet is being ACKed.
3. Turn – The turn on which the packet was sent. Old packets can be discarded
4. SEQ – The sequence number of the sent bytes. Used to detect out of order, dropped, or duplicate packets.
5. Checksum – Used to ensure that the contents of the packet were transmitted without errors.

**Actions**

An action is a single input to be simulated on all clients. This could mean moving a unit, beginning to build a structure, or purchasing an upgrade. UI interactions that do not affect the game simulation do not need to be sent as actions.

Each action has a header to indicate the type of action being performed and flags to determine what data is included in the packet. Actions with the same type can have a different meaning based on the flags. For example, a “move” action can could mean move to a location if the “target location” flag is included or mean move to a structure if the “target structure” flag is set.

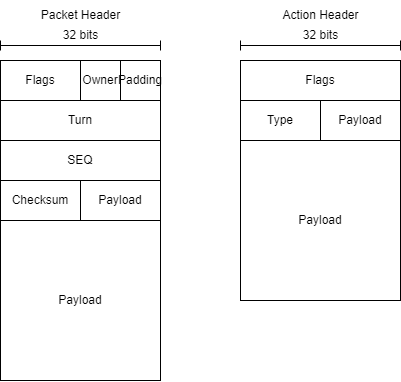
****

Figure : Packet and action header format

**Turn** **Structure**

In a communication turn, all clients send input to each other, decide which inputs will be performed, then simulate the game using those inputs. The following steps are taken:

1. Broadcast all buffered actions
2. Receive any actions from other clients sent last turn
3. Simulate a game step, using actions sent two turns earlier
4. Send a turn done message
5. Wait until turn time is up
6. If any clients are not done, wait and potentially drop them out if they take too long

**Results**

**Successes**

I was successful in designing the packet structure for different kinds of messages, reliability, and connection over UDP. I have a working serialization and deserialization of packets using this structure. I have also designed the protocol for a single communication turn and made good progress on the implementation of a turn but was unable to finish it.

**Challenges Faced**

The main challenges I faced were regarding using Unreal Engine. I had very little prior experience in using Unreal Engine and its C++ API. I had many problems to work through while learning the system such as adding the correct dependencies for socket programming to work, figuring out the built-in memory management, and figuring out what networking features were already included.

I found that I needed to create my own network library for connection over UDP, which caused many design challenges and took a large amount of time and extra research. I was unable to finish this library with the time I had, which stopped me from being able to test much of the rest of the protocol implementation.

**Future Improvements**

There are a number of improvements that could be made in the future to this project:

1. It would be useful to be able to split large packets before being sent over the network.
2. I was unable to test the turn time since I didn’t finish getting the network library done. I would like to experiment with the turn duration to see how fast it can get before having to pause while waiting for packets.
3. It could be useful to separate turn length from simulation steps. If the turn length is too long, the simulation could become too choppy. If needed, a single input turn could be spread over two or three simulation steps if needed. This is the approach described in the article about Age of Empires.
4. In the future the packet headers could be organized to be more layered. Information regarding reliability could be in a separate header than the one describing if it is a turn done message or action.

**Conclusion**

I was able to successfully design most of the protocol for a P2P RTS game. However, due to many challenges faced during implementation, mostly from learning the new software and designing an abstraction for a reliable connection over UDP, I did not yet get a working implementation in the time I had. I would like to continue working on this project to develop a working implementation and comparison of different turn durations.

**References**

* <https://en.wikipedia.org/wiki/UDP_hole_punching>
* <https://en.wikipedia.org/wiki/STUN>
* <https://www.gamedeveloper.com/business/opinion-synchronous-rts-engines-2-sync-harder>
* <https://zoo.cs.yale.edu/classes/cs538/readings/papers/terrano_1500arch.pdf>
* <https://blog.photonengine.com/bannermen-a-classic-rts-game-using-lockstep-with-photon-and-unreal-engine/>