Apply the Design Pattern of Entity Component System (ECS) for a cross-platform network layer

STUDENTS: Roee Horowitz, Erik Huang, Roger Liao

Summary

- Minecraft's game engine is built on an ECS architecture, currently its network and replication system is outside of the ECS
- Our project had 3 main objectives
  1) To build a sandbox multiplayer game with an ECS engine usging ENTT.
  2) To build a replication system around that game.
  3) To recommend a design pattern for an ECS replication system for Minecraft.

Terminology

Serialization

Serialization: The process of translating an object into a format that can be stored and reconstructed later.

Deterministic Lockstep

A game networking technique in which the only messages sent from the user to the server are controls. The server simulates the game and responds to controls by sending game updates to all users.

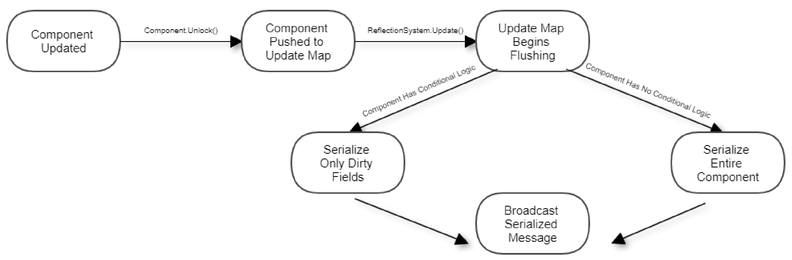
Replication

Sharing information to ensure consistency across all nodes in a networked system.

Entity Component System
An alternative to standard inheritance. Components hold only data and no behavior, systems hold only behavior and no data, and entities are only 64-bit identifiers.

Repplication Pipeline

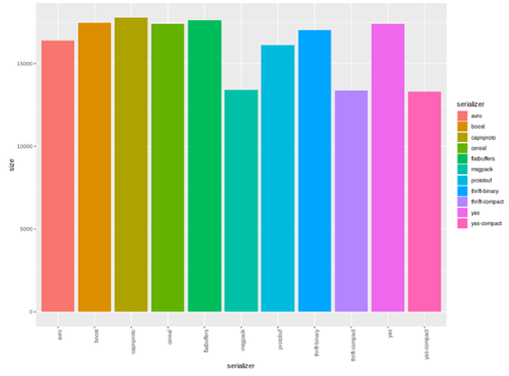
- On updating of an entitys networked component, the replication system is notified by enqueing the entity id and component id onto an update map.
- On game tick the map is flushed, all new data is serialized and packetized, and the updates are broadcasted to all clients.



FSM for the Replication System

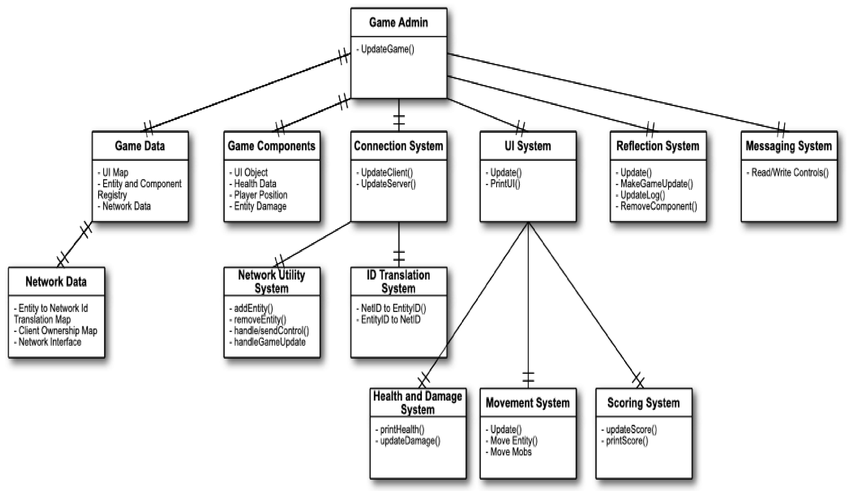
Replication and Serialization

- In ECS, data is held in components. To replicate an entity, we only need to replicate the components.
- To be sent over the network, component data must be serialized.
- We compared the bandwidth vs. Runtime of serialization libraries and chose MsgPack (JSON style).
- Our initial choice was Protobuf, a protocol buffer-based library, but moved away as it didn't make sense in a data driven system.



Comparison of Various Serialization Libraries

Game Architecture

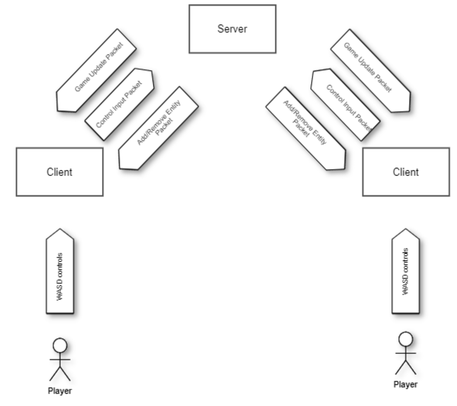


UML Diagram for the Game's Architecture

- The Game is made of 4 primary systems and several auxility subsystems.
- The Game Data is shared acrossed all primary systems.
- Not all systems are being used in the final version as we designed different systems for different serialization libraries like ProtoBuffer and MessagePack.
- In the final version, the reflection system designed for MessagePack is in use.

Networking Structure

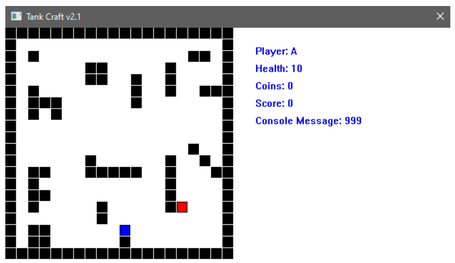
- Our game uses deterministic lockstep network design
- The only messages sent by a user are login and controls
- Game logics are entirely processed by the server
- Component updates are broadcasted back to the users via the replication system.
- Cheating is limited, since the server has complete authority over how the game is played. Clients can only send input data over the wire.



Visualization of the Network System

Conditional Logic in Serialization

- 20% of Minecraft's components have some sort of conditional logic in how they are serialized.
- Our system allows conditional serialization.
- Additionally, developers can choose to only serialize certain fields at runtime, lowering bandwidth.



Example of the Game UI in Action

Future Work, References, and Acknowledgments

- The current system assumes a reliable data transfer protocol
- Future work would be invested in porting the design to an unreliable protocol

References:
https://github.com/thekvs/cpp-serializers
http://www.jenkinssoftware.com/
https://github.com/skypjack/entt
https://developers.google.com/protocol-buffers
https://msgpack.org/index.html

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

ADVISOR: RANIA HUSSEIN
INDUSTRY MENTORS: ROBERT SANCHEZ, RADHA KOTAMARTI

SPONSOR: MICROSOFT