Worlds: A Distributed MMO

Ryan Walker
ryan.cjw@gmail.com

Abstract. This paper overviews a protocol that aims to manage the large scale economic components of a distributed MMO. Following the implementation of the Worlds protocol it becomes possible to fairly scale a universe based on games developed by completely independent parties.

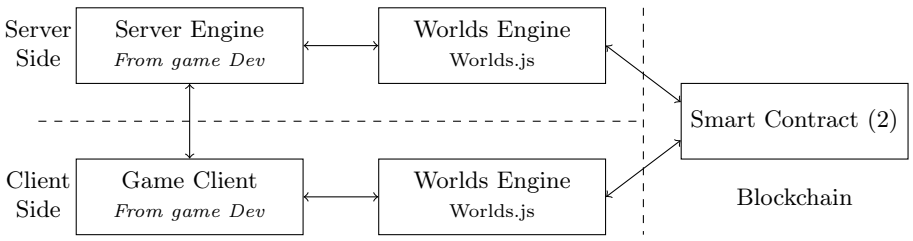
1 Worlds

1.1 Introduction

For an open software ecosystem to grow organically there should be outlets for contribution. In the context of a massively multiplayer online game the outlets become more complicated then a simple code repository. Fair game mechanics are built on a fragile ecosystem that have negative consequences if managed improperly. Forming consensus on a network is trivially done with conventional methods. A server maintains a secure connection with a player who pipes actions to the server. The server then provides ground truth for the network. Our model replaces a centralized organization with a smart contract which forms consensus and balances the economics of the universe. The user experience is designed by anyone who wishes to contribute.

1.2 Overview

A world, defined as w_k, is a node that designs a user experience and hosts players. Worlds provide the game client which the players interact with. In addition they govern their world and create the elements that make it unique. A world can have up to four adjacent worlds determined by itself. This forms a network, which allows players to explore and move common items (5), experience (4) and currency (6) between worlds.



The worlds engine is an application that manages the elements that are common throughout all worlds, such as player keys, hashing, signing and fetching the files required to run the Game Client. The world designer has full creative freedom over the game client and server engine. This enables the highest flexibility and the most unique universe. It is possible for clusters of worlds to share a single game client, yet design their own experience. Conversely, adjacent worlds might have completely different game clients, where players might move from a text based MUD into a fully 3D world.

2 Worlds Smart Contact (WSC)

The worlds smart contract is used to balance the economy across the network. It contains functions and data elements that are used as various forms of proof that require interoperability between worlds. An appropriate blockchain with the following requirements has not yet been selected, it must satisfy the following requirements.

- Low networks fees
- Fast transactions for in game purchases
- Decent data capacity to hold proofs

2.1 Staking

Staking is the fundamental mechanic that balances the economy. This allows a node to lock the WOR token against a hash. This hash can be a proof for experience, items, or any arbitrary chunk of data that that has meaning within the World universe. It's up to the World creators to interpret this data.

3 Actions

Actions are defined as anything a node can do to effect the network. Messages and data packages are to be considered actions in this context.

3.1 Direct Action

A direct action is any action that is travels between the Server Engine and the Game Client. These actions have the lowest latency. An example of an action like this might be the Server engine telling the Game Client of some in game event. Before there can be communication like this the there needs to be a symmetric key exchange.

3.2 Proof Action

Proof actions are actions that require the use of private keys for signing, private keys are managed exclusively by the worlds engine. It it possible for the Game Client to request the Worlds Engine to sign a package, however the player must approve this. For nodes that require higher security it is possible to hold the Worlds Engine offline, this inherently keeps their keys keys offline.

3.3 Contract Action

Contract actions are any actions which communicate with the Worlds Smart Contract. They can be issued from the Worlds Engine directly, or the Game Client can pipe action requests into the worlds engine. Like proof actions, contract actions must be approved by the player if they were issued by the game client.

4 Experience

Experience is distributed to players using experience packages. These are of the form (Figure 1). When a player is to received experience this package is hashed, signed and sent to the *Worlds Smart Contract* (2)(WSC). The world must stake some WOR in order to grant experience. Experience is distributed like items, although it is not transferable from player to player. It is possible for experience to be liquidated back into WOR by the players. The amount of experience is a function of the amount of WOR locked against the hash. It is possible for players to stake their own WOR and buy experience. However this experience is stamped with the public key of the issuer and it's possible that worlds might not accept this during the worlds transfer.

Fig. 1. Experience Package - Sword Fighting

```
ExpClass = 'SwordFighting'
PlayerPublic = 0x5d7ac22131ad370e59fddb5f6079a354dbdd2dd9
WorldPublic = 0xb1abdaf3ab936c99f5fd518122cf7d5b811a1a30
IssueTime = UnixTime #Time of Issue
Stake = 0.1WOR
ExpHash = hash(ExpClass | ... | Stake);
```

4.1 Receiving Experience

Players may receive lots of small chunks of experience during gameplay, this can be taxing on a blockchain. For this reason worlds may wish to issue a lumped sum package when the player leaves the world or requests their package. Even thought experience is kept off-chain during gameplay in this scenario, it's still possible that the benefits may be realized immediately.

5 Items

Items may be introduced by any node on the network. In order to spawn an item, the item package must be signed by the node and submitted to the *worlds smart contract*. This serves to proves ownership and lock the WOR associated

with the item. The smart contract then verifies that the WorldPublic field is the public key of the node issuing the item, assigns current unix time and then moves the WOR into the smart contract from the issuers account. The sha256 hash of the item package is kept on-chain, it is the responsibility of the owner of the item to keep the item package contents. If the user wishes to liquidate or transfer the item the item's hash is required, this is done to preserve space onchain.

Fig. 2. Item Package i - Wood

```
ItemName = 'Wood'
ItemClass = 'Material'
OwnerPublic = 0x5d7ac22131ad370e59fddb5f6079a354dbdd2dd9
PreviousOwnerPublic = 0xgf7dsa7ejdb5370e59fddb5f6079a354ugf84bdv
WorldPublic = 0xb1abdaf3ab936c99f5fd518122cf7d5b811a1a30
UnixTimeGen = # Time the item was created
UnixTimeTX = # Time the item was last transmitted
Stake = 1WOR
ItemHash = hash(ItemName | ... | Stake);
```

As with experience, Worlds have the option to credit or discredit certain items based on the WorldPublic field.

5.1 Item Transfers

It's possible to transfer items to other players simply by calling a function on the WSC.

5.2 Item Liquidation

It's possible to liquidate an item back into the WOR that's stakes against it. This is simply done be calling a function on the worlds smart contract.

5.3 Forging

It is possible to forge items by combining multiple items, thus forming a derivative item. In literal sense the act of forging it to call the function *ForgeItem(Item1, Item2, ...)* on the WSC. This will then make a derivative item which is shown in Figure 3. It is up to the worlds to decide on the item names and item classes of these subitems. As an example, a player may forge a sword by combining metal, fabric and a sharpening stone. As long at the world recognizes that the ForgeHash of the item do in fact form a sword, then the player has crafted a sword for use in these worlds. The power of the sword would be proportional to the amount of stakes WOR in all the items that were used to craft the sword.

Once items are forged into something they can no longer be used for anything else. It is possible to unforge an item.

Fig. 3. Forged Item i_f

```
ForgeHash = hash(i_0[ItemName]|i_0[ItemClass]|...|i_n[ItemName]|i_n[ItemClass])
PlayerPublic = 0x5d7ac22131ad370e59fddb5f6079a354dbdd2dd9
WorldPublic[0 .. n] = # Public Key of all the worlds that issued the sub items
ItemHash[0 .. n] = # Item hashes of items in the forge
UnixTimeGen = # Time the item was created
UnixTimeTX = # Time the item was last transmitted
Stake = sum(ParentItemStake)
ItemHash = hash(ForgeHash | ... | Stake);
```

5.4 Using Items

In order to use an items in a world, the player must present the items to the world for verification, this is done using ftp. During an initial connection the worlds server issues ftp credentials to player, these credentials are then used by the player to move items into the game. The world is responsible for ensuring that there are no double deposits. This can be done with a simple hash lookup comparing items that have been deposited in the past to items that are currently being deposited.

5.5 Item and Exp Value

The value and effect of items and experience will most likely not be directly correlated to the amount of WOR that is staked into them. The two other factors are Genesis time and Genesis world. As the game moves forward, many items will become rare or scarce.

5.6 Item Translations

Worlds maintain a list of all items they recognize, during a world transfer some player items may not be on this list. This means the player will not be able to use these items. To minimize this problem worlds may use an item translation. This is effectively a list of items that funnel into one.

6 Currency (WOR)

A common currency between worlds will be used for staking and integration into the platform. To prevent abuse this currency must be zero sum, meaning worlds cannot generate it and it must start with a fixed supply. Ideally it is tracked

through an existing blockchain. Worlds can introduce mechanics for generating revenue, like entrance fees, subscriptions, purchasing items, or keeping players items upon a death. Please read more in section 9.

7 Transport

To prevent players from playing on multiple worlds concurrently a player location is kept on the worlds smart contract. This location is the public key of the world that the player currently resides. In order for this to be valid it must be signed by the player and the world the player resides in. If a player wishes to travel to a world...

- The player must send a signed entry request to the world.
- If the world only allows travel from adjacent worlds, the player must reside in an adjacent world.
- The player provides proof of items and experience.
- The world might check the public key stamped the items/exp to ensure they are from creditable sources.
- Items or experience are staked if required to be.
- Player is granted access, WSC is updated with player location.

8 Connection

In order to make a smooth player experience there is a state machine that must be followed by all worlds. This is outlined below, SS indicated serverside while CL indicated client size.

1. **SS:** The game server establishes a socket to the Worlds Engine on the server-side. This is used to pipe data back and fourth.
2. **CS:** The Worlds Engine launches the game client, as commanded by the player
3. **SS/CS:** Communication is opened between the game client and the game server. This can be any protocol and is determined by the developer.
4. **CS:** The Worlds Engine generates a proof package and writes the file to player/player.json on the client side**
5. **CS/SS:** The game client sends the json serialized object from client to server.
6. **SS:** The game server verifies the proof through the worlds engine, if the proof in genuine the server instantiates the player into the world.
7. **CS/SS:** The player is given ftp credentials, these can be used to move items into a folder on the server.
8. **SS:** The server then verifies the items are real. If they are they are given the player in game.
9. **SS:** If a player is to receive items, they are placed in the players folder on the server. They can then be claimed by the player for transfer to another world.

9 World Incentivisation

If worlds want to distribute items or experience they must stake some WOR into items. As WOR has monetary value worlds need a way to generate some revenue. In order for a world to profit, the sum of all revenue generation methods must be less their their distributed items plus their overhead. Some revenue strategies are similar to existing games, some are completely new.

9.1 Burying

Some worlds may require you to "bury" some experience, items or WOR before entering. This is effectively temporarily moving these into possession of the world. Worlds may introduce in game mechanics which result in the retention of these items, such as a player death. The world then has freedom over what happens to these items. Some profit models are described below.

- Redistribute the assets to other player, skimming some of the top.
- Liquidate all the items for WOR.
- Give some of the items back to the player.

9.2 Fees

Just like conventional games, it is possible the worlds might charge an entrance fees. These could be one time or reoccurring. This would be transparent as the player moves about worlds. Ideally the fees would be built into the lore of the game.

9.3 In Game Purchases

Worlds might wish to sell items for WOR, as an example there might be an NPC store that sells some items which have special abilities within the world of issuance. The amount of WOR staked into the items would be less than the amount it is sold for. Neighboring world may wish to grant similar abilities in exchange for having some of their items more powerful in worlds adjacent to themselves. This may be considered pay-to-win, however the economics can be moderated by the ecosystem and community, as worlds have the option to deny items that are issued by certain worlds.

Static items are assets that may not transcend worlds, an example being virtual land or housing. These are provided solely by the world of issuance and the value is determined by that world. It is possible they may be kept in the WSC with a staked value of 0 WOR. As worlds are not required to stake WOR against these assets they may profit from selling them for WOR. Items of this class might prove to be an alternate revenue stream.

10 Outstanding Issues

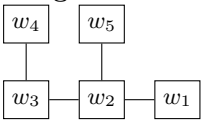
10.1 Malicious Worlds

Players may be required to bury assets to enter a world (9.1). As the universe grows, worlds may be required to manage lots of wealth, which opens the possibility of an exit scam or hacking.

10.2 Neighbor Conflict

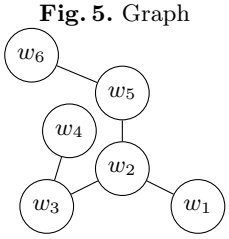
Depending on the topology of worlds there may be neighbor conflicts. In the grid model, Figure 4, w_5 may wish to be adjacent to w_2 but not next to w_4.

Fig. 4. GRID



Using a network, Figure 5 it's possible to treat the worlds like nodes on a graph. This allows four neighboring connections on every node.

Fig. 5. Graph



10.3 Developers

It is possible that development may never catch on, developers may be turned away from the staking mechanic. As they are use to distributing digital assets without consequences.

10.4 Blockchain Technology

We are keeping our eyes open for a suitable blockchain for developing the smart contract. But the space requirements are high, ideally players can bring their own blockchain storage, meaning players pay a small amount for each item they wish to keep.

10.5 Unification

Worlds may have issues forming consensus on forged items (5.3). The derivative hash will be the same if all of the items that go into the forge are the same from forge to forge. Although the resultant item of the forge can be controversial. It is possible that worlds can use item translations (5.6), these can be used to funnel many items into a single for use in the world.

10.6 Congruent Worlds

As it is possible for each world to have their own client the game may suffer from a disconnect, as players move from world to world and the engine fetches new clients for each world the universe may seem to be disconnected. This may be either positive or negative based on the player. After the development of the Worlds Engine and smart contract, we are confident that there will be clusters of worlds that use the same client for players that want more fluid game play.