Authority Assignment in Distributed Multi-Player Proxy-based Games

Sudhir Aggarwal Justin Christofoli
Department of Computer Science
Florida State University, Tallahassee, FL
{sudhir, christof}@cs.fsu.edu

Sarit Mukherjee Sampath Rangarajan
Center for Networking Research
Bell Laboratories, Holmdel, NJ
{sarit, sampath}@bell-labs.com

ABSTRACT

We present a proxy-based gaming architecture and authority assignment within this architecture that can lead to better game playing experience in Massively Multi-player Online games. The proposed game architecture consists of distributed game clients that connect to game proxies (referred to as "communication proxies") which forward game related messages from the clients to one or more game servers. Unlike proxy-based architectures that have been proposed in the literature where the proxies replicate all of the game state, the communication proxies in the proposed architecture support clients that are in proximity to it in the physical network and maintain information about selected portions of the game space that are relevant only to the clients that they support. Using this architecture, we propose an authority assignment mechanism that divides the authority for deciding the outcome of different actions/events that occur within the game between client and servers on a per action/event basis. We show that such division of authority leads to a smoother game playing experience by implementing this mechanism in a massively multi-player online game called RPGQuest. In addition, we argue that cheat detection techniques can be easily implemented at the communication proxies if they are made aware of the game-play mechanics.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication networks**]: Distributed Systems—*Distributed Applications*

General Terms

Games, Performance

Keywords

MMOG, Distributed multi-player games, authority, communication proxy, latency compensation

1. INTRODUCTION

In Massively Multi-player On-line Games (MMOG), game clients who are positioned across the Internet connect to a game server to interact with other clients in order to be part of the game. In current architectures, these interactions are direct in that the game clients and the servers exchange game messages with each other. In addition, current MMOGs delegate all authority to the game server to make decisions about the results pertaining to the actions that game clients take and also to decide upon the result of other game related events. Such centralized authority has been implemented with the claim that this improves the security and consistency required in a gaming environment.

A number of works have shown the effect of network latency on distributed multi-player games [1, 2, 3, 4]. It has been shown that network latency has real impact on practical game playing experience [3, 5]. Some types of games can function quite well even in the presence of large delays. For example, [4] shows that in a modern RPG called Everquest 2, the "breakpoint" of the game when adding artificial latency was 1250ms. This is accounted to the fact that the combat system used in Everquest 2 is queueing based and has very low interaction. For example, a player queues up 4 or 5 spells they wish to cast, each of these spells take 1-2 seconds to actually perform, giving the server plenty of time to validate these actions. But there are other games such as FPS games that break even in the presence of moderate network latencies [3, 5]. Latency compensation techniques have been proposed to alleviate the effect of latency [1, 6, 7] but it is obvious that if MMOGs are to increase in interactivity and speed, more architectures will have to be developed that address responsiveness, accuracy and consistency of the gamestate.

In this paper, we propose two important features that would make game playing within MMOGs more responsive for movement and scalable. First, we propose that centralized server-based architectures be made hierarchical through the introduction of communication proxies so that game updates made by clients that are time sensitive, such as movement, can be more efficiently distributed to other players within their game-space. Second, we propose that assignment of authority in terms of who makes the decision on client actions such as object pickups and hits, and collisions between players, be distributed between the clients and the servers in order to distribute the computing load away from the central server. In order to move towards more complex real-time networked games, we believe that definitions of authority must be refined.

Most currently implemented MMOGs have game servers that have almost absolute authority. We argue that there is no single consistent view of the virtual game space that can be maintained on any one component within a network that has significant latency, such as the one that many MMOG players would experience. We believe that in most cases, the client with the most accurate view of an entity is the best suited to make decisions for that entity when the causality of that action will not immediately affect any other players. In this paper we define what it means to have authority within the context of events and objects in a virtual game space. We then show the benefits of delegating authority for different actions and game events between the clients and server.

In our model, the game space consists of game clients (representing the players) and objects that they control. We divide the client actions and game events (we will collectively refer to these as "events") such as collisions, hits etc. into three different categories, a) events for which the game client has absolute authority, b) events for which the game server has absolute authority, and c) events for which the authority changes dynamically from client to the server and vice-versa. Depending on who has the authority, that entity will make decisions on the events that happen within a game space. We propose that authority for all decisions that pertain to a single player or object in the game that neither affects the other players or objects, nor are affected by the actions of other players be delegated to that player's game client. These type of decisions would include collision detection with static objects within the virtual game space and hit detection with linear path bullets (whose trajectory is fixed and does not change with time) fired by other players. Authority for decisions that could be affected by two or more players should be delegated to the impartial central server, in some cases, to ensure that no conflicts occur and in other cases can be delegated to the clients responsible for those players. For example, collision detection of two players that collide with each other and hit detection of non-linear bullets (that changes trajectory with time) should be delegated to the server. Decision on events such as item pickup (for example, picking up items in a game to accumulate points) should be delegated to a server if there are multiple players within close proximity of an item and any one of the players could succeed in picking the item; for item pick-up contention where the client realizes that no other player, except its own player, is within a certain range of the item, the client could be delegated the responsibility to claim the item. The client's decision can always be accurately verified by the server.

In summary, we argue that while current authority models that only delegate responsibility to the server to make authoritative decisions on events is more secure than allowing the clients to make the decisions, these types of models add undesirable delays to events that could very well be decided by the clients without any inconsistency being introduced into the game. As networked games become more complex, our architecture will become more applicable. This architecture is applicable for massively multiplayer games where the speed and accuracy of game-play are a major concern while consistency between player game-states is still desired. We propose that a mixed authority assignment mechanism such as the one outlined above be implemented in high interaction MMOGs.

Our paper has the following contributions. First we propose an architecture that uses communication proxies to enable clients to connect to the game server. A communication proxy in the proposed architecture maintains information only about portions of the game space that are relevant to clients connected to it and is able to process the movement information of objects and players within these portions. In addition, it is capable of multicasting this information only to a relevant subset of other communication proxies. These functionalities of a communication proxy leads to a decrease in latency of event update and subsequently, better game playing experience. Second, we propose a mixed authority assignment mechanism as described above that improves game playing experience. Third, we implement the proposed mixed authority assignment mechanism within a MMOG called RPGQuest [8] to validate its viability within MMOGs.

In Section 2, we describe the proxy-based game architecture in more detail and illustrate its advantages. In Section 3, we provide a generic description of the mixed authority assignment mechanism and discuss how it improves game playing experience. In Section 4, we show the feasibility of implementing the proposed mixed authority assignment mechanism within existing MMOGs by describing a proof-of-concept implementation within an existing MMOG called RPGQuest. Section 5 discusses related work. In Section 6, we present our conclusions and discuss future work.

2. PROXY-BASED GAME ARCHITECTURE

Massively Multi-player Online Games (MMOGs) usually consist of a large game space in which the players and different game objects reside and move around and interact with each-other. State information about the whole game space could be kept in a single central server which we would refer to as a Central-Server Architecture. But to alleviate the heavy demand on the processing for handling the large player population and the objects in the game in real-time, a MMOG is normally implemented using a distributed server architecture where the game space is further sub-divided into regions so that each region has relatively smaller number of players and objects that can be handled by a single server. In other words, the different game regions are hosted by different servers in a distributed fashion. When a player moves out of one game region to another adjacent one, the player must communicate with a different server (than it was currently communicating with) hosting the new region. The servers communicate with one another to hand off a player or an object from one region to another. In this model, the player on the client machine has to establish multiple gaming sessions with different servers so that it can roam in the entire game space.

We propose a communication proxy based architecture where a player connects to a (geographically) nearby proxy instead of connecting to a central server in the case of a central-server architecture or to one of the servers in case of distributed server architecture. In the proposed architecture, players who are close by geographically join a particular proxy. The proxy then connects to one or more game servers, as needed by the set of players that connect to it and maintains persistent transport sessions with these server. This alleviates the problem of each player having to connect directly to multiple game servers, which can add extra connection setup delay. Introduction of communication proxies also mitigates the overhead of a large number of transport sessions that must be managed and reduces required network bandwidth [9] and processing at the game servers both with central server and distributed server architectures. With central server architectures, communication proxies reduce the overhead at the server by not requiring the server to terminate persistent transport sessions from every one of the clients. With distributed-server architectures, additionally, communication proxies eliminate the need for the clients to maintain persistent transport sessions to every one of the servers. Figure 1 shows the proposed architecture.

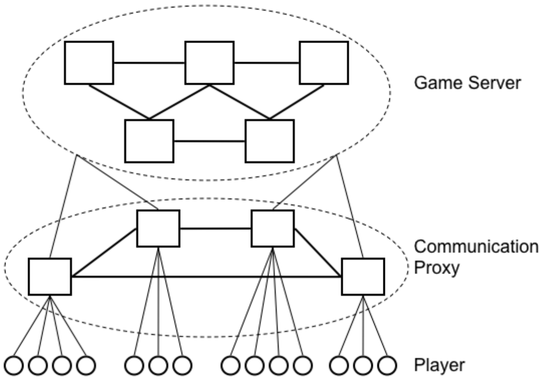


Figure 1: Architecture of the gaming environment.

Note that the communication proxies need not be cognizant of the game. They host a number of players and inform the servers which players are hosted by the proxy in question. Also note that the players hosted by a proxy may not be in the same game space. That is, a proxy hosts players that are geographically close to it, but the players themselves can reside in different parts of the game space. The proxy communicates with the servers responsible for maintaining the game spaces subscribed by the different players. The proxies communicate with one another in a peer-to-peer to fashion. The responsiveness of the game can be improved for updates that do not need to wait on processing at a central authority. In this way, information about players can be disseminated faster before even the game server gets to know about it. This definitely improves the responsiveness of the game. However, it ignores consistency that is critical in MMORPGs. The notion that an architecture such as this one can still maintain temporal consistency will be discussed in detail in Section 3.

Figure 2 shows and example of the working principle of the proposed architecture. Assume that the game space is divided into 9 regions and there are three servers responsible for managing the regions. Server S_1 owns regions 1 and 2, S_2 manages 4, 5, 7, and 8, and S_3 is responsible for 3, 6 and 9.

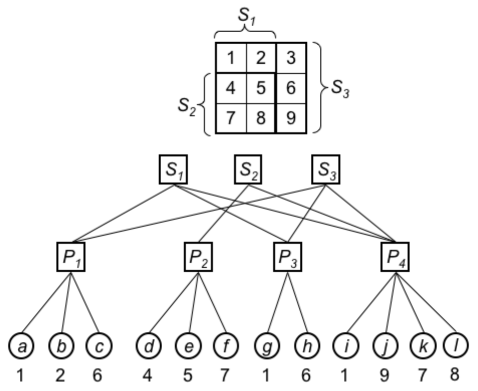


Figure 2: An example.

There are four communication proxies placed in geographically distant locations. Players a, b, c join proxy P_1, proxy P_2 hosts players d, e, f, players g, h are with proxy P_3, whereas players i, j, k, l are with proxy P_4. Underneath each player, the figure shows which game region the player is located currently. For example, players a, b, c are in regions 1, 2, 6, respectively. Therefore, proxy P_1 must communicate with servers S_1 and S_3. The reader can verify the rest of the links between the proxies and the servers.

Players can move within the region and between regions. Player movement within a region will be tracked by the proxy hosting the player and this movement information (for example, the player's new coordinates) will be multicast to a subset of other relevant communication proxies directly. At the same time, this information will be sent to the server responsible for that region with the indication that this movement has already been communicated to all the other relevant communication proxies (so that the server does not have to relay this information to all the proxies). For example, if player a moves within region 1, this information will be communicated by proxy P_1 to server S_1 and multicast to proxies P_3 and P_4. Note that proxies that do not keep state information about this region at this point in time (because they do not have any clients within that region) such as P_2 do not have to receive this movement information.

If a player is at the boundary of a region and moves into a new region, there are two possibilities. The first possibility is that the proxy hosting the player can identify the region into which the player is moving (based on the trajectory information) because it is also maintaining state information about the new region at that point in time. In this case, the proxy can update movement information directly at the other relevant communication proxies and also send information to the appropriate server informing of the movement (this may require handoff between servers as we will describe). Consider the scenario where player a is at the boundary of region 1 and proxy P_1 can identify that the player is moving into region 2. Because proxy P_1 is currently keeping state information about region 2, it can inform all the other relevant communication proxies (in this example, no other proxy maintains information about region 2 at this point and so no update needs to be sent to any of the other proxies) about this movement and then inform the server independently. In this particular case, server S_1 is responsible for region 2 as well and so no handoff between servers would be needed. Now consider another scenario where player j moves from region 9 to region 8 and that proxy P_4 is able to identify this movement. Again, because proxy P_4 maintains state information about region 8, it can inform any other relevant communication proxies (again, none in this example) about this movement. But now, regions 9 and 8 are managed by different servers (servers S_3 and S_2 respectively) and thus a hand-off between these servers is needed. We propose that in this particular scenario, the handoff be managed by the proxy P_4 itself. When the proxy sends movement update to server S_3 (informing the server that the player is moving out of its region), it would also send a message to server S_2 informing the server of the presence and location of the player in one of its region.

In the intra-region and inter-region scenarios described above, the proxy is able to manage movement related information, update only the relevant communication proxies about the movement, update the servers with the movement and enable handoff of a player between the servers if needed. In this way, the proxy performs movement updates without involving the servers in any way in this time-critical function thereby speeding up the game and improving game playing experience for the players. We consider this the "fast path" for movement update. We envision the proxies to be just communication proxies in that they do not know about the workings of specific games. They merely process movement information of players and objects and communicate this information to the other proxies and the servers. If the proxies are made more intelligent in that they understand more of the game logic, it is possible for them to quickly check on claims made by the clients and mitigate cheating. The servers could perform the same functionality but with more delay. Even without being aware of game logic, the proxies can provide additional functionalities such as time-stamping messages to make the game playing experience more accurate [10] and fair [11].

The second possibility that should be considered is when players move between regions. It is possible that a player moves from one region to another but the proxy that is hosting the player is not able to determine the region into which the player is moving, **a)** the proxy does not maintain state information about all the regions into which the player could potentially move, or **b)** the proxy is not able to determine which region the player may move into (even if maintains state information about all these regions). In this case, we propose that the proxy be not responsible for making the movement decision, but instead communicate the movement indication to the server responsible for the region within which the player is currently located. The server will then make the movement decision and then **a)** inform all the proxies including the proxy hosting the player, and **b)** initiate handoff with another server if the player moves into a region managed by another server. We consider this the "slow path" for movement update in that the servers need to be involved in determining the new position of the player.

In the example, assume that player a moves from region 1 to region 4. Proxy P_1 does not maintain state information about region 4 and thus would pass the movement information to server S_1. The server will identify that the player has moved into region 4 and would inform proxy P_1 as well as proxy P_2 (which is the only other proxy that maintains information about region 4 at this point in time). Server S_1 will also initiate a handoff of player a with server S_2. Proxy P_1 will now start maintaining state information about region 4 because one of its hosted players, player a has moved into this region. It will do so by requesting and receiving the current state information about region 4 from server S_2 which is responsible for this region.

Thus, a proxy architecture allows us to make use of faster movement updates through the fast path through a proxy if and when possible as opposed to conventional server-based architectures that always have to use the slow path through the server for movement updates. By selectively maintaining relevant regional game state information at the proxies, we are able to achieve this capability in our architecture without the need for maintaining the complete game state at every proxy.

3. ASSIGNMENT OF AUTHORITY

As a MMOG is played, the players and the game objects that are part of the game, continually change their state. For example, consider a player who owns a tank in a battlefield game. Based on action of the player, the tank changes its position in the game space, the amount of ammunition the tank contains changes as it fires at other tanks, the tank collects bonus firing power based on successful hits, etc. Similarly objects in the battlefield, such as flags, buildings etc. change their state when a flag is picked up by a player (i.e. tank) or a building is destroyed by firing at it. That is, some decision has to be made on the state of each player and object as the game progresses. Note that the state of a player and/or object can contain several parameters (e.g., position, amount of ammunition, fuel storage, points collected, etc), and if any of the parameters changes, the state of the player/object changes.

In a client-server based game, the server controls all the players and the objects. When a player at a client machine makes a move, the move is transmitted to the server over the network. The server then analyzes the move, and if the move is a valid one, changes the state of the player at the server and informs the client of the change. The client subsequently updates the state of the player and renders the player at the new location. In this case the authority to change the state of the player resides with the server entirely and the client simply follows what the server instructs it to do.

Most of the current first person shooter (FPS) games and role playing games (RPG) fall under this category. In current FPS games, much like in RPG games, the client is not trusted. All moves and actions that it makes are validated. If a client detects that it has hit another player with a bullet, it proceeds assuming that it is a hit. Meanwhile, an update is sent to the server and the server will send back a message either affirming or denying that the player was hit. If the remote player was not hit, then the client will know that it did not actually make the shot. If it did make the hit, an update will also be sent from the server to the other clients informing them that the other player was hit. A difference that occurs in some RPGs is that they use very dumb client programs. Some RPGs do not maintain state information at the client and therefore, cannot predict anything such as hits at the client. State information is not maintained because the client is not trusted with it. In RPGs, a cheating player with a hacked game client can use state information stored at the client to gain an advantage and find things such as hidden treasure or monsters lurking around the corner. This is a reason why most MMORPGs do not send a lot of state information to the client and causes the game to be less responsive and have lower interaction game-play than FPS games.

In a peer-to-peer game, each peer controls the player and object that it "owns". When a player makes a move, the peer machine analyzes the move and if it is a valid one, changes the state of the player and places the player in new position. Afterwards, the owner peer informs all other peers about the new state of the player and the rest of the peers update the state of the player. In this scenario, the authority to change the state of the player is given to the owning peer and all other peers simply follow the owner.

For example, Battle Zone Flag (BzFlag) [12] is a multi-player client-server game where the client has all authority for making decisions. It was built primarily with LAN play in mind and cheating as an afterthought. Clients in BzFlag are completely authoritative and when they detect that they were hit by a bullet, they send an update to the server which simply forwards the message along to all other players. The server does no sort of validation.

Each of the above two traditional approaches has its own set of advantages and disadvantages. The first approach, which we will refer to as "server authoritative" henceforth, uses a centralized method to assign authority. While a centralized approach can keep the state of the game (i.e., state of all the players and objects) consistent across any number of client machines, it suffers from delayed response in game-play as any move that a player at the client machine makes must go through one round-trip delay to the server before it can take effect on the client's screen. In addition to the round-trip delay, there is also queuing delay in processing the state change request at the server. This can result in additional processing delay, and can also bring in severe scalability problems if there are large number of clients playing the game. One definite advantage of the server authoritative approach is that it can easily detect if a client is cheating and can take appropriate action to prevent cheating.

The peer-to-peer approach, henceforth referred to as "client authoritative", can make games very responsive. However, it can make the game state inconsistent for a few players and tie break (or roll back) has to be performed to bring the game back to a consistent state. Neither tie break nor roll back is a desirable feature of online gaming. For example, assume that for a game, the goal of each player is to collect as many flags as possible from the game space (e.g. BzFlag). When two players in proximity try to collect the same flag at the same time, depending on the algorithm used at the client-side, both clients may determine that it is the winner, although in reality only one player can pick the flag up. Both players will see on their screen that it is the winner. This makes the state of the game inconsistent. Ways to recover from this inconsistency are to give the flag to only one player (using some tie break rule) or roll the game back so that the players can try again. Neither of these two approaches is a pleasing experience for online gaming. Another problem with client authoritative approach is that of cheating by clients as there is no cross checking of the validation of the state changes authorized by the owner client.

We propose to use a hybrid approach to assign the authority dynamically between the client and the server. That is, we assign the authority to the client to make the game responsive, and use the server's authority only when the client's individual authoritative decisions can make the game state inconsistent. By moving the authority of time critical updates to the client, we avoid the added delay caused by requiring the server to validate these updates. For example, in the flag pickup game, the clients will be given the authority to pickup flags only when other players are not within a range that they could imminently pickup a flag. Only when two or more players are close by so that more than one player may claim to have picked up a flag, the authority for movement and flag pickup would go to the central server so that the game state does not become inconsistent. We believe that in a large game-space where a player is often in a very wide open and sparsely populated area such as those often seen in the game Second Life [13], this hybrid architecture would be very beneficial because of the long periods that the client would have authority to send movement updates for itself. This has two advantages over the central-authority approach, it distributes the processing load down to the clients for the majority of events and it allows for a more responsive game that does not need to wait on a server for validation.

We believe that our notion of authority can be used to develop a globally consistent state model of the evolution of a game. Fundamentally, the consistent state of the system is the one that is defined by the server. However, if local authority is delegated to the client, in this case, the client's state is superimposed on the server's state to determine the correct global state. For example, if the client is authoritative with respect to movement of a player, then the trajectory of the player is the "true" trajectory and must replace the server's view of the player's trajectory. Note that this could be problematic and lead to temporal inconsistency only if, for example, two or more entities are moving in the same region and can interact with each other. In this situation, the client authority must revert to the server and the sever would then make decisions. Thus, the client is only authoritative in situations where there is no potential to imminently interact with other players. We believe that in complex MMOGs, when allowing more rapid movement, it will still be the case that local authority is possible for significant spans of game time. Note that it might also be possible to minimize the occurrences of the "Dead Man Shooting" problem described in [14]. This could be done by allowing the client to be authoritative for more actions such as its player's own death and disallowing other players from making preemptive decisions based on a remote player.

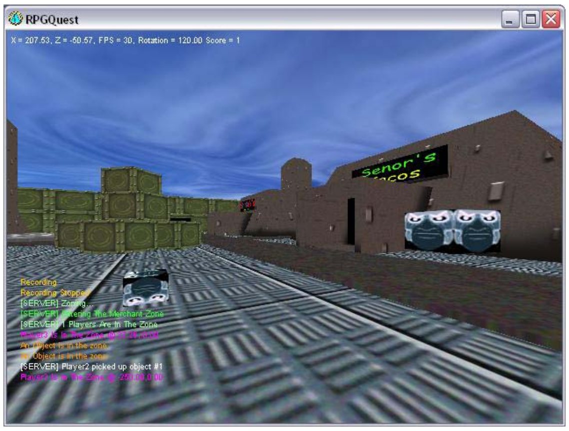
One reason why the client-server based architecture has gained popularity is due to belief that the fastest route to the other clients is through the server. While this may be true, we aim to create a new architecture where decisions do not always have to be made at the game server and the fastest route to a client is actually through a communication proxy located close to the client. That is, the shortest distance in our architecture is not through the game server but through the communication proxy. After a client makes an action such as movement, it will simultaneously distribute it directly to the clients and the game server by way of the communications proxy. We note that our architecture however is not practical for a game where game players setup their own servers in an ad-hoc fashion and do not have access to proxies at the various ISPs. This proxy and distributed authority architecture can be used to its full potential only when the proxies can be placed at strategic places within the main ISPs and evenly distributed geographically.

Our game architecture does not assume that the client is not to be trusted. We are designing our architecture on the fact that there will be sufficient cheat deterring and detection mechanisms present so that it will be both undesirable and very difficult to cheat [15]. In our proposed approach, we can make the games cheat resilient by using the proxy-based architecture when client authoritative decisions take place. In order to achieve this, the proxies have to be game cognizant so that decisions made by a client can be cross checked by a proxy that the client connects to. For example, assume that in a game a plane controlled by a client moves in the game space. It is not possible for the plane to go through a building unharmed. In a client authoritative mode, it is possible for the client to cheat by maneuvering the plane through a building and claiming the plane to be unharmed. However, when such move is published by the client, the proxy, being aware of the game space that the plane is in, can quickly check that the client has misused the authority and then can block such move. This allows us to distribute authority to make decisions about the clients.

In the following section we use a multiplayer game called RPGQuest to implement different authoritative schemes and discuss our experience with the implementation. Our implementation shows the viability of our proposed solution.

4. IMPLEMENTATION EXPERIENCE

We have experimented with the authority assignment mechanism described in the last section by implementing the mechanisms in a game called RPGQuest. A screen shot from this game is shown in Figure 3. The purpose of the implementation is to test its feasibility in a real game. RPGQuest is a basic first person game where the player can move around a three dimensional environment. Objects are placed within the game world and players gain points for each object that is collected. The game clients connect to a game server which allows many players to coexist in the same game world. The basic functionality of this game is representative of current online first person shooter and role playing games. The game uses the DirectX 8 graphics API and DirectPlay networking API. In this section we will discuss the three different versions of the game that we experimented with.



```
X = 207.53, Z = -50.57, FPS = 30, Rotation = 120.00 Score = 1
Recording
Recording Stopped
[SERVER] Zoning...
[SERVER] Entering The Merchant Zone
[SERVER] 1 Players Are In The Zone
Player3 Is In The Zone @ [illegible]
An Object is in the zone.
An Object is in the zone.
[SERVER] Player2 picked up object #1
Player3 Is In The Zone @ 250.00,0.00
```

Figure 3: The RPGQuest Game.

The first version of the game, which is the original implementation of RPGQuest, was created with a completely authoritative server and a non-authoritative client. Authority given to the server includes decisions of when a player collides with static objects and other players and when a player picks up an object. This version of the game performs well up to 100ms round-trip latency between the client and the server. There is little lag between the time player hits a wall and the time the server corrects the player's position. However, as more latency is induced between the client and server, the game becomes increasingly difficult to play. With the increased latency, the messages coming from the server correcting the player when it runs into a wall are not received fast enough. This causes the player to pass through the wall for the period that it is waiting for the server to resolve the collision.

When studying the source code of the original version of the RPGQuest game, there is a substantial delay that is unavoidable each time an action must be validated by the server. Whenever a movement update is sent to the server, the client must then wait whatever the round trip delay is, plus some processing time at the server in order to receive its validated or corrected position. This is obviously unacceptable in any game where movement or any other rapidly changing state information must be validated and disseminated to the other clients rapidly.

In order to get around this problem, we developed a second version of the game, which gives all authority to the client. The client was delegated the authority to validate its own movement and the authority to pick up objects without validation from the server. In this version of the game when a player moves around the game space, the client validates that the player's new position does not intersect with any walls or static objects. A position update is then sent to the server which then immediately forwards the update to the other clients within the region. The update does not have to go through any extra processing or validation.

This game model of complete authority given to the client is beneficial with respect to movement. When latencies of

100ms and up are induced into the link between the client and server, the game is still playable since time critical aspects of the game like movement do not have to wait on a reply from the server. When a player hits a wall, the collision is processed locally and does not have to wait on the server to resolve the collision.

Although game playing experience with respect to responsiveness is improved when the authority for movement is given to the client, there are still aspects of games that do not benefit from this approach. The most important of these is consistency. Although actions such as movement are time critical, other actions are not as time critical, but instead require consistency among the player states. An example of a game aspect that requires consistency is picking up objects that should only be possessed by a single player.

In our client authoritative version of RPGQuest clients send their own updates to all other players whenever they pick up an object. From our tests we have realized this is a problem because when there is a realistic amount of latency between the client and server, it is possible for two players to pick up the same object at the same time. When two players attempt to pick up an object at physical times which are close to each other, the update sent by the player who picked up the object first will not reach the second player in time for it to see that the object has already been claimed. The two players will now both think that they own the object. This is why a server is still needed to be authoritative in this situation and maintain consistency throughout the players.

These two versions of the RPGQuest game has showed us why it is necessary to mix the two absolute models of authority. It is better to place authority on the client for quickly changing actions such as movement. It is not desirable to have to wait for server validation on a movement that could change before the reply is even received. It is also sometimes necessary to place consistency over efficiency in aspects of the game that cannot tolerate any inconsistencies such as object ownership. We believe that as the interactivity of games increases, our architecture of mixed authority that does not rely on server validation will be necessary.

To test the benefits and show the feasibility of our architecture of mixed authority, we developed a third version of the RPGQuest game that distributed authority for different actions between the client and server. In this version, in the interest of consistency, the server remained authoritative for deciding who picked up an object. The client was given full authority to send positional updates to other clients and verify its own position without the need to verify its updates with the server. When the player tries to move their avatar, the client verifies that the move will not cause it to move through a wall. A positional update is then sent to the server which then simply forwards it to the other clients within the region. This eliminates any extra processing delay that would occur at the server and is also a more accurate means of verification since the client has a more accurate view of its own state than the server.

This version of the RPGQuest game where authority is distributed between the client and server is an improvement from the server authoritative version. The client has no delay in waiting for an update for its own position and other clients do not have to wait on the server to verify the update. The inconsistencies where two clients can pick up the same object in the client authoritative architecture are not present in this version of the client. However, the benefits of mixed authority will not truly be seen until an implementation of our communication proxy is integrated into the game. With the addition of the communication proxy, after the client verifies its own positional updates it will be able to send the update to all clients within its region through a low latency link instead of having to first go through the game server which could possibly be in a very remote location.

The coding of the different versions of the game was very simple. The complexity of the client increased very slightly in the client authoritative and hybrid models. The original "dumb" clients of RPGQuest know the position of other players; it is not just sent a screen snapshot from the server. The server updates each client with the position of all nearby clients. The "dumb" clients use client side prediction to fill in the gaps between the updates they receive. The only extra processing the client has to do in the hybrid architecture is to compare its current position to the positions of all objects (walls, boxes, etc.) in its area. This obviously means that each client will have to already have downloaded the locations of all static objects within its current region.

5. RELATED WORK

It has been noted that in addition to latency, bandwidth requirements also dictate the type of gaming architecture to be used. In [16], different types of architectures are studied with respect to bandwidth efficiencies and latency. It is pointed out that Central Server architectures are not scalable because of bandwidth requirements at the server but the overhead for consistency checks are limited as they are performed at the server. A Peer-to-Peer architecture, on the other hand, is scalable but there is a significant overhead for consistency checks as this is required at every player. The paper proposes a hybrid architecture which is Peer-to-Peer in terms of message exchange (and thereby is scalable) where a Central Server is used for off-line consistency checks (thereby mitigating consistency check overhead). The paper provides an implementation example of BZFlag which is a peer-to-peer game which is modified to transfer all authority to a central server. In essence, this paper advocates an authority architecture which is server based even for peer-to-peer games, but does not consider division of authority between a client and a server to minimize latency which could affect game playing experience even with the type of latency found in server based games (where all authority is with the server).

There is also previous work that has suggested that proxy based architectures be used to alleviate the latency problem and in addition use proxies to provide congestion control and cheat-proof mechanisms in distributed multi-player games [17]. In [18], a proxy server-network architecture is presented that is aimed at improving scalability of multiplayer games and lowering latency in server-client data transmission. The main goal of this work is to improve scalability of First-Person Shooter (FPS) and RPG games. The further objective is to improve the responsiveness MMOGs by providing low latency communications between the client and server. The architecture uses interconnected proxy servers that each have a full view of the global game state. Proxy servers are located at various different ISPs. It is mentioned in this work that dividing the game space among multiple games servers such as the federated model presented in [19] is inefficient for a relatively fast game flow and that the proposed architecture alleviates this problem because users do not have to connect to a different server whenever they cross the server boundary. This architecture still requires all proxies to be aware of the overall game state over the whole game space unlike our work where we require the proxies to maintain only partial state information about the game space.

Fidelity based agent architectures have been proposed in [20, 21]. These works propose a distributed client-server architecture for distributed interactive simulations where different servers are responsible for different portions of the game space. When an object moves from one portion to another, there is a handoff from one server to another. Although these works propose an architecture where different portions of the simulation space are managed by different servers, they do not address the issue of decreasing the bandwidth required through the use of communication proxies.

Our work differs from the above discussed previous works by proposing **a)** a distributed proxy-based architecture to decrease bandwidth requirements at the clients and the servers without requiring the proxies to keep state information about the whole game space, **b)** a dynamic authority assignment technique to reduce latency (by performing consistency checks locally at the client whenever possible) by splitting the authority between the clients and servers on a per object basis, and **c)** proposing that cheat detection can be built into the proxies if they are provided more information about the specific game instead of using them purely as communication proxies (although this idea has not been implemented yet and is part of our future work).

6. CONCLUSIONS AND FUTURE WORK

In this paper, we first proposed a proxy-based architecture for MMOGs that enables MMOGs to scale to a large number of users by mitigating the need for a large number of transport sessions to be maintained and decreasing both bandwidth overhead and latency of event update. Second, we proposed a mixed authority assignment mechanism that divides authority for making decisions on actions and events within the game between the clients and server and argued how such an authority assignment leads to better game playing experience without sacrificing the consistency of the game. Third, to validate the viability of the mixed authority assignment mechanism, we implemented it within a MMOG called RPGQuest and described our implementation experience.

In future work, we propose to implement the communications proxy architecture described in this paper and integrate the mixed authority mechanism within this architecture. We propose to evaluate the benefits of the proxy-based architecture in terms of scalability, accuracy and responsiveness. We also plan to implement a version of the RPGQuest game with dynamic assignment of authority to allow players the authority to pickup objects when no other players are near. As discussed earlier, this will allow for a more efficient and responsive game in certain situations and alleviate some of the processing load from the server.

Also, since so much trust is put into the clients of our architecture, it will be necessary to integrate into the architecture many of the cheat detection schemes that have been proposed in the literature. Software such as Punkbuster [22] and a reputation system like those proposed by [23] and [15] would be integral to the operation of an architecture such as ours which has a lot of trust placed on the client. We further propose to make the proxies in our architecture more game cognizant so that cheat detection mechanisms can be built into the proxies themselves.

7. REFERENCES

[1] Y. W. Bernier. Latency Compensation Methods in Client/Server In-game Protocol Design and Optimization. In *Proc. of Game Developers Conference'01*, 2001.

[2] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29, New York, NY, USA, 2002. ACM Press.

[3] G. Armitage. Sensitivity of Quake3 Players to Network Latency. In *Proc. of IMW2001, Workshop Poster Session*, November 2001. `http://www.geocities.com/gj_armitage/q3/quake-results.html`.

[4] Tobias Fritsch, Hartmut Ritter, and Jochen Schiller. The effect of latency and network limitations on mmorpgs: a field study of everquest2. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–9, New York, NY, USA, 2005. ACM Press.

[5] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 144–151, New York, NY, USA, 2004. ACM Press.

[6] Y. Lin, K. Guo, and S. Paul. Sync-MS: Synchronized Messaging Service for Real-Time Multi-Player Distributed Games. In *Proc. of 10th IEEE International Conference on Network Protocols (ICNP)*, Nov 2002.

[7] Katherine Guo, Sarit Mukherjee, Sampath Rangarajan, and Sanjoy Paul. A fair message exchange framework for distributed multi-player games. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 29–41, New York, NY, USA, 2003. ACM Press.

[8] T. Barron. *Multiplayer Game Programming*, chapter 16–17, pages 672–731. Prima Tech's Game Development Series. Prima Publishing, 2001.

[9] Carsten Griwodz and Pål Halvorsen. The fun of using tcp for an mmorpg. In *NOSSDAV '06: Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and VIdeo*, New York, NY, USA, 2006. ACM Press.

[10] Sudhir Aggarwal, Hemant Banavar, Amit Khandelwal, Sarit Mukherjee, and Sampath Rangarajan. Accuracy in dead-reckoning based distributed multi-player games. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 161–165, New York, NY, USA, 2004. ACM Press.

[11] Sudhir Aggarwal, Hemant Banavar, Sarit Mukherjee, and Sampath Rangarajan. Fairness in dead-reckoning based distributed multi-player games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–10, New York, NY, USA, 2005. ACM Press.

[12] Riker, T. et al. Bzflag. `http://www.bzflag.org`, 2000-2006.

[13] Linden Lab. Second life. `http://secondlife.com`, 2003.

[14] Martin Mauve. How to keep a dead man from shooting. In *IDMS '00: Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 199–204, London, UK, 2000. Springer-Verlag.

[15] Max Skibinsky. *Massively Multiplayer Game Development 2*, chapter The Quest for Holy Scale–Part 2: P2P Continuum, pages 355–373. Charles River Media, 2005.

[16] Joseph D. Pellegrino and Constantinos Dovrolis. Bandwidth requirement and state consistency in three multiplayer game architectures. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 52–59, New York, NY, USA, 2003. ACM Press.

[17] M. Mauve J. Widmer and S. Fischer. A Generic Proxy Systems for Networked Computer Games. In *Proc. of the Workshop on Network Games, Netgames 2002*, April 2002.

[18] S. Gorlatch J. Muller, S. Fischer and M.Mauve. A Proxy Server Network Architecture for Real-Time Computer Games. In *Euor-Par 2004 Parallel Processing: 10th International EURO-PAR Conference*, August-September 2004.

[19] H. Hazeyama T. Limura and Y. Kadobayashi. Zoned Federation of Game Servers: A Peer-to-Peer Approach to Scalable Multiplayer On-line Games. In *Proc. of ACM Workshop on Network Games, Netgames 2004*, August-September 2004.

[20] B. Kelly and S. Aggarwal. A Framework for a Fidelity Based Agent Architecture for Distributed Interactive Simulation. In *Proc. 14th Workshop on Standards for Distributed Interactive Simulation*, pages 541–546, March 1996.

[21] S. Aggarwal and B. Kelly. Hierarchical Structuring for Distributed Interactive Simulation. In *Proc. 13th Workshop on Standards for Distributed Interactive Simulation*, pages 125–132, Sept 1995.

[22] Even Balance, Inc. Punkbuster. `http://www.evenbalance.com/`, 2001-2006.

[23] Y. Wang and J. Vassileva. Trust and Reputation Model in Peer-to-Peer Networks. In *Third International Conference on Peer-to-Peer Computing*, 2003.