# Peer-to-peer support for instance-based massively multiplayer games

Marcos Rates Crippa, Fábio Reis Cecin, Cláudio Fernando Resin Geyer
Informatics Institute
Federal University of Rio Grande do Sul
Porto Alegre, RS, Brazil

## Abstract

Massively Multiplayer Online Games are a growing segment on the game industry, combining the excitement of 3D games with the opportunity of playing with thousands of players worldwide. The common network model used is a straightforward client/server one, where all the simulation and events are processed on the server. This creates a bottleneck on the server-side, and harshly affects the scalability of the whole system. When the broadband technologies available today and the increase in the number of players are considered, the server's complexity and size grows significantly, becoming too costly. One approach being proposed to overcome those difficulties is to change the network model into a P2P (Peer-to-Peer) model. In this article we propose a P2P network model that tries to provide scalability and security. We first examine in more details the disadvantages of the client-server model. After that, we talk about the instanced game model, which is the game model support by our software architecture. Finally, we describe the organization of our solution, and some interesting points about it and others solutions proposed.

**Keywords::** Massively multiplayer games, peer-to-peer, hybrid topologies, instanced games

**Author's Contact:**

{mrcrippa,fcecin,geyer}@inf.ufrgs.br

## 1 Introduction

Multiplayer games are a very popular game genre, due to their highly interactive nature. Among those games, a class that is growing in popularity is MMOGs (Massively Multiplayer Online Games). MMOGs are real-time multiplayer games played through the Internet, where a great number of players (usually thousands) play within a persistent-state virtual world. Successful examples include EverQuest [Entertainment a] and Ultima Online [Origin ].

The most common network model used on MMOGs is client-server. In this model, the client only sends its data to the server-side (that can operate on a single machine, a cluster or distributed on a grid) and receives frequent updates from the server. The server process all the data from the clients, and broadcasts all the results that occur on the virtual world back to the them. Some advantages of this model are: being simple in design; cheating can be efficiently detect and stopped; retaining control over access to the game; and being a predictable model.

The main disadvantage of the client-server model is the lack of scalability. The cost of maintaining the server-side become excessive with the increase in the number of clients. When talking about commercial games, the usual approach is to continuously expand the number of machines on the server-side. That is a reasonable solution, however it is not suitable for projects on a budget such as those from small companies or research groups.

One way of dealing with the above problem is to fully or partially distribute the MMOG simulation. All the changes on the game world would be registered and dealt within the client-side, with no interference of the server-side. Security and world state consistence are issues on that model, because there isn't a point where the changes in the virtual world can be evaluate and consider legal or illegal. The main challenge of our project is thus to provide a partially decentralized support model for MMOGs, while retaining the basic properties of the client-server model such as security, consistency and scalability [Schiele et al. 2007].

In this paper a hybrid model is proposed, combining the security and consistency of the client-server model with the scalability and flexibility of the distributed model. A game model which follows that directives, called instanced game model and inspired by the game Guild Wars [ArenaNet ], is described in the section 2 of this article. The P2PSE(acronym in portuguese for "P2P for Entertainment Software") architecture, which implements the instanced game model, is explained in section 3. Section 4 offers a discussion of our impressions with the limited testing that we managed to perform over the current (incomplete) implementation, and discusses our expectations towards gains when compared with pure client-server approaches. Section 5 presents conclusions on the topic.

## 2 The instanced game model

The P2PSE project is a distributed simulation model, and also a library (in reality, a stack of C/C++ libraries) that implements that model, which will be described in section 3. The library is designed to support a specific kind of MMOG,described in this section. The instanced game model is the price to pay, in consistency, for a simple approach in unifying security, consistency and scalability in a decentralized MMOG support model.

Several MMOGs, like PlanetSide [Entertainment b] and World of Warcraft [Blizzard ] offer to the player the illusion of one or several large and contiguous virtual spaces where all the gaming takes place. Those spaces are divided in segments, and each server machine or group of server machines are responsible for one segment. A warping system is nec-

essary to tied all segments together, allowing the player to move from one space to the other transparently.

The game Guild Wars introduces a new game model, which will be referenced to as the instanced game model [Cecin et al. 2006]. There is two virtual spaces on Guild Wars: the social space and the action space. The first kind is a medium-sized virtual space where a significant number of players can socialize, trade virtual goods and organize game sessions. Because of the nature of this space, low consistency requirements are necessary. Game sessions happen on the action space, which is a small-sized virtual space where a small number of players gather to play a game session. Higher consistency requirements are necessary on that space because of its fast nature.

The relationship between the two spaces is as follows. When a game session ends, the players involved in it return to the social space. All the traits of the characters (e.g. virtual world money, experience points and statistics) are updated with new information from the session. The social space is designed as a contiguous world, as mentioned in the beginning of this section. The action spaces are created dynamically to support temporary game sessions with a small number of players. This space is destroyed after the session ends.

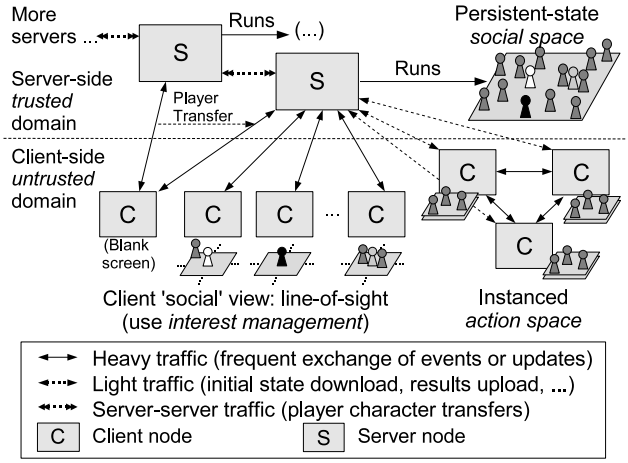


**Figure 1:** *The instanced 'social vs. action' MMOG and our approach for its distribution.*

Guild Wars, as far as we can tell, follows the client-server model. Our proposal is to used the instanced game model with a hybrid network architecture model. Since the interactions on the social space (chatting, trading) don't require high consistency, and there is a necessity of validation of who can play (game accounts,passwords), the server-side will be responsible for coordinating this part of the game. So even a server-side with modest processing capacity and network bandwidth could manage the social space if game quality is scaled down accordingly, for instance by sending less frequent updates to clients. The game sessions would be process only on the client-side machines, avoiding unnecessary processing and communication cost on the server-side. The clients groups would operate within a peer-to-peer dynamics.

# 3 The P2PSE project

The P2PSE architecture proposed on this article implements the instanced game model specified on section 2. A group of software libraries provide the functionalities associated with the game model. These libraries are organized in layers, each level setting a group of features to the upper one. There are three layers on the architecture, which are described below.

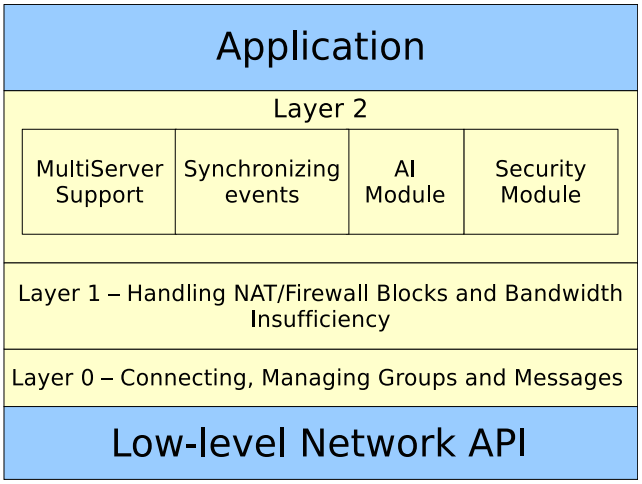


**Figure 2:** *The layer structure of the P2PSE archictecture.*

## 3.1 Layer 0

The first layer of the P2PSE architecture is responsible for the creation of the network topology seen on section 2. It operates assuming that the most basic network functions ( such as creating a socket and managing TCP/UDP packages and connections) are handle by an external library. All the communication necessary on the system will use the services provide by this low-level network API.

There are two main components on the game model: the server and the client. The server is responsible for the social space and for the clients playing in the action spaces. The client can either be in the social space, communicating directly with the server, or playing in a game session and communicating with others players. In order to differentiate all the roles that both server and client assume, Layer 0 has four main classes: Server, Group,Client and Peer.

The Server represents the server machine, and handles the connections and the social space. The Group resides within the server, and represents a group of clients. It encapsulates all the necessary functions to manage a group. The Client represents the client machine and handles the connection with the server. This class also represents one client when he is on a social space. And, finally, the Peer represents the client within a game session, and handles the connections with the others players.

## 3.2 Layer 1

Layer 1 of the P2PSE architecture is responsible for handling connection-related problems between the peers within a group. This layer will ensure that all peers are able to send and receive messages from each other.

We see two main problems that can cause a connection failure between two peers : NAT/Firewall block and bandwidth insufficiency. When the Layer 0 indicates that a message must be send from one peer to another, Layer 1 must verify if there is a direct connection between the peers. If this connection doesn't exist, an alternative path must be discover, using intermediate peers. After the path is discovered, the proper routing of the message can take place.

Among all peers within a group there are different bandwidth capabilities. It is very likely to occur a situation where a peer doesn't have enough upload bandwidth to deal with the number of messages that it has to send. Layer 1 will identify such situation and, instead of sending the messages directly to the their destinations, send them to a peer with good upload bandwidth capability, holding him responsible to send the message.

The algorithms and methods that are going to be used to implement this layer are been researched and developed.

### 3.3 Layer 2

The last layer of the P2PSE architecture will keep the action spaces consistent. Due to the fact that the simulation on the action spaces is carried out only by the peers, maintaining consistency becomes more difficult.

The main problem associated with a P2P approach to the simulation is that there isn't a centralized point where all the events are kept ordered. A player generates a event and has to notify all the others players on the group. But the events are generated in an arbitrary order and it's very difficult to maintain game consistency. All players will end up with unsynchronized versions of the simulation. Keeping all the local simulation synchronized requires a total event order within the group. One possible solution is proposed on [Cecin et al. 2006].

One optimization present on this layer that can help to reduce the amount of data the server has to handle in the social space is to determine an area of interest [Morse et al. 2000] for the player, and to adapt the frequency of updates messages to that area. All the information regarding players and environment that are far away from the player and can't affect him is sent less frequently than objects that are inside an area of interest surrounding the player. The size of this area is game-dependent. Identifying the best way to add this optimization on our architecture is a work in progress.

The server-side isn't necessarily represented by only one server. It's common to a player to be able to choose between multiple servers. All these servers belong to the same company, but differ on geographic location, maximum number of players playing and even network latency and bandwidth. However, the only responsibility the player has is to choose among one of the available servers: his account, which includes his character traits and his virtual goods should remain the same on all servers. Multi-server functionalities will be present on the Layer 2 to allow such system. Multi-server support is still being designed and implemented into the P2PSE architecture.

Everything mentioned so far collectively form the core of our proposed game architecture. An application that follows the model proposed could be builded using only that set of functionalities. However, when dealing with gaming, especially network gaming, cheating becomes a key problem [Kabus et al. 2005]. The architecture would be incomplete without some sort of tools to prevent any illegal modification on the game that favors one or more players. Cheating in multiplayer games can be carried out in many ways, one of them being the illegal modification of network messages.

The architecture will include support for marking messages so that, for instance, a malicious peer that is relaying messages from two other peers, at Layer 1 level, will not be able to modify the message. Optionally, it will be possible to also cypher the message so peer relaying messages won't be able to understand them.

If a player alters the messages, he could alter the behavior of his character and/or his traits within the game. A common modification is movement cheats. They can be either excessive speed, far beyond what's allow on the game, or "teleportations" through the map. In order to identify players cheating in such a way, the uncommon movement pattern by the player has to be detected. The P2PSE architecture will use Artificial Neural Networks (ANN) to prevent movement cheating. To automatically detect that a pattern has been broke, it is necessary to record a sample of the player's moves, and use it as input for a ANN. But there is a lot of noise and instability on network messages. To prevent the ANN to give false positives, a training is needed ( feeding the ANN with a correct pattern, specific to the game). Every player in a P2P group will monitor each other, and if a cheat is detect, the server will then take any necessary measures. The exact details of the AI module are being researched.

## 4 Discussion and related work

For comparison purposes, our project is a solution to the problem of supporting massively multiplayer on-line games with a decentralized (albeit partially) topology. There are several publications that offer solutions to this problem. The solution shown here attempts to solve the security, consistency and scalability problems at the same time [Schiele et al. 2007]. The security problem is that, when an MMOG decentralizes its simulation, it start relying on client (untrusted) machines for critical computations, and opportunities for players to cheat at the game are bound to appear. The scalability problem is dealt with ensuring that server machines are not overloaded, and that the peer-to-peer overlay (logical) topology chosen scales well. The consistency problem is ensuring that game players have a view of the game world that is sufficiently well synchronized with the views of other peers.

To achieve scalability and security, the architecture borrows from the game metaphor of Guild Wars, which separates the player experience in *action* spaces and in *social* spaces. By only decentralizing the simulation of action spaces, the security problem is greatly simplified, as all player goods trading is done on social spaces. Scalability is achieved since the social space is expected to have very few interactions (chat messages and low-fidelity 3D avatar interactions) and so running it as a server-side simulation doesn't incur significant costs. As for the consistency problem on the action spaces, there are solutions proposed to cope with that problem, so the

problem ends up being to find one that fit our model properly.

Of course, having a fully decentralized MMOG that is as secure, consistent and scalable as a partially decentralized MMOG will always be a good research objective. The more decentralized, the better, if all necessary requirements [Schiele et al. 2007] are met. Works that attempt full MMOG decentralization include NEO [GauthierDickey et al. 2004], SimMud [Knutsson et al. 2004], Mercury [Bharambe et al. 2002], Dynamic microcell assignment [De Vleeschauwer et al. 2005] and others. NEO is actually an event-ordering mechanism which is part of a larger MMOG model in development, which aims to be scalable and fully decentralized by using of distributed hash tables (DHTs).

Although we don't have corroborative results yet, it still possible to analyze the model proposed and elaborate some reasonable expectations. The main problem with client-server games is that the simulation processing is done on the server.So, besides security anf authentication issues, the server has to cope with keeping all the players updated and the virtual world consistent. Changing the model to a P2P approach transfers all that load of work to the peers, and makes the cost of the server inherently lower on our proposed architecture. Even if elaborated security and authentication measures are implemented, it is very unlikely that they will inflict on the server-side a greater workload than the simulation of the game.

# 5 Conclusion

We proposed a different approach to MMOGs, that uses P2P groups and transfers the simulation processing to them. In order to provide security and scalability (that are relevant problems on a decentralized model), the game model was restrict to the instanced game model. The existence of a server-side ( as oppose to purely decentralized P2P MMOGs) guarantees that, if deem necessary, the server can act as final arbiter. In addition to that, that allowed for a layer organization to our software architecture, where each layer has a defined and simple job. Despite the choice of a specific game model, most of the techniques shown here could be adapt in order to be used on purely decentralized P2P MMOGs.

We expect to obtain more data using a simulation based on our implementation of the Layer 0. A log of the messages on the P2P groups and between client-server, and the analysis of the traffic will show how much information the server has to deal with, and if the expectations cited in the last section were satisfied. The simulation is still under development, but its basic structure is defined: one process will simulate the server-side, managing the groups and sending update messages with a certain frequency ( to emulate the social space). Another process will create several clients and connect with the server. The clients will be distribute among the social space and various action spaces. Clients in the social space will send an update message with a certain frequency either to the server, if they are in the social space (they will do that to emulate movement and chatting in the social space) or will send updates messages to all of others peers on his group. To give the simulation a more realistic behavior, artificial events (like the destruction of group) will occur based on a probability of occurrence within a period of time. For instance, every 10 seconds, there is a 3% chance of a group to be destroyed.

Future work includes finishing up the simulation and researching bibliography to find out realistics values to all the variables on the simulation (update frequency, distribution of clients between social and action spaces, etc).Besides, more details and results are expect, based that most of the architecture is still under development.

# References


ARENANET. Guild wars. `http://www.guildwars.com/`.

BHARAMBE, A., RAO, S., AND SESHAN, S. 2002. Mercury: a scalable publish-subscribe system for internet games. *1st workshop on Network and system support for games*, 3–9.

BLIZZARD. World of warcraft. `http://www.worldofwarcraft.com/`.

CECIN, F., GEYER, C., RABELLO, S., AND BARBOSA, J. 2006. A peer-to-peer simulation technique for instanced massively multiplayer games. *10th IEEE International Symposium on Distributed Simulation and Real-Time Applications -Volume 00*, 43–50.

DE VLEESCHAUWER, B., VAN DEN BOSSCHE, B., VERDICKT, T., DE TURCK, F., DHOEDT, B., AND DEMEESTER, P. 2005. Dynamic microcell assignment for massively multiplayer online gaming. *4th ACM SIGCOMM workshop on Network and system support for games*, 1–7.

ENTERTAINMENT, S. O. Everquest. `http://www.everquest.com/`.

ENTERTAINMENT, S. O. Planetside. `http://www.planetside.com/`.

GAUTHIERDICKEY, C., ZAPPALA, D., LO, V., AND MARR, J. 2004. Low latency and cheat-proof event ordering for peer-to-peer games. *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, 134–139.

KABUS, P., TERPSTRA, W., CILIA, M., AND BUCHMANN, A. 2005. Addressing cheating in distributed MMOGs. *4th ACM SIGCOMM workshop on Network and system support for games*, 1–6.

KNUTSSON, B., LU, H., XU, W., AND HOPKINS, B. 2004. Peer-to-peer support for massively multiplayer games. *IEEE Infocom*.

MORSE, K., BIC, L., AND DILLENCOURT, M. 2000. Interest Management in Large-Scale Virtual Environments. *Presence: Teleoperators and Virtual Environments 9*, 1, 52–68.

ORIGIN. Ultima online. `http://www.owo.com/`.

SCHIELE, G., SUSELBECK, R., WACKER, A., HAHNER, J., BECKER, C., AND WEIS, T. 2007. Requirements of peer-to-peer-based massively multiplayer online gaming. *7th IEEE International Symposium on Cluster Computing and the Grid*, 773–782.