Project Proposal

Philippe David   Ariel Vardi

February 16, 2006

1 Motivation and objectives

In only 5 years, the Massively Multiplayer Online Games (MMOG) have become the fastest growing segment of the gaming industry. At first, reserved to only the hardcore gamers, the market has increasingly reached a broader portion of the market. The latest MMORPGs boast thousands and thousands of users playing simultaneously on a same world. The antique Client/Server architecture tends to show its limits with such a number of clients. The bandwidth requirement is huge, as well as the CPU resources needed to process the tremendous amount of data in real time. However, an analysis of the traffic on the client side shows a very low bandwidth needed compared to other real-time games like the now famous Counter-Strike. This lead us to think that MMOGs could very well be implemented as P2P applications. Indeed, the low bandwidth, and the nature of the data transferred make this kind of game the ideal application for a distributed system.

However, despite the obvious drawbacks of a client/server architecture for this kind of applications, it also has numerous advantages. It namely enable the company hosting the server to have a very strong control over what is happening in the world they have created. Indeed, the centralized server receives all the data sent by all the players connected to the world which would not be the case in a P2P architecture.

Several concerns are raised by a P2P architecture. How can we prevent players from cheating? How can we guarantee a uninterrupted experience to the user when one of the peers fails and is removed from the network? How can we keep the gamestate consistent through all the peers in a decentralized architecture? And finally, how can we make sure that the network latency between the peers won't affect their experience?

These are all the issues that we will try to address in our study.

2 Related works

[1] and [2] provide several traffic figures and analysis. One of the most important result is about the spacial and temporal locality of information exchanged. Based on the fact that most of the traffic to and from players contains information that has an impact on players and objects in a close area around the player, [3] presents a commonly used technique to decentralize the communications in MMOGs which is to partition the world into multiple regions. All players in a particular region will communicate which each other to spread the information about who is doing what. This way, the most intensive activity of the central server is decentralized among playing peers. Doing this is not without issues, one of them being keeping a consistent game state among players. In a centralized solution, the server make everything move at a fixed heart beat and is able to detect conflicting actions. In a decentralized solution, this is very tricky since when a player receives a message from another player, the message was sent a few ms before. This can lead to a lot of inconsistency of the game state among players. For example if two players go toward a same point, they will most certainly receive a message saying that the other player is at the same point before detecting a collision.

[4] introduced and tested a robust solution based on a responsible node in each region, with backup nodes and an adaptive tree-like communication when the number of nodes increase in this region. This solution meets the expectations in terms of scalability, as it provides very goods performance results and does not suffer from the game state consistency problem. However it does not address one of the issue introduced by giving high responsibility to a few nodes, which is security and preventing these nodes from altering data.

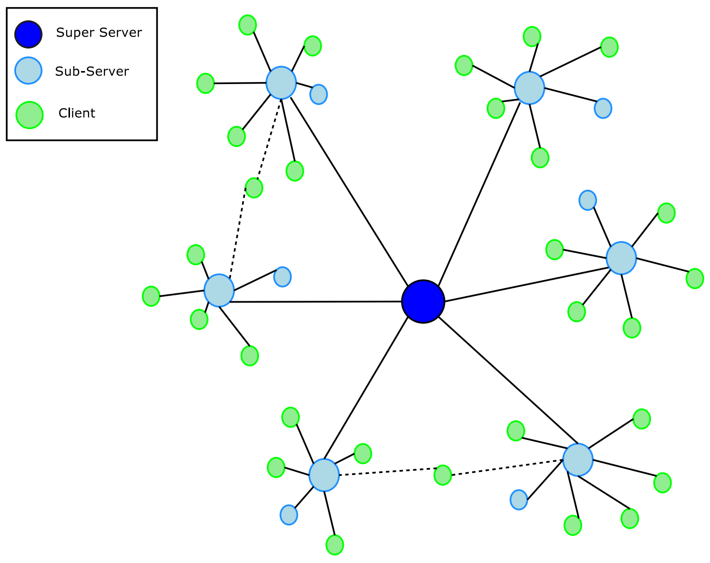
[5] introduce what they call booster boxes which are trusted servers run within the ISPs installations. This solution gives good latency results and the server is trustable but since the expensive communication goes through the trusted servers, their capacity must be scaled along with the number of players. This could eventually divide the prices of these costly servers among the game companies and ISPs but does not produce a solution that scales easily and automatically with the number of players.

[6] explored several multicast solutions to spread data in a region, based on minimum spanning trees and steiner minimal tree in networks with several heuristics. The game state consistency issue is addressed by using proxies. However his conclusion is that none of these multicast techniques is ideal for MMORPGs, so better heuristics has to be found.

3 Plan of action

We plan to develop an infrastructure with a simple server holding persistent data about the world and characters and clients that are able to act very simply in this world with respect to the rules. We chose to study two topologies, both based on the partition of the world and a group of peers in each region.

The first topology will be based on [4]: a responsible node in each region with backup node. Our goal is to study what power and bandwidth capacity a peer should have to be elected responsible or backup node and what percentage of players need to meet this capacity in order to have enough peers that can play these roles in the game. We will also research how to make this solution secure if the responsible node or its backup nodes try to cheat.

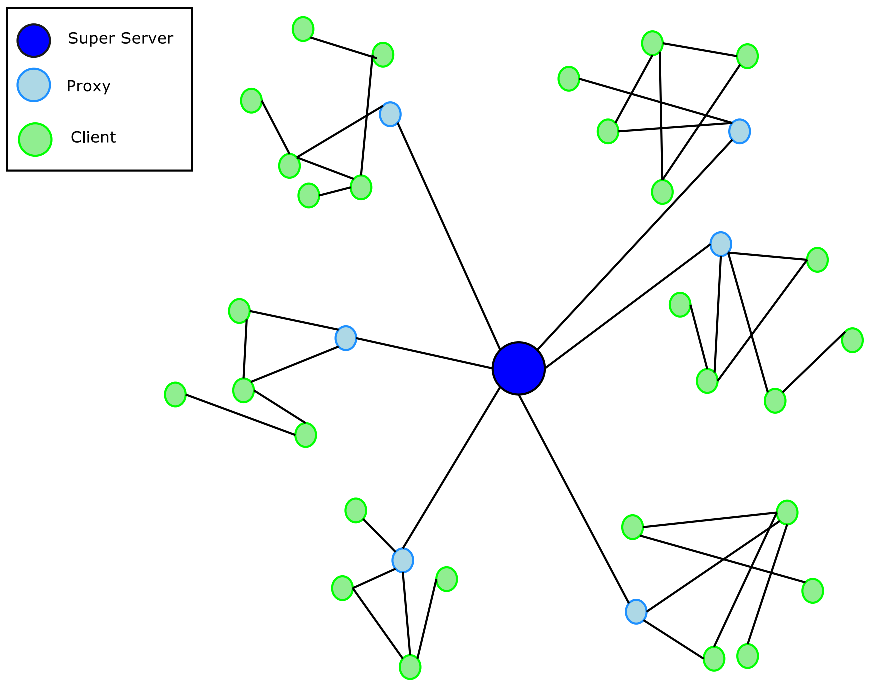


The clients are connected to the subservers which are also clients of the games. The subservers report to the Super Server. When a client is geographically located between two Subservers, it is temporarily connected to two subservers before switching to one of them only (dotted line). The blue clients are the backup subservers that will manage the clients in the area if the main subserver fails.

Figure 1: Topology 1

The second topology will be based on [6]: each group of peers in a region use a more decentralized communication style. The study will focus on the topology that can be used to keep a low latency and a coherent game state.

We plan to develop a design in which peers communicate directly with their closest neighbors and accept to receive information about further players through a few hops. To prevent inconsistency in the game state, one or several proxies in each region will interact with the concerned peers when some poten-



Peers communicate directly with their closest neighbors and accept to receive information about further players through a few hops. To prevent inconsistencies in the game state, one or several proxies in each region will interact with the concerned peers when some potential conflicts are detected.

Figure 2: Topology 2

tial conflicts are detected. Determining the exact topology to achieve this is part of the project.

In these two options we will also study what the optimal number of peers are in each region, and how much bandwidth these solution require for each peer or type of peer.

Our project will span on 9 weeks starting from February 16th and ending on April 20th.

- Week 1: Finalization of the specifications
- Week 2: Design of the data structures
- Week 3-4: Design and implementation of the client/server
- Week 5-6-7: Design and implementation of the sub-servers, dynamic load balancing, trust grading mechanism

- Week 8-9: Test and writing of the final report

The project will be written in Java and will be hosted if possible on a cluster hosted by Georgia Tech. Otherwise, our personal machines will be used.

4 Evaluation and Testing method

During the last two weeks of our project, we will evaluate the efficiency of our solution in terms of network usage, CPU load, as well as cheating prevention.

A comparison between our architecture and a more traditional one will be done. We will then see if we have been able to improve the efficiency of the system while preserving the coherence of the game-state and by preventing cheating.

To measure network usage, a solution based on the tcpdump tool will probably be used. Several cases of network failure will also be simulated in order to test the resiliency of our system. The CPU load will be measured on every type of peer and be compared to the client and server CPU load on a traditional architecture. Finally, we will see how our application behaves when a player tries to cheat.

5 Bibliography

1. Traffic Characteristics of a Massively Multi-player Online Role Playing Game, Jaecheol Kim, Jaeyoung Choi, Dukhyun Chang, Taekyoung Kwon, Yanghee Choi, Eungsu Yuk

2. Game Traffic Analysis: An MMORPG Perspective, Kuan-Ta Chen, Polly Huang, Chun-Ying Huang, Chin-Laung Lei

3. Peer-to-Peer Support for Massively Multiplayer Games, B. Knutsson, H. Lu, W. Xu, B. Hopkins

4. A Distributed Event Delivery Method with Load Balancing for MMORPGs, Shinya Yamamoto, Yoshihiro Murata, Keiichi Yasumoto and Minoru Ito

5. Network Infrastructure for Massively Distributed Games, Daniel Bauer, Sean Rooney, Paolo Scotton

6. Game State and Event Distribution using Proxy Technology and Application Layer Multicast, KnutHelge Vik