A Generic Proxy System for Networked Computer Games

Martin Mauve
University of Mannheim
L15, 15
68161 Mannheim, Germany
mauve@informatik.uni-mannheim.de

Stefan Fischer
TU Braunschweig
Muehlenpfordtstr. 23
38106 Braunschweig
stefan.fischer@tu-bs.de

Jörg Widmer
University of Mannheim
L15, 15
68161 Mannheim, Germany
mauve@informatik.uni-mannheim.de

ABSTRACT

In this work-in-progress report we present the general outline of a research project which aims at providing proxy support for networked computer games. The problems of both client-server and fully replicated architectures are discussed and we reason that employing proxy technology, which has been successfully used for other networked applications, is advantageous for this class of applications as well. In particular we describe how a proxy system for networked computer games can help with providing congestion control, achieving robustness, minimizing the impact of network delay and providing fairness.

Keywords

Networked Computer Games, Proxy, Congestion Control, Robustness, Latency Optimization, Fairness

1. INTRODUCTION

Networked computer games form a very challenging class of distributed applications. They require low network latency and fairness, they have to be protected from players who do not want to follow the rules, and the data that they transmit usually cannot be scaled down and should not be delayed. Furthermore, many online games such as Ultima Online [11], Everquest [13] and Diablo II [5] have been so popular that it took several months after their initial launch to provide sufficient bandwidth and server capacity to handle all the players.

Currently there are two different architectures used for networked computer games. Commercial products are typically based on a centralized client-server architecture, where a central server maintains the game state. The game clients run on the computers of the individual players. They receive information about the game state from the server and may change it by sending their local actions to the server. In the academic environment, there have been prototypes of networked games that rely on a replicated architecture without a centralized server, e.g., MIMAZE [6]. In this architecture, all applications cooperate to maintain a shared game state.

As we shall discuss, both alternatives exhibit significant problems when used for large scale games over very heterogeneous networks (such as the Internet). We therefore propose to use a proxy architecture which combines the advantages of the centralized and distributed approaches. The main idea of this architecture is to move intelligence and server functionality to the border of the network. This concept has been tremendously successful for other networked applications, such as web-access or multimedia content distribution [1].

2. EXISTING ARCHITECTURES

Commercial networked computer games generally employ a centralized client-server architecture. This is true for games with many thousands of players participating in the same game (e.g., Ultima Online, Everquest, and Diablo II) as well as games with only a very limited number of players (e.g., Quake [12]). The main benefits of this architecture are quite compelling: maintaining the consistency of the game state is relatively easy when a server holds the master copy of the state. For the same reason, illegal manipulations of the game state by the players are extremely difficult. Finally, conceiving business models for this architecture is fairly straight forward – one can charge for the access of the server or sell the server software.

Since all these issues are vital to the success of commercial networked computer games, developers of such applications typically accepted the quite substantial drawbacks: a centralized server is a bottleneck for the overall game. Almost all games with a large number of players have had major problem coping with this for the first couple of months of their operation. Latency for the actions of remote players is typically very high, since all actions will be sent to the server before they are distributed from there to other players. This information might therefore take a very long detour. At the same time, players with different network delays are treated unfairly. Players with a small latency to the server usually have an advantage over other players with higher latencies. Finally, due to the high demands interactivity poses on the information dissemination, there are no

appropriate congestion control mechanisms for this application class. This makes the deployment of large scale networked computer games a significant hazard to the service provider hosting them.

Distributed architectures for networked computer games maintain the game state in the game instances that are run by the players. Thus, there is no need for a central server. The application instances communicate directly with each other, using IP multicast if available. This approach has been used primarily for academic research projects such as MIMAZE [6], DIVE [7] or MASSIVE [8]. The main advantages of this architecture are that there is no central bottleneck and that player actions are transmitted directly to the application instance of the other players, avoiding detours that would otherwise increase the latency.

However, even though research prototypes have proven that a distributed approach is superior with respect to those two aspects, this architecture is generally not used in commercial products. The main reason is, that for commercial purposes the benefit of easier manageability of the centralized architecture outweighs the advantages of a distributed gaming framework. Furthermore it has been discussed in [10] that in the current IP multicast environment it is difficult to effectively consider the different interests of receivers in large scale sessions, i.e., to make sure that receivers get only the data they are interested in.

Summarizing, it can be reasoned that neither centralized nor distributed architectures are sufficiently well suited for large scale commercial computer games. Both exhibit substantial problems, either from a scaling or from a deployment perspective. We therefore propose to investigate new architectures that can mitigate the aforementioned deficiencies and try to combine the benefits of both approaches.

3. PROXY ARCHITECTURE FOR NETWORKED COMPUTER GAMES

The aim of our research project is to investigate the suitability of a proxy architecture for the application class of networked computer games. As depicted in Figure 1, a central server is in charge of maintaining the game state, being the final authority on consistency and prevention of illegal state manipulations. A client may connect directly to the server and communicate in the same way as in a centralized client-server architecture. However, a player may also connect to a proxy instead of the main server. It is expected that such proxies are located close the players (e.g., at their respective service-providers). The proxies can be thought of as extensions of the server. The server can trust them to a certain degree, since the proxies are not under the control of the players. Consequently, some server functionality can be delegated to the proxies and is therefore located closer to the players. Furthermore, the proxies themselves are interconnected. Therefore they form an overlay network which may be used to alleviate some of the inherent problems of a centralized architecture.

It is the aim of our research project to develop such a proxy system and investigate to what extend it can improve upon conventional architectures for networked computer games. In particular, this architecture can help to provide congestion control, enhance robustness, minimize the impact of network latency, improve fairness, and protect against cheating. For these components we aim to realize generic functionality which is usable by any networked computer game. In addition, the dynamic distribution of game dependent functionality to the proxies may further increase their flexibility and possible uses.

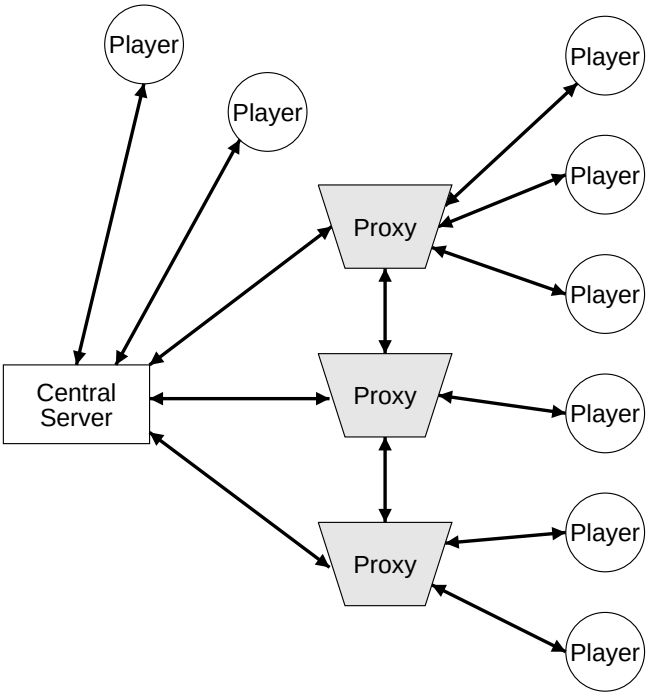

Figure 1: Proxy Architecture

3.1 Congestion Control

Realizing an appropriate mechanism for congestion control in networked computer games is difficult since these applications do not tolerate delayed transmission of information as it is done for example by TCP. On the other hand, the amount of data sent and received by a player can usually not be scaled as is done to adjust the rate of an audio or video transmission to the current network conditions.

The only viable mechanism for congestion control is to resort to some form of access control. One networked game where access control is used to safeguard against the overload of the server is Diablo II. This game has been so popular at launch that it was initially overrun by players, bringing the network and the servers to a complete halt. The developers were forced to add a hard limit to the number players allowed to play at any time. Not only is such a simple measure insufficient to solve the problem of network congestion and fair competition with other network traffic, but also it turned out that even rejecting incoming connection requests at the central server could exceed the server's capacity during peak times.

The presence of a proxy system can alleviate the problem of congestion. Since the proxies are interconnected it is often

possible to route traffic around congested areas. In case all possible paths to a server are congested, or the server cannot handle the current load even with the help of the proxies, individual proxies can perform access control. This allows the server to continue operation without being flooded by connection requests. The proxies only admit additional players as long as the generated traffic is fair to competing flows and can be handled by the server.

3.2 Robustness

In addition to using the overlay of proxies to avoid congested areas of the network, it can improve the overall robustness of the architecture. It was shown by Andersen, Balakrishnan, Kaashoek, and Morris [2] that an overlay network can significantly improve network fault detection and recovery. Their twelve node overlay network could recover from all of the measured 32 network outages by routing over an intermediate overlay node. Fault detection and rerouting took on the order of several seconds while conventional routing mechanisms often take minutes before they settle on an alternative route (if at all).

Today, network outages are relatively common and even in a well engineered networks they are hard to prevent completely. If the network of proxies can detect such outages sufficiently fast, transparently rerouting clients to a different proxy can hide faulty network behavior from the user and maintain playability even during partial network outages.

3.3 Latency Optimization

In order to investigate latency it is important to understand that there are two different ways in which the state of the game can change non-deterministically: The first is a change caused by user actions and the second a change through random events.

In a client-server architecture, user actions must be brought to the attention of the server and later on to the game clients of the other players. The latency of a user action is determined by the time it takes for the action (or its effect) to be received by the remote players. This includes its transmission to the server and the distribution from the server to the other clients. In the presence of a proxy architecture the latency can be reduced significantly: it is now possible for the local proxy of the player who performed the action to transmit this information to the server as well as directly to the proxies of other players. This avoids an unnecessary detour before the action is distributed to the other players and thus reduces perceived latency. In order to determine those players interested in a given message existing grouping strategies for network games could be used [14, 3].

Random events in a client-server architecture are typically transmitted by the server at the point of time when they occur. Therefore they will arrive late at the clients, delayed by the network latency. While a server could usually transmit these random events ahead of time (e.g., in form of random number seeds), clients would then be able to decode the additional information ahead of time and gain an unfair advantage over other players. With the help of proxies, this form of cheating can be easily prevented by sending random events to the (trustworthy) proxies ahead of time. Since proxies should be placed close to the game clients, they can forward a random event at the time it takes place without unduly delaying event execution.

3.4 Fairness and Cheating Prevention

In a distributed application where the participants do not cooperate to solve a common problem but compete against each other, three closely related characteristics are desirable:

- All users must have the same chance of winning the competition.
- It must be possible to prevent or at least discover non-rule-conforming client behavior.
- It must be possible to identify players who do not conform to the rules in order to prevent further participation in the game.

We believe that these issues will be significantly easier to address with the help of proxies that are not under the control of the players and are therefore more trustworthy than the gaming applications running on the players' computers.

Fairness in networked computer games can only be judged in the context of a notion of time. Essentially, it is necessary to determine who issues operations at what point in time, independently of the network delay between players or between a player and the server. To reach this goal, each user action is associated with a *timestamp* which is used to determine the ordering and temporal distance of user actions. In order to prevent cheating, in a centralized architecture the timestamp of a user action is typically considered to be the time of the arrival of that operation at the server. This creates unfairness due to the different network latency that individual players experience between themselves and the server.

In a distributed architecture, the timestamping of user actions becomes even more problematic. Baughman and Levine propose the use of a discrete time, which is divided into time frames [4]. Each user of a distributed application has the opportunity to execute at most one operation during a time frame. However, this is quite problematic since high network latency or underprovisioned clients can lead to an unnaturally slow execution of the operations which is not acceptable for many games.

With a proxy architecture a solution to this problem is rather straight forward: the proxy timestamps every user action. Thus, players can no longer gain an advantage by modifying the timestamps of their actions. Furthermore, the proxies can make sure that operations of remote players are not played out too early by a game client. This can simply be done by buffering operations of other players until they should be visible to a player served by the proxy.

The same rules apply to other (potentially data-intensive) game information that needs to be made available to a client at a specific point in time but not before. A typical example for such data is a level map that is often distributed to the clients as a whole, even before it is completely explored,

to reduce communication cost. Cheating players can now attempt to decode information that should not be available to them (an existing problems for games such as Diablo II). With a proxy architecture, revealing a map gradually as a player explores a level is a feasible alternative. It is no longer necessary to distribute any information to a client that should not yet be available.

Proxies can also be used to check if the behavior of a player conforms to certain rules. The server no longer has to perform this very time consuming tasks and can in turn support a larger number of simultaneous clients. Proxies can be used to detect whether the players use so called "bots" or try to manipulate the game's protocol; it acts as a special type of firewall for the game server. A gaming proxy can even act as a form of packet normalizer [9], preprocessing client information and ensuring that only valid client data is distributed to other clients and the server.

Once cheating or faulty behavior is observed by a proxy, it is necessary to identify the corresponding client to be able to take appropriate actions, such as terminating the connection to the client or preventing the client from connecting to the server again. Such a mechanism should not fail in the face of spoofed addresses or fake user IDs. Since proxies are located closer to the clients (e.g., at the same ISP) and are more likely to have spare capacity which can be used for tracking clients than the server, reliable identification of cheating clients can be greatly simplified.

The challenging task for all of the aforementioned mechanisms will be to keep the functionality as game independent as possible. Hence, it is necessary to allow a dynamic configuration of the proxies for a given game.[1] An important task in this context is to find a suitable format to describe these rules. XML is one example for a language that could be used for the description of these rules.

4. CONCLUSION

In this work-in-progress report we presented the general outline for a research project on generic proxy support for networked computer games. We reason that neither centralized client-server architectures nor fully replicated applications sufficiently solve all of the problems encountered in the area of large scale online computer games. By moving *server* functionality to the boundaries of the network we believe that several important problems of networked computer games can be solved, including congestion control, robustness, latency minimization, providing fairness, and cheat prevention. Information distribution strategies can be adjusted to the new architecture to only transmit game state and information to the clients at the specific point of time when this information should be made available. Through the placement of proxies close to the clients, it is further possible to move functionality from the *clients* to the proxies to gain more control over the players action, to more reliably prevent cheating, and to improve game performance.

[1]Nevertheless, to make an overlay network of proxies a commercially feasible alternative, proxies should be able to support a multitude of different game architectures simultaneously.

5. REFERENCES

[1] AKAMAI. Akamai homepage. www.akamai.com.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. 18th ACM SOSP*, 2001.

[3] J. Aronson. Using Groupings for Networked Gaming. http://www.gamasutra.com/features/20000621/aronson_pfv.htm, June 2000.

[4] N. E. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. In *Proc. Infocom*, 2001.

[5] Blizzard. Diablo II homepage. www.blizzard.com/worlds-diablo.shtml.

[6] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. *IEEE Networks magazine*, 13(4), July/August 1999.

[7] E. Frécon and M. Stenius. DIVE: A Scalable network architecture for distributed virtual environments. *Distributed Systems Engineering Journal*, 5(3):91–100, 1998.

[8] C. Greenhalgh, J. Purbrick, and D. Snowdon. Inside massive-3: Flexible support for data consistency and world structuring. In *Proceedings of the Third ACM Conference on Collaborative Virtual Environments (CVE 2000)*, pages 119–127, Semptember 2000.

[9] M. Handley, C. Kreibich, and V. Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proc. USENIX Security Symposium*, 2001.

[10] B. Levine, J. Crowcroft, C. Diot, J. Garcia-Luna-Aceves, and J. Kurose. Consideration of receiver interest for ip multicast delivery. In *Proc. of INFOCOM 2000*, 2000.

[11] Origin. Ultima online homepage. www.uo.com.

[12] I. Software. ID software homepage. www.idsoftware.com.

[13] Verant. Everquest homepage. www.everquest.com.

[14] L. Zou, M. Ammar, and C. Diot. An Evaluation of Grouping Techniques for State Dissemination in Networked Multi-User Games. In *Proc. of MASCOT*, 2001.