A novell interest management algorithm for distributed simulations of MMGs

Carlos Eduardo B. Bezerra, Fábio R. Cecin, Cláudio F. R. Geyer
Federal University of Rio Grande do Sul
Bento Gonçalves, 9500, Porto Alegre, RS, Brazil,
Email: {carlos.bezerra, fcecin, geyer}@inf.ufrgs.br

***Abstract*—Traditionally, a central server is utilized to provide support to MMGs (massively multiplayer games), where the number of participants is in the order of tens of thousands. Much work has been done trying to create a fully peer-to-peer model to support this kind of application, in order to minimize the maintenance cost of its infrastructure, but critical questions remain. Examples of the problems relative to peer-to-peer MMG support systems are: vulnerability to cheating, overload of the upload links of the peers and difficulty to maintain consistency of the simulation among the participants. In this work, it is proposed the utilization of geographically distributed lower-cost nodes, working as a distributed server to the game. The distribution model and some related works are also presented. To address the communication cost imposed to the servers, we specify a novell refinement to the area of interest technique, significantly reducing the necessary bandwidth. Simulations have been made with ns-2, comparing different area of interest algorithms. The results show that our approach achieves the least bandwidth utilization, with a 33.10% maximum traffic reduction and 33.58% average traffic reduction, when compared to other area of interest algorithms.**

I. Introduction

In the last years, electronic games have become very popular, specially massively multiplayer games, where the number of simultaneous players is in the order tens of thousands [1]. Worlf of Warcraft [2], Lineage II [3] and Guild Wars [4] are successful examples.

Usually, the network support for this kind of application consists in a central server with plenty of resources (processing power and available bandwidth), which accepts connections from the clients. Each player interacts through one of these clients, which sends his actions to the server, that processes them, calculating their influence over the game, and broadcasts the action's result to every client. Due to the number of simultaneous participants that this kind of game usually has, these tasks demand a significant amount of resources from the server, which receives and processes the actions from all the players and broadcast state updates to all of them.

Recently, some alternatives to the central server approach have been researched. One of them is to distribute among the own participants the game simulation and the task to broadcast the state updates when they perform actions. Communication between them is in a peer-to-peer manner, forming a decentralized network [5]. This approach would be the ideal if it did not have not some critical inherent drawbacks. For example, as the player computers participate on the simulation, they need to agree on the resulting state of the game match. If this agreement does not happen, some inconsistency on the game state among the players may occur.

There is another question, concerning the number of trasmissions that each peer needs to execute. In the client-server model, each client only needs to send his actions to the server, which processes them and broadcasts the resulting game state to every player. In the peer-to-peer model, each peer becomes responsible for processing its own player's actions and send the resulting state to everyone else to whom that state change might be relevant. The problem is that we cannot assume that every client has enough bandwidth to do this. Besides, without a central server, the game becomes dependant of the peers' simulation, which can be corrupted in order to result in an invalid state which benefits a player, or even invalidate the whole game session.

Besides the peer-to-peer model, another alternative is to use a distributed server, in which nodes connected between them divide the game simulation and the state update broadcasts [6]. Such approach allows for the use of lower-cost computers to form the distributed server system, reducing the cost of the support infrastructure. Also, consistency maintenance becomes easier than with the peer-to-peer model, because a single server decides the simulation for its assigned portion of the world, and problems related to malicious players may be abstracted, since the servers are able to verify the simulation and detect cheating. Finally, with less bandwidth and processing power requirements for the clients, the game becomes accessible for a wider public.

However, to avoid the distributed server system maintenance cost to be the same as the central server's, it is necessary to perform some optimizations, in order to reduce the necessary bandwidth for each server node. The present work proposes an algorithm to reduce the bandwidth usage caused by the game traffic between servers and clients, through a refinement of the players' interest management technique [7]. The basic principle of this technique is that each participant of the game receives only state updates that are relevant to them. Simulations have been made, comparing our proposed algorithm with conventional ones, obtaining significant results.

This paper is organized as follows: in section II, some related works with proposed distribution models are presented; in section III, some necessary definitions are made; in section IV, we propose a distribution model to be used as basis of our approach; in section V, we review some existing interest management algorithms; in section VI, we present our approach in detail; in sections VII and VIII, the simulation and its results, respectively, are described and in section IX, we present our final considerations.

II. Related Work

In the past few years, many works have been done trying to distribute the massively multiplayer games support. One approach is the peer-to-peer model, which has some issues concerning game state consistency among the the participating peers, vulnerability to cheating, and increased bandwidth usage by each peer. There are some proposals that try to minimize these problems. One of then is presented in [5], where is suggested the division of the virtal simulated game environment in regions, and inside each region, a peer is chosen to be the region coordinator. Its function will be manage the players' interest, checking to what peers each update really needs to be sent. In this way, the peers' upload bandwidth usage is reduced. However, this usage will be still greater than with a server to perform the broadcasts. In the peer-to-peer model, each player usually must send updates to more than one other player. Besides, the chosen peer must be trustable to be the peer coordinator.

Another work focused on the peer-to-peer model [8] uses an approach similar to the one presented in [5], but suggests, for each region of the virtual environment, the creation a "zoned federation" formed by peers chosen among the participants in that region. Since different nodes manage that region, and will need to agree in order to proceed the game, the simulation becomes more trustable. However, the risk of the chosen peers commit collusion cheating [9] is not eliminated. Besides, the agreement itself between the peers - that provides increased reliability - creates a fair amount of extra traffic between the participants of the federation, possibly delaying each simulation step.

A key problem in the peer-to-peer architectures, in what concerns the use of interest management, is that some peers are responsible for part of the simulation and for decide to whom each state update would interest. Assuming that there are only trustable players, the interest management technique may be useful. However, supposing that a player is malicious, he could hack his client software and avoid sending state updates to certain players, who would be in disadvantage. Consequently, the distributed server model is considered more suitable to use the interest management technique.

An example of this kind of model is described in [6], where a distributed architecture for MMGs is proposed. It is also based on the division of the virtual environment in regions, each one with a server node assigned to it. The player whose avatar is situated in a region of the game world should connect to the server responsible for that region. This way, each server would group different players, based on their locality on the virtual space. To achieve consistency between the different server nodes performing the simulation, it is used the concept of locks. When a server node needs to alter the state of some entity of the game, it first needs to obtain exclusive access to that entity. To do that, it negociates with the other server nodes which might also need to make some change in the same object, and then obtains the lock. When it finishes changing the state, it releases the lock and notifies the other servers

The first major restriction in the proposal of [6], however, is the assumption that all server nodes are connected with one another through a high velocity and low latency network, what cannot be assured when using lower-cost geographically distributed nodes. Another problem is that the scalability question is addressed simply via the expansion of the virtual environment area, supposing that the players will spread over it. Finally, they suggest to solve the hotspot problem through successive recursive environment partitioning, until the number of players per server is below a certain threshold. However, there is a practical limit to the repartitioning of the virtual environment, and they do not suggest what to do when this limit is reached.

III. Definitions

It is necessary to describe the network support model on which the proposed interest management algorithm is intended to be utilized. Throughout the text, some terms will be used, and their definitions are given here:

Avatar is the player representation in the virtual environment. Through his avatar, the player interacts with the game world and with the other players. Examples of avatars are the characters in MMORPG (massively multiplayer online role-playing games) games like World of Warcraft.

Entities are the constituent parts of the virtual world. Examples of entities are the avatars of the players as well as avatars controlled by artificial intelligence of the server - monsters of MMORPGs, for example - and objects in the environment, such as doors, weapons and items with which avatars can interact.

State is the set of properties that can be observed in the various entities of the game. The overall state of the simulated world is composed of individual states of the different entities present in it.

The players interact with the game world through **actions**. An action is a command of the player as, for example, move your avatar for particular location in the virtual world, attacking another player, take some object available in the environment and so on. In general, actions change the state of one or more entities in the game.

A **region** is a partition of the virtual environment, under the responsibility of a single server. Thus, players whose avatars are located in the same region will have its interaction improved, since their clients are connected to the same server.

The **border** between two regions is the line that divides the areas that these regions occupy. When an avatar is located near a border, the server responsible for the region beyond this border is notified about the presence of that avatar by the server where it is.

IV. Distribution Model

This work is based on a virtual environment partitioned into regions, each managed by a server. The regions are contiguous,

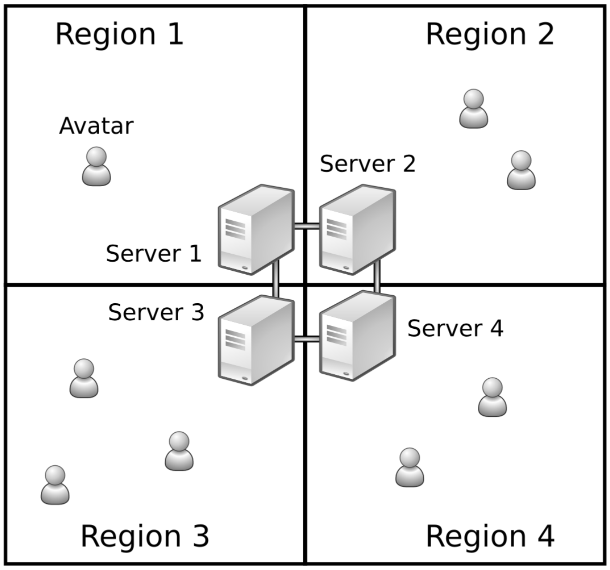

Fig. 1. Distribution model

exploring the locality of the players' avatars. Thus, avatars next to each other will probably be located in the same region and, therefore, their clients will tend to be connected to the same server, so that their interaction is faster (Fig. 1). Situations in which two players interacting with each other are connected to different servers imply a higher bandwidth usage, as it is required some sort of negotiation between the servers to which the different players are connected, in order to maintain the states of simulation in both servers identical. Also, the hop count of each message between the clients of these players is increased, since there is one more intermediary.

One issue that relates to that type of partitioning of the virtual environment concerns the borders between regions. If an avatar of a client connected to a server is close to the border of a region with another, which is managed by another server, it will be necessary to have an exchange of information between these servers. This information consists of state updates of the entities that are interacting with each other despite being located in different regions. For example, let S_A be the server responsible for the region R_A where is located the avatar of the client C_A and let be S_B the server responsible for another region, R_B, where is located the avatar of the client C_B. When the avatar of C_A gets close the border with R_B, S_A sends to S_B a message notifying it about the presence of that avatar near its region's border. If the avatar of C_B also gets close to the border with R_A and to the avatar of C_A, S_B sends another message, notifying S_A. Then, the avatars of C_A and C_B start exchanging state updates through their servers.

Regarding the simulation of the actions performed by players whose avatars are located in different regions, it must be decided how it will be performed. As the focus of this work is not the simulation itself, but the bandwidth usage optimization through a new interest management algorithm, it was decided that the simulation will be performed by the server to which the client of that player is connected. Thus, if the player whose avatar is in region R_i performs an action near the border involving entities in R_j, it is the server S_i who will decide the outcome of these actions, forwarding only the already calculated new state to S_j.

This way, the details of this mechanism will not result in relevant changes to the interest management. When a player J, whose avatar is near the border of a region, performs actions whose results must be broadcasted to players with their avatars in other regions, the server S, responsible for J, simulates his actions, calculates the resulting state and simply sends it to the neighbor server, as if it was sending to its own clients. Similarly, when S receives the resulting state of the action of a player who is connected to a neighbor server, it broadcasts the state to the clientes connected to it as if a player in its own region had executed that action.

V. Interest Management Algorithms

In order to provide an identical sense of the environment among the players, each one of them must maintain a copy of the entities' states, which must be the same for everyone. The simplest way to do this is to broadcast the states of all entities to all clients. The problem of this approach is that it generates a heavy traffic between the servers and clients, preventing the game to well, as the number of participants increase. To save bandwidth, both of the players, as the servers that intermediate them, a technique known as interest management is employed. This technique reduces the number of updates that the players will receive - and send, in the case of a peer-to-peer architecture.

In short, the interest management technique works as follows: for every state change of each entity, it is calculated for whom it will be important. For example, if an avatar is miles away from another in the virtual environment, it is most likely that their state changes is irrelevant to each other. Thus, it is not necessary for them to exchange information on their state. This principle - locality - is used as the main criterion in the interest management algorithms.

The algorithms described in the following sections are mainly based on the euclidian distance between each avatar and all other entities in the virtual environment. This could create a scalability problem due to the processing of the distances, but a distributed architecture is assumed, where the processing can and should be parallelized. In the distribution model defined previously, each server controls a region of the map. Therefore, each one of them manages only a subset of the entities of the game, checking only the distances between each pair of them, in addition to the entities that are in a neighbouring region, close to its border.

In the next sections, some versions of this technique, such as circular area based and avatar's field of view based interest management, will be presented. In the section VI, it is introduced the approach of attenuating the frequency of updates, and a proposed algorithm will be described in detail.

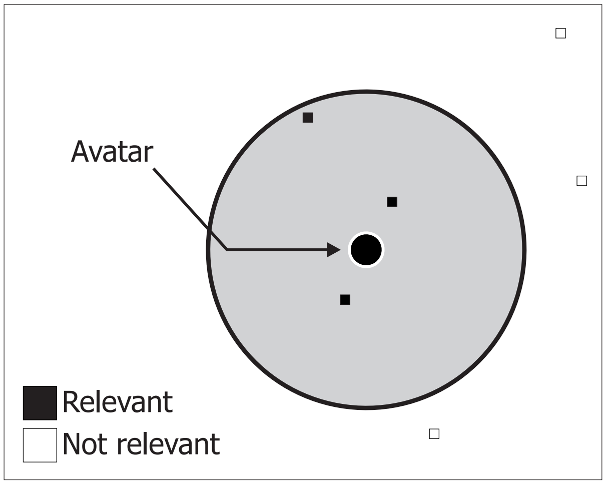

Fig. 2. Circular area of interest

A. Circular area of interest

The simplest way to execute interest management is to define a circular area, whose center is determined by the coordinates of the avatar's location in the virtual environment. After that, the euclidian distance between the avatar and every other other entity in the game world is calculated. Let A be an avatar, whose area of interest is a circle of radius rad. If an entity E is at a distance less than rad from A, then its state updates will be relevant to A. A will not receive state updates from entities which are at a distance greater than rad. Figure 2 illustrates this type of area of interest.

B. Field of view

A more refined method to manage the interests of the avatars is to take into account what each player can see. The area within which the player perceives state changes can be defined as a circular sector. This is similar to the circular area of interest described in the previous section, but takes into account that the player can only see objects that are located in front of his avatar.

One issue to be considered, however, is that the player will not receive state updates from entities entities which are close to his avatar, but behind it. This could cause some problems. For example, if the avatar turns 180 degrees, the player might not be able to see a particular entity that should be there, needing some time to receive its state information. This happens because, despite this entity have been near the avatar, the player did not receive its information yet because it was behind him, outside his field of view. In Figure 3 it is illustrated an area of interest that takes into account the player's view angle.

VI. Graded Area of Interest

The principle behind the approach proposed here is based on the fact that the more distant an entity is from the avatar in the virtual environment, the lower its update frequency for that avatar may be. Therefore, the state updates from entities which are more distant may be received with a longer interval

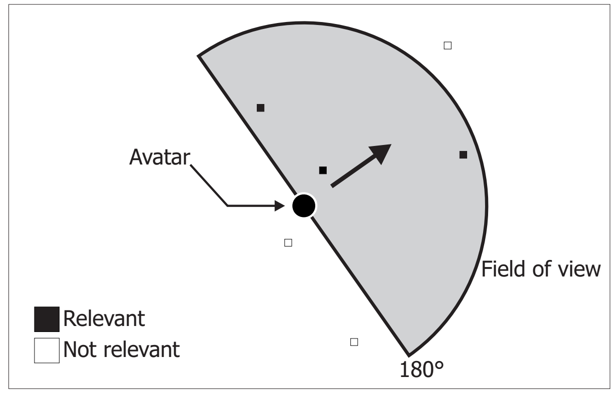

Fig. 3. Field of view based area of interest

between them. On the other hand, if an entity is very close, it is desirable that the player receives its most recent state change information as soon as possible, to view any alterations quickly.

To achieve this objective, it is necessary to define some parameters:

Relevance - real value ranging from 0 to 1 inclusive, which determines how much an entity's state is relevant to an avatar.

Update frequency - number of state updates, received by a player from each one of the entities in the virtual environment, divided by time.

Normal update interval - lowest time interval between the arrival of two consecutive state updates of the same entity to a client. The normal update interval is used when the state has a relevance value of 1. Thus, it determines the maximum update frequency.

View distance - determines the maximum distance from which an avatar may be from the entities so its player may see them.

Critical distance - it is the radius of the circle around the avatar, inside which all entities have a relevance value of 1.

Before sending the state of an entity to a client, the last transmission time is checked. The next transmission is then scheduled to occur after a certain interval. If the relevance of that state is 1, the normal update interval will be used. If it is less than 1, the normal interval is divided by the relevance. For example, let the normal update interval of a game be 200 ms. If the avatar A_i, which just sent a state update packet to A_j, is at a distance from A_i such that its relevance is 0.5, the next transmission will only happen after an interval of 200/0.5. So, the player controlling A_j will only receive an update of A_i within 400 ms. Despite this interval is still less than a half second, it represents a reduction of the state update frequecy of A_i in 50%. As they are at a greater distance from one another, and the interval was increased only by 200 ms, this variation will probably be imperceptible by the player who controls A_j.

It is important to note that the reduction of the entities' update frequency may be combined with other interest management techniques. In [7], various interest management algorithms are described, and they can be further improved

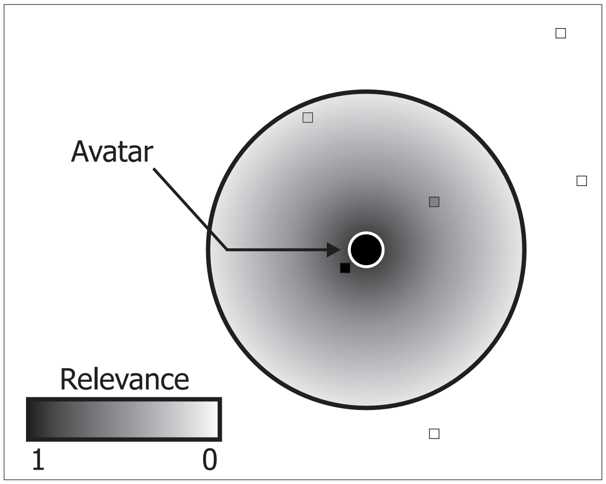

Fig. 4. Circular area of interest with update frequency attenuation

if the idea of different transmission intervals based on the relevance of the updates is used. Generally the state of each entity is classified into one of only two extremes: it is relevant or it is not relevant, ignoring the fact that there is a wide range of intermediate values. The question is how to define the relevance value for each state update. In the next sections, it will be presented two examples of algorithms which define a method to determine this value, as well as the area of interest used by each one. In section VI-B, it is specified the proposed A^3 algorithm, which employs, besides other principles, the update frequency attenuation. The simulations and their results are shown in sections VII and VIII, respectively.

A. Circle with attenuation

A simple form of using various update intervals based on the relevance value would be utilizing the circular area of interest. To obtain the relevance of an entity relative to an avatar, the value may be set to 1 when the entity is in the same position of the avatar and gradually reduce as it gets more distant, until it reaches 0 and stops decreasing no matter how more distant it gets. This is a way that, although simple, has shown a significant reduction in traffic between clients and servers. In Figure 4, it is illustrated how the area of interest would be, with entities' state update frequency attenuation for a given avatar.

B. A^3 algorithm

The interest management algorithm proposed in this article, A^3 (view angle with close area and update frequency attenuation), takes into account three main factors:

- Avatar's view angle, to determine which entities the player must be able to see because they are in front of his avatar and within the maximum view distance;
- Close area, whose purpose is improve the game quality in the space near the avatar. Its radius is the critical distance, defined previously;
- Update frequency attenuation.

The resulting area of interest takes then the shape of a circular sector, which represents the player's field of view. The origin of this circular sector is also the center of a smaller circle, which defines the close area of the avatar. In the close area, all entities have a relevance value of 1, so their update frequency to that avatar is set to the maximum value. This way, the most updated game state becomes available to the player in the area around his avatar, improving the interaction with entities next to it. Even if some of them are outside the player's field of view, he will be able to see them if his avatar turns rapidly to their direction. Figure 5 illustrates the area of interest which has just been defined.

As for entities that are outside the area nearby, but still within the avatar's field of view, their relevance must be calculated. It is proposed that the relevance of each entity decline gradually in accordance with the distance between it and the avatar in question. The further they are, the less frequent will be their state updates. This is possible because even if the interval update is doubled, it will most likely still be a fraction of a second, which is hardly noticeable for a player whose avatar is located at a great distance from the entity in question. Moreover, short delays between the arrival of updates of state can easily be masked by techniques of interpolation, such as dead-reckoning [10]. The algorithm 1 defines the operation of our interest management technique.

Algorithm 1 Calculate relevance of entity E to avatar A

$dist \leftarrow distance(A, E)$
if $dist \leq critical\_distance$ **then**
  $relevance \leftarrow 1$
else
  if A can see E in its field of view **then**
    $relevance \leftarrow 1 - \frac{dist - critical\_distance}{view\_distance - critical\_distance}$
    if $relevance < 0$ **then**
      $relevance \leftarrow 0$
    end if
  else
    $relevance \leftarrow 0$
  end if
end if

VII. Simulation

Para efetuar a simulação do algoritmo proposto, foi necessário primeiro criar um modelo de ambiente virtual a simular, com diversos avatares presentes, pois o algoritmo é baseado nas informações de localização e ângulo de visão. O ambiente consiste em um espaço bidimensional, que corresponde à região gerenciada por um dos servidores. Nela, há diversos avatares presentes, cujo número varia de uma simulação para outra. Cada avatar escolhe aleatoriamente um ponto de destino no ambiente e segue até lá. Ao chegar no destino, permanece parado por um tempo aleatório, que pode ser zero, e então escolhe uma nova localização para se dirigir.

Para simular a técnica proposta, foi utilizado o simulador de rede ns-2 [11]. Este simulador permite criar código específico

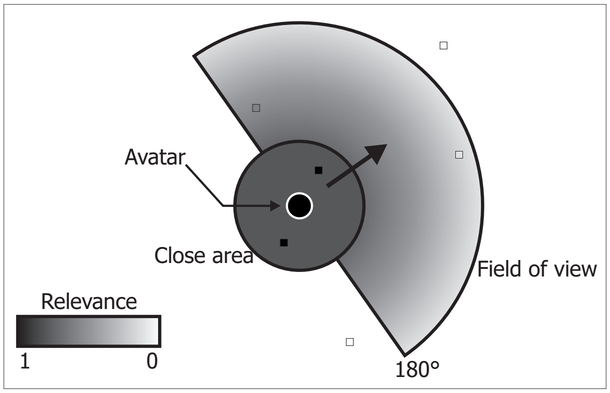

Fig. 5. A^3 area of interest

da aplicação que será simulada. No caso, foi simulado um servidor, que deveria enviar atualizações de estado para um cliente, responsável por um dos avatares na região. Baseado na localização dos outros avatares e no algoritmo de gerenciamento de interesse escolhido, o servidor decidia quais outros avatares tinham um estado relevante para o cliente em questão. Com isso, obtém-se a ocupação de largura de banda de envio necessária para um único cliente. Não se julgou necessário simular simultaneamente todos os clientes conectados àquele servidor, pois todos os avatares têm o mesmo perfil. Para encontrar a carga total no servidor, basta multiplicar a banda de envio necessária para um cliente pelo número de clientes presentes na região.

Outra questão é que o consumo de largura de banda de envio do servidor é muito maior que o de recepção - se ele recebe n ações, cada uma oriunda de um dos n clientes, é necessário, no pior caso, enviar $O(n^2)$ atualizações de estado, pois cada jogador precisaria do estado de todos os outros. Assim sendo, foi necessário apenas medir a banda de transmissão utilizada.

Em trabalhos como [12], [13] e [14], é analisado o tráfego de rede gerado por jogos em larga escala. Baseado nestes trabalhos, e adotando uma postura conservadora, foram decididos os seguinte parâmetros para serem utilizados na simulação:

- Intervalo normal de atualização: 250 ms;
- Tamanho do pacote de atualização de estado de uma única entidade: 100 bytes;
- Duração de cada sessão de jogo simulada: 20 min;
- Área do ambiente virtual: 750 x 750;
- Alcance da visão: 120;
- Distância crítica: 40;
- Ângulo de visão: 180°.

Foram executadas diversas simulações, com o objetivo de comparar os algoritmos de gerenciamento de interesse apresentados. O número de avatares presentes no ambiente foi uma das variáves analisadas, para verificar a escalabilidade. Os algoritmos comparados foram os baseados em círculo, círculo com atenuação, ângulo de visão e o algoritmo proposto, A^3. Para demonstrar o quanto cada um destes reduz o tráfego, foram feitas simulações também em que não é empregado nenhum tipo de gerenciamento de interesse, e o servidor envia para o cliente atualizações de estado de todas as outras entidades do jogo.

VIII. Results

Os resultados foram coletados da seguinte maneira: para encontrar a largura de banda utilizada em média para envio, foram somados todos os pacotes de cada sessão e dividido pelo tempo que foi simulado; para determinar a largura de banda máxima utilizada, foi verificado, segundo a segundo, quantos bytes foram enviados e foi selecionado o máximo.

Nas tabelas I e II, são apresentados os dados coletados de largura de banda máxima e média, respectivamente, utilizada com os quatro algoritmos simuladas - área em círculo (C), área em círculo com atenuação (C & A), área do campo de visão (FoV) e área do campo de visão mais área próxima mais atenuação (A^3) - além de mostrar quanto seria a largura de banda utilizada se nenhuma técnica fosse empregada (None). Os valores estão em bytes/s. Nas figuras 6 e 7 são mostrados os gráficos correspondentes.

TABLE I
Largura de banda máxima utilizada

Avatars	None	C	C & A	FoV	A^3
25	9400	8500	5700	7100	4700
50	19300	17000	10300	12300	8100
75	29100	23600	16600	17800	11300
100	38800	32500	20500	23000	15500
125	48600	37400	24300	29500	19700
150	58300	47400	29900	32900	22700
175	67700	56100	34300	32400	21500
200	77600	62300	37500	41200	28900

TABLE II
Largura de banda utilizada em média

Avatars	None	C	C & A	FoV	A^3
25	9221	4715	2759	2534	1700
50	18826	9350	5442	4949	3303
75	28432	13963	8315	7619	5137
100	38037	19324	11029	9928	6739
125	47642	23138	13871	12434	8290
150	57247	29031	16432	15085	10062
175	66853	34697	19661	23060	14250
200	76458	38600	23450	21491	14413

Apenas usando diferentes freqüências de atualização no gerenciamento de interesse baseado em círculo, reduziu-se em 41.59% a largura de banda de envio utilizada em média pelo servidor por cliente. A utilização máxima de largura de banda também foi reduzida, em 36.19%. Estes valores representam a média de redução de uso de largura de banda para os diferentes números de clientes.

No que diz respeito ao algoritmo proposto A^3, obteve-se uma redução de uso médio de largura de banda de envio de 63.51% e 33.58%, comparado respectivamente com o algoritmo de área de interesse circular e baseado em ângulo de visão. Reduziu-se também o pico de utilização em 52.03% e 33.10%, comparado com os mesmos algoritmos. Na tabela III, são mostrados os percentuais médios de economia de largura

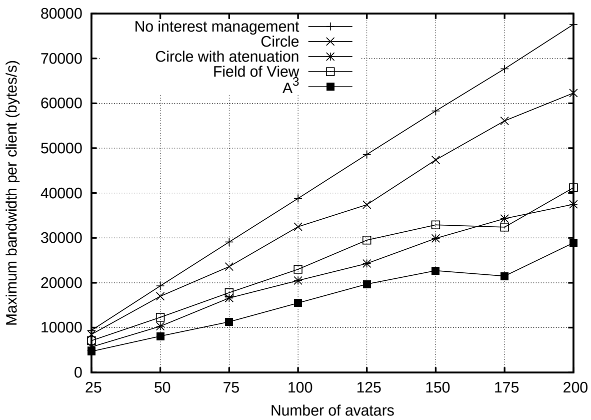

Fig. 6. Simulation Results: maximum bandwidth usage

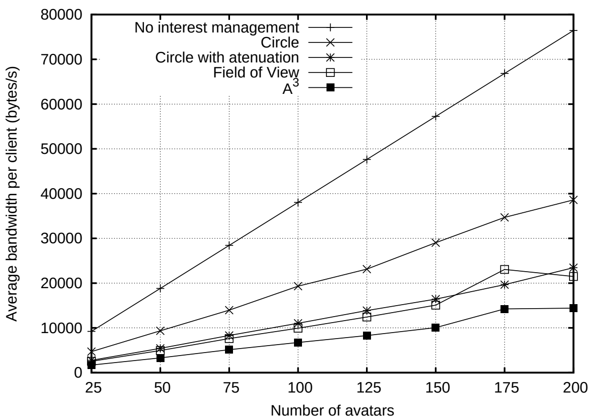

Fig. 7. Simulation Results: average bandwidth usage

de banda máxima e média com o algoritmo A^3, em relação aos outros algoritmos apresentados.

Observou-se também que os valores médio e máximo observados diferem, mesmo quando não é utilizado nenhum algoritmo de gerenciamento de interesse, ou seja, o cliente recebe atualizações de estado de todas as entidades presentes no jogo, com a freqüência normal. Além disso, com 200 avatares no ambiente, com estado de 100 bytes, cuja atualização é enviada a cada 250 ms, o servidor deveria alocar $199 \times 100 \times 4$ bytes/s para cada cliente, ou seja, 79600 bytes/s. No entanto, observou-se que a utilização máxima e média, com 200 avatares presentes e nenhum gerenciamento de interesse, foi de 77600 e 76458, respectivamente. Isso acontece porque o ns-2 é um simulador de eventos discreto, e o servidor simulado foi programado para checar o schedule de envios a cada 10 ms. Em conseqüência disto, cada atualização de estado pode ter tido seu intervalo aumentado em até 10 ms, o que explica os valores encontrados.

IX. Conclusion

Foi apresentado um algoritmo de gerenciamento de interesse, o A^3, cuja idéia principal é adaptar a freqüência de atualização de estado das entidades do jogo de acordo com

TABLE III
Economia de largura de banda com o algoritmo A^3

Utilização	None	C	C & A	FoV
Máxima	60.10%	52.03%	24.81%	33.10%
Média	81.64%	63.51%	37.48%	33.58%

sua relevância para o cliente que receberá as atualizações. O formato da área de interesse utilizada pelo algoritmo A^3 consiste em um setor de círculo, correspondente ao campo de visão do jogador, mais um círculo de raio menor, que corresponde à área próxima ao avatar daquele jogador. O objetivo deste círculo menor é o de manter o estado naquela região, que é considerada crítica, o mais atualizado possível. Somando-se essas características, chegamos a um algoritmo que obteve redução da utilização máxima da banda de envio do servidor de 52.03% e 33.10%, comparados com o gerenciamento de interesse baseado em círculo e em campo de visão, respectivamente, e de 63.51% e 33.58% de utilização média, comparados com os mesmos algoritmos.

References

[1] F. Cecin, R. Real, R. de Oliveira Jannone, C. Geyer, M. Martins, and J. Barbosa, "FreeMMG: A Scalable and Cheat-Resistant Distribution Model for Internet Games," *IEEE Int. Sym. on Distributed Simulation and Real-Time Applications*, pp. 83–90, 2004.

[2] Blizzard, "World of warcraft," 2004, http://www.worldofwarcraft.com/.

[3] NCsoft, "Lineage ii," 2003, http://www.lineage2.com/.

[4] ArenaNet, "Guild wars," 2005, http://www.guildwars.com/.

[5] G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis, "Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming," *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pp. 773–782, 2007.

[6] M. Assiotis and V. Tzanov, "A distributed architecture for MMORPG," *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, 2006.

[7] J. Boulanger, J. Kienzle, and C. Verbrugge, "Comparing interest management algorithms for massively multiplayer games," *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, 2006.

[8] T. Iimura, H. Hazeyama, and Y. Kadobayashi, "Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games," *Proceedings of ACM SIGCOMM 2004 workshops on NetGames' 04: Network and system support for games*, pp. 116–120, 2004.

[9] J. Yan and B. Randell, "A systematic classification of cheating in online games," *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pp. 1–9, 2005.

[10] J. Smed, T. Kaukoranta, and H. Hakonen, "A Review on Networking and Multiplayer Computer Games," *Turku Centre for Computer Science*, 2002.

[11] S. McCanne, S. Floyd *et al.*, "Network simulator ns-2," *The Vint project, available for download at http://www.isi.edu/nsnam/ns*.

[12] Y. Yu, Z. Li, L. Shi, Y. Chen, and H. Xu, "Network-Aware State Update For Large Scale Mobile Games," *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pp. 563–568, 2007.

[13] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, and E. Yuk, "Traffic characteristics of a massively multi-player online role playing game," *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pp. 1–8, 2005.

[14] P. Svoboda, W. Karner, and M. Rupp, "Traffic Analysis and Modeling for World of Warcraft," *Communications, 2007. ICC'07. IEEE International Conference on*, pp. 1612–1617, 2007.