A^3: a novel interest management algorithm for distributed simulations of MMOGs

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 MMOGs (massively multiplayer online 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 MMOG 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 game server. The distribution model and some related works are also presented. To address the communication cost imposed to the servers, we specify the A^3 algorithm, which is a novel refinement of the area of interest technique, significantly reducing the necessary bandwidth. Simulations have been made with ns-2 and their results demonstrate 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 MMOGs (massively multiplayer online games), where the number of simultaneous players is in the order tens of thousands [1]. World of Warcraft [2], Lineage II [3] and Guild Wars [4] are successful examples of MMOGs.

Usually, the network support for this kind of application consists of 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 actions' results to everyone. 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 broadcasts 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 transmissions 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 peer has enough bandwidth to do this. Besides, without a central server, the game becomes dependent on 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 to one another divide the game simulation and the state update broadcasts [6]. Such approach allows 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 the participants of the game need only to receive state updates that are relevant to them. Simulations have been made, comparing our proposed algorithm to existing 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 optimization; in section V, we review some existing interest management algorithms; in section VI, we explain the principle of using variable update intervals; in section VII, we present in detail the A^3 algorithm; in sections VIII and IX, the simulation and its results, respectively, are described and in section X, we make our final considerations.

II. Related Work

In the past few years, many works have been done trying to distribute the network support for massively multiplayer online games. One approach is the peer-to-peer model [8], [9], [10], which has some issues concerning game state consistency among 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 them is presented in [5], where it is suggested the division of the virtual simulated game environment into regions, and for each region, a peer is chosen to be its coordinator. The function of this peer will be manage the players' interest. It will check who really needs each state update, so each player does not receive every update packet. This way, the peers' bandwidth usage is reduced. However, it will be still higher than with a server performing all 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 region coordinator.

Another work focused on the peer-to-peer model [11] uses an approach similar to the one presented in [5], but suggests, for each region of the virtual environment, the creation of a "zoned federation" formed by peers chosen among the participants in that region. Since different nodes will work together, and thus will need to agree in order to proceed with the game, the simulation becomes more trustable. However, the risk of the chosen peers commit collusion cheating [12] 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 interests. 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 [6], [13], [14], [15] 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 MMOGs is proposed. It is also based on the division of the virtual environment into 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 negotiates 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 to one another through a high speed and low latency network, what cannot be assumed 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 hot spot 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's representation in the virtual environment. Through his avatar, a player interacts with the game world and with the other players. Examples of avatars are the characters in MMORPGs (massively multiplayer online role-playing 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 all the entities present in it.

The players interact with the game world through **actions**. An action is a command of the player as, for example, moving his avatar to a particular location in the virtual world, attacking another player, taking 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.

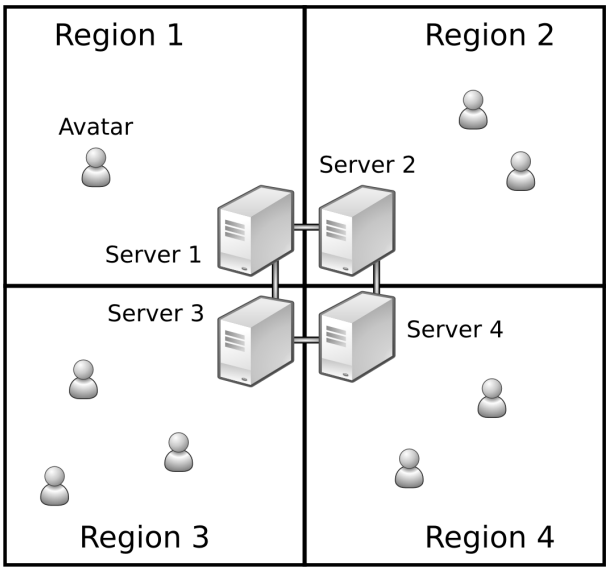

Figure 1. Distribution model

IV. Distribution Model

This work is based on a virtual environment partitioned into regions, each managed by a different server. The regions are contiguous, 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 (Figure 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 the simulation identical in both servers. 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 is close to the border between two regions, it will be necessary some exchange of information between the servers which manage them. 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_i be the server responsible for the region R_i where the avatar of the client C_i is located and let S_j be the server responsible for another region, R_j, where the avatar of the client C_j is situated. When the avatar of C_i gets close to the border with R_j, S_i sends to S_j a message notifying it about the presence of that avatar near its region's border. If the avatar of C_j also gets close to the border with R_i and to the avatar of C_i, S_j sends another message, notifying S_i. Then, C_i and C_j start exchanging state updates of their avatars through the 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 P, 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 P, 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 clients 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 local 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 scale well, as the number of participants increase. To save bandwidth, both of the players, as of 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 to whom it will be important. For example, if an avatar is miles away from another one in the virtual environment, it is most likely that their state alterations are 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 section VI, it is introduced the approach of attenuating the frequency of updates, and in section VII our proposed algorithm is described in detail.

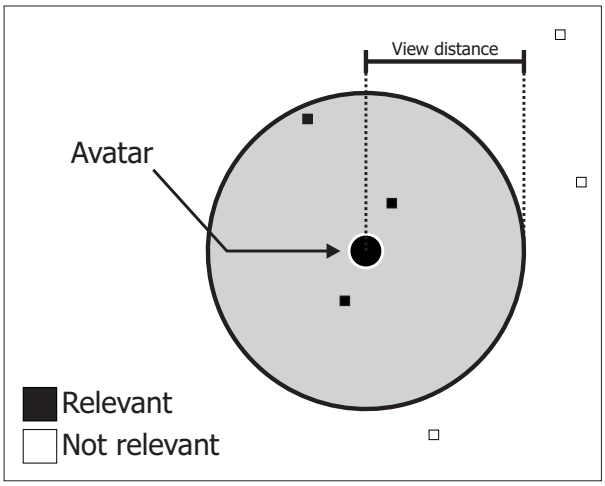

Figure 2. Circular area of interest

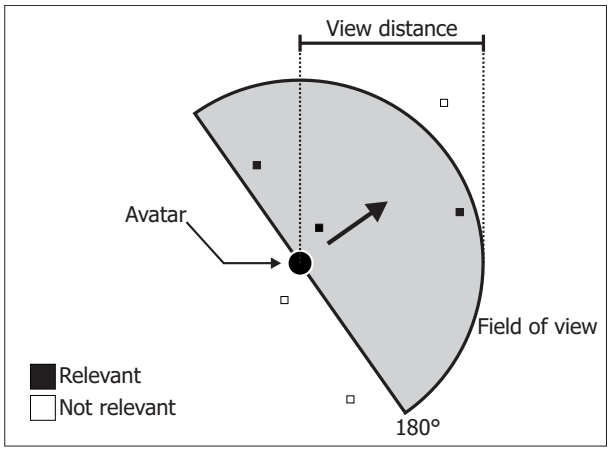

Figure 3. Field of view based area of interest

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 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 based area of interest

A more refined method to manage the players' interests is to take into account what each one of them 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 considers that the player can only see objects that are located in front of his avatar.

One issue to be observed, however, is that the player will not receive state updates from entities that 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, though the entity was near the avatar, it was outside the player's 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 between them. On the other hand, if an entity is very close, it is desirable that the player receives its most recent state 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 divided by 0.5. So, the player controlling A_j will only receive an update of A_i after 400 ms. Despite this interval is still less than a half second, it represents a reduction of the state update frequency 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

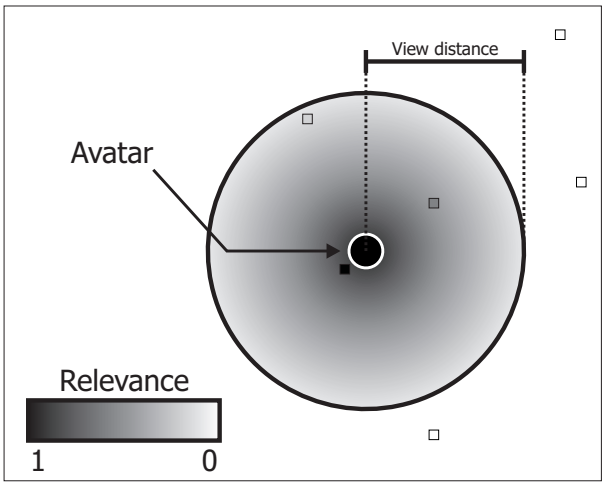

Figure 4. Circular area of interest with update frequency attenuation

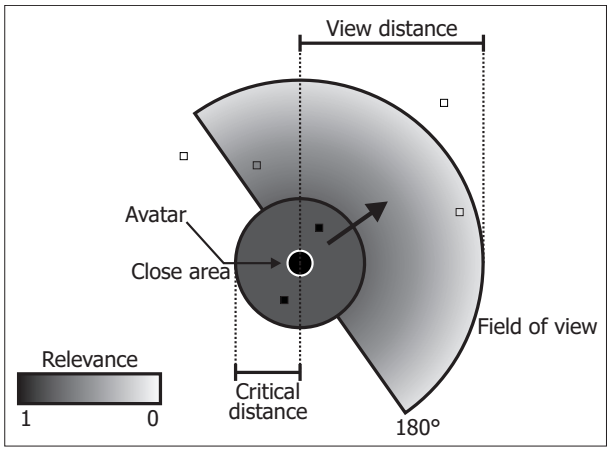

Figure 5. A^3 area of interest

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.

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 they are in the same position and gradually reduce it as the distance between them increases, until it reaches 0 and stops decreasing no matter how farther the entity goes. 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 this area of interest would be, with entities' state update frequency attenuation for a given avatar.

In the next section, it will be presented the proposed A^3 algorithm, which employs, besides other principles, the update frequency attenuation. The simulations and their results are shown in sections VIII and IX, respectively.

VII. The A^3 algorithm

In the previous sections, we have hinted that filtering based on an avatar field of view of 180 degrees, for instance, instead of using a full circle around it could result in significant reduction in update traffic from the server to the avatar player's machine. However, that would bring an important drawback, which is the large but temporary inconsistencies that would ensue whenever a player would turn around his avatars quickly to see what remotely-controlled objects are behind it. We have also reminded that reducing update frequencies for objects that are farther away from an avatar is a good optimization, especially for large virtual worlds where it may be on an open field with potentially hundreds of other avatars in its sighting distance. In this case, the player will not benefit at all from receiving updates at maximum frequency from objects that are sufficiently distant.

In our envisioned distributed MMOG scenario, the field of view optimization cannot be missed. We need to use the avatar's field of view information to further improve on filtering gains. Furthermore, as it has the mentioned drawback, we combined it with a smaller circle around the avatar. This addresses the field of view blind spot because, even if the game allows players to turn their avatars 180 degrees quasi-instantly, the only objects which will be affected negatively due to abrupt turning will be the more distant ones. Some time will be taken for the client to inform the server about the new facing angle of its avatar and to receive the state updates of the entities which were far behind it. Nevertheless, this delay will affect only distant objects, what is acceptable for most games, even action or shooter ones, which are the most demanding in terms of consistency due to presence of long-range hit-scan weapons, for instance.

Therefore, 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 to improve the game quality in the space near the avatar. Its radius is the critical distance;
- Update frequency attenuation.

The resulting area of interest then takes 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 area close to 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 up-to-date 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 described.

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 as the distance between it and the avatar in question increases. The farther 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 state updates can be easily masked by extrapolation techniques, such as dead-reckoning [16]. 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

VIII. Simulation

In order to perform the simulation of the proposed algorithm, it was first necessary to create a simulated virtual environment, with avatars present on it, since the algorithm is based on locality and view angle information. The environment consists of a two-dimensional space, which corresponds to the region managed by one of the servers. There are various avatars present, whose number varies from one simulation to another. Each one of them randomly chooses a destination point in the environment and then moves there. When it reaches its destination, it stays for a random time value, which may be zero, and then chooses a new destination to move to.

Ns-2 simulator [17] was used to simulate and compare the interest management algorithms described in this paper. This simulator allows the user to create code of the specific application which will be simulated. In our case, it was simulated a server managing a certain region where there was an avatar whose client should receive state updates of the other avatars present in that region. Based on their locations and on the chosen interest management algorithm, the server decided which other avatars had a relevant state to the client in question.

Through the simulations, it was obtained the server upload bandwidth usage for one client. It was not considered necessary to simulate every client connected to the server simultaneously because every one of them had the same behavior. To find the total upload bandwidth usage, we just multiply the value found in the simulation by the number of clients connected to that server. Also, there was no point in simulating or measuring the download bandwidth usage of the server, since it would keep receiving the players' actions at the same rate no matter what interest management algorithm was used.

Some works, such as [18], [19] and [20], analyse the network traffic generated by large scale games. Based on these analysis, and adopting a conservative posture, the following parameters have been decided to be used in the simulations:

- Normal update interval: 250 ms;
- State update for one entity: 100 bytes UDP packet;
- Duration of each simulated game session: 20 min;
- Virtual environment area: 750 x 750;
- View distance: 120;
- Critical distance: 40;
- View angle: 180°.

Several simulations were executed, in order to compare the described interest management algorithms. The number of avatars in the environment was varied in order to measure the scalability. The algorithms compared were the ones with circle, circle with attenuation and field of view based area of interest and the proposed algorithm, A^3. To demonstrate how much traffic reduction each one of these achieve, simulations in which no type of interest management was used - and the server sent to the client state updates of all entities in the environment - were also performed.

IX. Results

The results were collected as follows: to measure the average upload bandwidth usage by the server, the sizes of all packets sent in the session were summed up and then divided by the session duration; to determine the maximum usage, it was measured how many bytes had been sent each second and the highest value was selected.

In tables I and II, the collected maximum and average upload bandwidth usage data are presented for each number of simulated avatars and interest management algorithm - circular area based (C), circular area based with attenuated update frequency (C & A), field of view based (FoV) and A^3 algorithm - and it is also show how much would be the upload bandwidth usage if no technique at all was employed (None). In Figure 6 and Figure 7, the corresponding graphics are given.

Table I
Maximum bandwidth utilization (bytes/s)

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
AVERAGE BANDWIDTH UTILIZATION (BYTES/S)

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

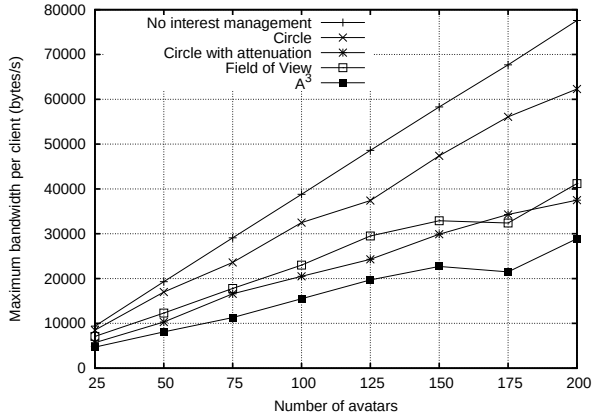

Figure 6. Simulation Results: maximum bandwidth usage

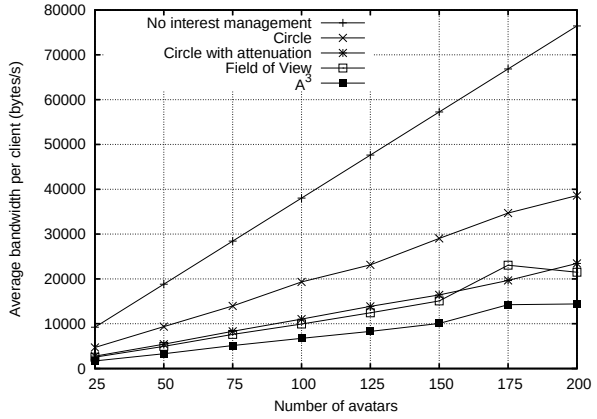

Figure 7. Simulation Results: average bandwidth usage

Just by using different update frequencies with circular area based interest management, the average upload bandwidth usage by the server decreased 41.59%. The maximum upload bandwidth usage was also reduced, decreasing 36.19%. These values represent the average usage reduction among all different numbers of avatars, compared to the circular area of interest algorithm with no frequency update attenuation.

Regarding the proposed algorithm, A^3, it has been obtained an average upload bandwidth usage reduction of 63.51% and 33.58%, compared to the circular area of interest and the field of view based algorithms, respectively. The maximum upload bandwidth usage was also reduced, decreasing 52.03% and 33.10%, compared to the same algorithms. Table III shows the maximum and average bandwidth saving percentage of the A^3 algorithm compared to the other techniques, for average and maximum usage. For example, the maximum bandwidth utilization with A^3 is 24.81% less than with a circular area of interest based algorithm with update frequency attenuation.

Table III
BANDWIDTH SAVING WITH A^3 ALGORITHM

Usage	None	C	C & A	FoV
Maximum	60.10%	52.03%	24.81%	33.10%
Average	81.64%	63.51%	37.48%	33.58%

It was also observed that the average and maximum usage values differ, even when no algorithm is used to manage interest, that is, the client receives state updates of all entities in the game, with maximum frequency. With 200 avatars in the environment, 100 bytes state update packet size and 250 ms normal update interval, the server should send 80000 bytes per second - $200 \times 100 \times 4$ - to each client. However, it was observed that the maximum and average usage, with 200 avatars present and no interest management, was 77600 and 76458, respectively. It happens because ns-2 is a discrete event simulator, and the simulated server was programmed to check the update schedule every 10 ms. As a result of this, each state update may have had its interval increased by up to 10 ms, which explains the values found.

X. CONCLUSION

In this work, we presented the A^3 interest management algorithm, whose main idea is to vary the state update frequency of the game entities according to their relevance to each client that will receive the updates. The A^3 area of interest consists of a circular sector, corresponding to the player field of view, plus a smaller circle, which represents the area close to that player's avatar. The goal of this close area is to maintain the game state inside it the most up-to-date possible, in order to improve the game near the avatar. Joining these characteristics, we obtained an algorithm that achieved a significant reduction of the maximum upload bandwidth utilization by the server, which decreased 52.03% and 33.10%, compared to circle and field of view based interest management algorithms, respectively. The average upload bandwidth utilization by the server also decreased 63.51% and 33.58%, compared to the same algorithms.

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] K. Endo, M. Kawahara, and Y. Takahashi, "A distributed architecture for massively multiplayer online services with peer-to-peer support," *Proceedings of NetCon?'05*, 2005.

[9] T. Hampel, T. Bopp, and R. Hinn, "A peer-to-peer architecture for massive multiplayer online games," *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, 2006.

[10] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," *IEEE Infocom*, 2004.

[11] 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.

[12] 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.

[13] B. Ng, A. Si, R. Lau, and F. Li, "A multi-server architecture for distributed virtual walkthrough," *Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 163–170, 2002.

[14] S. Fiedler, M. Wallner, and M. Weber, "A communication architecture for massive multiplayer games," *Proceedings of the 1st workshop on Network and system support for games*, pp. 14–22, 2002.

[15] S. Rieche, M. Fouquet, H. Niedermayer, L. Petrak, K. Wehrle, and G. Carle, "Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games," *Consumer Communications and Networking Conference, 2007. CCNC 2007. 2007 4th IEEE*, pp. 763–767, 2007.

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

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

[18] 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.

[19] 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.

[20] 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.