2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies

Occlusion Based Message Dissemination Method in Networked Virtual Environments

Kari Vatjus-Anttila, Timo Koskela, Seamus Hickey and Jarkko Vatjus-Anttila
Center for Internet Excellence
University of Oulu
Oulu, Finland
Email: firstname.lastname@cie.fi

***Abstract*—Virtual spaces that host hundreds or even thousands of users create enormous amounts of traffic, which usually is broadcasted to the entire environment and congests the network medium very quickly. In this paper, a novel interest management (IM) algorithm called EA^3 is presented and evaluated. EA^3 is based on an existing IM algorithm, called A^3 which uses euclidean distance to discover entities of interest. A^3 was improved by utilizing a ray visibility algorithm, which helps to filter out traffic originating from entities behind obstacles. The evaluation of EA^3 and A^3 was conducted using an extensible networked virtual environment called realXtend. The results show that, when either EA^3 or A^3 is used, the overall network throughput can be lowered significantly. In a city-like test scene, the drop was approximately 50 per cent with A^3 and 62 per cent with EA^3 compared to the situation when no IM algorithm was used. In terms of CPU load EA^3 is slightly heavier to use when there is a lot of open space inside the virtual environment.**

Keywords*—3D; virtual world; virtual space; interest management; performance evaluation

I. Introduction

Virtual spaces, which are also called Networked Virtual Environments (NVE), can be defined as synthetic worlds which allow users to interact with each other over the network [9]. Virtual spaces have been evolving from experimental testing platforms to vast growing worlds with a massive number of simultaneous users [17]. In virtual spaces, users can do whatever they like such as make friends, play games or do business and work. Users may even feel that virtual spaces are their only reality, in which they feel comfortable [11].

The users of a virtual space use a program that simulates the environment by rendering images, which represents the current situation from the users' point of view [8]. Normally, users interact with each other using a personal representation of themselves called avatars. These avatars and other objects inside a virtual space are all called entities.

Typically, a virtual space is based on a client-server approach because of its simplicity and security. All of the traffic goes through the server so the clients do not communicate with each other directly. In the client-server model, the actions are always authorized by the server before the client can proceed. This helps to eliminate cheating or other malicious acts made by the client. [4]. In a typical client-server architecture, the message dissemination is not very optimal. Usually, the messages are broadcasted to the entire virtual space [4]. This basically means that increasing the user counts increases the amount of traffic, typically about to the square of the number of users inhabiting the virtual space [3]. In [11], it was stated that the current virtual spaces have approximately 1 billion users who generate huge amount of traffic.

For alleviating the network load, the virtual space may be regionalized where each region has its own coordinating server pool. The regionalization approaches are usually based on quadtree or octree data structures. Although regionalization improves the scalability of a virtual space, it does not filter the network traffic in any way. Thus, there is a possibility that some regions get overcrowded. These situations can be avoided by using a filtering mechanism that prevents propagation of unnecessary messages. These filtering mechanisms are called as interest management (IM) algorithms.

Roughly, IM algorithms can be classified as four different types which are either based on proximity [13], regions [7], occlusion [6] or a combination of these [8]. Proximity-based IM is based on distance between entities. In region-based IM, the environment is divided into regions where rules determining the entities of interest are precomputed (for instance, what regions are visible from every other region). It should be noted that region-based solutions that rely on precomputations are not feasible for extensible virtual spaces, where the environment can change dynamically. Finally, occlusion-based IM accepts messages from entities that are visible to the user.

In this paper, we present and evaluate a novel IM algorithm called EA^3 that is based on an existing IM algorithm, called A^3 [5]. A^3 uses proximity to discover entities of interest, however it does not take obstacles into account. A^3 was improved by utilizing a ray visibility algorithm, which helps to filter out traffic originating from entities behind obstacles. Both EA^3 and A^3 were implemented into realXtend extensible virtual space that was used as a test bed to evaluate and validate the effectiveness of the EA^3 algorithm. EA^3 was evaluated against the baseline (no IM enabled) and the existing A^3 algorithm. The results of this paper show that by combining occlusion-based IM with proximity-based IM, the use network bandwidth can be further decreased in extensible virtual spaces at the expense of some processing power.

This paper is organized as follows: in section 2, the existing literature about IM is presented. In section 3, the proposed EA^3 algorithm is introduced. In section 4, the experimental setup and the testcases are described. Section 5 presents and analyzes the results gained from the experiments, and finally, in section 6 conclusions are drawn from the results and future work is proposed.

II. Related Work

IM algorithms can be categorized into three broad categories based on how they work. These categories are proximity- [1], [5], [13], [14], region- [1], [8], [10], [16] and occlusion-based [6], [7] IM. Proximity-based IM is usually

978-0-7695-5090-9/13 $26.00 © 2013 IEEE
DOI 10.1109/NGMAST.2013.17

the most simplest form where updates are filtered based on distance between entities. In region-based IM, the environment is divided into regions using various technques available. Finally occlusion-based IM takes obstacles of the environment into account when detecting interesting entities.

In [13], a proximity-based IM approach was implemented. Their approach used prediction to determine interesting entities. The message exchange in their IM algorithm is based on three steps. The first step is used for entity discovery where the message exchange occurs at a low frequency. In the second step, messages are sent at a variable frequency between the entities that are most likely to be interacting in the near future. In the third step, messages are sent at a high-frequency between entities that are interacting with each other. However, this paper did not provide any test results to indicate that this approach is effective. It only introduced the key concepts behind this algorithm, but it did not validate the effectiveness of the algorithm.

In [14], the IM algorithm which was introduced in [13] was used. The algorithm was refined to utilize a middleware solution to run the implemented IM algorithm. This allows the IM algorithm to be separate from the application and the same IM algorithm can be used in other applications as well. Initial experiments revealed that the approach is scalable and helps to increase the user count in a virtual space by increasing the number of servers.

In [1], a three-tiered approach to IM was proposed. This algorithm used the aspects of both region- and proximity-based IM. In this system, the data flow goes through three different tiers. In the first tier, the area is divided into dynamic regions. The second tier uses the data from the first tier to filter out data per entity basis. Finally, in the third tier, which is built on top of the first and the second tier, the data is filtered by per protocol basis. In the first tier, the region can be divided as the developer sees fit, either with quadtree, octree or with n-tree. The experiments produced results which show that the three-tier approach is effective. This research did not provide any statistical results but stated that using this approach the system could handle thousands of entities with ease. However, the environment was not clearly described whether it was based on 2D or 3D setting but the results suggest that the system that was used in the experiments was very light weight and implemented only the necessary functions to operate correctly.

In [16], it was proposed that the environment would be regionalized based on the user's behaviour. The collected data was analysed and the world was regionalized in a heuristic way. In [7], the environment was divided using a Delaunay tetrahedrization where polygons are decomposed into triangles. Using this method the virtual space can be regionalized in a very high-grained way. Also, in [8], Delaunay tetrahedrization was used in the regionalization of the virtual space. Results gained from this research showed a significant drop in traffic amounts, over 90 per cent, when used together with an occlusion-based IM algorithm. However, as the visibilities between the regions are precomputed, these results are not valid in the case of extensible virtual spaces that can change dynamically. Finally in [10], the authors divided the virtual environment into hexagon shape tiles and each hexagon had its own multicast group. When a member of the multicast group sends a message, the message is delivered to the subscribers of the group. If the hexagons are correctly sized, the number of subscriptions and unsubscriptions can be limited.

The most powerful IM techniques are based on occlusion, where updates originating from entities that the client does not see are denied. Occlusion-based IM can be implemented in various ways, for example using regionalization, where visibilities between regions are precomputed beforehand [6]. This approach performs quite poorly in a virtual space which is able to change its form dynamically, because whenever the environment changes, the visibility data has to be recomputed. However, when used in a static setting, this approach is very efficient. The most general occlusion based IM algorithm can be implemented using ray casting and is called ray visiblity. In [6], it was stated that ray visibility implements theoretically a perfect IM algorithm because it detects precisely interesting entities based on what the client sees. The complexity, however, is quite high and it quickly becomes very heavy to use when the user count rises.

In [5], an algorithm called A^3 was presented. In their approach, they used two circular areas of interest where the first one is a sector corresponding to the client's field-of-view, and a smaller circle, which represents the critical area (CA) around the client. Both of these, the sector and the circle, are based on euclidean distance. The purpose of the CA is to keep the game state inside the circle up-to-date, in order to provide a better game play experience near the avatar. They also used a relevance factor to control the update interval of the entities in front of the client. The farther away the entity is from the client, the more irrelevant it is. Updates are sent more frequently when the entities are closer. If an entity is inside the CA, the updates from that entity are not throttled in any way. The results showed that using this algorithm traffic on the network medium per client was reduced by approximately 80 per cent on average compared to a situation when IM was not used at all. Because this algorithm is based on euclidean distance, the improvement for this algorithm would be to use an occlusion-based algorithm to detect the obstacles in the environment, e.g. using ray casting instead of euclidean distance.

III. Proposed Interest Management Algorithm

The proposed IM algorithm extends the A^3 algorithm. A^3 is improved by changing it to utilize the ray visibility algorithm instead of euclidean distance in detecting entities in front of the client. This improved version of A^3 is called as EA^3. Figure 1 presents the A^3 algorithm and Figure 2 presents the EA^3 algorithm which detects obstacles in front of the client.

For EA^3, the CA works the same as for A^3. Entities inside the CA are considered to be very important and messages are sent between these entities at a high frequency without update attenuation. Usually, it is feasible to keep the CA rather small, because it is only intended to keep entities in the immediate vicinity of the client up-to-date. The entities that reside in front of the client are detected with the ray visibility algorithm. Ray visibility detects whether or not an entity is behind an obstacle and acts accordingly. The ray visibility algorithm uses AABB raycasting which is generally considered to be a lighter technique to use in terms of CPU usage compared to polygon level raycasting, even though it is not as precise as polygon

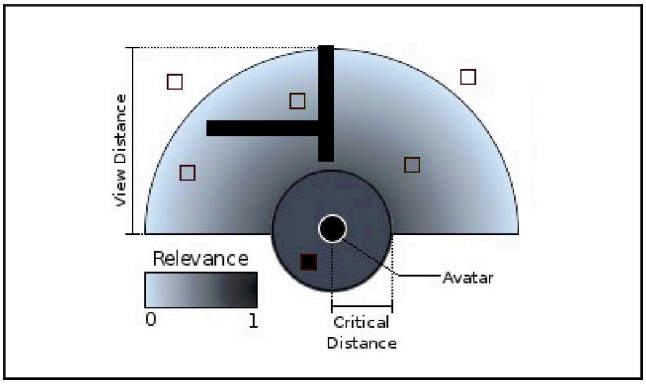

Fig. 1: The A^3 algorithm.

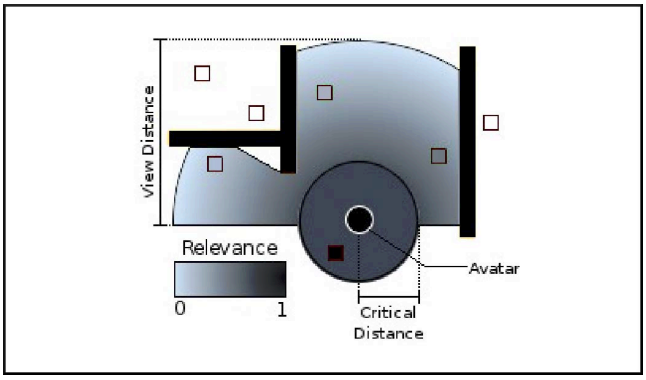

Fig. 2: The EA^3 algorithm.

level raycasting [2]. However, in the IMs case, this is not an issue because AABB raycasting is precise enough to detect if an entity is visible or not. All in all, the less the algorithm burdens the CPU, the better.

For each entity a relevance factor is also calculated, which controls the update frequency of the entity. This factor is dependent on the distance between the entity and the client. If the entity is considered invisible to the client or if it is beyond the maximum view distance (MVD) of the algorithm, the relevance factor is automatically set to zero. Relevance factor of zero always means that all updates are rejected from the given entity. The calculation of the relevance factor is presented in Equation 1. The relevance factor is always somewhere between 0 and 1 depending on how far the entity is from the client. Using this factor the relevance filter can calculate a new update frequency for the target.

$$relevance = 1 - \frac{distance - criticaldistance}{viewdistance - criticaldistance} \quad (1)$$

IV. Experiment Setup

In this section, the experiment setup is discussed. The aim of the experiments is to find out if EA^3 is more efficient than A^3. The hypothesis is that both EA^3 and A^3 filter the network messages and reduce the overall bandwidth usage of the server significantly. Furthermore, EA^3 is expected to bring better results than A^3 in situations when clients are occluded from each other.

A. Hardware

The experiments were executed with five PCs and a high end server each running Ubuntu 12.10 operating system. Each PC was connected locally to the server via 1 Gbit/s Ethernet link using a HP ProCurve 2510G-24 switch. The server had 64 GB of DDR3 memory and an Intel Core i7-3930K processor running at 3.2 GHz with available threads of 12. The PCs configurations varied, but all of them had 8 GB of DDR3 memory and an Intel Core i5 or i7 processor. The testing took place in laboratory settings. The network traffic is recorded from the server side using Wireshark network protocol analyser. Wireshark is a utility which can be used to analyse and capture network traffic from the network medium [15].

B. Testbed and Test Scene

realXtend open source extensible virtual space platform (version 2.3.3) was used as a test bed in the experiments. Two 3D scenes were used in the experiments. The first one presented in Figure 3 (called the 'corridor' scene) contained a long rectangle surface which was divided by a wall, hence creating two narrow corridors. The wall had gaps between sections so entities moving inside the two corridors would see each other. This synthetic scene was used to demonstrate the effect of obstacles when EA^3 is used. The second scene was

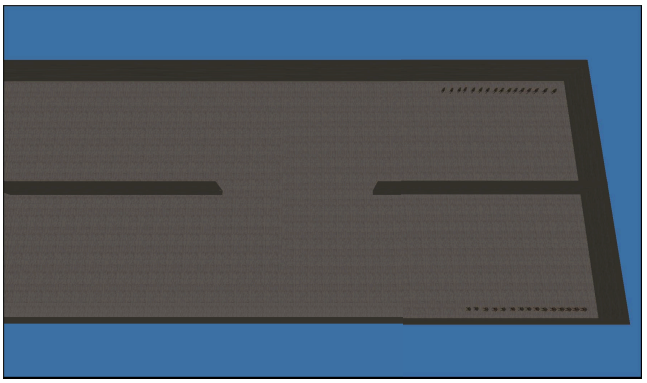

Fig. 3: The synthetic corridor scene.

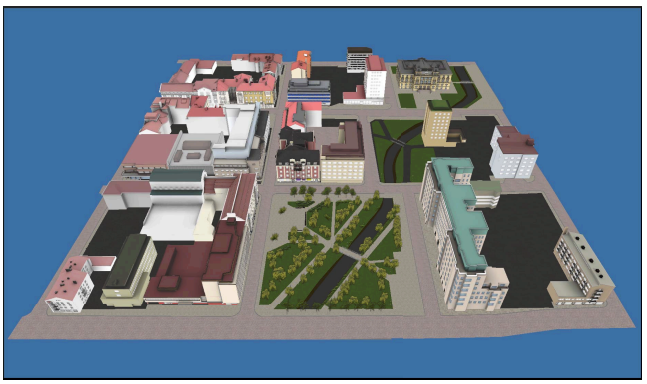

Fig. 4: The 3D city scene of 9 blocks.

a 3D city model which consists of nine block area. Figure 4 presents the 3D city scene used in the experiments. This scene was used to test the IM algorithms in a realistic environment.

C. Parameters for the EA^3 and the A^3 algorithms

The aim was to test the EA^3 algorithm in the same way the A^3 algorithm was tested in [5]. However, exactly the same test setup is quite impossible to create. The original A^3 algorithm was used in a virtual 2D space, whereas realXtend implements a fully featured 3D environment. Also, the simulation was done with the ns-2 network simulator in A^{3}s case [12]. The simulation of the whole realXtend platform in ns-2 is too complicated and time consuming to create, so ns-2 was put aside. Because A^3 was tested with ns-2, the user counts are much higher than realXtend can handle at the moment. That is why the experiments had to be run with a lower number of users, in this case with 30 avatars, to get an idea if the algorithm is effective or not.

The algorithms implemented in realXtend were tested with the following parameters. In the corridor scene, the CA was set to 2 units, so only the entities, that are in the immediate vicinity of the client are treated as very important and the MVD was set to 100 units. In the 3D city scene, the CA was set to 10 units and the MVD was first set to 100 units and later to 200 units to see how the range of the algorithms affect the overall bandwidth usage. The 3D city scene used longer ranges because of the larger scale of the scene. The length of one unit in this case can be whatever the virtual space 3D proportions are, but in the 3D city and the corridor scenes one unit can be converted to meters. The maximum update interval of the relevance filter was set to 250 milliseconds in both cases. The ray casting interval was also set to 250 milliseconds.

Ray casting is throttled because the algorithm becomes very heavy very quickly in terms of CPU usage if the ray-casting is not limited. Even though ray casts are throttled to 250 milliseconds, the algorithms precision should be within acceptable levels. If the ray cast interval is too long, for example 500 milliseconds and up, the quality of the simulation starts to deteriorate, because the ray visibility algorithm cannot test if the entity is really visible or not often enough. This results, for instance, in avatars warping around corners in front of the client instead of coming around the corner smoothly.

D. Experiment

The experiments were run in both scenes first without any IM starting from 6 users and incrementing the user count by 6 each time, until the user count reached 30 users. During each increment, the network traffic was recorded from the server side for two minutes. Also profiling data was gathered using realXtend's own built-in profiler functionality. This data reveals the CPU usage for each algorithm. Next, the same experimentation procedure was repeated with A^3, and finally, with EA^3.

In the corridor scene, the avatars were spawned in two rows inside the corridors facing each other. This way, when the entities enter a section where the wall has a hole, it can be ensured that all of the entities see each other. The avatars that were spawned into the 3D city scene during the experiments were spawned randomly all over the scene. When all of the avatars were spawned, the server informed the avatars to start moving. In the corridor scene, the avatars were ordered to move in a straight line through the corridors. In the 3D city scene, the avatars were moving in a random pattern. After capturing the traffic for two minutes, the scene was reset for the next test case.

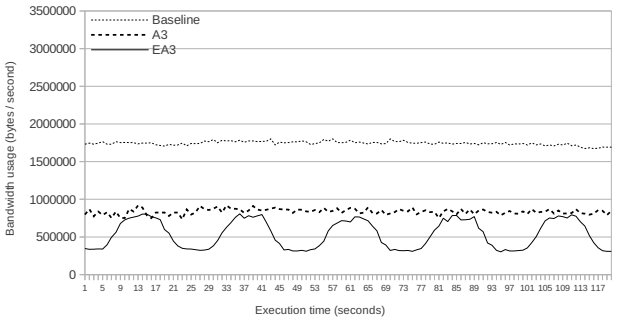

Fig. 5: Network traffic graph showing real-time results gained from the corridor scene.

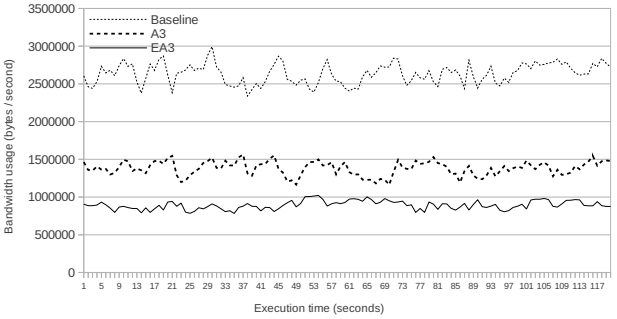

Fig. 6: Network traffic graph showing real-time results gained from the 3D city scene.

V. Results

In this section, the results gained from the experiments are presented and analysed. Figures 5 and 6 the presents the network traffic results from the test scenes and Figures 7 and 8 the CPU load data. All of the network traffic and CPU load data was recorded in the server side, which is the most reasonable place to capture the traffic, because realXtend uses a basic client-server architecture, and thus, the server acts as a central node in the network.

Figure 5 presents a real-time network traffic of the experiment when the server was hosting 30 clients in the corridor scene. In this scenario, the MVD of the algorithms is 100 units and the CA is 2 units. Figure 5 contains three curves which represent the baseline, A^3 and EA^3 results. Without any IM enabled, the results show that the clients produce approximately 1.6 to 1.7 megabits per second of network traffic. With A^3 enabled, the algorithm reduces the traffic amounts by approximately 50 per cent resulting in a constant

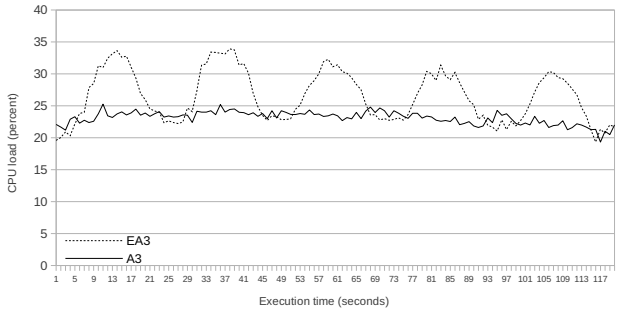

Fig. 7: Real-time CPU load of the A^3 and the EA^3 algorithms recorded from the corridor scene.

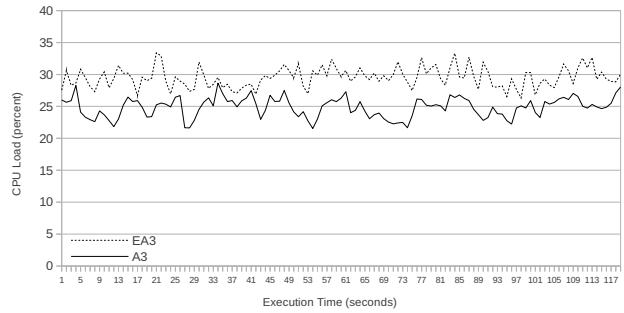

Fig. 8: Real-time CPU load of the A^3 and the EA^3 algorithms recorded from the 3D city scene.

1 megabits per second. Finally, when the EA^3 was enabled the results differ significantly from A^3. The traffic amounts fluctuate from 1 megabits per seconds to 0.4 megabits per second. This is mainly caused by the structure of the corridor scene; the center wall has five sections without walls. While the entities are travelling through an open section all of the entities see each other. This causes a spike in the network traffic amounts and EA^3 acts quite the same as A^3. When another wall occludes the view, the traffic amounts plunge to 0.4 megabits per second. EA^3 lowers the amount of traffic approximately 70 per cent against the baseline.

Figure 6 presents the real-time network traffic gathered from the 3D city scene but with MVD of 200 units and CA of 10 units. The results show that the clients produce a approximately 2.7 megabytes per second of network traffic. With A^3 enabled, the algorithm reduces the traffic amounts approximately 48 per cent resulting in about 1.4 to 1.5 megabytes per second. Finally, when the EA^3 was enabled the traffic amounts dropped to approximately 0.9 megabytes per second which is 35 per cent lower than A^3 and approximately 62 per cent lower than the baseline. EA^3 performs better than A^3 because the 3D city scene has lots of obstacles that occlude entities from each other. On average, the number of entities that are interesting to a client was approximately 13 entities when A^3 was used while EA^3 limits the number of interesting entities to approximately 3 entities per client.

Figure 7 presents the CPU load data acquired from the realXtend's profiling tool while the server was hosting 30 users in the corridor scene. The profiling data shows, how many per cent a specific block of code allocates time from the realXtend process. In this case, the IM related code was profiled and the data was collected in 1 second intervals. The figure shows the performance of both the A^3 and the EA^3 algorithms. The continuous line represents A^3 and the dashed line represents the performance of EA^3. The results clearly illustrate that the CPU load of A^3 varies between 23 to 25 percent. The CPU load consists mainly of calculating distances between entities. In EA^3s case, there is a greater fluctuation in the CPU load varying between 23 to 30 per cent. The load peaks are caused by the open areas in the corridor scene where every entity sees every other entity. In a walled situation, it can be seen that the CPU load drops to about the same level as A^3. This happens because the ray query is terminated quickly after colliding with the center wall thus lowering the CPU load.

Figure 8 presents the CPU load data gained from the 3D city scene. The figure shows the performance of both the A^3 and the EA^3 algorithms. The continuous line represents A^3 and the dashed line represents the performance of EA^3. The results illustrate that the CPU load of A^3 varies between 20 and 25 per cent. The CPU load consists mainly of calculating distances between entities. With EA^3, the CPU load varies from 27 to 30 per cent. EA^3 is slightly heavier to use than A^3 because the algorithm has to raycast entities in addition to calculating distances. However, if an entity is not visible to the client, the relevance filtering part is not executed at all to ensure that unnecessary calculations are not done. The slightly heavier CPU usage of EA^3 is not a surprise because as in [5] it was stated that even though ray visibility algorithm represents a "perfect" IM algorithm, it tends to be quite heavy to use when considering the CPU load.

Figure 9 and 10 presents the network traffic amounts of the algorithms using the 3D city scene. In this experiment, the aim was to see how the bandwidth usage is impacted when avatar counts rise. The experimentation was started with 6 avatars and was increased by 6 until the virtual space hosted 30 avatars. After each increment, network traffic was measured with and without any IM algorithms enabled. Figure 9 presents the experiment where the MVD was set to 200 units and the CA to 10 units. The advantage of EA^3 is clearly visible when compared to A^3 when the number of avatars is increased. In Figure 10, the MVD was set to 100 units and the CA to 10 units. In this scenario, the difference between EA^3 and A^3 is slightly harder to see because the MVD is shorter than the inter block distance, which accounts for a large part of the occlusion benefit. Even though the MVD is short, EA^3 still performs more efficiently than the A^3 algorithm. The higher the viewing distance is, the larger the difference is between A^3 and EA^3. The difference increases because A^3 does not consider obstacles so the number of interesting entities is much larger than with its counterpart EA^3, which scans only the visible parts of the scene. Thus, increasing the viewing distance does not affect the performance of EA^3 as much as it affects A^3.

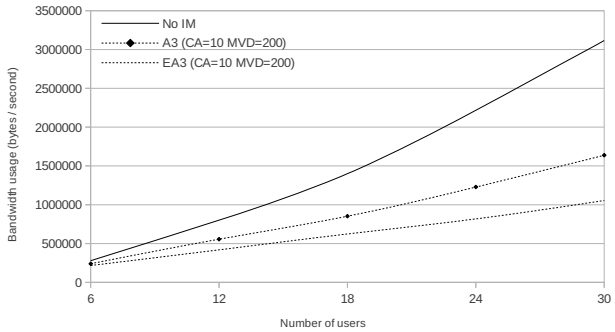

Fig. 9: Network traffic when CA is 10 and MVD is 200.

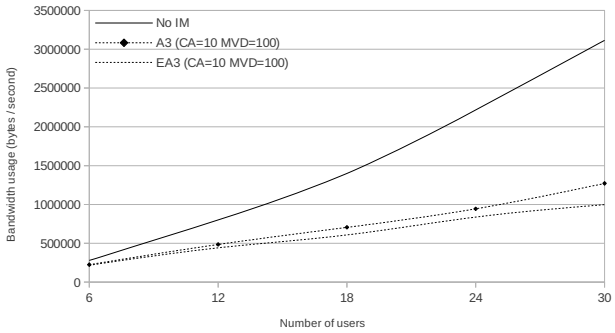

Fig. 10: Network traffic when CA is 10 and MVD is 100.

VI. Conclusion

This paper presents and evaluates a novel IM algorithm called EA^3 that is used in extensible virtual spaces to filter out unnecessary network traffic. EA^3 is based on an existing IM algorithm, called A^3 which uses euclidean distance to discover entities of interest. A^3 was improved by using ray visibility, which is able to filter out messages coming from entities behind obstacles. EA^3 and A^3 were implemented and evaluated in realXtend extensible virtual space platform.

The evaluation results show that both EA^3 and A^3 performed efficiently compared to a situation when no IM algorithm is used. In a city-like test scene, the network bandwidth savings were approximately 50 per cent with A^3 and 62 per cent with EA^3 compared to the situation when no IM algorithm was used. The effectiveness of EA^3 against A^3 is heavily dependent on the context where they are compared. In a scene, where there are no obstacles of any kind, the EA^3 algorithm behaves the same way as the A^3 algorithm. However, having obstacles in the scene greatly improve the efficiency of EA^3. The CPU load of the algorithms was also evaluated. A^3 burdened the CPU by approximately 20 to 25 per cent while the load with EA^3 was between 27 to 30 per cent. Additionally, it is important to notice that when the server is filtering the traffic, it also affects the client side by lowering the number of acknowledgements the client has to send. This is important for mobile devices, where transmission of data is a significant contributor to the overall energy consumption. Finally, all of the IM related computation is done on the server side and thus mobile clients are not burdened with this.

Acknowledgement

The authors would like to thank the staff at the Intel and Nokia Joint Innovation Center for their support. This work has been carried out in the Tekes Chiru Project.

References

[1] H. Abrams, K. Watsen, and M. Zyda. Three-tiered interest management for large-scale virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '98, pages 125–129, New York, USA, 1998. ACM.

[2] U. Assarsson and T. Moller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools*, 5(1):9–22, 2000.

[3] D. Bauer, S. Rooney, and P. Scotton. Network infrastructure for massively distributed games. In *Proceedings of the 1st workshop on Network and system support for games*, NetGames '02, pages 36–43, New York, USA, 2002. ACM.

[4] A. Berson. *Client/Server Architecture*. McGraw Hill, 1992.

[5] C. E. Bezerra, F. R. Cecin, and C. F. R. Geyer. A3: A novel interest management algorithm for distributed simulations of mmogs. In *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications.*, pages 35–42, 2008.

[6] J. Boulanger. Interest management for massively multiplayer games, 2006.

[7] J. Boulanger, J. Kienzle, and C. Verbrugge. Comparing interest management algorithms for massively multiplayer games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, USA, 2006. ACM.

[8] B. Hariri, S. Shirmohammadi, and M. R. Pakravan. A distributed interest management scheme for massively multi-user virtual environments. In *VECIMS Virtual Environments, Human-Computer Interfaces and Measurement Systems*, page 111, July 2008.

[9] S. Hu and G. Liao. Scalable peer-to-peer networked virtual environment. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, NetGames '04, pages 129–133, New York, USA, 2004. ACM.

[10] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. Barham. Exploiting reality with multicast groups. *IEEE Computer Graphics and Applications*, 15(5):38–45, 1995.

[11] M. Mantymaki. *Continuous Use and Purchasing Behaviour in Social Virtual Worlds*. Turku School of Economics, 2011.

[12] S. McCanne and S. Floyd. Network simulator ns-2, 2011.

[13] G. Morgan and F. Lu. Predictive interest management: An approach to managing message dissemination for distributed virtual environments. Technical report, Richmedia, 2003.

[14] G. Morgan, F. Lu, and K. Storey. Interest management middleware for networked games. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 57–64, New York, USA, 2005. ACM.

[15] A. Orebaugh, G. Ramirez, and J. Burke. *Wireshark and Ethereal network protocol analyzer toolkit*. O'Reilly Media, Canada, 2007.

[16] A. Steed and R. Abou-Haidar. Partitioning crowded virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '03, pages 7–14, New York, USA, 2003. ACM.

[17] C. W. Thompson. Next-generation virtual worlds: Architecture, status, and directions. *Internet Computing, IEEE*, 15(1):60–65, 2011. ID: 1.