A Computational Geometry technique for Supporting Multiplayer Online Games

Farhad Ranjbar Helan, Shahriar Lotfi

Electronic, Computer Engineering, Information Technology Department
Islamic Azad University Qazvin Branch, Iran
Computer Sciences Group, Tabriz University, Iran
Ite1867@eaedu.org, Farhad.Ranjbar@qazviniau.ac.ir, Shahriar_Lotfi@tabriz.ac.ir

Abstract– This paper presents a geometric technique to support multiplayer online games on a peer-to-peer system. Assumption in this paper is based on the fact that players are more interested in their region of visibility and influence than in the other regions. We use a computational geometry technique – Voronoi Diagram – to partition the game space into regions. The players in a region communicate with other players through the coordinator of the region. The resulting system scales up with the number of players and is able to distribute region updates in a scalable manner. We also propose techniques for fault tolerance in the wake of node failures. We have simulated a simple game to prove the efficiency of this technique.

[Ranjbar Helan F, Lotfi L. **A Computational Geometry technique for Supporting Multiplayer Online Games.** *Life Sci J* 2013;10(6s):151-158] (ISSN:1097-8135). http://www.lifesciencesite.com. 24

Keywords – Peer-to-Peer, Multiplayer Game, Voronoi Diagram, Net Games.

I.INTRODUCTION

This paper uses a computational geometric technique – Voronoi Diagram [6] – to partition the game space into regions based on some locality properties and support it on a peer-to-peer structure. The players participating in the game form an overlay structure based on their location in the game space. Thus all the players contribute memory, CPU cycles to manage the shared game state.

The game space in a multiplayer online game (MOG) is shared and inhabited by thousands of players. Some popular games like Ultima Online and Quake have recorded 200,000 simultaneous users. Apart from the exciting story line and graphics, MOGs are a shared distributed application with some private state maintained locally, and the shared state communicated to other players.

Traditionally the MOGs were supported by a client – server architecture. Since the single server solution did not scale well, mirror server architectures were proposed. But the mirror servers constrain the number of simultaneous players in a geographic location. Cluster of server architecture was proposed to share the load by dividing the game space into regions. But even this solution does not scale with the number of simultaneous players in a given region of the game space.

Multiplayer Games are natural application for peer-to-peer systems. Game players have incentive to provide resources for managing the shared state because the participation in resource sharing is limited to the duration of the game play. A recent proposal [7] to support MOGs on a peer-to-peer system divides the game space into a fixed number of regions. Although this architecture scales with the number of players, it does not fully exploit the locality property. The coordinator of a region may not be playing in the region but still has to handle the burden of state management and communication in the region.

Games are different from the existing peer-to-peer applications that mostly harness only the storage and bandwidth of the peers. Games utilize memory and CPU cycles to manage the shared game state. Hence some of the problems that have to be addressed are:

- Performance - games have frequent updates that must be quickly propagated to other players. Further the propagation of updates must scale with the number of players.
- Scalable State Management – the state maintained by the peers must be based on their current location. Also each peer must manage state only for the region of game space that is closest to it.
- Fault Tolerance - replicating game state to improve availability has some problems. With high frequency of updates maintaining a large number of synchronous replicas in the system become a performance bottleneck.
- Security – with the state being maintained on the peers,instead of the central game server, player get increased opportunity to cheat.

We propose a technique to partition the game space and assign coordinators for regions based on some closeness and locality properties. As we shall see in the later sections, this technique addresses the first two problems well. We also address the problem of fault tolerance with an initial proposal.

We present an architecture marrying MOGs, peer-to-peer systems and Voronoi Partitioning technique and we also provide an initial evaluation of the technique to demonstrate its feasibility.

The rest of the paper is organized as follows: Section II discusses the background and related work in detail, Section III describes some of the terms used and our design, Section IV discusses some of our proposals, which forms future work, Section V gives some initial evaluation. We conclude in last Section.

II. RELATED WORK

Traditionally MOGs have been supported using a client – server architecture, where the server keeps player account information and handles all shared state and communication between players. This architecture clearly does not scale up with the players. To achieve scalability the servers have to be over-provisioned for the worst case scenario. Further this architecture suffers from a single failure point and has little fault tolerance.

Mirror Server architecture was proposed to isolate players based on their geographic locations. Players typically join the closest geographic mirror. Hence the number of simultaneous players in a geographic region is constrained. Further the mirrors have to be synchronized and this synchronization is normally done on a high speed backbone.

Later, Server clusters were used instead of a single server to achieve scalability. But even this scheme limits the number of simultaneous players in a region. All the client-server architectures lack flexibility and have to be over-provisioned for peak loads. Further the client-server model limits the deployment of user designed game extensions, which is an important trend in game development and design. Since a centralized game server is required to host the core game, the development is slowed down considerably.

Recently there have been proposals for peer-to-peer gaming systems with application layer multicast. One such system is SimMud [7], which is built on top of Pastry [12] and uses Scribe [3] application layer multicast for communication. The game space is divided into a fixed number of regions and each region is managed by a node assigned to be the coordinator for that region. Players in a region of the game space form a multicast group and communicate using Scribe multicast system. Players, on switching between regions, leave the multicast group in the old region and join the multicast group in the new region.

Even though the above described system takes into account the locality of interest in a MOG, it does not fully exploit it. A player may be a coordinator for a region, but may be playing in another region of the game space. But still the player has to maintain state and handle communication for that region. Hence in this system players maintain state for regions of game space this is no longer close to them. As the players move in the virtual space, their region of influence and interest continuously change, but this system does not take this into consideration.

We target Multiplayer Online Games, which currently use the client-server architecture or a peer-to-peer architecture described above. Although the peer-to-peer architecture lowers the deployment cost with all nodes providing CPU and memory, it incurs a security risk because the game state is distributed to peers. Hence some techniques like Run Time Verification [5] for Anomaly Detection proposed by Honghui Lu et al., may be applied to our system.

Replication is an integral part in any peer-to-peer file sharing system [4], [8], [13] for both improved availability and performance. However these systems are read only system, whereas a gaming system has frequent updates. As a result our system must maintain data consistency while tolerating network and node failures. Our approach is to maintain the replicas at the neighbors. The intuition in maintaining the replicas at the neighbors is that when a node leaves or fails, the Voronoi region has to be refined and the refined mesh has the neighbors taking over the region of the failed node. However consistency requirements require us to design a consistency mechanism with a small window of vulnerability.

Fault tolerant consistent data services can be built with quorum systems [9]. In these systems, updates cannot proceed if the number of nodes in a region is not large enough to form a quorum.

Group communication and interest management is used in some distributed game implementations including AMaze [1] and Mercury [2]. The SimMud system as discussed before makes use of Pastry and Scribe. Since our system partitions the game space into regions using Voronoi Diagram, it is closer to CAN [11] than to any other DHT like Chord [14] or Pastry.

CAN is a scalable, robust and self organizing DHT that considers a d-dimensional Cartesian coordinate space. The coordinate space is completely logical and bears no resemblance to any physical coordinate system. At any point in time, the entire coordinate space is partitioned among all the nodes in the system such that every node owns its individual distinct zone within the overall space. The node that owns a particular zone stores all keys that map to any point within the zone. Hence the lookup for an object may be routed through the CAN infrastructure until it reaches the node whose zone the point P, onto which the object maps, lies.

Routing in CAN works by following the straight line path through the Cartesian Space from the source to the destination coordinates. A CAN node maintains a coordinate routing table that holds the IP address and the virtual coordinate zone of each of its immediate neighbors in the coordinate space. This neighbor information is sufficient to route between any two arbitrary locations in the space. Using its neighbor coordinate set a node routes a message towards it destination by simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates.

For our purpose we need to consider a 2-d CAN like system to perform the message routing. Since the Voronoi Partitioning divides the 2-d game space into regions similar to zones in CAN, we can use CAN type routing to communicate with players in other regions. Further in our system each node knows the address of its neighbors. Message routing to nodes that are far off in the game space is needed for team player strategy games like War Craft. In these games, it becomes essential to have communication between team mates, who may be at different regions of the game space at a given time.

III. OUR SOLUTION

This section proposes a mechanism to support multiplayer games, based on voronoi partitioning approach. The contents of this section so organized to act as a step by step guide leading us to development of the proposed solution. Analyzing the problem and challenges involved in the support of massively multiplayer games on peer to peer system, we explain what motivates the design of proposed solution. Following that is a detailed explanation of voronoi partitioning, its application to support massively multiplayer games, a communication mechanism amongst peers involved in the game.

A. Multiplayer Games on Peer-to-Peer System

In multiplayer games, actions and state of a player need to be communicated to other players. For example, during a fight between two players if one player shoots at other; the other player must be communicated that it has been shot at. (Typically, a player is controlled by a peer and hence the terms peer and player are used interchangeably in the further paper.) Considering this, one obvious approach to support such games on peer-to-peer system is to have every player communicate with every other player in the network. But it is clearly not scalable to massively multiplayer game as a peer may not be capable of handling communication with tens of thousands of other peers involved, at the same time. It may lead to explosion in the number of messages leading to network congestion, packet drops and increased latencies. To make the system scalable, a peer must be required to communicate with only a subset of other peers involved in the game. A challenge here is to find such a subset of nodes so as to enable communication within the latency constraints imposed by the game design and to minimize messages within the network.

B. Region of Influence (RoI)

Consider a game field as shown in Figure 1 with players A-G. The region of influence (RoI) of a player is defined as region in which the player's actions may can be seen or perceived by other players. This region is dependent upon game terrain design. Above figure shows region of influence of A. Actions of player A will be perceived by only those players who are within this region. Thus, only Players B and C can perceive A's actions. In other words, B and C form subset of nodes A should communicate to. In addition to this, A may be required to send some status updates to other players not in this region. For example, if A and E are team-mates then A will be required to send its status updates to E. A may also be required to send status updates to F, like number of kills in a First Person Shooter game like Quake. F may not be interested in the actions taken by A but only in the state of A resulting from these actions. Although the subset of peers, a node should communicate to, is completely dependent on game design, it will always be required to communicate with the players within its RoI. This is because if, say, A decides to move forward, then the movement will be seen by B and C on their display. But G will not be interested in this update because A is not displayed on its screen.

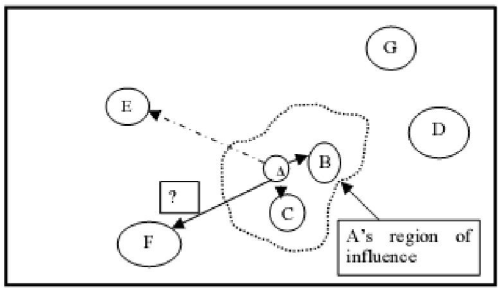

Fig. 1 Region of Influnce

B. Voronoi Partitioning

Voronoi Partitioning is a general concept applicable to n-dimensional space. This section, however, explains and defines it only for 2-dimensional space to maintain simplicity and relevance to this project. figure 2 a screenshot of applet VoroGlide [15]demonstrating voronoi partitioning. Consider a set of input vertices, represented as dots, in a 2-dimensional plane. The figure shows the partitioning of the plane into polygonal regions. The partitioning is done such that any point within a polygon is closest to input vertex enclosed within that polygon than to any other input vertex. It is restated, for the ease of understanding, that the input vertices are the points represented as dots in figure 2.

Formal definition of voronoi partitioning is as follows:

Definition: Give a set of points in a 2-dimensional plane, voronoi partitioning of the plane is sub-division of plane into polygonal regions where each region is a set of points that are closer to some input vertex than to any other input vertex.

To apply voronoi partitioning to divide the plane of the game area, consider that the input vertices represent positions of the players at any particular instant. The voronoi partitioning thus divides game area into polygonal regions, such that, any point in a polygon is closest to some player than to any other player. This player, which is closest to all points within a polygon than any other player, is assigned as coordinator for the region.

Voronoi partitioning can lead to formation of very small regions, in case of clustering of nodes. Figure 2 shows such a clustering and breakup of the plane into small regions. This can lead to formation of regions which are smaller than RoI of a player. Such a small fragmentation is unnecessary and as will be clear later when communication mechanism is explained, it may lead to additional communication overhead. To avoid this, adjacent regions can be merged together to form a larger region. One answer to decide which regions to merge could be to merge the regions with smallest area. This however incurs additional overhead of determining such regions. Alternative approach is to merge any two regions. This can then be implemented recursively to merge multiple regions. Since the regions are small, merging of any two adjacent regions may serve the purpose equally well. Both the techniques have various issues and trade-offs but those are neither discussed nor addressed as the problem is outside the scope of this project. This results in some players without coordination responsibilities.

One major disadvantage of voronoi partitioning is the time complexity associated with it. Fortune's algorithm [6] has the best known time complexity of O(n log n). This is for the static points. For nodes moving at variable speed, which is typical of any game, the upper bound is predicted to be cubic. Hence, there is need for an efficient distributed implementation of algorithm to make it scalable. No such algorithm exists to the authors' knowledge

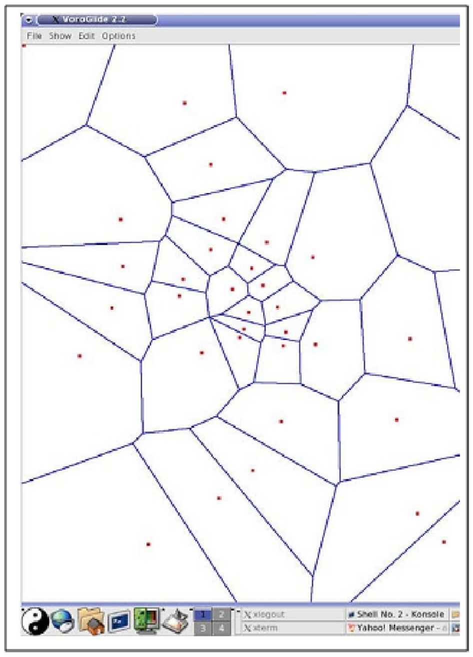

Fig. 2 Sample Voronoi Partition

. However, not all games have players moving continuously. While First Person Shooter (FPS) games like Quake have players continuously changing their position, their activity is largely limited to a particular region in case of strategy games like Age of Empires. Voronoi partitioning can be efficiently implemented for such games. Another way to overcome this problem is to consider only a fraction of nodes as input vertices for partitioning. This also results in nodes without any coordination responsibilities at a particular instance.

C. Communication

Explained first are the terms used frequently in the subsequent
report.

- Player: Any node which is alive and participating in the game.
- Coordinator: A process running on player node with responsibilities similar to server of client-server model, within specific region.
- Neighbor: Coordinator of adjoining region with respect to the node under consideration.

Consider the state of the game after it has been divided into voronoi partitions and coordinators assigned for the regions. Each coordinator is informed of its neighbors.

A player sends its status update to the coordinator of the region it is in, currently. For the moment, assume that player knows the coordinator. This status message is multicast to other nodes in the region as well as to neighbors. Figure 3 explains the message flow.

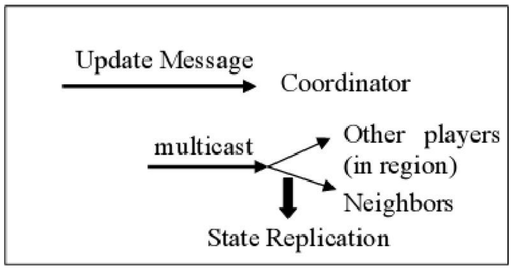

Fig. 3 Message Flow in the System

As a player moves around, it may move across a region at some point. At this point, it should be informed of the new coordinator it should contact to. This is done by the coordinator of the region it was in, previously. This is explained with the figure 4.

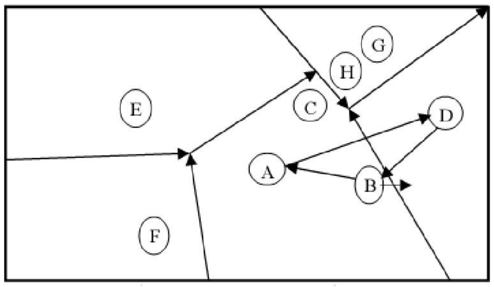

Fig. 4 Movement across regions

Consider player B moving from A's region to D's region Suppose A and D happen to be coordinators. B sends its status update coordinator A notifying its position and irection of motion. From this information A infers that B is moving into D's region and hence, sends a join request to D on behalf of B. D then sends the join approval message to B. However, this coordinator switch is performed after time ə, as the player may cross the region and return immediately. If the coordinator of a region leaves the region, the particular area may be repartitioned to reflect the current state of the game. We advocate periodic refining of the partitioning to appropriately reflect the current state of the game and thereby maintaining the properties of closeness and relevance. Obviously, there is additional overhead and latency involved when a player tries to switch region. Hence, if voronoi regions are very small then players will switch regions more frequently. This increases communication overhead.

D. Node Join and Leave

The system has a bootstrap node whose address is known to other nodes. This can be thought of as a web server hosting the game any node can join the game by sending a Join Request message to this node. The player can select its position or it may be randomly assigned some position by the coordinator. Alternatively, a player can join by sending a Join Request message directly to the coordinator of the region it wishes to join provided it knows the coordinator. A player may leave the game by sending leave message to coordinator. A coordinator can leave by informing the bootstrap node. The bootstrap node, in turn, can take action to select new coordinator of the region. It can be a node which was closest to the coordinator before the coordinator left or the bootstrap node may choose to repartition the space.

IV. RESULTS

We present the experimental results obtained with a prototype implementation of our system in this section. We have used a LAN environment to perform the initial evaluation of our prototype. We concentrate on the networking aspect of results, mainly the latencies experienced by the players, and the messages sent and received by the coordinators.

All experiments were performed on nodes with 1.2 GHz Pentium III CPUs and 512 MB of main memory. The machines run Linux 2.4.17 and Sun JDK 1.4. Our prototype system is written fully in Java and each node runs in a separate Java Virtual Machine. We run two processes in each node – one to act as a coordinator for the region and another to act as a player in the region. Each of these processes runs in a separate Java Virtual Machine.

We analyze the effect of total population on the latencies experienced by the players and the messages sent and received by the players. We also compare our results with SimMud, run on a UDP communication system. In every instance, our initial results collaborate with our hypothesis that Voronoi Partitioning technique provides good closeness and locality property.

Since we have not implemented the dynamic distributed partitioning scheme, we study the effect of static partitioning by modeling the game to make sure the players stay within their respective regions. In our model simulated players eat and fight every 10 seconds and always stay within their region. We are restricting the players to their regions to perform an initial study of our scheme.

Similarly in our game three or four position updates would be sufficient, but we send updates every 100 milliseconds to stress our system to see how our scheme can be applied to other gaming environments that need more frequent updates.

Each region of the game space is described using an MxN array depending upon the area spanned. Associated with each node is a map of the entire game space, consisting of immutable landscape information. In our implementation, object updates are sent in 200 bytes serialized Java records. The object arrays, which contain the mutable object information, are inherently sparse, and in packed format the messages to transfer objects are around 20 KB. In our simulations, we randomize the actions performed by the players and average the results over 10 runs. We measure 300 seconds of simulated game play.

A. Join Latency

Join Latency is the time between sending a Join Requet message to join the game and receiving a Join Approval message from the coordinator. Join Request messages are sent both at the time of joining the game and while switching between regions. But since we do not allow our players to switch regions, it is the initial delay in joining the game.

We measure the Join Latency as the number of players in the system increase. We compare our latencies with that of SimMud, where we assume one region and vary the number of players. Our implementation has as many regions as there are players. Figure 5 presents latencies experienced as a function of players in the region.

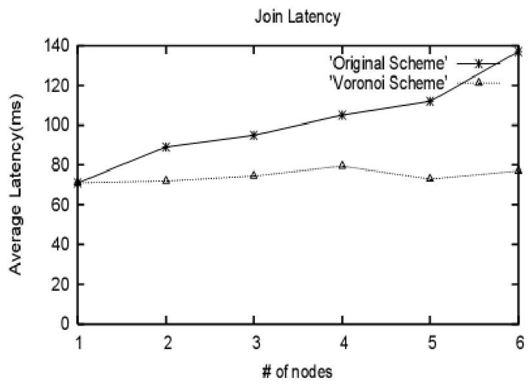

Fig. 5 Join Latency as a function of Number of Players

As seen from the graph, the Join Latency in the Voronoi Partitioning Scheme remains more or less constant. This can attributed to partitioning the game space into regions based on closeness. In our implementation, say with 6 nodes, the entire space is managed by all of the nodes, as against the original scheme, where in one coordinator manages multiple players. The small fluctuations in the curve is attributed to network dynamics.

B. Attack Latency

Attack Latency is the time between sending and Send Attack message and receiving a Attack Reply message. We measure the Round Trip Time (RTT) of attack and the one way latency can be approximated as half the RTT. Players attack one another when within a certain region of visibility γ. We present the latencies experienced in Figure 6.

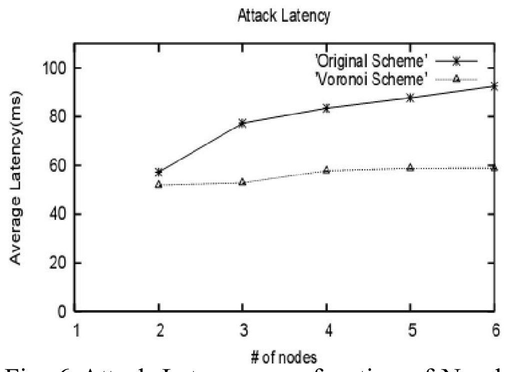

Fig. 6 Attack Latency as a function of Numbers of Players

The Attack Latency also remains more or less a constant. As discussed before, this is due to the intuitive closeness property provided by the Voronoi Partitioning. Attack Latency is critical in most First Player Shooter games like Quake, and hence it is important to have little or no fluctuations in it.

C. Eat Latency

Eat Latency is the time between sending a Food Request message and receiving a Food Reply message from the coordinator. A Food Request message is sent to the coordinator on sensing food in a region ə around. Only the food in the region of visibility γ can be consumed at any point. The latencies are presented in Figure 7.

The Eat Latency experienced by the players remains a constant in our scheme. Players maintain state for region of game space that is closest to them, and hence there is no load on a single “coordinator”. Voronoi Partitioning distributes state among players based on locality and closeness. So each node maintains only that state, which is of interest to it. As players move they no longer have to handle communication and maintain state for the old region.

D. State Transfer Latency

As players move in the game space, the state maintained at each node changes. State Transfer Latency measures the time equired to obtain new state as there is movement in the ame space. Figure 8 illustrates the time taken to perform state transfer.

Again, as seen from the graph, our implementation gives near constant performance. No single node has to handle the transfer between players. Each player maintains and exchanges almost constant amount of state.

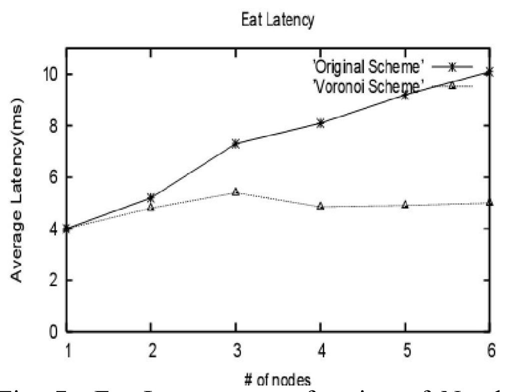

Fig. 7 Eat Latency as a function of Number of Players

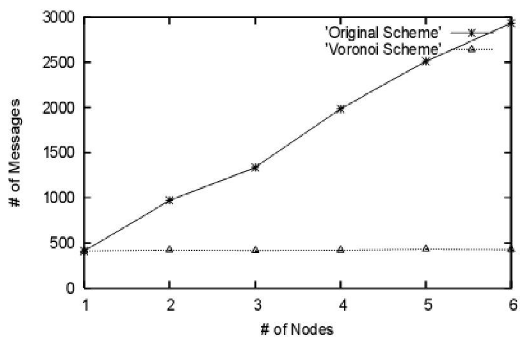

Fig. 8 State Transfer Latency as a function of Number of Players

V. FUTURE WORK

One major challenge to the proposed solution is an efficient implementation of voronoi partitioning. There is need to enable nodes to determine their voronoi regions in a distributed manner. Also, current work does not address the problem of fragmentation due to clustering of nodes in a particular area. There is need to develop an efficient solution to overcome the problem of clustering.

Current work does not implement fault tolerance. In future, we plan to implement and test fault-tolerance mechanism proposed earlier.

The communication based on voronoi partitioning is similar to Content Addressable Network (CAN) [11]. Hence, it can implement application layer multicast on the lines similar to CAN. Such a mechanism will help to communicate status updates to distant regions. For example, updates to team members which are in other corner of the playing area.

Peer-to-peer systems are highly susceptible to cheating. Empowering nodes to control game state can lead to the node manipulating the state to suit it goal. For example, a player may move through a wall, gain infinite health or drop other players' packets. Cheat prevention is one of the major challenges in supporting multiplayer games on peer-to-peer system. This work has not focused on the problem of cheat prevention. Future work aims to provide a robust cheat prevention mechanism.

VI. CONCLUSION

This work presents the implementation and evaluation of a computational geometry technique – Voronoi Diagram- to partition the game space and support it on a peer-to-peer system. We exploit the locality of interest and closeness property, and design a scalable mechanism to maintain game state and quickly propagate updates to the players.

Our initial results are promising and closeness property provided by the Voronoi Diagram can be used to efficiently support gaming systems on a peer-to-peer structure. Measurements show that latencies experienced by the players remain more or less constant. Further message exchanged is also independent of the number of simultaneous players.

In conclusion, we have demonstrated a new technique to efficiently support gaming systems on a peer-to-peer overlay. But much work needs to be done to perform the Voronoi Partitioning in a scalable, distributed, and dynamic manner to achieve the full benefits of the partitioning. In our present implementation, each node knows only the neighbors. We have not implemented any routing mechanism to achieve communication between players in arbitrary regions of game space. Routing can be performed as in CAN to achieve communication between any two arbitrary regions. Further replication must be done in an efficient manner to achieve fault tolerance, in the wake of node failures. Security is one another important aspect, that needs attention in a peer-to- peer gaming system.

We have performed an initial evaluation with limited number of nodes and by considering static partitions. The results show that our approach holds promise, but further evaluation needs to be done before strong claims can be made. We have to experiment with a larger number of nodes with a dynamically refining mesh. Further a LAN environment assumes near uniform latencies. So experiments need to be performed on a Wide Area test bed. A completely distributed dynamically refining mesh is needed to completely validate our system.

REFERENCES

[1] E. J. Berglund and D. R. Cheriton. Amaze: A multiplayer computer game. IEEE Software, 2(1), 1995.

[2] Ashwin R. Bharambe, Sanjay Rao, and Srinivasan Seshan. Mercury: a scalable publish-subscribe system for internet games. In Proceedings of the first workshop on Network and system support for games, pages 3–9. ACM Press, 2012.

[3] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In Infocom'03, April 2011.

[4] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In Proceedings of SOSP'01, October 2011.

[5] Margaret DeLap, Bjorn Knutsson, Honghui Lu, Oleg Sokolsky, Usa Sammapun, Insup Lee and Christos Tsarouchis. Is Runtime Verification Applicable to Cheat Detection?. In the proceedings of Netgames '04, ACM SIGCOMM 2011 Workshops, pp.134-138, August 2011, Portland, Oregon.

[6] Steven Fortune. A Sweepline Algorithm for Voronoi Diagrams. In the proceedings of Symposium on Computational Geometry, Yorktown Heights, NY, 2012.

[7] Bjorn Knutsson, Honghui Lu, Wei Xu and Bryan Hopkins .Peer-to-Peer Support for Massively Multiplayer Games. INFOCOM 2011, March 2004, Hong Kong, China.

[8] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In Proceedings of ASPLOS. ACM, November 2012.

[9] Nancy Lynch, Dahlia Malkhi, and David Ratajczak. Atomic data access in content addressable networks. In Proceedings of the 1st International Workshop on Peer-to-Peer, March 2012.

[10] Katherine L. Morse. Interest management in large-scale distributed simulations. Technical Report ICS-TR-96-27, University of California, Irvine, 2010.

[11] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In Proceedings of the 2011 conference on Applications, technologies, architectures, and protocols for computer communications, pages 161–172. ACM Press, 2011.

[12] Antony Rowstron and Peter Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), November 2012.

[13] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, A large-scale, persistent peer-to-peer storage utility. In Greg Ganger, editor, Proceedings of SOSP- 01, volume 35, 5 of ACM SIGOPS Operating Systems Review, pages 188–201, New York, October 21–24 2012. ACM Press.

[14] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In Roch Guerin, editor, Proceedings of SIGCOMM-01, volume 31, 4 of Computer Communication Review, pages 149–160, New York, August 27–31 2011. ACM Press.

[15] http://www.pi6.fernuni-hagen.de/GeomLab /VoroGlide/index.html.en

3/5/2013