# Area-Based Gossip Multicast

Christian Seeger
Techn. Univ.
Darmstadt, Germany
c.seeger@stud.tu-darmstadt.de

Bettina Kemme
McGill Univ., Montreal, Canada
kemme@cs.mcgill.ca

Patric Kabus
Techn. Univ.
Darmstadt, Germany
pkabus@dvs.tu-darmstadt.de

Alejandro Buchmann
Techn. Univ.
Darmstadt, Germany
buchmann@dvs.tu-darmstadt.de

## ABSTRACT

This paper presents *areacast*, a dynamic, area-based gossip multicast that enables a node to multicast messages only to nodes with a shared interest. This multicast can be used in multiplayer games where players want to send position updates, that is information that they have moved in the virtual game world, to players that reside close to them in the game world. As nodes continuously change their position the set of nodes in their neighborhood changes, too. *areacast* is able to handle this dynamism in a truly peer-to-peer fashion as each node keeps track of its current neighbors detecting new neighbors very fast. We evaluate our approach using game simulations where players either walk randomly in the game field or flock to a few hotspots. The results show that *areacast* is able to disseminate messages very fast to neighbors while not overloading the network.

## 1. INTRODUCTION

Gossip-based broadcast, such as *lpbcast* [3], uses a peer-to-peer approach to disseminate messages to a large group of nodes. Each node is connected to some other nodes in the system such that the entire system builds a connected graph. A node sends its messages to a random subset of its neighbors which in turn forward it to a subset of their neighbors. Thus, the message is slowly disseminated through the entire system. A continuous exchange of neighbor information guarantees that the network remains connected despite node churn. The reliability of message dissemination is high and nearly every messages reaches every node.

The question arises whether gossip-based broadcast can be used in the context of massively multiplayer games (MMGs). In an MMG, disseminating messages about the current position of players is one of the most common message types and doing so without server intervention is very desirable. However, *lpbcast* and other existing protocols are not suitable in this context. The reason is that these protocols lead to a high degree of redundancy as nodes often receive a message more than once. As the message throughput in MMGs is high (position updates are sent continuously), individual nodes would not be able to handle the large amount of messages they would receive. Even if every position update is only received once, this still overloads an individual node if there are many players in a game. Current peer-to-peer games usually support game sizes with tens of players but not with hundreds or thousands of players.

Standard server-based MMGs handle this scalability problem by defining a visibility range for each player: a player can only see other players and activity in its current neighborhood, and thus, the server sends to each player only actions that occur in this neighborhood. As the server has global knowledge, its interest management module can easily detect what are the actions that are of interest for each player depending on its location in the game. That is, a position update is not *broadcast to all players* in the game but only *multicast to a group of players*.

The area-based gossip multicast *areacast* that we propose does exactly the same but in a truly distributed fashion. Each node, representing one player, keeps dynamically track of other players in its current neighborhood, detecting fast when new players approach and discarding information about players that leave. Then, it multicasts position updates mainly to players in its neighborhood. It might not reach all neighbors if bandwidth requirements are too stringent. However, if a message does not reach a close-by player, it is likely that the next message will, still keeping the information quite accurate. Furthermore, message forwarding is used to provide further reliability and to detect new approaching neighbors. However, forwarding is restricted since old messages are not of big relevance. In summary, *areacast* is a completely decentralized area-based gossip protocol enabling nodes to have accurate information about other nodes in their neighborhood even if bandwidth capacity is limited.

## 2. AREA-BASED GOSSIP MULTICAST

We consider three different areas of interest for a player (see Figure 1): interaction range ($IR$), vision range ($VR$) and outside the vision range ($Inf$). The interaction range around a player defines an area where players could directly interact or compete with each other. It is very desirable that a player has up-to-date information about other players in this range in order to react adequately to their actions. The vision range defines an area where players can see each other but not interact. In this range, the need for current information about another player decreases with the distance to the local player. The closer the other player is, the more likely it will come into interaction range soon and the local player needs


*NetGames'08* Worcester, MA, USA

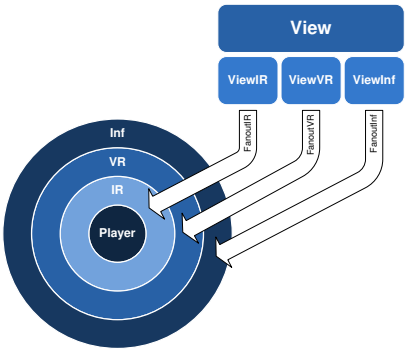


**Figure 1: Views according to distance in the game**

to beware of its actions. Outside the vision range, players cannot see nor interact with each other. Thus, it is not necessary to exchange position updates.

Having these areas of interest in mind, the design of *areacast* was guided by three desired properties. Firstly, the position updates of a player should reach the neighbors of the player in the IR and VR but there is no need that nodes outside the vision range receive the updates. That is, *areacast* is a *multicast* to neighbors and not a broadcast. Secondly, *areacast* must be dynamic as a player moves continuously and thus, the group of its neighbors changes frequently. Thirdly, updates should be received fast. If resources are scare than it is more important to get current position updates fast than to get all position updates but with a long delay. Also, there is no use to get an old position update if a more recent one has already been received.

## 2.1 Overview

In the following, we assume that each player resides on a different node. We use the terms node and player interchangeable although in the strict sense, a player refers to the gaming application and a node refers to the *areacast* communication infrastructure.

The basic idea of *areacast* is as follows. Each node maintains three views: *viewIR*, *viewVR* and *viewInf*. Each of them represents a set of nodes. Nodes belong to the corresponding view depending on their position in the virtual world. The union of all three views is a subset of all nodes in the system. A node only keeps track of another node if the latest information it has received from that node is relatively recent. Each node now periodically sends a gossip message with it own position update together with position updates it has received recently from other nodes to a random subset of nodes from each of its three views. How many nodes of each view are chosen are parameters of the system denoted as *fanoutIR*, *fanoutVR* and *fanoutInf*, respectively. As we see later, it will be mostly nodes in *viewIR* and *viewVR*, but also some of *viewInf*. The total number of nodes to send the gossip to also depends on the bandwidth capacity. When a node receives a gossip message it extracts the position updates and delivers those to the game application that are not outdated. Outdated means a more recent update has been received and delivered before. Furthermore, the node also keeps the position updates in a local buffer so that it can forward them together with its own position update in the next gossip. Only recent position updates, i.e., updates that were forwarded at most one or two times, are kept in that buffer. Older messages are no more forwarded. Therefore, they don't congest the network. Purging old position updates also guarantees that the message buffer remains small. Finally, the node also analyzes the position updates in order to keep its views up-to-date.

The algorithm has several important features. Firstly, nearby players will typically receive position updates fast, as a node sends messages mostly to its neighbors in *viewIR* and *viewVR*. Secondly, a player will detect other players that enter its vision range from outside very fast. If a node has no knowledge about another node that just enters its vision range, it is very likely that a third node transitively forwards gossips and thus introduces both to each other. Thirdly, despite having a preference for close nodes, the network does not partition. Games usually have areas where players tend to gather (we call them *hotspots*). If a player only keeps track of its nearby neighbors, then players at a hotspots might form a partition which is disconnected from the rest of the network. In order to avoid this, our solution keeps also track of some remote players.

## 2.2 Details

The system starts up with an initial number of nodes. Each node knows a limited number of other nodes building an initial connected overlay. Each node sends to its neighbor nodes its initial position in the game leading to an initial game configuration. Over time, each node will get to know more nodes in its neighborhood. New nodes can join the system by knowing a small number of nodes as starting point and starting the *areacast* protocol.

Our algorithm works in rounds whereby each round defines a time-interval. In each time-interval a node can receive and process several gossip messages and adjust its views. Once per round it sends one gossip message to other nodes.

At the beginning of a round, each node sends a *gossip message* consisting of a set of *position updates*. The gossip contains the current position update of the local player plus a random set of recently received position updates stored in the local *position update buffer*. The number of position updates in the gossip message is a parameter of the algorithm. Every position update consists of a unique player id, a player specific sequence number which is increased with every update sent, the player position, the age of the update and the network node id (e.g. IP address). Together, the sequence number and the player id uniquely identify an update and are referred to as *update id*. The update age is initially zero and increased every time the update is forwarded.

The gossip message is sent to random nodes from the three local views. The number of nodes chosen from each view is determined by the corresponding fanouts (*fanoutIR*, *fanoutVR* and *fanoutInf*).

Whenever a new gossip message is received, each included update is compared to a list of already received update ids. For this the node maintains an *update id list* containing for each player at most one update id reflecting the update with the largest sequence number received so far. Only if the new update is the most up-to-date update received from this player so far, it will be added to the position update buffer, the update id list and delivered to the application. Furthermore, *areacast* reads out the position information and inserts the player into the proper view (*viewIR, viewVR* or *viewInf*). After processing all updates included in a gossip message, updates which are older than a certain age are

deleted from the buffer. Also the list of already received update ids will be trimmed to a maximum size.

At the end of a round, just before sending the next gossip, the views are updated. Each view checks the position of the nodes it contains and moves them to one of the other views if necessary. If the age of the last position update received from a node exceeds a certain limit, the node is purged.

# 3. P2PGAME

In this section, we present our test environment P2PGame, which provides a basis for the evaluation of our *areacast* network protocol. It is based on the *PeerSim*[1] simulator, which provides a round-based modus where each round a node can send messages, receive messages and do some local processing. In our simulation, a message is received one round after it is sent. P2PGame runs on each node two components. The network layer implements the *areacast* protocol. The game layer simulates a multiplayer game and evaluates the goodness of the network protocol.

## 3.1 Game Application

The game application consists of a rectangular field on which players move freely and independent of other players. There are two different play modes. In *random mode*, each player moves with discrete steps into a random direction, and changes its direction after each round with a certain probability. In the *hotspot mode* there are additional points on the field called hotspots. Each player chooses one hotspot, moves to it and then performs random moves within a certain range of the hotspot. After a random exposure time the player moves to another hotspot. Having most of the players in a small set of areas resembles the distribution of players in many game applications.

The application provides the network layer once per round a position update of the local player. At start of the game, the local application only knows the position of the local player. Then, new players are added and their positions adjusted whenever the network delivers new position updates.

## 3.2 Protocol Quality

The basis for our measure of protocol quality is the function $PositionAge(p, q, r)$ which indicates how old the information is player $p$ has about player $q$ when $p$ is in round $r$. $r$ can be considered the current age of the game perceived by $p$. $PositionAge(p, q, r)$ is calculated as $r$ minus the game age at which $p$ has received the last position update from $q$ plus the number of hops this message took on the network. If $p$ has not yet received a position update from $q$, $PositionAge(p, q, r)$ is set to the current game age $r$ at $p$.

The protocol quality is now determined by the position age and the distance between players. We first define the protocol quality $PQ(p, q, r)$ perceived by player $p$ in regard to player $q$ in round $r$. (a) If a player $q$ is in the interaction range of $p$ then $PQ(p, q, r) = PositionAge(p, q, r)$, that is, the more accurate $p$'s knowledge about $q$ is, the smaller the $PQ$ value, and the higher the quality. (b) Player $p$ should still know about a player $q$ in its vision range outside the interaction range, but the importance of accuracy of the information decreases with the distance between $p$ and $q$. Thus, if a player $q$ is in the vision range outside the interaction range of $p$, then $PQ(p, q, r) = PositionAge(p, q, r)^{(1-\frac{dist(p,q)-IR}{VR-IR})}$.

[1]http://peersim.sourceforge.net

| Paramater | Value |
|---|---|
| Number Players | **100** / 1000 |
| Remove from View | 3 |
| Standard Game Size | $1000x1000$ |
| Vision Radius | 200 |
| Interaction Radius | 50 |
| Simulation Rounds | 500 |
| Overall fanout | **5 / 10** |
| maxUpdatesPerGossip | **60 / 10** |
| maxUpdateAge | **3 / 2** |

**Table 1: Simulation Parameters**

(c) Players outside the vision range are not considered in the calculation of the protocol quality.

The overall protocol quality $PQ(p, r)$ for $p$ in round $r$ is the average over all $PQ(p, q, r)$, $q$ being in the interaction or vision range of $p$ during round $r$. The overall protocol quality $PQ1(r)$ for a given simulation round $r$ is the average over all $PQ(p_i, r)$, $p_i$ being one of the $n$ players in the game.

$$PQ1(r) = \frac{1}{n} \sum_{i=1}^{n} PQ(p_i, r) \quad (1)$$

To get the protocol quality up to a given simulation round $R$, we calculate the average of $PQ1$ values over all simulation rounds up to $R$ and call it $PQ2$:

$$PQ2(R) = \frac{1}{R} \sum_{r=1}^{R} PQ1(r) \quad (2)$$

A perfect protocol quality would be one requiring that all nodes within the vision range get each message within one hop (with the exception of nodes that are at the exact vision range distance, which can have a position age of 2). In a client/server model, where each player $p$ sends its position updates to a central server who then forwards it to all players in $p$'s vision range would have $PositionAge$ to always be 2, and thus, the protocol quality be around 1.4-1.5 depending on the ratio of players in the interaction vs. vision range.

# 4. EXPERIMENTAL RESULTS

We conducted a wide range of experiments. Table 1 summarizes the parameters and their settings. Our simulation setup consists of 100 nodes in most experiments (two experiments test 1000 players). A player $p$ removes a player $q$ from its view if its knowledge about $q$ is older than three rounds. In general, the performance was relatively insensitive to this parameter. The game size is chosen so that despite the relative small number of total players each player has a reasonable number of other players in its interaction and vision range. We run each experiment for 500 rounds. Our experiments are conducted with fanouts of 5 or 10, the max. number of position updates in a gossip ($maxUpdatesPerGossip$) is either 60 or 10, and position updates are discarded either after three or after two hops ($maxUpdateAge$). Further forwarding values resulted in worse performance. We used a wide set of values for *fanoutIR*, *fanoutVR* and *fanoutInf*.

## 4.1 Fanout 10

We first analyze performance for a total fanout of ten, i.e., a node sends the gossip message to a total of ten other

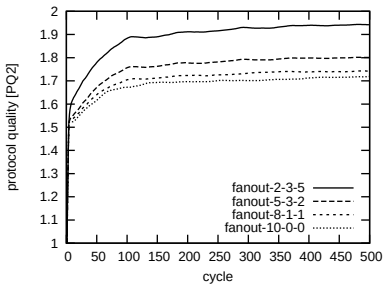


**Figure 2: *Fanout 10*: hotspot.**

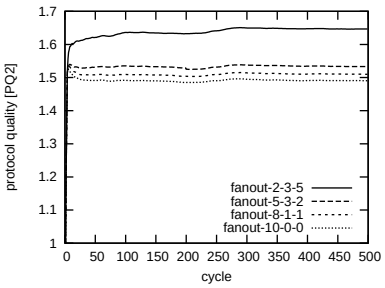


**Figure 3: *Fanout 10*: random.**

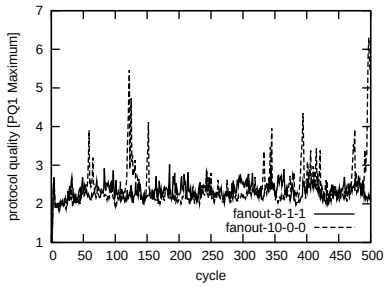


**Figure 4: *Fanout 10*: Max. $PQ1$, hotspot**

nodes. This fanout can be split differently among the three different views. The figures show in the legend the distribution as "fanout-$fanoutIR$-$fanoutVR$-$fanoutInf$". In this experiment, a gossip message contains up to 60 position updates and a position update travels at most 3 hops. Figure 2 reports the $PQ2$ values in hotspot mode for each simulation round, denoted as cycle in the figures. All protocols have very low $PQ2$ at the start, then the values increase and finally stabilize. The increase at the beginning is due to the fact that players that are not yet known have as $PositionAge$ the game age which increases with every round. Later on, all players get to know their neighbors and the absolute game age has no more influence. The worst performance (highest PQ2 value) is achieved when 5 gossips go to nodes outside the vision range and only 2 (3) go to the interaction range (vision range). Sending all messages to players in the interaction range has best performance. The same behavior can be observed in random mode, as Figure 3 shows, where focusing completely on the interaction range (cp. *fanout-10-0-0*) achieves a protocol quality of 1.49. This behavior confirms our presumption that sending messages to a player's direct neighborhood provides better results than a totally randomized approach like *lpbcast*. Although players in the vision range don't get position updates directly, they are likely to receive them through forwarding. In summary, we consider the results as very good as they are very close to what could be achieved in a client/server system.

The best protocol quality is reached with the whole fanout at the interaction range. But this configuration could cause network partitions, as mentioned before. Therefore, we have to take a look at the $PQ1$ values. Growing values indicate a network partition. Figure 4 is a comparison between the fanout configurations *fanout-8-1-1* and *fanout-10-0-0* in hotspot mode. In this case, the ordinate represents the maximum $PQ1$ value any node reached in a given cycle. Network partitions don't appear as each node eventually gets to know each other node in is neighborhood. Peaks are likely to occur when players leave one hotspot and move to another as they have to get to know many nodes in their new neighborhood. Having a more distributed fanout of *fanout-8-1-1* allows to get to know these neighbors faster, and thus, the peaks are generally smaller than with *fanout-10-0-0*.

## 4.2 Fanout 5

In order to understand the trade-offs and implications of the choice of fanout value, we also tested *areacast* with a fanout of 5, i.e, only half as many gossips are sent than in the previous experiment, considerably reducing the bandwidth requirements. Figures 5 and 6 report the protocol quality $PQ2$ for hotspot and random mode. Except of the *fanout-5-0-0* configurations the characteristics are the same as with a fanout of ten. The more messages are sent within the interaction range, the better the protocol quality.

However, if all messages go to the interaction range, then the protocol quality breaks down and becomes erratic. With a fanout of 5, sending only to nodes in the interaction range is not enough to propagate the game information fast. The effect is more severe in the hotspot mode than in the random mode. Random mode can provide better results because all players move permanently on the field and so they have consistently new players in their interaction range. Thus, the players are more easily able to propagate player positions from their former position to players in their current position. In hotspot mode player movements from one hotspot to another are not so frequent. Players in the interaction range are mostly the same and so there is less information exchange between hotspots. Therefore, the protocol quality is inferior than in random mode. Figure 7 represents a comparison of maximum $PQ1$ values between *fanout-2-2-1* and *fanout-5-0-0* in hotspot mode. The very high peaks of *fanout-5-0-0* underline our assumption of less information exchange between the hotspots. Such a performance degradation could not be observed for *fanout-10-0-0* as ten gossip messages per round in a network of 100 nodes was enough to propagate to a sufficiently large number of nodes using the forwarding mechanism.

## 4.3 Gossip Configuration

Apart of choosing a smaller fanout, we can reduce bandwidth demands by (i) reducing the number of position updates per gossip ($maxUpdatesPerGossip$), and (ii) by reducing the number of hops a position update may travel ($maxPositionAge$). In this experiment, we run tests with $maxUpdatesPerGossip = 60/10$, and $maxPositionAge = 3/2$. We use random mode and a fanout of *8-1-1* as it showed stable results in the previous experiment. In Figure 8, each of the four bars reflects the $PQ2$ value for one of the configurations at round (cycle) 500.

First, we can note that the differences are generally small and range from 1.51 to 1.525. With a $maxPositionAge = 3$, 10 updates per gossip (b) provide slightly worse performance than 60 updates (a), as with 10 updates not all updates in the buffer can be selected for forwarding.

With 60 updates per gossip (a and c), the performance is the same whether we have $maxPositionAge$ set to 3 or 2. Although 3 hops forward a position update more often, it

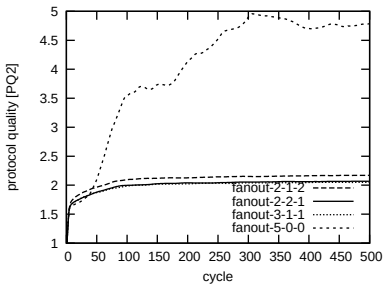


**Figure 5: *Fanout 5*: hotspot.**

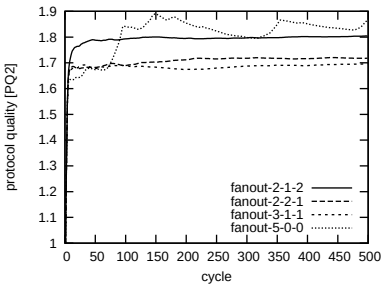


**Figure 6: *Fanout 5*: random**

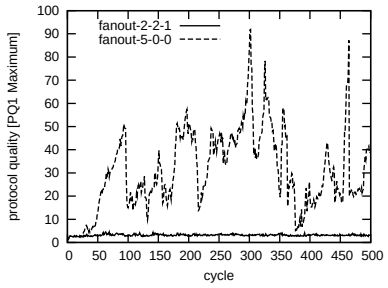


**Figure 7: *Fanout 5*: Max. $PQ1$, hotspot.**

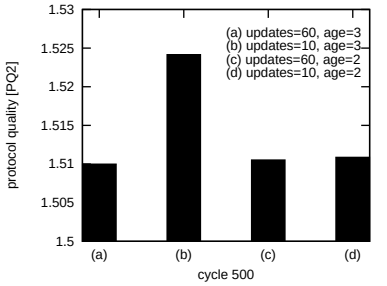


**Figure 8: Number of updates and hops**

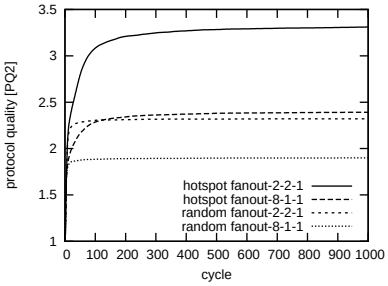


**Figure 9: 1000 Players**

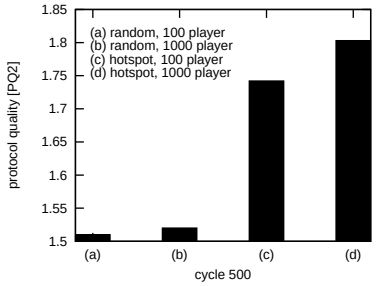


**Figure 10: Scaling the game field.**

| maxUpdatesPerGossip | Fanout 5 | Fanout 10 |
|---|---|---|
| **10** | 1.58 KByte | 3.17 KByte |
| **60** | 9.52 KByte | 19.04 KByte |

**Table 2: Outbound traffic**

does not lead to performance gains as the update is often already obsolete when being received.

Finally, when we compare the two configurations with 10 updates per gossip, we observe that forwarding more often ($maxPositionAge = 3$) actually results in worse performance. When the gossip may only contain 10 updates and we allow three hops, then old updates compete with new updates as not all updates of the buffer fit into the gossip. In contrast, when an update is only forwarded once, then the gossip can be filled with more recent position updates that have a more positive effect on the PQ2 value. In fact, with $maxPositionAge = 2$, the performance with 60 updates per gossip is the same as with 10 updates per gossip.

In summary, including few updates in a gossip and forwarding a position update only once can provide good results with little overhead.

## 4.4 Traffic

The size of a gossip is determined by the size and the number of the contained updates. Each update consists of the player id (64 bits), sequence number (64 bits), position (64 bits), age (4 bits) and the node id (64 bits). Crucial for the outgoing traffic are the $fanout$ and the maximum number of updates per gossip. Table 2 shows the resulting maximum outbound traffic per node and round. Assuming a very low outbound capacity of 15 KByte per second, *areacast* can handle 4 rounds per second for a fanout of 10 and 10 updates per message. This is a very practicable result.

Because of the protocol's probabilistic character an absolute value for incoming traffic cannot be calculated. In our simulations we never had more than 43 (18) KBytes incoming traffic for a fanout of 10 (5), and 60 updates.

## 4.5 Scalability

To test the scalability of *areacast*, we have performed two tests. First, we have used the same game world but used 1000 players over a simulation of 1000 rounds for both hotspot and random. Figure 9 represents the protocol quality reached for this scenario with a fanout of *2-2-1* and *8-1-1*. With a fanout of 10, $PQ2$ degrades about 37 percent in hotspot and about 26 percent in random mode compared to 100 nodes. Given that not only the total number of players but also the number of players in each others vision range increased 10-fold the results are acceptable. However, a fanout of 5 does not provide good results. Increasing $maxUpdateAge$ would not help as messages become too old. Thus, increasing the fanout is the better solution.

A more realistic test of scalability increases not only the number of players but also the game world itself. In such a scenario, while the total number of players increases, the number of players in the vision range remains the same. This makes sense: scaling a game allows a player to potentially meet with many more players, but at any given time, it will only interact with and see few. Figure 10 shows the $PQ2$ value at 500 rounds with a fanout of *8-1-1* for four different configurations. The first bar shows the random game mode with 100 players and the standard game size, the second shows random with 1000 players and a game size 10x the standard game size. The last two bars show the hotspot mode with 100 players / standard game size, and 1000 players / 10x game size, respectively. A gossip message can contain up to 60 updates with an age of at most 3 hops.

For random mode, we can see that the performance is nearly the same independently of the number of players as both have the same number of players in interaction and vision range, and all receive position updates very fast. For hotspot, the 1000 player configuration provides slightly worse results. The likely reason is that players now need longer to travel from one hotspot to the other, so it takes longer to adjust their views and get to know their neighbors. In summary, we can see that our approach shows excellent scalability in cases where the number of players within the interaction and vision range remains the same.

# 5. RELATED WORK

SimMud [8] distributes the game world in zones and each server is master of one zone serializing conflicting actions. Game state is disseminated within a zone using Scribe [1] resulting in multi-hop latency. In contrast, [5] handles position updates via the master node leading to a 2-hop latency. The vision range is always restricted to the zone the player resides in. In MOPAR [11], a player's vision range can span several zones but moving from zone to zone is expensive. Our approach has no predefined zones but uses one single continuous space, handling dynamism with ease.

Most similar to our work are [4, 6, 7]. All three propose a decentralized approach where players keep track of each other in their continuously changing neighborhood. However, connectivity requires the maintenance of complex and compute intensive structures such as Voronoi diagrams [4] or convex hulls [7]. In [6, 7] only overlay maintenance is considered, multicasting position updates is not included. In [6], the neighborhood always consists of a fixed number of players while we support arbitrary number of neighbors. None of the three approaches provides overhead or performance evaluations while we provide a full-fledged performance analysis.

Commercial systems rely heavily on the client-server architecture. P2P systems are only provided in very small scale, e.g., Z-Net supports up to 32 players [10]. All players interact directly with each other. In contrast, as we mainly interact with players in the vision range, the total number of players in the system can be much higher. Additionally, we allow the players in the vision range to continuously change. Thus, our approach is much more flexible.

We have already discussed that probabilistic multicast similar to the already presented *lpbcast* [3] provide high reliability and a large degree of redundancy which is not exactly what is needed in MMG. There also exist tree-based multicast protocols [1]. However, they assume that group membership is rather static which is not the case in our application. Group communication systems (GCS) [2] offer primitives to multicast messages within a group of sites. However joining or leaving the group are usually expensive operations. Also, in the context of MMG, one still has to determine when players have to join which groups.

In the area of wireless ad-hoc networks, the term Geocast refers to multicasting to all nodes currently residing in a geographical area. The main challenge in Geocast is to route the message from an arbitrary sender to some nodes in the requested area [9]. These nodes can then simply use wireless broadcast to forward the message to other nodes in the area. In contrast, our focus is exactly on finding and multicasting to neighbors in the virtual world (neighbor nodes in the virtual world are not neighbors in the physical world).

# 6. CONCLUSIONS AND FUTURE WORK

Our area-based gossip protocol provides fast and truly distributed dissemination to nodes in a continuously changing neighborhood while keeping the network connected. Messages are forwarded for reliability and connectivity but forwarding is limited as it often does not benefit protocol quality relevant for games. The performance of *areacast* is very close to that of a client-server system.

We see this paper as a first step to further explore probabilistic mechanisms for data dissemination in peer-to-peer based multiplayer games. So far, we are able to scale to 1000 players with very good performance. Can we adapt to scale even further? Can we further optimize view management, fanout decisions, and forwarding strategies? There are also some fundamental topics that need further consideration. For instance, cheating has not yet been considered. As redundancy is a natural component of probabilistic mechanisms to achieve reliability, it could be exploited to counter cheating attacks. Another issue is the treatment of game events that need conflict resolution, such as picking up objects or shooting. We could simply use a central server for them, but more elegant mechanisms might be possible that fit better into the dynamic peer-to-peer architecture.

# 7. REFERENCES


[1] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), Oct. 2002.

[2] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[3] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003.

[4] S.-Y. Hu and G.-M. Liao. Scalable peer-to-peer networked virtual environment. In *NETGAMES*, 2004.

[5] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *NETGAMES'04*.

[6] Y. Kawahara, T. Aoyama, and H. Morikawa. A peer-to-peer message exchange scheme for large-scale networked virtual environments. *Telecommunication Systems*, 25(3):353–370, 2004.

[7] J. Keller and G. Simon. Solipsis: A massively multi-participant virtual world. In *Int. Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA)*, pages 262–268, 2003.

[8] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *INFOCOM*, 2004.

[9] C. Maihöfer. A survey of geocast routing protocols. *IEEE Communication Surveys & Tutorials*, 6(2):32–42, 2004.

[10] Quazal. `www.quazal.com`.

[11] A. P. Yu and S. T. Vuong. Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *NOSSDAV*, pages 99–104. ACM, 2005.