A Case for Mutual Notification

A survey of P2P protocols for Massively Multiplayer Online Games

Stephan Krause
Institut für Telematik
Universität Karlsruhe (TH)
Germany
stkrause@tm.uka.de

ABSTRACT

Massively Multiplayer Online Games and Virtual Worlds are among the most popular applications on the Internet. As player numbers increase, the limits of the currently dominant client/server architecture are becoming obvious. To overcome those limits, the research community has developed protocols for these applications based on peer-to-peer technologies. However, no consensus has been found yet on how the potential of peer-to-peer can be optimally used for these applications.

In this paper, we compare and evaluate three classes of proposed architectures that within themselves share common design principles. One representative protocol of each class is examined in greater detail. The performance of these protocols is then evaluated in different scenarios in a series of simulations. We show, that the architecture with the best performance in message delay is the one relying on mutual notification for detecting new neighbors and on direct connections to all neighbors for exchanging event messages. Furthermore, this architecture is still competitive regarding the required bandwidth.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer Communication Networks—*Distributed Systems*

General Terms

P2P MMOG Protocol Survey

Keywords

MMOG, P2P, Simulation, Mutual Notification

1. INTRODUCTION

In recent years, Massively Multiplayer Online Games (short: MMOGs) have become a popular genre among computer gamers. Additionally, virtual worlds like Second Life have created much interest in mainstream media. Traditionally, MMOGs and Virtual Worlds rely on central servers or server farms to disseminate game events among players. As player numbers increase, however, these approaches reach their limits. Not only is maintaining a huge server infrastructure a significant cost factor in providing a MMOG, the capacity of the server also limits the number of concurrent players in a game. Most MMOG developers resort to techniques known as instancing. Instead of providing one shared world for all players, they provide several copies of the game world, each with only a fraction of the players. Players in different instances of the game world can usually not interact.

P2P technologies have emerged as a possible way to solve the scalability problem of client/server based architectures. There is a growing interest in P2P protocols for MMOGs and Virtual Worlds in the research community. Several different architectures have been proposed. However, there is no consensus yet on which of these architectures will perform best, especially because the many different possible scenarios may favor different protocols.

In this paper, we present a survey and a simulation based evaluation of possible protocol architectures for P2P based MMOGs. In section 2, we introduce three different P2P protocols belonging to three different architectural categories. In section 3, we give an overview of the simulation environment, and we present the simulation results in section 4. Section 5 concludes the paper.

2. OVERVIEW OF P2P PROTOCOLS FOR MMOG

Over the last few years several P2P protocols for MMOGs and Virtual Worlds have been developed in active research. As simulating all proposed protocols is not feasible, we categorized the P2P protocols and selected one representative protocol from each category. The following subsections will introduce these categories: ALM based protocols, supernode based protocols and mutual notification based protocols.

2.1 ALM Based Protocols

Application Layer Multicast (short: ALM) based protocols [10][17] disseminate game events and messages using standard ALM techniques. In most cases, the game space is divided into a number of subspaces. Each of these subspaces is represented by a dedicated multicast group. If an event takes place inside one subspace, the corresponding message is sent via the subspace's multicast group to all players that are interested in events in this subspace.

A player is usually only interested in events that happen inside his visual range. This range is called Area of Interest, or AOI. In many cases, the AOI is fully inside a single subspace. If a player's AOI intersects the border to another subspace, however, he also has to subscribe to the other subspace's multicast group. To avoid a flood of subscription and unsubscription messages if a player moves close to the border between subspaces, usually an *unsubscription range* slightly larger than the AOI is defined. A player only unsubscribes from a subspace if the subspace is not only outside his AOI but also outside of his unsubscription range.

SimMUD [10] is a typical example for an ALM based protocol. In SimMUD, the game space is divided into fixed regions with unique IDs. The ID of the subspace is hashed by a collision resistant hash function like SHA-1. The resulting value serves as the group ID of the associated multicast group. If additional state is required for the region (e.g., the location of non-player objects such as chests), this information is stored at the root of the multicast tree.

SimMUD relies on Scribe [3] for dissemination of the multicast messages. Scribe in turn is built on top of the peer-to-peer key based routing protocol Pastry [15]. In Pastry, each node has an unique ID. Nodes can send messages to arbitrary keys; these will be delivered to the node with the numerically closest ID. As it is infeasible for a node to know all other participants of the network, each node has a routing table limited to $\log(N)$ entries. If a message is sent to the network, it will be recursively routed to the routing table entry with the numerically closest ID. Pastry guarantees that this will take at most $O(\log(N))$ steps.

Scribe exploits this recursive routing mechanism. When a node wants to join a multicast group, it sends a JOIN message to the group's ID. Each node that routes this message becomes a forwarder for the multicast group. Thus, a forwarding tree with a depth of $O(\log(N))$ is created without any additional messaging overhead.

If a node fails, this will be detected by his children in the tree. These will rejoin the overlay network. As the new JOIN message is routed, the damaged multicast tree will repair itself. If the failing node has been root of a multicast tree, the new root will detect this by receiving the JOIN messages from the former children of the failed node. As the new potential root is always the numerically second closest node to the group ID, a multicast root can also backup additional informations for the subspace at this node in advance. Hence, the new multicast root can take over the multicast group seamlessly.

2.2 Supernode Based Protocols

As in ALM based protocols, in supernode based protocols the gamespace is usually divided into subspaces. The subspaces can be either of a fixed size [18] or dynamically based on player density [7]. Each subspace is assigned to a responsible node, or supernode. This supernode will receive all game event messages relevant for his subspace and distribute them to all subspace members. Players have to register at the supernodes of the subspaces of their interest. The subspaces a node is interested in are determined analogous to ALM based protocols, i.e. the concepts AOI and unsubscription range apply.

In case of dynamic subspace sizes, overloading of the supernode is prevented by limiting the number of players per subspace: If too many players are inside a certain subspace, it will be divided into a number of smaller subspaces. With fixed subspaces, other mechanisms are needed.

As an example for this class of protocols, we have chosen the publish/subscribe based approach presented in [18] (we will call this protocol PubSubMMOG from now on). It uses fixed sized subspaces and a load balancing approach. If the number of players in a subspace becomes too high, the supernode will start to request so-called *intermediate nodes*. These intermediate nodes will form a load balancing tree, alleviating the supernode's load.

For a better allocation of global resources PubSubMMOG utilizes a central entity called lobby server. All nodes register their capacities (bandwidth, available CPU resources, etc.) at this lobby server. If new supernodes or intermediate nodes are needed, the lobby server tasks the node with the most resources available.

If the supernode fails, the message dissemination inside the supernode's subspace stops. To avoid a disruption of the game for players, each supernode in PubSubMMOG has a designated backup node. Whenever a player joins or leaves the subspace, the game state is synchronized between supernode and backup node. If the backup node detects the failure of the supernode, it can instantly replace the old supernode and resume the stalled message dissemination, thus minimizing the impact of a supernode failure.

PubSubMMOG incorporates a timeslot mechanism to allow for the aggregation of messages at the supernode: The supernode will accept messages only during the first half of a timeslot. Then it will relay all received messages at once. Messages coming too late will be discarded. While this aggregation makes a global ordering of events easier and may help the message dissemination, it limits the maximum latency: players, whose messages need more than timeslot/2 to reach the supernode will not be able to publish their event messages.

2.3 Mutual Notification Based Protocols

In contrast to ALM and supernode based protocols, mutual notification based protocols do not divide the gamespace. Instead, all players send the game event messages directly to all other players inside their AOI. This minimizes the delay of event propagation. As a prerequisite for this, all players have to be aware of all other players inside their AOI. To learn about players recently moved into a player's AOI, each player has to rely on his neighbors for information.

We have chosen *Vast* [8] to represent the class of mutual notification based protocols. In Vast, each player computes a Voronoi Diagram [6] of all his known neighbors. Nodes whose Voronoi region intersects the outer border of the player's AOI are called *boundary neighbors*, nodes whose Voronoi region is adjacent to the player's own Voronoi region are called *enclosing neighbors*. Note: A neighbor's position can be outside a player's AOI. Also, a node can be boundary neighbor and enclosing neighbor at the same time.

Every time a player moves he notifies all his neighbors about his movement. Every time a player receives a movement notification, he updates his local Voronoi diagram. By calculating the differences between the old and the new diagram he can deduce whether one of his neighbors has to be informed about the moving neighbors' presence. In general, only boundary neighbors have to send information about their enclosing neighbors. To reduce traffic this check only has to be done by boundary neighbors with their enclosing neighbors.

Joining nodes can enter the overlay by greedily routing a JOIN message to the closest player. Failed players can be detected by the absence of movement messages.

3. SIMULATION

We evaluated the protocols introduced in section 2 by running simulations with different scenarios. For all simulations we used OverSim [2] as the simulation environment.

3.1 Simulation parameters

The simulations were done using OverSim's *SimpleUnderlay*. In this underlay model, nodes are placed into a two dimensional space. Message delay is calculated using the nodes' access net delay and the nodes' euclidean distance. This allows simple, yet very realistic delay calculations for Internet-like settings [12]. Node positions were chosen to fit the measurements from CAIDA's skitter project [11].

Node joins and leaves were initiated by OverSim's *ParetoChurn* churn generator. This churn generator simulates heavy tailed node session times, using a Pareto distribution for calculating a node's live and dead time [19]. Heavy tailed session times in MMOGs are supported by several measurement studies [13][4]. Our average mean session time was 100 minutes [4]. The churn generator was configured to create on average 500 live nodes. Each simulation run lasted two hours of simulated time.

For a realistic simulation of a MMOG we implemented a simple gaming application. This application is aware of a player's position on a virtual game field, as well as the position of all players in his vicinity; the AOI size is set to 40 meters. Players move according to a random waypoint model. As players in MMOGs tend to play together in groups, we extended this movement model to a group based random waypoint. Here, a configured number of players agree on a common waypoint. To avoid players walking in a straight line, a flocking algorithm [14] is applied to the moving players. The walking speed of all players is set to $5m/s$, which is roughly the running speed used in World of Warcraft. As players move, the gaming application generates position updates which are then sent to the peer-to-peer protocol layer. In Vast and SimMUD, this is done five times a second, which is a realistic average movement update frequency for MMOGs [4]. Due to the limitations in PubSubMMOG's timeslot model, only two updates per second are sent here[1].

In PubSubMMOG and SimMUD, the game map is divided into $10 * 10$ subspaces.

3.2 Scenarios

The most important factor affecting the performance of a peer-to-peer protocol is the player density. This factor can be changed in two different ways. Given a fixed number of players, the first way is to change the size of the game area. Smaller areas mean a higher global node density. In protocols using subspaces, this means players are more often part of more than one subspace (thus increasing the average number of players per subspace) and have to switch subspaces more frequently. In protocols with a direct connection between neighbors, a higher density leads to an increase of players inside the AOI.

The other way to change the player density is employing group-based movement schemes and changing the size of the groups. A bigger group size means higher variance of the player density, with areas crowded by a high number of players and others completely empty.

Our simulations were done on a total game area of $1,000m * 1,000m$, $5,000m * 5,000m$ and $10,000m * 10,000m$. Players formed groups of 1, 5, 15, 25 and 40.

4. RESULTS

The most important criteria for judging the performance of network protocols for MMOGs and Virtual Worlds are the event message propagation delay, i.e. the time an event message needs to reach the players, and the required bandwidth. While players tolerate a small event message delay, high latencies impair the players' gaming experience [5]. In P2P protocols for MMOGs, the message dissemination is done by the players. As these usually have limited bandwidth capacities, the bandwidth demands of these protocols must not exceed a certain threshold.

In our simulation, we measured the difference between the creation time of a movement event and the time of reception of this event by other players, as well as the bandwidth consumption. The figures show the behaviour of these values dependent on the group size for different playground sizes.

4.1 Delay

Studies suggest that in MMOGs players will tolerate event message propagation delays of up to 500 ms [5]. In fact, even smaller latencies will be detected by players [16].

Figure 1 depicts the average time it takes for a movement indication to reach a distant player on a playground of $10,000m * 10,000m$.

The best performing protocol is Vast. As in all mutual notification protocols, it takes a movement notification packet only one overlay hop to reach its destination. Thus, the movement delay is the same as the average delay between two randomly chosen players in the overlay, which is slightly above 150ms. Obviously, the delay is independent of the group size.

In SimMUD, the ALM based protocol, a movement notification needs an average of 360ms to reach the other players. However, due to the unpredictable structure of Scribe's multicast trees, there is a lot of variation in the delays. Because the depth of the multicast tree is independent of the number of players in a subspace, the delay is not affected by bigger group sizes.

As in PubSubMMOG the supernode constructs a load balancing tree if too many nodes enter his subspace, the delays increase with greater group size. The delays start at 430ms for a group size of one, reaching 470ms for a group size of 40. A major fraction of this delay is not caused by message propagation times but by PubSubMMOG's timeslot concept: Each movement message has to wait for an average timeslot/2 until it is relayed by the supernode. This means that without the timeslots, movement messages could reach their destinations 250ms faster.

Figure 2 shows the results for a playground size of $5,000m * 5,000m$. Like in the scenario above, Vast stays at 150ms regardless of the group size. SimMUD's delay is slightly higher than before, this time remaining at about 370ms. PubSubMMOG shows a slight increase in delay, as well, now ranging from 450ms to 500ms.

Figure 3 shows the results for a playground size of $1,000m * 1,000m$. Vast's delay is still not affected, neither by the

[1]See section 2.2 for more details.

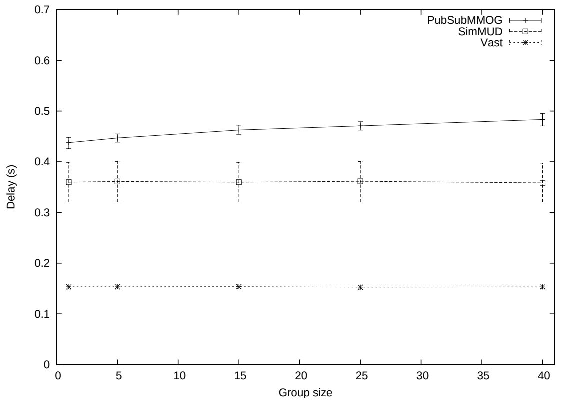

Figure 1: Average event message propagation delay, playgroundsize: $10,000m * 10,000m$

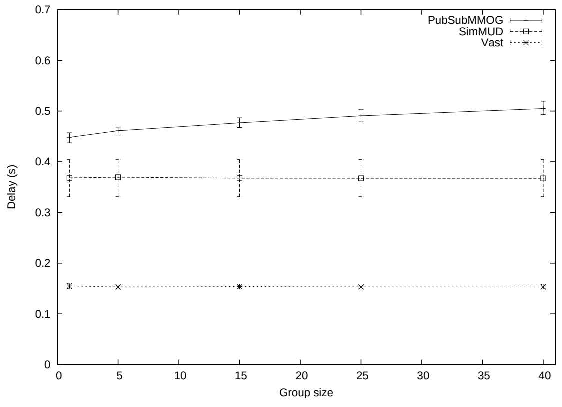

Figure 2: Average event message propagation delay, playgroundsize: $5,000m * 5,000m$

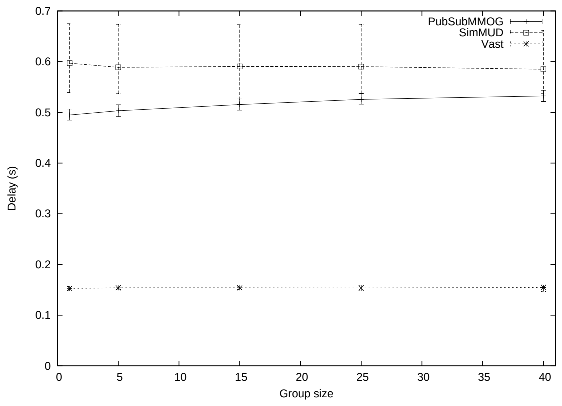

Figure 3: Average event message propagation delay, playgroundsize: $1,000m * 1,000m$

smaller playground size nor the group size. In this scenario, SimMUD's performance is significantly worse than in the scenarios above: the average delay is slightly below 600ms, which is an increase of about 60%.

As in the scenario above, PubSubMMOG shows a slight increase in the average delay: the average delay starts at 490ms for a group size of one, and reaches 520ms for a group size of 40.

Summary: Due to its one-hop design, Vast outperforms all other protocols by a factor of 2 to 4 depending on the scenario. Additionally, the performance is not affected by local or global player density.

SimMUD suffers the highest deterioration in performance when player density increases. While on a bigger playground the delays are all in the well playable range, in the $1,000m * 1,000m$ scenario the average delay, nearly 600ms, is too high for most applications.

Global density is not a big problem for PubSubMMOG: While delays increase with smaller playground sizes, the difference is barely noticeable. Due to the construction of the load balancing tree, the delays increase with local player density. However, as the depth of the tree increases logarithmically with the number of players in a subspace, the delay increase stays within reasonable boundaries. The biggest drawback of PubSubMMOG is the subspace concept, as it adds an average of 250ms to all delays. Without this additional delay, movement indications would have reached players at an average of 200-270ms.

4.2 Bandwidth

During our simulations, bandwidth usage with all protocols in all scenarios remained within reasonable boundaries. However, we observed significant differences between the evaluated protocols. Due to the limitation in PubSubMMOG to deliver more than two movement updates per second[2], its bandwidth usage is not directly comparable to Vast and SimMUD. As with these protocols 2.5 times as many event messages have to be distributed to the players, the results from PubSubMMOG have to be multiplied by that factor to become roughly equivalent.

Figure 4 shows the average bandwidth per node in dependency of the group size in a playground of $10,000m * 10,000m$. Vast uses approximately 1,1kByte/s. As in sparse scenarios Vast connects to a higher number of neighbors outside a player's AOI, increasing group size has no significant influence on bandwidth usage.

SimMUD's bandwidth use stays at a slightly better 900 Bytes/s, also unaffected by the group size. However, it shows a significantly higher variance than the other two protocols: nodes which are involved in disseminating messages, for example the root of a multicast tree, need a much higher bandwidth than nodes which act only as leaves in these trees. As nodes have no influence on who becomes involved in a multicast tree, there are no means to evenly spread the load among players.

In contrast to the other protocols, PubSubMMOG's bandwidth requirements rise with increasing group size: starting with 300 Bytes/s, or 750 Bytes/s when compensating for the lower amount of generated movement messages for a group size of one, the bandwidth requirements rise up to 570 Bytes/s, or an equivalent of 1,4 kByte/s, for a group size of 40.

This behavior can be explained by the increase of the local density of players when increasing the group size. Bigger groups lead to a number of empty subspaces, while other subspaces will be swarmed by players. In these subspaces, a linearly increasing number of intermediate nodes is needed

[2]See sections 2.2 and 3.1 for details.

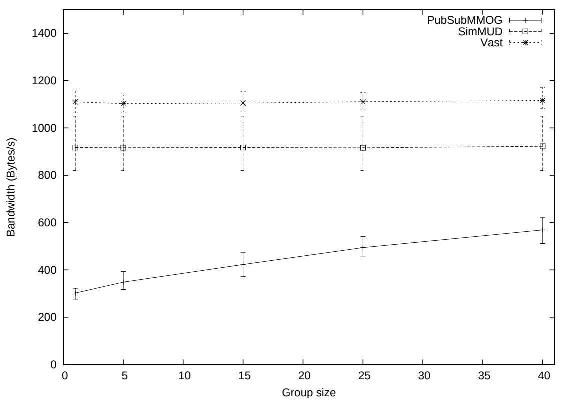

Figure 4: Average bandwidth, playground-size: $10,000m * 10,000m$

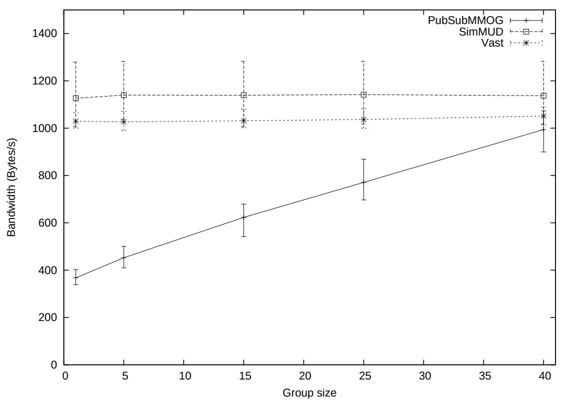

Figure 5: Average bandwidth, playground-size: $5,000m * 5,000m$

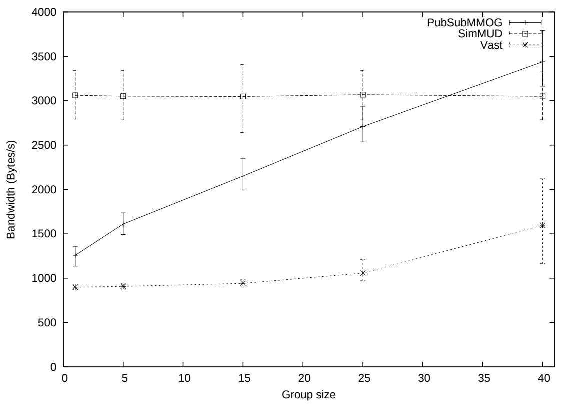

Figure 6: Average bandwidth, playground-size: $1,000m * 1,000m$

to construct the load balancing tree. Thus, the total bandwidth needed to deliver the movement messages increases.

Figure 5 shows the bandwidth measurement results on a smaller playground of $5,000m*5,000m$. Vast bandwidth requirements do not change much compared to the $10,000m * 10,000m$ playground scenario. There is still no influence of the group size.

SimMUD shows a slight increase in traffic. It averages slightly above 1,1 kByte/s: Because of a smaller playground, the subspaces are smaller. While the average number of players inside a subspace remains constant, a player has to subscribe to multiple subspaces more often because he is near a subspace boundary.

Looking at PubSubMMOG, the influence of group size becomes even more obvious. With a group size of one, the traffic, 350 (875) kByte/s is not much higher than in the first scenario. However, with a group size of 40 the bandwidth requirements rises to 1 kByte/s (2.5 kByte/s).

Figure 6 shows the results from the last scenario with a playground size of $1,000m*1,000m$. In this scenario, Vast's bandwidth usage starts to be affected by group size: With a group size of 40, Vast uses 1,5 kByte/s, which is about 50% more than with a group size of 1. Additionally, the variance starts to get significant. This is caused by consistency problems that arise in scenarios with high node densities[1]: some players will be dropped and be unable to find new neighbors, thus sending no data at all. At some additional traffic costs, these consistency problems might be alleviated[9].

SimMUD is the only protocol with a bandwidth requirement still not affected by the group size: it levels at slightly above 3 kByte/s. However, this is about three times as much as SimMUD used in the scenarios above.

PubSubMMOG sees a dramatic increase in traffic as well: it starts at 1.25 kByte/s (3.1 kByte/s) for a group size of 1, reaching 3.5 kByte/s (8.75 kByte/s) for a group size of 40.

Summary: For a wide range of parameters, Vast's traffic usage is not affected by local or global player density. Only if the local player density becomes high[3], Vast starts to generate significantly more traffic.

SimMUD seems to be completely unaffected by local player density. However, an increase of the global player density will severely affect SimMUD.

PubSubMMOG is affected by both global density and local density. Especially with bigger group sizes, the dynamic construction of load balancing trees seems to be even more inefficient than SimMUD's implicit multicast trees.

5. CONCLUSIONS AND FUTURE WORK

We have shown with our simulations, that Vast performs best regarding the delay criterion. As all mutual notification protocols share the design principle of one-hop communication for event messages, this result can be generalized. Additionally, mutual notification ensures that the message delay is not affected by global or local player density, as long as a player does not exhaust his bandwidth capacity. As shown in the simulations, the bandwidth requirements of a mutual notification protocol are in most cases moderate, and only increase in case of extreme local and global player density. However, in these cases the bandwidth demand was smaller than the demand of the other protocols.

ALM based protocols like SimMUD show moderate delays and average bandwidth requirements. As they are based on well researched ALM protocols, they are in most cases easy to implement and stable, making them a good alternative for specific scenarios. However, they do not cope well with high

[3]500 players in an area of $1000m * 1000m$ corresponds roughly to the most crowded area in World of Warcraft during peak time.

global density, which leads to high bandwidth consumption and almost unacceptable delays.

Most simulation results for PubSubMMOG show a relatively bad performance. However, most problems arose from PubSubMMOG's poor timeslot design. Without that, the delays of PubSubMMOG would have been fairly competitive, albeit not as good as Vast's results. As an important advantage over ALM based SimMUD, the delay did not increase much when confronted with high player densities. High local densities had some impact, but only increased latencies logarithmically. This advantage came at the cost of a fairly high bandwidth consumption.

As mutual notification based protocols seem to be the most promising candidates for low-delay multiplayer gaming, we would like to further evaluate this class of protocols. A problem in these networks can be stability [9]. Increasing stability will have some impact on the required bandwidth. Therefore, evaluations will have to be conducted on how to maximize stability while not exceeding a player's available bandwidth.

6. REFERENCES

[1] Backhaus, H., and Krause, S. Voronoi-based adaptive scalable transfer revisited: gain and loss of a voronoi-based peer-to-peer approach for mmog. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (New York, NY, USA, 2007), ACM, pp. 49–54.

[2] Baumgart, I., Heep, B., and Krause, S. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA* (May 2007), pp. 79–84.

[3] Castro, M., Druschel, P., Kermarrec, A.-M., and Rowstron, A. Scribe: a large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on 20*, 8 (Oct 2002), 1489–1499.

[4] Chen, K.-T., Huang, P., and Lei, C.-L. Game traffic analysis: an mmorpg perspective. *Comput. Netw. 50*, 16 (2006), 3002–3023.

[5] Claypool, M., and Claypool, K. Latency and player actions in online games. *Commun. ACM 49*, 11 (2006), 40–45.

[6] Fortune, S. A sweepline algorithm for voronoi diagrams. In *SCG '86: Proceedings of the second annual symposium on Computational geometry* (New York, NY, USA, 1986), ACM, pp. 313–322.

[7] GauthierDickey, C., Lo, V., and Zappala, D. Using n-trees for scalable event ordering in peer-to-peer games. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video* (New York, NY, USA, 2005), ACM, pp. 87–92.

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

[9] Jiang, J.-R., Chiou, J.-S., and Hu, S.-Y. Enhancing neighborship consistency for peer-to-peer distributed virtual environments. In *ICDCSW '07: Proceedings of the 27th International Conference on Distributed Computing Systems Workshops* (Washington, DC, USA, 2007), IEEE Computer Society, p. 71.

[10] Knutsson, B., Lu, H., Xu, W., and Hopkins, B. Peer-to-peer support for massively multiplayer games. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies 1* (March 2004), –107.

[11] Ma, A. Caida : tools : measurement : skitter. http://www.caida.org/tools/measurement/skitter/, June 2008. accessed on 2008-06-08.

[12] Ng, T., and Zhang, H. Predicting internet network distance with coordinates-based approaches. *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE 1* (2002), 170–179 vol.1.

[13] Pittman, D., and GauthierDickey, C. A measurement study of virtual populations in massively multiplayer online games. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games* (New York, NY, USA, 2007), ACM, pp. 25–30.

[14] Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 25–34.

[15] Rowstron, A., and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001. Proceedings* (2001), vol. Volume 2218/2001, Springer Berlin / Heidelberg.

[16] Smed, J., Kaukoranta, T., and Hakonen, H. Aspects of networking in multiplayer computer games. In *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century* (Hong Kong SAR, China, Nov. 2001), L. W. Sing, W. H. Man, and W. Wai, Eds., pp. 74–81.

[17] Vik, K.-H. Game state and event distribution using proxy technology and application layer multicast. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia* (New York, NY, USA, 2005), ACM, pp. 1041–1042.

[18] Yamamoto, S., Murata, Y., Yasumoto, K., and Ito, M. A distributed event delivery method with load balancing for mmorpg. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* (New York, NY, USA, 2005), ACM, pp. 1–8.

[19] Yao, Z., Leonard, D., Wang, X., and Loguinov, D. Modeling heterogeneous user churn and local resilience of unstructured p2p networks. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 32–41.