Adaptive Update Propagation for Low-Latency Massively Multi-User Virtual Environments

Richard Sueselbeck, Gregor Schiele, Sebastian Seitz and Christian Becker
University of Mannheim
Mannheim, Germany
{ richard.sueselbeck | gregor.schiele | sebastian.seitz | christian.becker }@uni-mannheim.de

***Abstract*—Massively Multi-User Virtual Environments (MMVEs) are highly interactive systems. They require the propagation of state updates to users with little delay. In this paper we propose a novel update propagation approach for MMVEs that enables such low-latency propagation while offering the scalability needed to support MMVEs with massive user numbers. Our approach combines peer-based and server-based update propagation into a hybrid system. It adapts itself dynamically to the available system resources and the current situation in the virtual world. We describe our approach in detail, evaluate it and discuss further steps towards a low-latency update propagation system.**

I. Introduction

Massively Multi-User Virtual Environments (MMVEs) allow thousands of users worldwide to interact with each other in a common virtual environment in real-time. Such systems are highly interactive. Users expect the virtual environment to react to their actions immediately. Thus, updates of the state of an MMVE must be propagated with little delay. To do so, a suitable update propagation subsystem is needed, which ensures that all users receive all updates they require in time. Current MMVEs are based on client/server-architectures. A central server cluster collects all user actions and sends update messages containing an updated state of the virtual world to all clients. However, this approach introduces an additional delay for update propagation since updates are first sent to the server and then back to the clients. Lower delays are possible by using directly connected peer computers. Peers send updates directly to each other. Unfortunately, due to the limited upload bandwidth of the peers, this approach is not scalable enough to be usable for MMVEs with massive user numbers.

In this paper we propose an update propagation subsystem for MMVEs that combines both of these approaches into a hybrid system. Based on a central server we dynamically shift the responsibility for update propagation between the server and individual peers such that the resulting propagation delay is minimized. At the same time we maintain the scalability of the server-based architecture. Our approach is based on the notion of so-called Areas of Propagation (AoP). An AoP determines the area in the virtual world for which a peer can distribute updates directly. That is, if an update occurs on a peer, the peer first checks if the update will influence only peers that are in its own AoP. If so, it delivers the update to them directly. Otherwise, it notifies the server. The size of a peer's AoP is selected dynamically depending on the peer's available bandwidth and the density of peers in the virtual world. This allows to communicating directly with the maximum possible number of peers in the virtual environment, while maintaining the scalability of previous approaches.

Our contributions in this paper are as follows: first, we introduce AoPs, our concept to integrate peer-to-peer-based and server-based update propagation. Second, we present an approach to create and maintain AoPs at runtime. Third, we show how to use AoPs to create a scalable and low delay update propagation system. Finally, we give a short evaluation of our approach and compare it to pure P2P-based and pure server-based update propagation.

Our work is part of the peers@play project [3], a cooperative project of the Universities of Mannheim, Duisburg-Essen and Hannover to develop protocols and algorithms for highly scalable and interactive MMVEs. We build upon existing functionality from this project to realize our approach. Where necessary, we describe this functionality briefly.

II. Related Work

There have been a number of proposals for hybrid or P2P-based MMVE architectures. HYMS [4] is a hybrid approach, which divides the virtual world into cells. Normally the server is responsible for update propagation, but when a suitable client is available, it takes responsibility for update propagation in the cell. The other clients in this cell then connect to this client instead of the server. This reduces server load, but does not decrease latency as updates are still sent via another system. Rooney et al. [5] is another hybrid approach, in which server-based Multicast Reflectors provide both update filtering and propagation for subregions within the virtual world. The focus of this approach is to move the processing of the game state to the peers, while update propagation is still handled by the server. Several fully P2P-based approaches use a coordinator-based approach, in which a selected peer forwards update messages to all peers in a subregion of the virtual world. In Knutsson et al.'s [6] proposal the majority of updates take between one and six hops to be delivered. Due to relaying, some updates require more than 50 hops to reach their destination peer. Iimura et al [7] and the MOPAR scheme [8] also divide the world into several regions and send all updates in a region via a coordinator node. For the purposes of latency, these proposals deliver similar results than a server-based architecture. In the VON [9] and Solipsis [10]

978-1-4244-4581-3/09/$25.00 ©2009 IEEE

projects a peer communicates directly with its neighbor peers. Both approaches determine these neighbors by partitioning the virtual world using Voronoi diagrams. In VON each peer only communicates directly with a small number of neighbors called enclosing neighbors. Updates to other peers are sent via a forwarding model. Depending on the position of the destination peer, a message may need to be forwarded multiple times, leading to high delay. Similarly, a peer in Solipsis only communicates with a small number of neighbors. Messages are delivered via a greedy routing algorithm. With the addition of long-range links, Solipsis needs a polylogarithmic number of hops on average to deliver an update. In contrast, our approach dynamically adapts the update propagation to the available system resources and the current situation in the virtual world, thus minimizing delay.

III. System Model

Our system consists of a set of end user computers that are connected via the Internet. We therefore assume that the system cannot utilize a global multicast. Each computer executes the MMVE software and acts as a peer in the system. We also assume the existence of a central MMVE server, possibly realized using a server cluster. The MMVE software consists of several subsystems. A user interface presents the MMVE to the user and allows him to issue commands, e.g. moving his avatar or chatting with other users. Commands lead to update messages being created, which in turn are handed to the update propagation subsystem for distribution to the correct peers. The propagation subsystem builds upon a basic network abstraction subsystem, which allows peers to send messages to other peers, regardless of networking details like NAT or firewalls. Our project partners at the University of Duisburg-Essen have developed and implemented such a subsystem in previous work [12] and integrated it into the existing peers@play MMVE prototype.

IV. Our Approach

Our goal is to provide an update propagation system for highly interactive and scalable MMVEs. Such a system has to perform two functions: first it must determine to which peers a given update should be propagated (update filtering). Second, it must deliver the update to these peers with as little delay as possible (update propagation). Concerning the first function, a typical concept for determining the target peers of a given update are so-called Areas of Interest (AoI) [13] [14]. An AoI is a spatial area around a virtual object in which updates affect the object. Thus, any update occuring inside an object's AoI should be propagated to it. Updates occuring outside the AoI are discarded. In previous work we have developed an update filtering mechanism based on this concept [15].

In this paper, we focus on update propagation, i.e., how to deliver updates with little delay. The easiest way to achieve this would be a fully connected P2P overlay network in which all peers are directly connected to each other. Using this network, each peer sends its updates directly to all other peers. This approach does not scale however, as the individual peers do not have sufficient resources to communicate with all other peers in the system. Therefore many MMVEs are based on a client/server architecture. All state updates are first sent to the server, which then forwards them to all peers. However, this indirect distribution via the server introduces additional delay, as the update messages must first be send to the server and then to the peers. In short, the server-based approach induces additional delay, while the fully connected P2P approach does not scale for a large number of peers.

We propose to combine these approaches into a flexible hybrid system, in which updates are distributed either directly between peers or via a server. The actual weight between both approaches is determined dynamically for each peer and adapted at runtime. This allows us to balance scalability and resulting delay flexibly for each update. In this section we describe this hybrid approach and show how the P2P overlay and the server are dynamically integrated. The remainder of this section is structured as follows: first we describe our pre-existing update filtering algorithm, i.e. how the set of peers that should receive a given update is determined. Then we introduce the main concept of our approach, the so-called Area of Propagation (AoP). The AoP specifies how the responsibility for update propagation is distributed between peers and the server. We also detail how the AoP is determined at runtime. Finally, we describe how an update is propagated in the system using the AoP.

A. Update Filtering

The goal of our update filtering algorithm is to determine which peers are interested in a specific update. This significantly reduces the number of updates that are sent to peers and thus enhances the system's scalability.

Our algorithm is based on two types of areas in the virtual world. The **Area of Interest** (AoI) is defined as the area of the virtual world, in which a certain peer can perceive events. This means that the peer is interested in all updates that either occur within this area or influence it. The size of this area depends on the peer it is associated with. For example, a peer standing in a closed room does not perceive anything outside of the room, thus the room determines its AoI. A peer standing on a mountain can see far away events, leading to an AoI that is several miles in diameter. We extend this concept with a second type of area, called **Area of Effect** (AoE). We define an AoE as the area in the virtual world, in which a certain event affects the world. The size and shape of this area depends on the event it is associated with. As an example, picking up an object influences an area that is equivalent to the size of the object. Both AoI and AoE can have arbitrary shapes. In this paper we restrict ourselves to circular shapes.

In summary, the AoI is associated with a peer and represents the part of the virtual world in which the peer can be affected by events. The AoE is associated with an event and represents the part of the virtual world in which the event can affect peers. This means that by delivering an event to all peers who's AoI intersects with the event's AoE, we can ensure that the event is delivered to all peers that are interested in it. The algorithm to

determine all peers p that a given update u should be delivered to is thus very simple: it compares the AoE of u with the AoI of each p. If they intersect, u is delivered to p. Otherwise, it is not delivered.

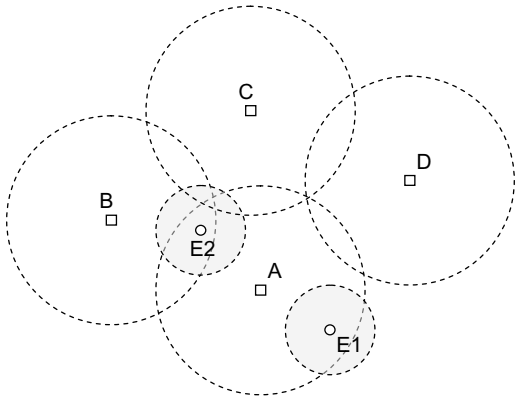


Fig. 1. AoI/AoE Example

Figure 1 illustrates this with an example. Peer A is generating events E1 and E2. The circular areas around the events represent their AoEs, while the circular areas around peers A, B, C and D represent their AoIs. As can be seen, event E1's AoE does not intersect with any peer's AoI. This means there is no peer which has interest in it. Therefore no update needs to be sent for this event at all, conserving bandwidth. The AoE of event E2 intersects with the AoI of both Peer B and Peer C. This means that both peers are affected by the event and need to receive its corresponding update. Peer D however is not interested in the event. Peer A therefore delivers the update to peers B and C.

Using this update filtering algorithm we can determine the target set of peers that should receive a given update. With this information, we now need to deliver the update to these peers.

B. Area of Propagation

We begin the presentation of our update propagation approach by introducing our main novel concept, the so-called **Area of Propagation** (AoP). Afterwards we describe how to use AoPs for achieving low delay update delivery.

In our approach, each peer is associated with its own surrounding AoP. A peer's AoP is the spatial area in the MMVE for which the peer has all necessary knowledge to perform both update filtering and propagation on its own. More specifically, this means that the peer knows the current location and AoI of all peers in the AoP. The peer uses this knowledge to filter updates and deliver them to peers in the AoP directly. Updates affecting the MMVE outside the AoP are forwarded to the server and processed there. The size of the AoP is determined by two factors: its peer's maximum available upstream bandwidth and the location of its neighboring peers. The available upstream determines the number of neighbors n to which the peer can send updates simultaneously. Given this upper limit, we choose the peer's AoP as a circular area whose diameter is set such that the AoP contains at most n neighbors.

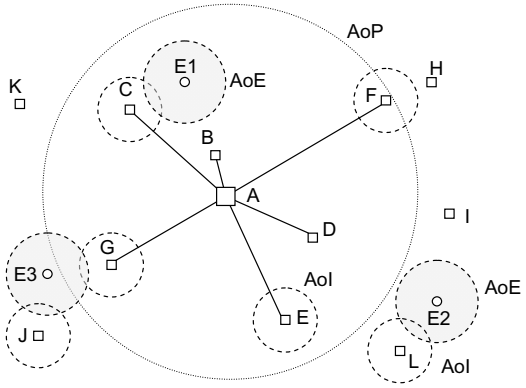


Fig. 2. AoP Example

Figure 2 illustrates this concept. Peer A has sufficient bandwidth to send updates to six neighbors. Its AoP is therefore set to contain its six closest neighbors B to G. Updates affecting these peers are sent to them directly, resulting in little delay. This is the case for update E1. However, there is not enough bandwidth to communicate directly with peers H to L, as they are outside A's AoP. To deliver an update to them, A sends the update to the server which determines the target peers and forwards the update to them. An example for this is update E2. A special case occurs if the event is relevant both for peers inside and outside the AoP, e.g. event E3. In this case A delivers the update to all relevant peers in its AoP, while the server delivers it to all relevant peers outside the AoP.

A more detailed description of this algorithm – including several additional special cases – is given in Section IV-D.

C. AoP Generation

In this section we describe how the AoP is created and maintained. As discussed previously, a peer's AoP is determined dynamically depending on its available upstream bandwidth and the location of peers in the virtual world. In this section we present the algorithm to do so. This algorithm determines the list of neighbors that are included in a certain peer's AoP. It is executed regularly by the server for all peers. If the members of a peer's AoP have changed, the server sends the peer an updated list of the members of its AoP. The server also sends all movement updates of the AoP members to the peer. This algorithm is given in Algorithm 1. The algorithm has one input parameter, the peer p for which to determine the AoP. It uses the set of all peers ($peers$) to return the updated AoP for p ($p.aop$).

First, the algorithm calculates p's available upstream bandwidth by taking its maximum upstream bandwidth of p and subtracting the maximum bandwidth that is required by p's events. This subtraction is done because the peer needs bandwidth reserves to send updates to the server, as the server needs to know the location of all peers in order to execute

this algorithm. This means that at a minimum each peer needs to send position updates to the server regularly. In addition p needs bandwidth to send event updates to the server, in case they occur outside its AoP. In some instances the AoP can be so small that all events occur outside the AoP. Therefore we reserve enough bandwidth to send all updates to the server. The algorithm then takes p's available upstream bandwidth and divides it by the maximum bandwidth that is required by p's events. This gives us the maximum number of direct communications that the peer can support ($maxComm$).

```
Input: a peer (p)
Data: an array of all peers (peers)
Result: the updated AoP of p (aop)
begin
    p.aop = {};
    usedBw = maxEventBw + maxPosUpdateBw;
    availBw = p.upstream - usedBw;
    maxComm = round(availBw / p.maxEventBandwidth);
    peers.sortByDistanceTo(p);
    for int i = 0; i < maxComm; i++ do
        p.aop.add(peers[i]);
    end
    return p.aop;
end
```

Algorithm 1: AoP Generation Algorithm

The algorithm then sorts all peers by their distance to the input peer and calculates the AoP. To do so, it iterates through the sorted list of peers, beginning with the closest neighbor peer. As long as $maxComm$ is not reached, it adds neighbors to the AoP. Based on the output of the algorithm, the server constructs a list of the members of the AoP and sends it to the peer. When the peer has received this list, it determines the radius of its AoP by measuring the distance to the most distant peer in the AoP. It sets the radius of the AoP to this distance. Note that a special case occurs if some peers have exactly the same distance to p, e.g. peers H and J in Figure 1. This can lead to a non-circular AoP if the first peer not in the AoP has the same distance as the last peer in the AoP. To solve this case, the server reduces the size of the AoP before sending the information to the peer by iteratively removing the last peer from the AoP until the problem is solved.

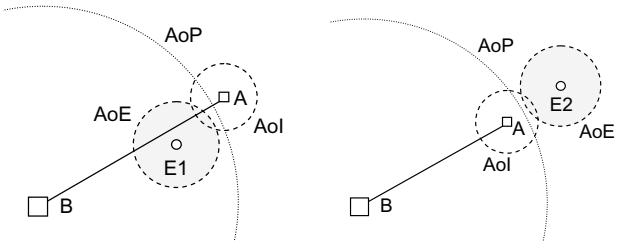
D. AoP-based Update Propagation

After detailing how the AoP of each peer is determined, we specify our approach for update propagation in more detail. We assume that the server executes Algorithm 1 regularly and that each peer knows the size and members of its current AoP. When an event occurs on one of the peers, it initiates the Update Propagation Algorithm given in Algorithm 2. The algorithm first compares the event's AoI with the peer's AoP. There are three possible cases: (1) the AoI can be completely within the AoP, (2) completely outside of the AoP, or (3) overlap its border and thus be partially inside and partially outside the AoP. If the AoI is completely within the AoP (Case 1), then the peer can determine the update's recipients and deliver the update without the assistance of the server. This is the case, because the peer has all relevant information to do so and has enough bandwidth to send the update directly. If the event's AoI is completely outside of the AoP (Case 2), then the peer cannot deliver the event itself, since it does not know any of the potential recipients. Therefore, the peer sends the update to the server. The server then determines the update's recipients and delivers it to them. Finally, if the event's AoI overlaps the edge of the AoP (Case 3), some of the event's potential recipients are members of the AoP, while others are not. Thus, the update needs to be delivered both by the peer and the server. First, the peer determines all target peers inside its AoP and sends the update to them. In addition, the peer sends the update to the server. The server then determines all target peers outside the AoP and sends the update to them.

```
Input: an update (u) and its AoE (aoe)
if u.AoE is inside (p.AoP - max radius of AoI) then
    local peer executes Algorithm 1;
else if AoE is outside AoP then
    deliver update to coordinator;
    coordinator executes Algorithm 1;
else
    local peer executes Algorithm 1;
    deliver update do coordinator;
    coordinator executes Algorithm 1;
end
```

Algorithm 2: Update Propagation Algorithm

There are two special situations to consider, both shown in Figure 3. The first one (see Figure 3a) can occur in Case 1. If a peer A is located just outside of another peer B's AoP, A's AoI may intersect with B's AoP. Thus, A may be interested in events that occur near the border of B's AoP, even if the event's AoE is completely inside the AoP. For example, this is the case for event E1 in the figure. To solve this situation, we modify the check for Case 1 as follows. Instead of checking whether the event's AoE is completely within the AoP, we also check if the event's AoE is close enough to the edge of the AoP to potentially intersect with another peer's AoI (using a predefined maximum AoI size). If this is the case, then the peer can not deliver this update completely by itself. Instead, the event is treated as if it belongs to Case 3.



(a) Case I (b) Case II

Fig. 3. Special Situations for Propagation

The second special situation (see Figure 3b) can occur in Cases 2 and 3. If a peer A is located just inside another peer B's AoP, A's AoI may be partially outside the AoP. In this case it is interested in some events that are completely outside of the AoP. To solve this, while checking which peers are target peers for an event, the server not just considers all peers outside the AoP, but also those peers whose AoI is partially

outside of the AoP.

V. Evaluation

In this section we evaluate our approach and show that it achieves low average update message delay while maintaining scalability even under heavy load. We also take a look at the influence of event locality on the average delay. We first describe our evaluation setup, then present our results and discuss them briefly.

A. Evaluation Setup

For our evaluation we implemented our approach prototypically and integrated it into our existing peers@play prototype, which provides us with a suitable communication layer as discussed in Section III. Our evaluation requires both a large number of peers in the system as well as the ability to get realistic delay measurements. Our experimental setup was designed to satisfy both criteria. To achieve a large number of peers we ran up to 400 instances of our prototype on an IBM Blade Center with 6 Blades (each with 2 Intel Xeon QuadCore CPUs with 2.33 GHz and 6 GB RAM). Clearly, the delay between these instances is unrealistically low, as they all run on the same local network or even system. Thus, to perform realistic delay measurements, we set up two additional peers at two separate off-site locations. The first peer was located at another university connected to the Internet with a high-speed broadband connection. This peer served as our server. The second peer was connected to the Internet through a standard home cable Internet connection. This peer served as our measuring node. We measured the delay of all incoming updates at this measuring node. This gives us update delays as experienced by an individual participant in the virtual environment. Any update sent directly to or from the measuring node has to traverse an Internet connection, thus taking a realistic amount of time to reach its destination. The same is true for any message sent to or from the server. Note that since all updates always traverse the same two Internet connections, our setup does reduce jitter compared to a real-world scenario, where the updates would traverse several different connections. As this does not affect the evaluation results regarding scalability and average delay, we found this to be acceptable.

For selecting our evaluation parameters, we assume that only a small fraction of a peer's upstream bandwidth is available for update propagation, as users often run other Internet applications in parallel with the MMVE and the MMVE itself may need significant bandwidth for other features, e.g. content streaming. Thus we restricted the available upstream for update propagation to 32Kbps per peer. We set the maximum event bandwidth ($maxEventBw$) used by each peer to 2 Kbps. This is identical to the event bandwidth used by the popular MMVE World of Warcraft [16]. For simplicity we set the maximum bandwidth used by a peer to send position updates to the server ($maxPosBw$) to 2 Kbps as well. Our virtual environment is 5000x5000m in size. Peers are placed statically and randomly in this area and have a circular AoI with a radius of 200m. Each peer regularly generates events with an AoE radius of 100m. This ensures that updates are propagated even for scenarios with few peers. Clearly, the performance of our approach depends highly on the percentage of events that occur inside the peer's AoP. Therefore we introduced a parameter into our test system which allows controlling this percentage.

We compared our approach to both a fully connected P2P-based system and a purely server-based architecture. To achieve the P2P-based system, we set the AoP of each peer to include all other peers, leading to a fully connected overlay network. Similarly, to turn our approach into a server-based system, we set the AoP of each peer to include no neighbors at all, thus sending all updates via the server.

B. Evaluation Results

We performed two series of measurements. In the first series we measured the average update propagation delay depending on the number of peers in the system. To do so, we increased the number of peers from 10 to 400. For these measurements we generated 20 percent of all events outside the peers' AoPs, thus accounting for the fact that most events are generated close to the originating peer.

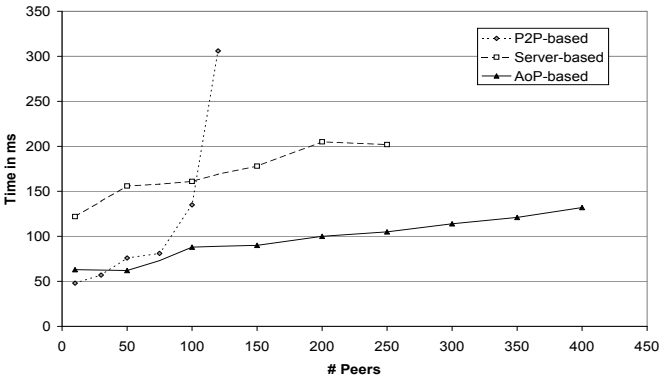


Fig. 4. Comparison of Average Delay

The resulting latencies are shown in Figure 4. The continuous line shows our AoP-based approach, the dotted line shows the P2P-based and the dashed line the server-based approach. As can be expected, the average update propagation delay increases with the number of peers in the system for all three approaches due to higher load on the blades. For less than 50 peers, the P2P-based approach delivers slightly lower delays than our approach. However, its delay quickly increases as more peers are added to the system and the peers start getting overloaded. With 120 peers, the average delay is already above 300ms. We omitted results for more peers, as these are above multiple seconds and thus off the chart.

Compared to the server-based approach, our approach delivers significantly lower delays in all cases. As only 20% of the updates are generated outside of the AoP and thus sent via the server, average delay is significantly reduced. Due to our test environment, the server was not able to handle more than 250 peers, resulting in delays of several seconds. We omitted

these results as we think they are not representative of server performance in a real system. Using our AoP-based approach the server still performed well with 400 connected nodes, as the peers relieve the server of a considerable amount of load.

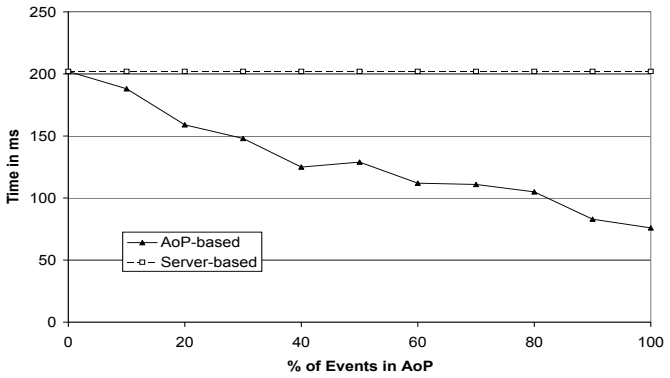


Fig. 5. Dynamic Adaptation with AoP

In our second measurement series, we evaluated the effect of event locations on the system's performance. To do so, we increased the percentage of events generated inside a peer's AoP from 0% to 100%. The number of peers in the system was set to 250, the largest amount of peers that our server hardware supports. This accounts for the worst case, in which all messages are sent via the server. As Figure 5 shows, our approach adapts itself dynamically to the locality of the events. If 0% of the events are generated outside of the AoP, all updates are sent via the server. Consequently, our approach shows the same delay as the server-based approach. With an increasing percentage of updates being sent directly, the average delay decreases continuously until 100% of the events are generated inside the AoP.

Our approach introduces an additional overhead for the server. The main computational overhead is the computation of the peers' AoPs. Since the server has to compare all peer positions in a zone this algorithm has a complexity of $O(n^2)$ for n peers per zone. In our evaluation this resulted in an increased CPU load of approximately 5% on the server. In practice we do not expect this to be a major issue as the calculation can be distributed across the server cluster and the zone size can be dynamically adjusted. Nevertheless, we are currently looking at ways to optimize this part of our approach

C. Summary

In conclusion, our approach performs as expected. It provides delays that are close to those of a fully connected P2P-based approach for scenarios with few peers. These delays are significantly lower than those of a server-based system. At the same time our approach is able to deliver updates with the same delay as a server-based approach for scenarios with many peers, which a P2P-based system cannot handle. In addition to the number of peers, the performance of our approach depends on the location of updates relative to their originating peer. If all events are created outside a peer's AoP our approach effectively falls back to a pure server-based approach, leading to the same delays. In practice most MMVEs generate the majority of their events near the originating peer. This leads to significantly lower average delays.

VI. Conclusion and Future Work

In this paper we presented a novel update propagation approach for low latency MMVEs. Our approach dynamically adapts itself to the available upload bandwidth of the participating peers and the current situation in the virtual world. By shifting the responsibility for update propagation dynamically between peers and the server, we are able to combine the low latency of P2P-based propagation systems with the scalability of server-based approaches. To do so, we introduce a special area in the virtual world, the AoP. Inside its AoP, a peer is able to handle update propagation on its own. Outside the AoP, the server is responsible for update propagation. To adapt itself to changing scenarios, the system can dynamically change the size of a peer's AoP. Currently we are extending our existing prototype to include a more sophisticated update filtering algorithm based on non-circular AoIs and AoEs. In the future we want to integrate our approach with zone-based MMVEs without a central server. In such systems, the server functionality is divided between dynamically elected superpeers who each manage a single zone.

Acknowledgment

The authors would like to thank Torben Weis, Arno Wacker and Sebastian Holzapfel for their support during the development of the prototype and the evaluation.

References

[1] Blizzard Entertainment, "http://www.worldofwarcraft.com."
[2] Linden Lab, "http://www.secondlife.com/."
[3] Peers@Play Project, "http://www.peers-at-play.org."
[4] K.-c. Kim, I. Yeom, and J. Lee, "Hyms : A hybrid mmog server architecture," *IEICE Transactions on Information and Systems*, 2004.
[5] S. Rooney, D. Bauer, and R. Deydier, "A federated peer-to-peer network game architecture," *IEEE Communications*, 2004.
[6] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," *INFOCOM '04*.
[7] 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*.
[8] 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 '05*.
[9] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "Von: A scalable peer-to-peer network for virtual environments," *IEEE Network*, 2006.
[10] D. Frey, J. Royan, R. Piegay, A. Kermarrec, E. Anceaume, and F. L. Fessant, "Solipsis: A decentralized architecture for virtual environments," in *MMVE '08*.
[11] J.-F. Chen, W.-C. Lin, T.-H. Chen, and S.-Y. Hu, "A forwarding model for voronoi-based overlay network," *ICPADS '07*.
[12] A. Wacker, G. Schiele, S. Holzapfel, and T. Weis, "A NAT traversal mechanism for peer-to-peer networks," in *MMVE '08*.
[13] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*. ACM Press, 1999.
[14] K. L. Morse, L. Bic, and M. Dillencourt, "Interest management in large-scale virtual environments," *Presence: Teleoperators & Virtual Environments*, 2000.
[15] F. Heger, G. Schiele, R. Süselbeck, and C. Becker, "Towards an interest management scheme for peer-based virtual environments," in *CoMMVE '09*.
[16] P. Svoboda, W. Karner, and M. Rupp, "Traffic analysis and modeling for world of warcraft," in *IEEE ICC '07*.