Proceedings First International Workshop on Massively Multiuser Virtual Environments

March 8, 2008

Contents

Foreword **2**

Program committee **3**

Scalable Reputation Management for P2P MMOGs,
Guan-Yu Huang, Shun-Yun Hu, Jehn-Ruey Jiang **3**

Clustering Players for Load Balancing in Virtual Worlds,
Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Timo Teifel, Georg Carle **9**

Consistency Management for Peer-to-Peer-based Massively Multiuser Virtual Environments,
Gregor Schiele, Richard Sueselbeck, Arno Wacker, Tonio Triebel, Christian Becker **14**

Data Aggregation Method for View Range Computation on P2P-based VCS,
Ryo Nishide, Dai Ito, Masaaki Ohnishi, Shinichi Ueshima **19**

An Implementation of a First-Person Game on a Hybrid Network,
Anthony Steed, Bingshu Zhu **24**

Solipsis: A Decentralized Architecture for Virtual Environments,
Davide Frey, Jérôme Royan, Romain Piegay, Anne-Marie Kermarrec, Emmanuelle Anceaume, Fabrice Le Fessant **29**

The HyperVerse - Concepts for a Federated and Torrent Based "3D Web",
Jean Botev, Markus Esch, Alexander Höhfeld, Hermann Schloss, Ingo Scholtes **34**

Towards an Authentication Service for Peer-to-Peer based Massively Multiuser Virtual Environments,
Arno Wacker, Gregor Schiele, Sebastian Schuster, Torben Weis **40**

Index of authors **46**

Foreword

With the increasing number of potential users of virtual and augmented reality systems, as indicated by the recent booming of massively multiuser online societies, the design of distributed, massively multiuser virtual environments (MMVEs) becomes more and more important, posing new requirements on both distribution platforms and virtual reality systems. Facing these challenges is a community-spanning effort, necessitating the pooling of the resources and experiences of the virtual reality and the networking and distributed computing communities. The aim of this workshop is to provide a link between these communities to foster the development of highly distributed, flexible and robust virtual environments. We aim to gather the practitioners and researchers in these fields under one roof to discuss their findings, incite collaboration and move the state of the art forward.

Program committee

Dewan Tanvir Ahmed, University of Ottawa
Gregor Schiele, University of Mannheim
Arno Wacker, University of Duisburg-Essen
Christian Bouville, IRISA
Bing-Yu Chen, National Taiwan University
Kuan-Ta Chen, Academia Sinica
Yiorgos Chrysanthou, University of Cyprus
Mike Eissele, University of Stuttgart
Abdennour El Rhalibi, Liverpool John Moores University
Joerg Haehner, University of Hannover
Aaron Harwood, University of Melbourne
Jehn-Ruey Jiang, National Central University
Pedro Morillo Tena, University of Valencia
Jauvane C. Oliveira, LNCC
Juan Manuel Orduna, University of Valencia
Shervin Shirmohammadi, University of Ottawa
Gwendal Simon, ENST Bretagne
Sandeep Singhal, Microsoft Corporation
Shinichi Ueshima, Kansai University
Krzysztof Walczak, Poznan University of Economics
Suiping Zhou, Nanyang Technological University
Daniel Weiskopf, University of Stuttgart
Ben Leong, National University of Singapore
Shun-Yun Hu, National Central University

Scalable Reputation Management for P2P MMOGs

Guan-Yu Huang[1], Shun-Yun Hu[2], Jehn-Ruey Jiang[3]
Department of Computer Science and Information Engineering
National Central University, Taiwan, R.O.C.
[1]aby@acnlab.csie.ncu.edu.tw [2]syhu@yahoo.com [3]jrjiang@csie.ncu.edu.tw

***Abstract*—Networked virtual environments (VEs) such as Massively Multiplayer Online Games (MMOGs) have become very popular in recent years. However, existing client-server architectures suffer from resource constrains when the number of concurrent users increases. Research on peer-to-peer virtual environments (P2P-VEs) thus tries to create more scalable and affordable VEs via the resource sharing of mutually cooperating clients. However, P2P approaches face the problem of client misbehavior where clients may not properly process the game rules. Without the monitoring and control from servers, the misbehavior could negatively affect a game's normal operations. In this paper, we present REPS, a distributed reputation management system for P2P-based MMOGs that allows trustworthy clients be identified. Based on the mutual rating and reputation query among users, REPS provides a scalable and reliable reputation mechanism that helps users to estimate the trustworthiness of others, so that subsequent grouping, trading, or superpeer selection decisions are made more reliably.**

I. Introduction

Massively Multiplayer Online Games (MMOGs) such as *World of Warcraft* [1] and *Second Life* [2], where over hundreds of thousands of players assume virtual identities and engage in various interactions, have become very popular in recent years. These virtual worlds are very attractive as they provide immersive 3D environments that people can constantly explore together. As of 2008, there are more than 12 millions registered Second Life accounts and over 10 millions paying subscribers in World of Warcraft. As user population grows, the traditional client-server architecture will suffer from the server's limited bandwidth and processing power. To solve this problem, peer-to-peer virtual environments (P2P-VEs, e.g., SimMud [3], Colyseus [4], and VON [5]) have thus been proposed.

In client-server architectures, the server receives and processes all the user-generated events. This ensures that the action of each participant is monitored, and game rules are executed objectively as the designers have intended. Cheating is also restricted as all important processing is done by the servers. However, P2P-VEs do not have such fairness guarantee because most server functions are now assumed by some clients. A client may modify any information that it possesses and may even assume new identities when it has cheated for private benefits. Although, most players may not go to great length to cheat, as modifying the game code requires certain technical skills. But even if only a small number of users are successful at cheating, gameplay can still be disrupted seriously.

Fortunately, we observe that the nature of MMOGs is highly social, and users often invest large amount of time and energy to build their in-game persona to ensure their standings in the virtual world. Users often are also bounded by guilds or other social organizations, as opinions from other users affect one's reputation and social experiences even more than other in-game activities. In other words, there exists strong social forces in successful MMOGs where active users typically value highly their status and reputations among peers. Such reputation thus may be exploited to facilitate certain game operations, such as the selection of trustworthy clients for important functions. We have seen similar mechanism in online marketplaces such as eBay and Yahoo Auctions, where online reputations based on mutual user ratings are used to estimate the trustworthiness of a user. If such reputation mechanism can be adopted in P2P MMOGs, it might help users to make decisions on whether to interact with a particular peer, or to select trustworthy clients for assigning more responsibilities.

In this paper we propose REPS, a reputation management system for P2P MMOGs based on peer-rated reputations. Each user has a reputation value based on other users' subjective opinions during their interactions. Reputation values are stored at some trustworthy neighbors called *trust nodes*, so that they may be accessed distributively without requiring a server. To select the trust nodes, we use *Neighbor Trust node Selection (NTS)* to choose trustworthy peers. NTS uses statistical regression method to choose trustworthy users, so that only users matching the strictest reputation criteria are chosen.

The rest of this paper is organized as follows. Section II provides background on reputation management and P2P-VEs. Section III presents our problem formulation and challenges in distributed reputation management. We describe the design of REPS in Section IV, and discuss its characteristics in Section V. Conclusions are given in Section VI.

II. Background

A. Reputation management

Recently, many P2P-based reputation systems have been proposed, often in the context of e-commerce (e.g., [6], [7], [8], [9], PeerTrust [10], Beta [11]). The goal of these systems is to compute the reliability of a peer and predict its future behavior of a specific identifier based on past interactions with the peer. The users are often buyers and sellers in an existing distributed or semi-distributed e-commerce environment. The reputation value represents a summary view for the user's

behavior, and can be used as the reference to warn or convince other users. By quickly identifying whether an user is trustworthy, interactions with *malicious users* who would cheat for private benefits can be avoided.

A peer's reputation value in these systems is calculated by collecting the local evaluations from other users. For example, in [12] and [8], the sum of the rating scores from every transaction is used to compute each user's reputation value. To make reputation values globally accessible and reliable, PeerTrust [10] normalizes the values by specific weights computed from each user's global reputation.

Some recent approaches like [13], [14], [15] and [16] use the Bayesian method that takes a binary input (i.e., positive or negative) to predict the cheating probability of the next transaction with a user from past experience. [17] provides the QoS experience vectors to perform reputation evaluation on many levels to determine more precise reputation values.

Besides evaluating other users, users will also need to know someone's reputation value for various tasks. To query reputations in P2P environments, a decentralized method is often used to aggregate reputation scores from various places to compute the global reputation value. In a client-server architecture, the server stores all the reputation data, and users just need to query the server. However, in decentralized environments, often a P2P storage such as Chord [18], CAN [19], or P-Grid [20] is used to distributively store the reputation data. For example, [17] uses Chord to find the successors of each user A, where A stores all reputation records evaluated by other users on its successors. When another user B needs to know A's reputation, it will hash A's identifier to aggregate the reputation evaluations from A's successors. Similarly, [12] uses the identifier hash to discover successors for storing the reputation values with CAN or Chord.

Some other problems also exist in P2P reputation management, for example: how to distinguish honest from dishonest people, or to detect dishonest ones pretending to be honest [21]? How to filter extreme (i.e. too positive or negative) or fake reputation evaluations in order to ensure correctness? And how to prove that a reputation management is reliable for a given application? There are many works that discuss these general problems for reputation management (e.g., [6], [22] and [8]). We will discuss how REPS deal with these problems in P2P MMOG scenarios.

B. P2P-VEs

In P2P-VEs, every user has a visibility range called *area of interest* (or AOI, see Fig. 1). The AOI is often circular, and other users within the AOI are called the *AOI neighbors*. Users can exchange messages to comprehend the environment around them, and see the dynamic updates from other AOI neighbors. The key to scalable P2P-VEs is based on the fact that users have limited view within their AOI and only need to observe changes within the AOI. The scalability of the whole environment thus can be extended if each user only exchanges messages with its AOI neighbors, without going through the server.

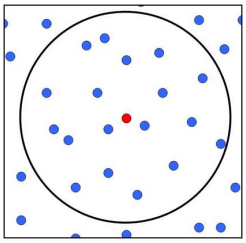


Fig. 1. Large circle is the AOI of the center user.

In some approaches (e.g., [3], [23], [24], [25]), the whole world is divided into serval disjoint sub-regions in order to manage information updates distributively. Some participants with better capacities are chosen as *superpeers* to relay information (e.g., position updates and the event notifications) for other users. Lo et al. [26] define a superpeer as a special role that can provide services to non-superpeers and describe several superpeer selection methods.

For the many P2P-VE schemes that adopt superpeers, whether the selected clients are trustworthy is essential for the system's. One of the implications for REPS thus is a reliable method to select trustworthy nodes that could assume important superpeer functions.

III. Problem Formulation and Challenges

Our goal is to build a scalable reputation management system that supports P2P MMOGs by developing a distributed method to rate, store, and query reputation values. The main problem is how to store the reputation scores on reliable peers and query them effectively. We first assume the following for our scenario:

1. Every user has a fixed AOI radius because most current MMOGs' use a fixed visual range, where users see each others only when they are within each other's AOI. Between two mutually visible users, certain game-specific interactions can occur (e.g., talking, fighting, trading, etc.). The users within AOI, or *AOI neighbors*, change periodically as users move around in the virtual world.

2. We assume that there exists some P2P-VE overlays that provide a list of AOI neighbors for each user (e.g., SimMud [3], Colyseus [4], VON [5]). So any user may connect and exchange messages directly with its AOI neighbors.

3. Two mutually visible users can rate each other with a score of positive, neutral, or negative (+1, 0, -1) based on their past interactions. A reputation record follows the form of (*rater, rated-user, evaluation*), where *rater* is the user who makes the rating, *rated-user* is the user evaluated by the rater and *evaluation* records the actual rating.

4. A user can only give a single rating to another user. However, the rating can be changed later at any time if the original rater wishes to.

5. We assume that if a user's reputation exceeds certain threshold, the probability of cheating is low, as more effort than others has been spent to build the reputation (Fig. 2).

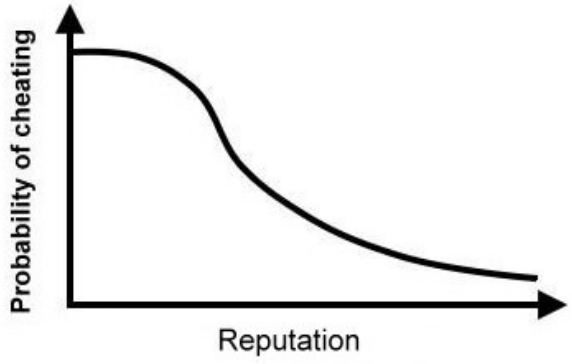


Fig. 2. Relation between probability of cheating and reputation value.

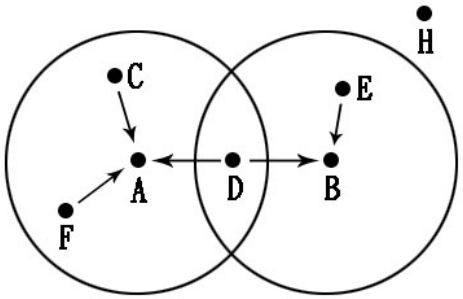


Fig. 3. The rating condition in REPS

To build a scalable reputation system for P2P MMOGs for the above scenario, we outline some challenges below:

Reputation evaluation User experiences are the basis for the reputation values in a reputation system. How to efficiently and precisely represent user perceptions thus is an important problem. Reputations are also meaningless if most users do not provide ratings. In MMOGs, players often focus more on the game itself than miscellaneous activity such as reputation evaluation, mechanisms to encourage user rating thus is needed. To provide suitable evaluations, we need a simple and efficient method that allows user evaluations be done conveniently, and reputation values be aggregated efficiently.

Storage and query How to store and query reputations in a fully distributed environment is the main challenge for a P2P reputation system. To ensure that the system would scale, we need to store the data with a distributed method while avoiding any server or client overloads. To efficiently query reputation data from other users, two problems need to be addressed: how to find the users that store the reputation data, and how to collect the data with minimal delays.

Reliability There are more transactions like trading, talking and grouping [27] in MMOGs than in online auction sites, where over 90% of users have only transacted once [28]. Therefore, ensuring that a reputation system provides reliable and trustworthy information is very important. In P2P environments, users may modify the reputation data they keep for private benefits. This would cause misunderstandings among users and unbalanced gameplay. Prevention, detection, and recovery of cheating behavior thus are needed.

IV. Design of REPS

A. Local reputation evaluation

In REPS, users rate one another when they are within each others' AOI, because interactions can only occur with AOI neighbors. For example, in Fig 3, users C and F could rate A because they are within A's AOI. The rating should also occur with a probability related to the level of interactions. The more exciting the interactions and the more unfamiliar the users are to each other, the higher the probability for rating. The interacting users will generate a **rating right** authorized by the rated user to the potential rater, so that it can give a rating at some convenient time. The rating right is used to prevent some raters to rate users that they have never interacted with. The rating right contains the rated user's name and IP address and is recorded at the rater so that the rater can choose a time that is convenient to give ratings. For example, if user C rates user A with a score of 1, then a rating record of (C, A, 1) will be stored at A's trust nodes, who would update A's reputation based on A's last reputation value. Each user can have only one rating about another user, but can also change the rating when impression for the other user has changed. This way users may mutually monitor each other's behaviors.

B. Reputation storage and query

In order to scalably store and query the reputation records, a user identifies and chooses N trustworthy users as its **trust nodes** from its current and recent AOI neighbors (called the *potential neighbors*). Trust nodes are chosen from potential neighbors according to **Neighbor Trust node Selection (NTS)** that will be described next. Once reliable peers are found through NTS, they are recorded in a **trust list** containing the chosen trust nodes' identifiers and IP addresses. The trust list is stored at the user to allow easy inquiry by others. To give a rating to user A, raters first query A's trust list from A, and then send their ratings (i.e., reputation records) directly to all trust nodes to update A's reputation value.

When a user B is within user A's AOI, B can request for A's trust nodes in order to query for A's reputation value. User A would send its current trust list to B, where B randomly chooses n (where $1 \leq n \leq N$) trust nodes from the list to query. The chosen trust nodes will then respond the reputation value of A to B. The reason for asking reputation values from n trust nodes is to prevent any trust nodes from manipulating the stored reputation values. A reputation value is recognized only if it is returned by a majority of the trust nodes.

A user's potential neighbors would expire after a certain timeout, but may be renewed if the neighbors revisit a user's AOI. This has the effect that a trust node will also expire if it has not been a user's recent AOI neighbor. By limiting the time a node may be a trust node, users with high reputation

TABLE I
EXAMPLE OF PROPORTIONALITY MISREPRESENTATION

User	total score, TS(u)	number of ratings, V(u)	P(u)
A	30	100	0.3
B	9	10	0.9

can be saved from being always selected as trust nodes and bombarded with requests.

C. Neighbor Trust node Selection (NTS)

In REPS, each user requires N trust nodes that are chosen via *Neighbor Trust node Selection* (NTS). Many existing reputation systems use peer rating to devise the reputation (e.g. *PeerTrust* [10]), where an user i can give another user u a score $S(u,i)$ of either positive or negative, and the reputation is simply the summation of all scores, or a *total score* (TS). Other methods also exist in auction sites such as eBay or Yahoo Auctions, where a ratio $P(u)$ indicates the proportion between the total score $TS(u)$ and the total number of ratings $V(u)$. The higher the $P(u)$, the more trustworthy a given user u is.

$$TS(u) = \sum S(u,i)$$

$$P(u) = \frac{TS(u)}{V(u)}$$

But which is better? If A scores 30 out of 100 ratings, and B scores 9 out of 10 ratings. According to $TS(u)$, A is more trustworthy as its total rating is higher than B's. But the ratio $P(u)$ of B is higher than A's, which indicates that B may be more trustworthy. Yet since 100 people are willing to rate A and only 10 persons have rated B, the significance of A's rating may be higher. Some proportionality misrepresentations thus exist in existing approaches (Table I).

Ideally, we would like to combine the effects of both the total score and the ratio of positive rating, as they are both meaningful for a person's reputation. However, we do not know which is more important as it may differ across regions or MMOGs, where the willingness to rate can vary. We thus design *Neighbor Trust node Selection(NTS)* that combines both $TS(u)$ and $P(u)$ in a flexible way. A simple way to conceptualize NTS is in Fig. 4, where the x-axis represents all possible ratio values and the y-axis represents all possible total scores. There is also an area called *trust region* where a user u can be selected to become a trust node if its reputation point lies within the trust region (i.e. $P(u) > P_{bound}$ and $TS(u) > TS_{bound}$, where P_{bound} is between 0 and 1 and TS_{bound} is between the most negative and the most positive ratings). If we want to select N trust nodes, we can select N points (i.e. clients) from the trust region. If more trust nodes are needed, the area of the trust region is adjusted by changing P_{bound} and TS_{bound}. To adjust P_{bound} or TS_{bound}, we define the value m as the absolute value of the regression coefficient that represents the slope of the regression line for all points in the trust region, where $\bar{P}$ is the average $P(u)$ and $\bar{TS}$ is the average $TS(u)$ of all users within the trust region:

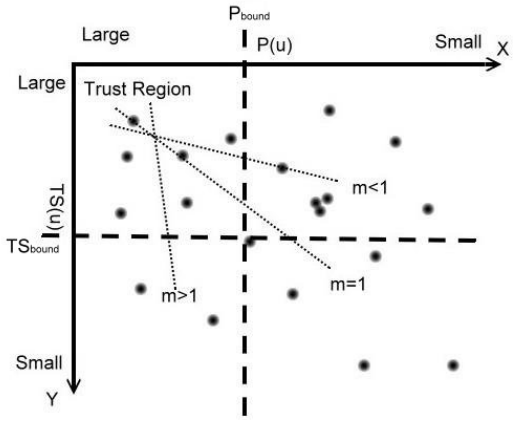


Fig. 4. Trust plane in REPS

$$m = \left| \frac{\sum (P(u) - \bar{P})(TS(u) - \bar{TS})}{\sum (P(u) - \bar{P})^2} \right|$$

The regression coefficient shows the pattern of distribution for all reputation points, and taking absolute values means that NTS only cares about the direction of the distribution but not the shape of the regression line. If $m > 1$, the trend for points in the trust region is towards $TS(u)$, its weight thus should be increased. If $m < 1$, it means that the point positions are tilting towards $P(u)$ in the trust plane, and NTS should increase the weight for $P(u)$. The actual adjustments ΔP_{bound} and ΔTS_{bound} for P_{bound} and TS_{bound} are adjustment ratios (i.e., they are percentages of the change in the values of P_{bound} and TS_{bound}), and depend on the value of m, where ΔP_{bound} / ΔTS_{bound} = m. NTS increases ΔP_{bound} and ΔTS_{bound} simultaneously with a fixed ratio m until the number of the candidate points matches the system demand. Likewise, ΔP_{bound} and ΔTS_{bound} could also decrease with the ratio m when the required trust nodes are less.

When the number of AOI neighbors is not large enough, trust nodes are chosen randomly until the number of users exceeds a threshold δ, then NTS is used again. When NTS is first used, we initialize m, P_{bound} and TS_{bound} to be 1.0, 1.0 and n where n is the number of current online users and the area of the trust region would be 0. We then reduce P_{bound} and TS_{bound} to extend the trust region by ΔP_{bound} / ΔTS_{bound} = 1 in order to find new trust nodes or remove old ones, as the set of potential neighbors change with time.

V. DISCUSSIONS

A. Reputation evaluation

REPS uses direct rating as the evaluation method, where users give a simple score (-1, 0, 1) to represent their impressions for each reputation evaluation. It is thus very simple to integrate one's reputation value. A user's trust nodes can update reputation values directly and individually whenever they get a new reputation record. The rating right control lets

users to recognize which users can rate them and ensures that only users who have interacted can rate each other. REPS thus provides a simple and effective method to represent and compute the reputation value.

B. Storage and query

Users in REPS store their reputation data on their trust nodes, this prevents a user from modifying its own reputations, as other users store and query reputation values directly with the trust nodes. A rated user provides only the trust list to a rater, and is not responsible to maintain his or her reputation value.

Querying reputation data can also be done efficiently as a querying user only needs to obtain the trust list, then it can ask some chosen trust nodes directly. As the number of users increases in a system, the number of queries may also increase for a given user. The overhead for each trust nodes can be reduced by increasing the number of trust nodes N for each user, so that more trust nodes may share the load of querying.

C. Reliability

The effect of malicious users on the system is reduced in REPS due to the *mutual monitoring* among users. As everyone can rate another user and update their scores when new situations occur, a cheating user will soon be rated very negatively if some misbehavior is discovered. The cheater's reputation will reduce rapidly and its privileges or responsibilities could be removed.

For the storage of reputation values, as they are stored on multiple trust nodes, improper modifications by any single trust node is masked from the correctly maintained records in other trust nodes. Trust node misbehavior thus will impact the system minimally and can be quickly detected. As reputations are stored and accessed at trust nodes instead of the rated user, users also cannot manipulate their own reputation values for unfair benefits. On the other hand, it is in a user's best interest to provide a list of trustworthy trust nodes to any potential rater, as it will surely wish that its reputation value is recorded and accessed objectively.

VI. Conclusion

REPS provides reputation management to support P2P MMOGs by allowing users to rate each other after some interactions, and select trustworthy nodes based on these ratings. Through the use of trust nodes, reputation records can be stored and accessed distributively without relying on a centralized server. Reputation values can thus be built and used in a scalable way. We also present Neighbor Trust node Selection (NTS) that chooses the trust nodes by combining two intuitive factors (e.g., a user's total score and positive rating ratio) and adjusts each factor's weights to adapt for different scenarios. Dynamic adjustments of the trust region finds the minimum area that satisfies a given number of required trust nodes, effectively selecting trustworthy nodes using the strictest criteria. We plan to evaluate REPS's effectiveness via simulations as our future work.

References

[1] "World of warcraft," http://www.worldofwarcraft.com/.

[2] "Second life," http://secondlife.com/.

[3] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *Proc. INFOCOM*, 2004, pp. 96–107.

[4] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A distributed architecture for online multiplayer games," in *Proc. NSDI*, 2006, pp. 155–168.

[5] S. Y. Hu, J. F. Chen, and T. H. Chen, "Von: A scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.

[6] Y. Atif, "Building trust in e-commerce," *IEEE Internet Computing*, vol. 6, pp. 18 – 24, 2002.

[7] A. Josang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, 2007.

[8] C. Dellarocas, "Analyzing the economic efficiency of ebay-like online reputation reporting mechanisms," in *Proc. 3rd ACM conference on Electronic Commerce*, 2001, pp. 171 – 179.

[9] K. Aberer and Z. Despotovic, "Managing trust in a peer-to-peer information system," in *In Proc. ACM CIKM*, 2001, pp. 310 – 317.

[10] L. Xiong and L. Li, "Peertrust: Supporting reputation based trust for peer-to-peer electronic communities," *IEEE TKDE*, vol. 16, pp. 843–857, 2004.

[11] R. Ismail and A. Josang, "The beta reputation system," in *Proc. 15th Bled Conference on Electronic Commerce*, 2002.

[12] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proc. WWW*, 2003.

[13] S. Buchegger and J.-Y. L. Boudec, "A robust reputation system for p2p and mobile ad-hoc networks," in *Proc. Second Workshop on Economics of P2P Systems*, June 2004.

[14] L. Mui, M. Mohtashemi, C. Ang, P. Szolovits, and A. Halberstadt, "Ratings in distributed systems: A bayesian approach," 2001.

[15] S. Ganeriwal and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," in *Proc. SASN '04*, Washington, D.C., USA, October 2004.

[16] R. Zhou and K. Hwang, "Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing," *IEEE Transaction on Parallel and Distributed Systems*, vol. 18, no. 4, pp. 460–473, 2007.

[17] Y. Zhang and Y. Fang, "A fine-grained reputation system for reliable service selection in peer-to-peer networks," *IEEE Transaction on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1134–1145, 2007.

[18] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. ACM SIGCOMM*, 2001, pp. 149–160.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM, August 2001*.

[20] K. Aberer, "P-Grid: A self-organizing access structure for P2P information systems," *LNCS (CoopIS 2001)*, vol. 2172, pp. 179–194, 2001.

[21] M. Srivatsa, L. Xiong, and L. Liu, "Trustguard: Countering vulnerabilities in reputation management for decentralized overlay networks," in *Proc. 14th Intl World Wide Web Conf*, 2005, pp. 422–431.

[22] Y. Yan, A. El-Atawy, and E. Al-Shaer, "Ranking-based optimal resource allocation in peer-to-peer networks," in *Proc. INFOCOM*, 2007, pp. 1100–1108.

[23] S. Yamamoto, Y. Murata, K. Yasumoto, and M. Ito, "A distributed event delivery method with load balancing for mmorpg," in *Proc. Netgames*, 2005.

[24] H. Lee and C. Sun, "Load-balancing for peer-to-peer networked virtual environment," in *Proc. Netgames*, Oct. 2006.

[25] S.-Y. Hu, S.-C. Chang, and J.-R. Jiang, "Voronoi state management for peer-to-peer massively multiplayer online games," in *Proc. NIME*, 2008.

[26] V. Lo, D. Zhou, Y. Liu, C. GauthierDickey, and J. Li, "Scalable supernode selection in peer-to-peer overlay networks," in *Proc. of HOT-P2P*, 2005, pp. 18 – 27.

[27] K.-T. Chen, P. Huang, and C.-L. Lei, "Game traffic analysis: An MMORPG perspective," *Computer Networks*, vol. 51, no. 3, 2007.

[28] P. Resnick and R. Zeckhauser, "Trust among strangers in internet transactions: Empirical analysis of ebays reputation system," *The Economics of the Internet and E-Commerce,*, vol. 11, pp. 127–157, 2002.

Clustering Players for Load Balancing in Virtual Worlds

Simon Rieche, Klaus Wehrle
Distributed Systems Group
RWTH Aachen University
{rieche,wehrle}@cs.rwth-aachen.de

Marc Fouquet, Heiko Niedermayer, Timo Teifel, Georg Carle
Computer Networks and Internet
University of Tübingen
{fouquet,niedermayer,carle}@informatik.uni-tuebingen.de

***Abstract*—Massively Multiplayer Online Games (MMOGs) have become increasingly popular in the last years. So far the distribution of load, caused by the players in these games, is not distributed dynamically. After the launch of a new game, the introduction of new content, during special ingame events, or also during normal operations, players tend to concentrate in certain regions of the game worlds and cause overload conditions. Therefore we propose the use of structured P2P technology for the server infrastructure of the MMOGs to improve the reliability and scalability. Previous work segmented the game work into rectangular areas; however this approach often split a group of players to different servers, causing additional overhead.**

This work presents a cluster-based Peer-to-Peer approach, which can be used for load balancing in MMOGs or in other virtual worlds. The system is able to dynamically adapt to the current state of the game and handle uneven distributions of the players in the game world. We show through simulation, also with traces from real online games, that the cluster-based approach performs better than the previous P2P-based systems, which split the world in rectangular areas.

I. Introduction

Computer games have changed in the last years due to the constantly rising speed and decreasing costs of internet connections. Ever more games offer the possibility, to play with other players together over the internet. Particularly in Massively Multiplayer Online Role-Playing Game (MMORPGs) a virtual world is created, in which thousands of players "live". For the players the shared virtual world offers the possibility that they can talk with other human players, build groups, friendships and fight together.

Traditionally, a cluster of servers contains one virtual world of a Massively Multiplayer Online Game (MMOG). But such an infrastructure is inflexible and error-prone. One would rather like to have a system that allows disconnecting a node at runtime while others take over its tasks. Also server-based MMOGs can have performance problems if players concentrate in parts of the game world or if some worlds are overpopulated. Thus, there is a need for load balancing mechanisms, which Peer-to-Peer (P2P) systems quite naturally support.

In previous work we showed how such a game can be hosted on a P2P-based infrastructure [1]. By using the structured P2P System Content Addressable Network (CAN) [2] as a basis, we split the game world into disjunctive zones and distribute them on different nodes of the P2P network. This happens automatically in such a way that all existing servers have nearly the same load. But the load balancing separates the world in a quad tree-like fashion and does not take the structure of the map into account, which may lead to many changes of players between the servers. In this case the old server has to transfer all the state information of the player to the new server, which is now responsible for this person. So a main task is to minimize the handovers of players from one server to another. Another problem are players who are located close to the border of the area that one server controls. These players can see parts of the game world on the other side of the border, therefore they need to be informed about all updates of dynamic content, i.e. player movements that happen there.

In MMORPGs it can be frequently observed that players form groups and walk over the map together. This particularly becomes problematic, if a whole group of players moves over a border and thus a large number of players have to be transferred to the new server. Thus we propose a P2P infrastructure for MMOGs which takes this behavior into account.

The distribution of load on the existing servers is thereby not done by dividing the map into different areas, but via dividing the players into clusters. Each server receives a group of players, who are close together. This group is one cluster. If the group moves on the map, the responsible area of the server moves with this group. If individual players depart from the group, they are handed over to another group as soon as they are closer to the new group than to the previous one. Figure 1 shows as an example several clusters in a virtual world.

The rest of this paper is organized as follows: First we discuss related work in Section II. Section III shows our

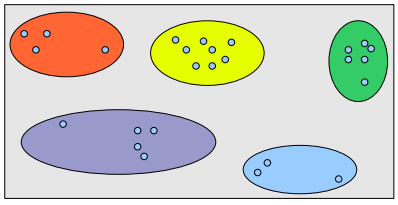


Fig. 1. Several clusters on the map of the virtual world.

approach to use structured P2P Systems for MMOGs, Section IV how the clusters are build, and Section V how to find clusters in the neighborhood of the map. In Section VI the load balancing of the cluster is described. Section VII describes the evaluation, including simulations that use player traces from a real MMOG, and finally, Section VIII provides conclusions.

II. Related Work

Some efforts have been undertaken to design a MMOG on a P2P basis, with the server tasks being shared among the player's PCs. In [3] and [4] the game world is divided into zones. Then some peers become zone owners that take responsibility for computing the server tasks for such a zone. While this approach is fascinating on one hand, it suffers from a number of practical problems. Players constantly connect to and disconnect from the game, often without warning if the PC crashes or suddenly gets disconnected from the network. This means that always backup machines are needed to replace disconnected zone owners. Allowing player's computers to calculate parts of the game mechanics makes it harder to avoid cheating. Another problem is that persistent player data, for example the progress of the player's characters in a role-playing game, need to be saved in a way that makes sure that no information is lost, as players may have invested a lot of work into the game. At the same time one has to prevent players from cheating by modifying data, which would be possible if it was stored on the player's local hard drives. This suggests that some kind of infrastructure provided by the game manufacturer would still be needed; a full P2P approach does not appear practical, so we focus on a hybrid P2P approach. Another challenge is the low upstream bandwidth of most current Internet connections, probably only peers with a good connectivity could be considered for becoming zone owners.

Solipsis [5] uses a different approach, as it tries to build a pure P2P network-based on the neighbourship relations of the player's avatars. Each peer has direct connections to all other peers that are in visible range of the player's avatar. There is a real implementation of Solipsis, however currently it is little more than a distributed chat client. If one wanted to make it a "real" MMOG, one would be confronted with the same problems as described above. It is especially difficult to make such an approach cheat-proof [6].

In our previous work [1] a structured P2P technology is used for the organization of the underlying infrastructure and thus for the reduction of downtimes in MMOGs. By using a CAN-based approach we split the game world in disjunctive zones and distribute them on different nodes in the P2P network. Thus, we get the possibility to dynamically connect and disconnect machines to and from the peer-cluster and to load-balance the game according to the actions of the players.

The P2P technology makes it possible to run multiple game worlds on a pool of servers. The physical location of these servers is of minor importance, they do not have to run in the same data center. Placing the servers of a single game world at different locations introduces additional overhead; however this may be justified by enhanced reliability or maybe by an improved locality. Even if one whole location should be disconnected from the network, the servers at other locations could take over seamlessly and without loss of data.

But it can be observed that players act in groups in most online games. Since those players can see all players in the same group on the virtual map, many messages have to be sent over the server to inform where the other players are. If there is a border between these players in the map the additional data has to be exchanged between the two servers, which are responsible for the different areas.

Many algorithms exist for load balancing in structured P2P systems [7]–[9]. However, most of these systems are not applicable for online games. Our approach is based on the virtual server (VS) approach [7] where multiple partitions of a Distributed Hash Table's (DHT) address space are managed in one node. Thus, one physical node may act as several independent logical nodes. So each VS will be considered by the underlying DHT as an independent node.

III. Cluster-based P2P Infrastructure Support for MMOGs

In our approach, the game world is defined by a map, which is managed by servers. Each server is responsible for the region, where the players of its group are located. Conceptually areas without players don't need a responsible server. Non-player characters and other objects with a state can be handled as players. So the game world is distributed on an infrastructure of different peers or servers. This infrastructure is not necessarily located in a single data center. However in most cases it makes sense to locate the peers that are responsible for adjacent regions in the same network to minimize delay and costs for internal traffic between the peers. But our approach could also be used for a super-peer network in a more distributed setting for more delay-tolerant games.

As in the virtual server (VS) approach multiple peers can be run on one physical server to better distribute the load. Since no central server should be used for the whole game, the movement of players from one cluster to another is done only by the two participating (virtual) servers. Each VS knows the coordinates of the players of its own group and the positions of some players of neighbouring groups which can be seen by the players of its group. Neighbouring virtual servers periodically exchange this information, to make sure that the positions of all players are rendered correctly on the player's computer.

IV. Building Clusters

For building the clusters, we measure the distance between the players. Simply measuring each player's distance to the centroid of a cluster (cf. Figure 2) leads to a suboptimal solution since groups may be split in the middle of two virtual servers. So this approach is not flexible enough to handle large or deformed clusters.

Figure 3 shows how the mapping of players to groups should be determined. Players belong to the same group, if they have a small distance to each other. The form of the group can be arbitrary, since the distance is not computed to a central

point, but to each player's neighbours. The required minimum distance to separate two players into different clusters is variable, so the size of the groups can be changed e.g. when adding new nodes. Therefore this parameter can also be used for the load distribution. Players, who have the same distance to two large groups, can be taken by any of these groups. A VS is not assigned to a fixed location on the map - as players move around, the VS assignment follows them.

V. Finding Neighbouring Clusters

Since there is no central server, each VS must be able to decide, which other virtual servers of the P2P structure are direct neighbours on the virtual map. Direct neighbours of a VS A are those servers, which have players in their groups, who can move to the players of group A without being transferred to another group of a VS B before arriving at A. In many cases these players are also close together, however there may also bigger areas on the map without any players. A cluster needs connections to all direct neighbours and all clusters are connected in one graph.

It is not sufficient to simply use the distance to a neighbour's centroid to test if two groups are direct neighbours since the groups do not necessarily have to be circular. Also groups, which are far from each other, may be direct neighbours, if no other group is between them.

So in order to test whether a VS A is a direct neighbour to B, the canonical approach would be to compare the positions of all players. For the players, with minimal distance, it must be tested if no players of another VS C lie between them. The test for the neighbourship relation can be optimized, as it is sufficient to test the players that form the convex hulls of the two groups instead of comparing all players.

VI. Load Balancing using Clusters

As mentioned before, our proposed system works with a VS approach. A cluster of players is equivalent to one virtual server. When the load of a physical server exceeds a threshold, for example when too many players are in its clusters, the load is reduced by three mechanisms:

- Moving whole clusters from one node to another.
- Moving one or some players from one cluster to another.
- Splitting a cluster into two parts, and moving one of them to another physical server.

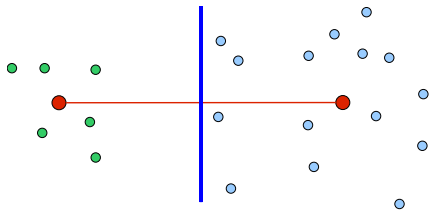


Fig. 2. Building clusters using the distance from the centroid.

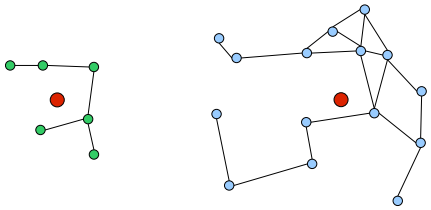


Fig. 3. Building clusters using the distance between players in a group. For illustration the centroids of the groups are also shown.

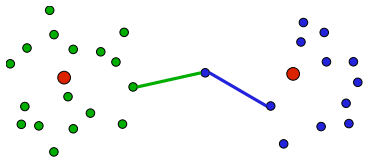


Fig. 4. Moving one player from one cluster to another.

A. Moving Clusters

The virtual server approach [7] is based on the idea of managing multiple partitions of a structured P2P address space in one node. Thus, one physical node may act as several independent logical nodes. Each VS will be considered as an independent node by the underlying structured P2P System. Within a CAN system, one VS is responsible for a zone of the address space, whereas the corresponding physical node may be responsible for several different and independent zones. The basic advantage of this approach is the simplicity of placing and transferring virtual servers among arbitrary nodes. This operation is similar to the standard join or leave procedure in a structured P2P system. Every participating node manages many virtual servers so load can be moved between nodes by moving a whole VS to another node.

B. Moving Players

Another simple operation to balance the load of the clusters is to move some players from one group to another. Figure 4 shows a player in the middle of two clusters. So depending on the load of each cluster the player can be moved to the group with the lower number of players. If some players move away from a cluster together, also all of them can be moved to the new cluster together (cf. Figure 5).

C. Splitting Clusters

Additionally, clusters with too many players can be split and one part can be sent to another server. This is done by

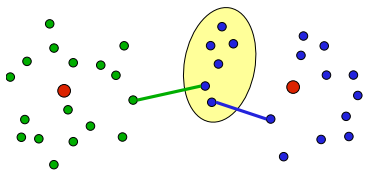


Fig. 5. Moving a part of a group to another cluster.

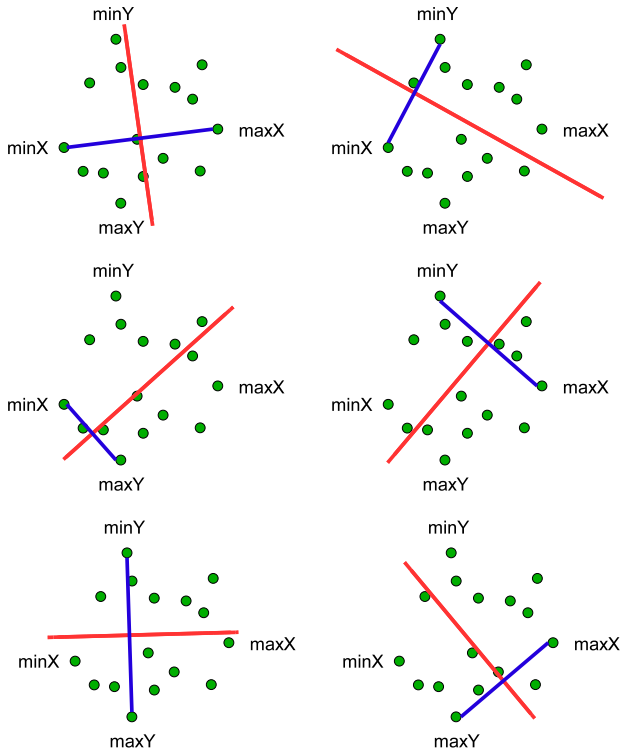


Fig. 6. The virtual server tests six possibilities for splitting a group based on four points. The two points, which are used as starting points to calculate the groups in each round, are connected with a line (blue). The perpendicular bisector of the side drawn in addition isolates the two groups. Each player belongs to the group of the starting point where he is nearer to. So the perpendicular bisector of the side splits the two groups, since after this line the player would be nearer to the other staring point.

first selecting the – usually four – players with minimal and maximal X- and Y-coordinates in the map.

For every one of the six combinations, two players p and q are selected and a split is calculated by simply assigning each other player r either to p's or q's group depending on r's distance to p and q.

At the end the combination of those two nodes p and q is chosen, which has the maximal distance between the two new groups. This is done to minimize the possibility that players will move from one group to the other in the future. Figure 6 shows as an example all six possibilities to split a group.

Also, adjacent clusters with low numbers of players can be merged for a lower number of internal messages.

VII. Evaluation

We focus in this evaluation part, due to space limitations, on the comparison of the cluster-based approach and the CAN-based approach with rectangular areas. In [1] we showed already that a P2P approach for a MMOG is practicable.

A. Traces

We evaluated our approach using a simulation with artificially generated as well as real traces of user-movement:

- **Random walk trace data** is a collection of generic data representing movement of users according to the Random Walk Mobility Model.
- **Random waypoint trace data** has been generated using the Random Waypoint Mobility Model.
- **Freewar trace data** consists of real player-movements traced in the online game Freewar [10] over a period of up to five hours. Approximately 400 players were online in this period of time, but the number of concurrent players varies over time since users join or leave the game. Also the Freewar traces show an extremely uneven distribution of the players over the world with some hotspots in cities.

B. Comparison

We compare the two approaches based on the following criteria using the Omnet++ network simulator:

- The number of ForeignPositionChange (FPC) messages, sent during the simulation. These are sent, whenever a player moves to the peripheral area of his group and can be seen by players of a neighbouring group. A FPC causes the transmission of one message per player and movement (time step) between the neighbouring virtual servers. So the number of FPC messages indicates how well the clusters were separated in the simulation. In a real game this number of FPC messages is one of the most important factors, since many FPC messages between the clusters will delay the response to the users.
- The number of players moved from one VS to another. The counter of handovers is increased with each handover of a single player. When parts of a group are transferred between two virtual servers at once, this counter is increased by the number of moved players. Besides the used bandwidth, the transfer of a player may cause a short delay for the customer. Therefore the number of handovers should be minimized.
- The number of players moved when complete virtual servers are moved between two nodes. This is separately counted in the counter handoversOnMove.

Figure 7 shows the number of handovers of players for a random walk trace. It shows that the cluster-based approach performs better than the CAN-based approach with rectangular areas. Additional, less FPC messages are sent in this simulation (cf. Figure 8). This random walk trace consists of 219,828 movement actions by 500 different players. All graphs are cumulative, i.e. they show the number of messages since the start of the simulation run.

We also simulated a scenario with random waypoint trace data. This trace again consists of 500 players, which move around in the game world for about five hours. Again, the cluster-based method performs better than the CAN-based method. The number of handovers using cluster-based method is 14,889 compared to 28,198 with the CAN-based method. The number of FPC messages is 79,492 for the cluster-based method, compared to 379,392 for the CAN-based method.

In a simulation with the Freewar traces there are less FPC messages sent by the cluster-based method than by the

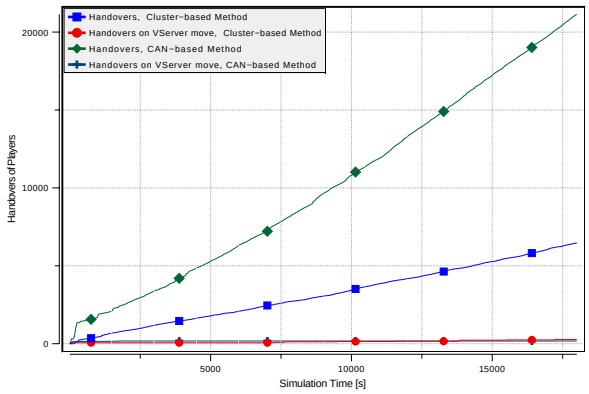


Fig. 7. Comparison of player handovers from one cluster to another with the CAN-based rectangular areas approach and the cluster-based approach with random walk trace data.

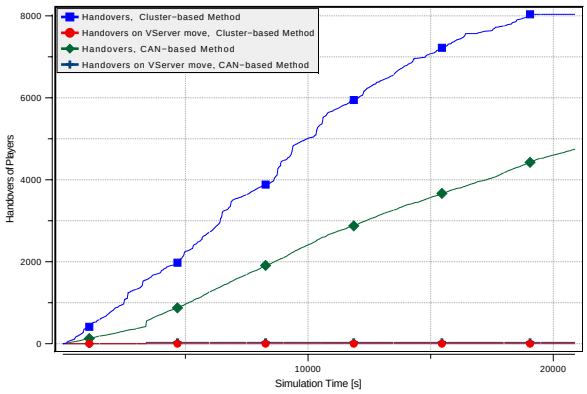


Fig. 9. Comparison of player handovers from one cluster to another with the CAN-based rectangular areas approach and the cluster-based approach with the Freewar trace data.

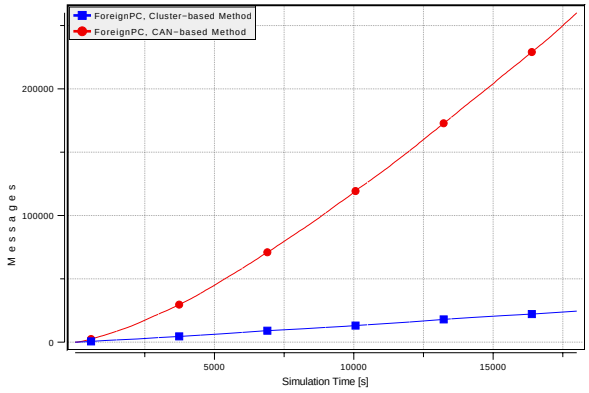


Fig. 8. The number of FPC messages with the CAN-based rectangular areas approach and the cluster-based approach with random walk trace data.

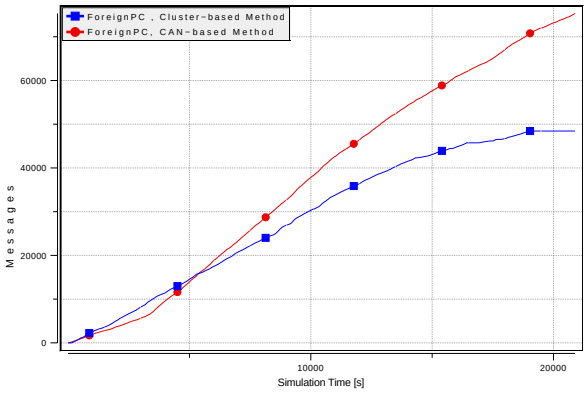


Fig. 10. The number of FPC messages with the CAN-based rectangular areas approach and the cluster-based approach with the Freewar trace data.

CAN-based algorithm (cf. Figure 10). As described in section VII-B, this is one of the main motivations for the cluser-based approach in order to improve the interactive experience for the players. But in the Freewar simulation the cluster-based method (cf. Figure 9 has a worse performance than the CAN-based approach with respect to player handovers. The CAN-based method causes 4,813 handovers and 75,325 FPC messages. The cluster-based method causes 8,038 handovers and 48,441 FPC messages.

So whether our CAN-based or the cluster-based approach is more suitable for a practical game depends on the typical behavior of the players in the game world and on the costs of FPC messages versus player handovers.

VIII. Conclusions

In this paper we use a structured P2P technology for the organization of the infrastructure and thus for the reduction of downtimes in MMOGs. By using a cluster-based approach we split the game world in groups of players and not in rectangular disjunctive zones. The clusters are distributed on different nodes of the P2P network. Thus, we get the possibility to dynamically connect and disconnect machines to and from the peer-cluster and to load-balance the game according to the actions of the players. The evaluation shows a better behavior than the previous CAN-based approach, which created rectangular areas.

References

[1] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle, "Peer-to-Peer-based Infrastructure Support for MMOGs," in *Proc. of CCNC*, Las Vegas, 2007.

[2] S. Ratnasamy, P. Francis, *et al.*, "A Scalable Content-Addressable Network," in *Proc. of the ACM SIGCOMM*, San Diego, 2001.

[3] T. Iimura *et al.*, "Zoned federation of game servers: a P2P approach to scalable MMOGs," in *Proc. of NETGAMES*, Portland, 2004.

[4] H. Lu, "Peer-to-Peer Support for Massively Multiplayer Games," in *Proc. of IEEE INFOCOM*, Hong Kong, 2004.

[5] J. Keller and G. Simon, "Solipsis: A Massively Multi-Participant Virtual World," in *Proc. of PDPTA*, Las Vegas, 2003.

[6] C. GauthierDickey, D. Zappala, *et al.*, "Low Latency and Cheat-Proof Event Ordering for P2P Games," in *Proc. of NOSSDAV*, Ireland, 2004.

[7] A. Rao, K. Lakshminarayanan, *et al.*, "Load Balancing in Structured P2P Systems," in *Proc. of IPTPS*, Berkeley, 2003.

[8] D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," in *Proc. of IPTPS*, San Diego, 2004.

[9] J. Byers, J. Considine, *et al.*, "Simple Load Balancing for DHTs," in *Proc. of IPTPS*, Berkeley, 2003.

[10] J. Cernik, "Freewar - MMORPG Browsergame," www.freewar.de.

Consistency Management for Peer-to-Peer-based Massively Multiuser Virtual Environments

Gregor Schiele*, Richard Süselbeck*, Arno Wacker†, Tonio Triebel*, and Christian Becker*

*University of Mannheim
Mannheim, Germany
{gregor.schiele | richard.sueselbeck | tonio.triebel | christian.becker}@uni-mannheim.de

†University of Duisburg-Essen
Duisburg, Germany
arno.wacker@uni-due.de

***Abstract*— Massively Multiuser Virtual Environments (MMVEs) require a seamless and consistent execution. To provide this, the MMVE must include a sophisticated consistency management. This management must adapt its behavior to different activities carried out in the MMVE as well as different system loads. As an example, a movement activity must be handled differently to picking up an object. During times of high system load, a lower level of consistency might be acceptable if this enables the MMVE to continue operating with low network delay. Developing such a consistency management is a complex and time consuming effort. To support MMVE developers with this task, we propose the creation of a dedicated consistency management infrastructure. This infrastructure can be configured by developers for their specific MMVE and makes it easier to maintain a consistent MMVE state.**

I. Introduction

A Massively Multiuser Virtual Environment (MMVE) allows thousands of users worldwide to interact with each other in a common environment in real-time. A crucial requirement for such systems is consistency [1]. As an example, if multiple users try to pick up the same object at the same time, the system has to guarantee that only one user actually receives the object. Otherwise, the users may become irritated by unusual effects in the environment's behavior.

At the same time, MMVEs must be highly responsive and progress seamlessly. Users expect the system to react promptly to their input and to progress without sudden state changes or rollbacks. To fulfill these conflicting goals, MMVE developers have to provide a complex consistency management which allows the MMVE to balance dynamically between consistency, responsiveness, and seamlessness. This is even more difficult to achieve in peer-to-peer (P2P) based MMVEs. In such systems, the MMVE is executed cooperatively on the hosts of its users and requires no central server. Therefore, there is no central entity to synchronize the participating entities and guarantee a consistent environment. To help reduce the development effort for such MMVEs, we propose the development of a novel consistency management infrastructure for P2P-based MMVEs. This infrastructure allows the application (i.e. the MMVE using our system) to choose between different consistency models at runtime.

Our work is executed in the context of the peers@play project [2]. The goal of the this project is the development of a comprehensive communication middleware for P2P-based MMVEs, including support for scalable update propagation, security, and consistency. In this paper we present the main design decisions and the architecture of our consistency management infrastructure. We present its main concepts and show how they can be used by developers.

The paper is structured as follows. First, we present our system model and discuss requirements for our consistency management infrastructure. We provide an overview of related work and present our approach. Finally, we offer a short conclusion as well as an outlook on future work.

II. System Model

Our system model consists of a number of users that want to use an MMVE, their devices, a communication network and the MMVE software. Users may be located at any place in the world. The number of users is a priori undetermined and can change dynamically. Each user owns a computing device, which executes the MMVE software and is connected to a common communication network, e.g., the Internet. We call such a device a peer. Each user is represented in the MMVE by a special character, called *avatar*. To interact with the MMVE, the user instructs his avatar to perform different activities, e.g., moving around or picking up an object. In addition, activities can be initiated by so-called non-player characters (NPCs). The state of the MMVE is distributed on the participating peers. If the user executes an activity, the system creates update events describing the resulting state changes. It sends these events to all other peers that hold a copy of this state. We assume the presence of a suitable P2P-based communication middleware to do so. The middleware determines the peers to which an event must be sent and provides all necessary communication services. In the context of the peers@play project, this functionality is provided by the peers@play middleware.

III. Requirements

A consistency management infrastructure in P2P-based MMVEs faces a number of requirements not encountered in

traditional client/server-systems. In the following we analyze these requirements in more detail.

1) Flexibility: Users perform a multitude of different activities in an MMVE, e.g. moving their avatar around, picking up objects, and interacting with other users or NPCs. The corresponding consistency requirements vary widely, depending on the activity's type and situation. As an example, the exact location of a user's avatar may not be crucial for nearby users, as long as they do not interact with it. However, once an interaction between users is initiated, position updates must be tightly synchronized. On the other hand, users are often willing to accept different delays for different activities. When a user picks up an object, he might tolerate several seconds delay. However, the same user will most likely not tolerate such a high delay each time he moves his avatar. In summary, both the required consistency and the maximum acceptable delay vary between different activities and situations. The consistency management must be flexible enough to allow the application to balance both these factors with each other.

2) Adaptability: As the MMVE is executed in a peer-to-peer system, the available devices and network resources can vary widely and without warning. If the user decides to log out of the system and turn off his computer, the system has to adapt dynamically to loosing this peer. In addition, user population in a given environment can vary. Thus, the system has to handle different situations concerning resource availability and system load. The consistency management must provide MMVE developers with means to react to high load situations to make sure that the MMVE maintains a satisfactory performance. As an example, the system might decide to lower the consistency requirements for certain activities in case of a high system load to maintain highest possible system responsiveness.

3) Extensibility: Different MMVEs may require different consistency models. To allow the consistency management infrastructure to stay independent of any specific MMVE, the infrastructure must enable MMVE developers to extend it with their own consistency models. As an example, an MMVE could include the possibility to buy objects from other users using real currency. To support such trading activities, the developer wants to include a consistency model that guarantees transactional properties, such as atomicity. In addition, it should be robust against attacks from other peers. On the other hand, a game-themed MMVE requires consistency models that are optimized towards latency.

IV. Related Work

In the past, a number of different approaches for consistency in MMVEs have been designed. These approaches can be divided into two classes, conservative and optimistic. Conservative approaches (e.g. [3],[4]) delay the processing of events until they have confirmed that it is safe to do so. Using this approach, if a peer receives an update event, it delays applying it to its local state until it can guarantee that it will never receive another update event that should have been applied previously. Conservative approaches are able to provide a high level of consistency, e.g. sequential or strong consistency. The ordering of events will always be the same on each peer. In addition, the user can be sure, that an activity will never be undone once it has been executed. However, a consistency model based on a conservative event synchronization approach might lead to high latencies, as faster peers could be required to wait for slower ones. In contrast, optimistic approaches (e.g. [5],[6]) process the events immediately, and try to correct possible inconsistencies through techniques such as rollbacks. They can be used to realize weaker consistency models, e.g. weak consistency. For the user, this means that an activity is executed instantly but that he can never be sure if it will be undone in the future. Hence, consistency models based on an optimistic event synchronization approach are specifically well suited for highly interactive activities that can be reversed without irritating the user too much. None of these isolated approaches are able to provide the necessary flexibility to support all different kinds of activities that are experienced in an MMVE.

For client/server-based MMVEs, several consistency infrastructures exist. Lu et al. [7] propose a consistency control mechanism that allows the application to select between two different levels of consistency, depending on, e.g., system load. The FITGap framework [8] allows states in an MMVE to be associated with different replication models, depending on their consistency requirements. In contrast to these centralized approaches, we provide a decentralized P2P infrastructure. A hybrid architecture was proposed by Pellegrino et al. [9]. It uses a central server to enforce a consistent game state using an optimistic synchronization protocol with rollbacks, while all other information is distributed via a P2P approach. Finally, OpenPING [10] is a middleware for networked games that includes different consistency models and offers dynamic adaptation through reflection. However, the supported consistency models are hard coded into the system and cannot be extended with additional models.

V. Our Approach

Our goal is to provide a consistency management infrastructure for P2P-based MMVEs. Our infrastructure should fulfill the requirements given before, namely: 1. flexibility, 2. adaptability, and 3. extensibility. In this section we discuss the main concepts and design decisions underlying our approach. Subsequently, we describe the architecture of the proposed infrastructure and illustrate how the different subsystems interact with each other using an example activity.

A. Design Rationale

Our system is based on a number of concepts that we discuss in the following. The main idea is to push application knowledge about the current situation of the MMVE and the performed activities into the consistency management to allow the latter to optimize system operation and minimize the latency experienced by users. In addition, the application is enabled to access information about the current situation in

the underlying P2P system in order to adapt its requirements to this situation.

1) Selectable Consistency Model: As discussed before, it can be insufficient to rely on a single consistency model. Instead, different situations and activity types should be handled with different consistency models. It may be acceptable in many situations to have loosely synchronized MMVE states between users that do not interact with each other. A user passing by another user's avatar does not need to have a perfectly synchronized view of the other user's current location. However, once both users interact with each other, e.g., by picking up the same object, a higher level of consistency is necessary. To facilitate this, two main approaches are possible. First, the system can provide a single integrated consistency model that provides different behaviors for different situations. Clearly, such a model would be complex and hard to maintain. The second possibility is to provide multiple independent consistency models and select the best model at runtime for each activity. We choose the second approach.

The MMVE can specify for each event which consistency model is required. This allows it to select the best consistency model for a given activity in a certain situation. Thus, if no strong consistency is required, the system can provide higher interactivity. For events for which strong consistency is mandatory, interactivity can be traded for consistency. Choosing the best model for a given situation requires MMVE-specific knowledge. Thus, to make consistency models reusable for different MMVEs, this decision should be separated from the models and pushed into the MMVE implementation. This approach fulfills the first requirement (flexibility).

2) Reflection Application Programming Interface (API): Allowing the application to select a consistency model enables the application developer to choose the best model for each activity. However, as described before, this selection should also depend on the current system load. If the system load is high, the application should be made aware of this and could react, e.g., by choosing a more resource efficient model. To enable this, we propose to offer a reflection API. Using this API, the application can query the consistency management on its current state. More precisely, the consistency management provides information about the currently expected delay for a given consistency model in a specified part of the MMVE. This delay depends, e.g., on the current system load in the corresponding P2P network. Clearly, the API could offer more detailed low level information, e.g., about the bandwidth currently available between specific peers. However, to use this information for selecting a suitable consistency, the application would require knowledge about the consistency model's implementation, e.g. what messages will be sent to whom. The reflection API fulfills the second requirement (adaptability).

We deliberately chose to push the decision how to react to a high system load into the application. Alternatively, this could be done by the consistency management internally. We argue, however, that only the application knows if selecting another consistency model is acceptable for a given activity. Furthermore, the application has to know which model is used to appropriately present the result of an activity to the user. As an example, consider an object that needs be removed from the user's inventory due to a rollback. To warn the user, the application could highlight this object in the graphical interface.

3) Consistency Plugins: So far, we have addressed the first two requirements, flexibility and adaptability. Still missing is to enable MMVE developers to extend the system with MMVE-specific consistency models – fulfilling the third requirement (extensibility). To do so, we propose to extract the implementations of consistency models from the infrastructure and place them into so-called consistency plug-ins. A consistency plug-in encapsulates a protocol implementing a specified consistency model. If an MMVE developer wants to add an additional consistency model, he can do so by implementing a consistency plug-in. He can also select the consistency models that are suitable for his MMVE and include only the plug-ins implementing them in the final product. Generic consistency models can be reused for multiple MMVEs, making development more efficient. Note that in contrast to other plug-in models, our plug-ins are selected at development time. At runtime, no new plug-ins are added. While offering this would allow updating the MMVE more easily, the ability to update code dynamically should be provided for the whole MMVE software and not for plug-ins, only. Therefore, we omit dynamic adding and removal of plug-ins to gain better system performance.

4) Consistency Sessions: All three requirements have now been addressed by the concepts discussed so far. To help the consistency management further optimize the MMVE execution, we provide an additional concept, called *consistency sessions*. Consistency sessions allow the application to group a number of events into a common context. As an example, the application can group events that are causally dependent on each other into a session. Thus it is possible to efficiently perform a partial rollback of these events. In our example of a user picking up an object, assume that the application uses a consistency plug-in with optimistic synchronization. Some time later, the application discovers that it must undo the pick up event. Normally, this would require to rollback the entire state of the MMVE. Alternatively, the MMVE could analyze all causal dependencies between this event and later ones to select the events to undo. If the application opens a new session before picking up the object, it can add all events that depend on the first one into it, and can easily determine the set of events to undo if necessary.

Note that placing an event into a consistency session does not mean that it will be delivered only to other members of the session. Reusing our example of a user fighting an NPC, both entities share a common session for the fight and movement events are tightly synchronized between them. If another user watches the fight, his peer should receive these events, too. However, he will most probably accept a lower level of consistency, if this prevents the fight from being slowed down. Therefore, the MMVE can select the required consistency model for an event per session. In addition, it can select the

consistency model that should be used for delivering the event to peers that are not included in a session. In our example, the MMVE would specify the use of strong consistency for movement events when sending them to entities in the session, i.e. the fighting user and the NPC, and no consistency when sending them to other entities, i.e. the observing user.

All events in a session must use the same consistency plug-in. While this restricts the usage of sessions, it makes implementing consistency plug-ins much easier. Otherwise, if a session contains events that use optimistic synchronization, and the MMVE tries to add an event that uses conservative synchronization, the plug-in implementing the conservative synchronization must guarantee that none of the earlier events in the session will ever have to be undone. Otherwise it would have to undo its own event, too, breaking its own semantic. To ensure this, the different consistency plug-ins would have to interact with each other. If an MMVE requires events in the same session to use different consistency models, the MMVE developer could implement a plug-in that integrates these models and makes sure that mixed sessions are executed correctly.

An event can be added into several sessions. This may lead to complex situations that must be handled by the MMVE. As an example, take an event that is added into a session with conservative synchronization and into a session with optimistic synchronization. Later, the system decides to roll back the events in the second session. However, the first session cannot be undone, as conservative synchronization was used. Currently such dependencies are not handled by the consistency management and must be dissolved by the MMVE. While we expect such situations to be rare, we plan to look into this issue more closely in the future.

5) Dynamic Relocation: As another optimization, we propose to let the consistency management initiate the dynamic relocation of objects in the MMVE to other peers. A prominent example for this is a user fighting an NPC in a game-themed MMVE. The activities of the user and the NPC have to be carefully synchronized to guarantee a consistent MMVE state and a satisfactory progression for the user. If the latency between the user's peer and the peer executing the NPC is high, this synchronization may decrease the system responsiveness unacceptably. The consistency management can detect this situation and try to optimize the communication delay. To do so, the management relocates the execution of the NPC to the user's peer. Thus, the network delay is reduced to zero and consistency and responsiveness can be provided efficiently.

As the overhead for relocating objects is usually high, the system should do so only if the resulting performance gain outweighs the initial effort. This depends on several factors, e.g. the current network delay, or how long the interaction between the peers will last. If synchronization between them is only required for a few events, no relocation is performed. On the other hand, if the peers are involved in a long term interaction, a relocation should be initiated. To detect such situations, we rely on the application. It notifies the consistency management about a potentially long running interaction. The management can then check if a relocation results in a sufficiently large performance enhancement and execute it.

B. System Architecture

Our proposed consistency management infrastructure consists of five system components as shown in Figure 1.

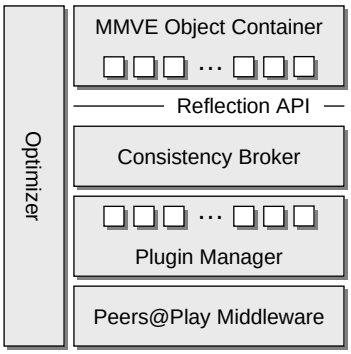


Fig. 1. Architecture of the consistency management infrastructure.

The *MMVE object container* manages MMVE objects. It allows objects to receive and send update messages if their state changes. In addition, the computation of an MMVE object's behavior can be relocated to another peer. To do so, it maintains the object's life cycle, transfers its current state and hands over control to the moved object. To use the object container, application developers inherit new objects from predefined classes.

Located below the object container, the *consistency broker* is the central contact point for the MMVE and its objects. Objects can register for incoming state update events, send update events to other peers, select the consistency models to use and notify the consistency management about possible MMVE object relocations. Furthermore, the broker implements the reflection API. MMVE objects can query the broker for the expected delay for a given consistency model. To support both discrete and continuous queries, the API offers both a synchronous system call as well as an asynchronous callback interface. To offer current delay information, the broker manages a local history of past interactions and exchanges such histories with remote brokers. Therefore a broker is able to provide the application developer with current information, even if the user's avatar has recently moved into another area. As an example, the MMVE might experience high load in a specific area in the MMVE. When a user moves his avatar from another low-load area into this one, his consistency broker exchanges historical data with others in the area and learns of this situation. It can notify the application of this, enabling the latter to initiate an appropriate reaction, e.g. requesting a lower level of synchronization for its activities. Finally, the consistency broker is responsible for forwarding update events between MMVE objects and plug-ins, and creates and maintains consistency sessions.

Consistency plug-ins are maintained by the *Plug-in Manager*. It offers an execution environment for plug-ins, main-

tains a list of installed plug-ins and mediates events between the consistency broker and the plug-ins. Each plug-in implements a consistency protocol, realizing a defined consistency model. To communicate with remote plug-ins, they interact with the peers@play communication middleware (see Section II), which determines the correct peers to send events to, etc.

Finally, the *Optimizer* adapts the system behavior to provide lower delays for executing interactions. It spans all system layers and combines information from different system components. Currently, it is used to initiate the relocation of MMVE objects. To do so, it accesses the communication middleware to learn about the current system state, e.g. network delay and bandwidth. To learn about long running interactions, it interacts with the consistency broker. If the Optimizer decides to relocate a MMVE object it contacts its local MMVE object container and initiates relocation. In the future, the Optimizer may contain additional optimization algorithms.

C. Example

After discussing the design rationale of our infrastructure and presenting its architecture, we give an example of an activity that is executed within our system. In this example, two users u_1 and u_2 attempt to pick up an object o at roughly the same time. The system needs to ensure that only one of the users receives the object.

First, the application selects as suitable consistency model. It decides that picking up an object is an event for which a strong consistency model is appropriate. The application queries the broker if the system is currently able to provide strong consistency, while maintaining adequate network delays. For the sake of simplicity, we assume that the application is satisfied with the current system state and a strong consistency model can be used. In addition to selecting a consistency model, the application also needs to determine if the event should be part of a consistency session. Since the application decided to use strong consistency, the ability to perform a partial rollback is not needed. Thus, the application decides not to include the event in a session.

Next, the application contacts the consistency broker and instructs it to enforce strong consistency by using the respective plug-in. It also hands the broker a set of context variables that are used by the strong consistency plug-in. In this example, the context contains an area of effect for the pick up event. This area determines all locations from which the object can be picked up. It is used later by the plug-in to determine which peers to synchronize with.

After being called by the broker, the plug-in determines all users and NPCs that are near o and thus would be able to pick it up. Here, this includes only u_2. Note that the plug-in does not know that u_2 has actually just tried to pick up o. It is only aware that she is in a position to do so, and therefore needs to be considered. The plug-in now contacts all affected users and NPCs (again only u_2) and notifies them that u_1 wants to pick up o at this time. In order to this, it uses the peers@play communication middleware. When the corresponding consistency plug-in at u_2's peer receives this request, it checks if it has already taken o or is in the process of doing so. Here, we assume that u_2 actually tried to pick up o a short while after u_1. Therefore u_1 is entitled to receive the object. The plug-in then passes this information back to the broker via the plug-in manager, which informs the application of the success of the event. During this entire process, u_1 has to wait for the confirmation of the consistency management. Only after the confirmation arrives, does the item actually appear in his inventory. The entire process looks almost identical for u_2, with the exception that the consistency management will report that o has already been taken. Thus, u_2 receives a message, that the object is no longer available.

VI. Conclusion and Future Work

In this paper we have proposed a consistency management infrastructure and presented its main concepts. We also gave an overview on the architecture of our infrastructure. Currently we are implementing the discussed concepts and components, as well as integrating a number of different consistency models. So far, our infrastructure offers no support for handling dependencies between different consistency plug-ins. As discussed before, this might be necessary if events are added into multiple consistency sessions. In future work we plan to extend our infrastructure with suitable coordination mechanisms for this.

References

[1] G. Schiele, R. Sueselbeck, A. Wacker, J. Haehner, C. Becker, and T. Weis, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proceedings of the Seventh International Workshop on Global and Peer-to-Peer Computing*, 2007.

[2] University of Mannheim, Duisburg-Essen and Lebniz University of Hannover, "peers@play homepage," published on the WWW at http://www.peers-at-play.org/, 2008.

[3] N. E. Baughman and B. N. Levine, "Cheat-proof playout for centralized and distributed online games," in *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-01)*. Los Alamitos, CA: IEEE Computer Society, Apr. 22–26 2001, pp. 104–113.

[4] J. Steinman, "Scalable parallel and distributed military simulations using the speedes framework," NASA, Tech. Rep., 1995.

[5] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Local-lag and timewarp: Providing consistency for replicated continuous applications," *IEEE Transactions on Multimedia*, vol. 6, pp. 47–57, 2004.

[6] E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin, "An efficient synchronization mechanism for mirrored game architectures," in *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*. New York, NY, USA: ACM, 2002, pp. 67–73.

[7] T.-C. Lu, M.-T. Lin, and C. Lee, "Control mechanisms for large-scale virtual environments," *Journal of Visual Languages and Computing*, vol. 10, pp. 69–85, 1999.

[8] A.-G. Bosser, *Technologies for E-Learning and Digital Entertainment*. Springer Berlin / Heidelberg, 2006, ch. A Framework to Help Designing Innovative Massively Multiplayer Online Games Interactions, pp. 519 – 528.

[9] J. D. Pellegrino and C. Dovrolis, "Bandwidth requirement and state consistency in three multiplayer game architectures," in *Proceedings of the 2nd ACM SIGCOMM workshop on Network and system support for games (NetGames'03)*, New York, NY, USA, 2003, pp. 52–59.

[10] P. Okanda and G. Blair, "OpenPING: a reflective middleware for the construction of adaptive networked game applications," in *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2004, pp. 111–115.

Data Aggregation Method for View Range Computation on P2P-based VCS

Ryo Nishide, Dai Ito, Masaaki Ohnishi, Shinichi Ueshima
Graduate School of Informatics, Kansai University
2-1-1 Ryozenji, Takatsuki, Osaka, 569-1095, Japan
Email: {fa4d003, fb6m124, fa4d001, ueshima}@edu.kansai-u.ac.jp

***Abstract*—Efficient data transfer is an essential topic to achieve scalability and data consistency to maintain the system in P2P-based Virtual Collaborative Space (VCS). In this VCS, each terminal requires the surrounding spatial data of its avatars for visualization of space. The congestion of avatars is then a serious problem when each terminal collects spatial data from the surrounding avatars. Thus, it requires a method to transfer data without delay and to relieve the load for terminals and networks. This paper proposes a data aggregation method on a P2P-based scalable geographic network to transfer the data efficiently at a congested area of avatars as nodes on a geographic network. The authors apply Skip Delaunay Network (SDN) generated from a hybrid structure of logical SkipNet and geographical Delaunay Network for remote access, and perform geocast for sending messages to a particular point or range on a plane. The authors conceive that multiple data paths to a common destination node construct a tree structure, in which the destination node is the root node, and nodes along the way are the internal nodes of the tree. Using the internal nodes for data aggregation, the proposed method can reduce frequent data transfer at a geographically crowded area of nodes. The authors show that data aggregation method on SDN can achieve both the long range contacts and reduction of CPU and network loads regardless of node distribution. The efficiency has been evaluated from the context of node congestion by examining the number of transferred data for methods with and without aggregation.**

I. Introduction

Virtual Collaborative Space System (VCS System), a system which uses the location and performs interaction on virtual space is gaining focus recently [1], [2]. It is a system with a set of interacting entities as avatars in virtual space. Users control these avatars from their terminals to walk-through in space, and perform interactions by sending messages to other users in virtual space.

Most of these VCS systems are built in C/S model [3], [4], which lacks scalability such as excessive cost for servers due to the increase of users. To overcome this problem, some efforts have been recently made for generating VCS systems on P2P setting [5–8], focusing on the characteristics as follows:

- Network scalability with respect to number of users
- System scalability according to spatial extension
- Distributive data management by space partitioning, and allotment of partitioned space to nodes

VCS systems have the same characteristic that each user requires only the local data of the surroundings. Thus, it is necessary to aggregate the surrounding data, and disseminate the data to a particular location or range on space. It is also necessary to cope with network congestion on P2P environment, by reducing inefficient data transfer for multihop communication.

We employ a well-known Delaunay diagram in computational geometry based on the adjacency of locations of avatars as nodes. We have proposed an autonomous and distributive generation algorithm of P2P Delaunay Network, which nodes generate such overlaid network cooperatively over 2D plane in P2P settings [9]. We have also proposed a Skip Delaunay Network (SDN) for reducing the number of hops for data transfer to remote nodes, using the hybrid structure of logical Skipnet and a geographical Delaunay network, and shown method to perform geocast to determine the directions to send data on geographical network [10].

If nodes within the view range increase, such problems as following become crucial in terms of network scalability, which requires a scheme to transfer data efficiently.

- Data Transfer Delay: Increase of number of hops
- Network Congestion: Data packet concentration to a specific peer

Furthermore, even though SDN can reduce the delay for data transfer, it cannot perform network load balancing for data packet concentration at a crowded area of nodes.

In this paper, we show data aggregation tree on SDN, which uses the scheme to temporarily cache multiple data and send them to the successor node at once. The paths for multiple data sent to the same destination node generate a tree structure, in which the root node is the destination, and each internal node of the tree is a node to cache multiple data and send them at once.

Using this scheme, we can reduce the frequency of data transfers between nodes, and cope with hotspots of nodes receiving data packets frequently. Our method employs advantageous features from both the SDN and aggregation method, by reducing the data transfer delay with SDN and handling network congestion with aggregation tree.

II. Features of Proposed Method

A. Generation of View

GUI Construction: In Virtual Collaborative Space, user's terminal requires the surrounding spatial data of its avatar to generate a view range for visualization of space. Thus, to construct a GUI, each user's terminal requires the spatial data managed by other users.

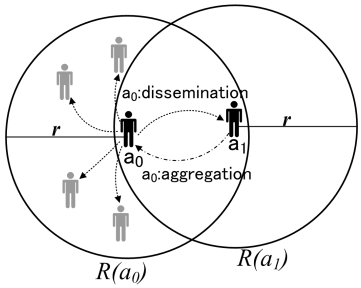
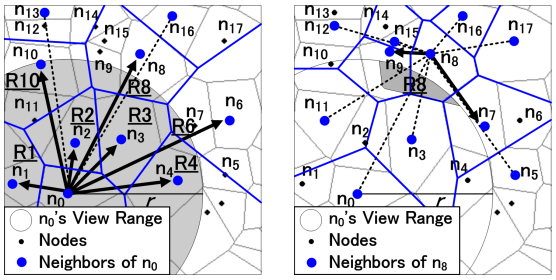


Fig. 1. a_0's Data Dissemination and Aggregation for Constructing a View



(a) n_0: Geocast in $R(n_0)$ (b) n_8: Geocast in R_8

Fig. 2. Geometric Routing to Nodes within View Range on SDN

To gather data from terminals of the surrounding avatars, it is necessary for other terminals to send their **(location, state)** data. On the other hand, it is also required to receive these data from the terminals of nearby avatars. Our model enables each terminal to generate a view with a certain range r by disseminating and aggregating these data to terminals of its surrounding avatars autonomously and distributively. Here, we assume the size and shape of view range is equal for every user's terminal.

A particular user's terminal which is controlling avatar a_i generates the view range $R(a_i)$, and sends data to every terminal which has avatar within $R(a_i)$. The number of terminals to send data depends on the density of avatars within the view range. Note that two particular avatars a_i, a_j mutually within their own view range require data of each other.

Fig. 1 shows an example for generating a view of avatar a_0. It shows that a_0 requires data of a_1 within the view range $R(a_0)$. On the other hand, a_1 requires a_0's data in the same way. Assuming that radius r of view range is the same with every avatar, a_0's terminal disseminates its data to its surrounding avatar's terminals, and aggregates other terminals' data within the view range $R(a_0)$ including a_1's data. In this way, the necessary spatial data are sent to all terminals with avatar in the view range.

Delaunay Network: Delaunay Network provides locality connections only with its adjacent neighbors, which can efficiently gather the surrounding spatial data required for generating a view. We use the locations of avatars controlled by users' terminals as nodes' location on Delaunay Network. Moreover, by assigning Voronoi Regions as the managing territories of space to the entire nodes, the entire data within the view range can be queried as the Voronoi regions of entire nodes cover the entire plane.

B. SDN as a Scalable Network

Geocasting within View Range on SDN

Delaunay Network requires a mechanism to access to remote nodes in case when node congestions occurred within the view range. Else, it can cause data transfer delay depending on the number of nodes within the view range. Thus, connections with remote nodes are necessary to send data within the congested area of nodes.

We use SDN, which is a hybrid structure of SkipNet and geographical Delaunay Network on a plane [10], for generating connections with remote nodes. Moreover, we perform geocast on SDN to send data to every node within the view range. Using SDN, we can build long range contacts (LRC) to send data to remote nodes, to deal with data transfer delay.

To perform geocast, each node generates Voronoi Diagram virtually with all of the neighbor nodes throughout the entire level of SDN. Using this Virtual Voronoi Diagram, we can determine the neighbor nodes to send data by extracting the intersection area A_i of query range and neighbor nodes' Virtual Voronoi Region. We send the data to every node, whose Virtual Voronoi Region intersects with the query range.

When data are sent to all neighbor nodes within view range, some nodes might receive the same data from multiple neighbor nodes. To avoid this, we set the direction to send data by replacing the query range with intersection area A_i, and send data with query range A_i to neighbor nodes. Thus, we can send data to a specific direction, and each query range data will be received only once.

We use the following notations to describe our method:

- NN: neighbor node set of the entire level of SDN
- $V.Vor(n_i)$: Virtual Voronoi Region of n_i
- $q(n_i)$: query range of n_i

Using these notations, we derive the following formula to determine the neighbor nodes $NN_{send}(n_i)$ to send data.

$$NN_{send}(n_i) = \{n_j \in NN | V.Vor(n_j) \cap q(n_i) \neq \phi\} \quad (1)$$

Instead of sending the view range to neighbor node n_j, we only send the intersection area $q(n_j) = \{V.Vor(n_j) \cap q(n_i)\}$ to node n_j. Note that this intersection area is used to set the direction to send data.

Here, we describe the method to perform geocast on SDN, to assure that n_0's data can be reached to every node within a certain view range r, using Fig. 2. Initially, n_0 generates $V.Vor$ with neighbor nodes of every level of SDN (blue dots on Fig. 2 left). In the figure, n_1, n_2, n_3, n_4, n_6, n_8, n_{10}, n_{13}, n_{16} are the neighbors of n_0. Among these neighbors, n_0 sends data to neighbors n_i with intersecting $V.Vor(n_i)$ and view range $R(n_0)$, which refers to all the neighbors of n_0 except n_{13} and n_{16}. The data includes the intersection area data R_i on the figure.

Then, node n_i which has received R_i data verifies their intersections with their neighbor's $V.Vor$ and R_i, and sends

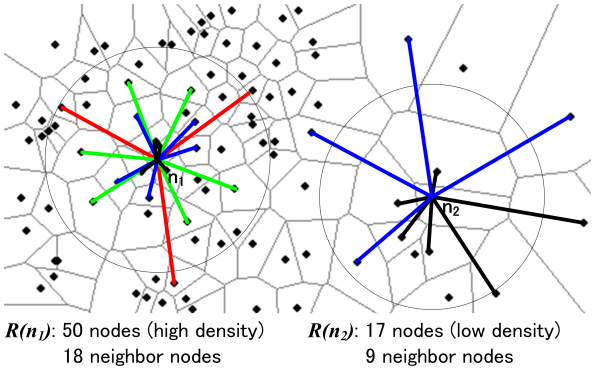


Fig. 3. Node Congestions and Number of Neighbors (Color = Level)

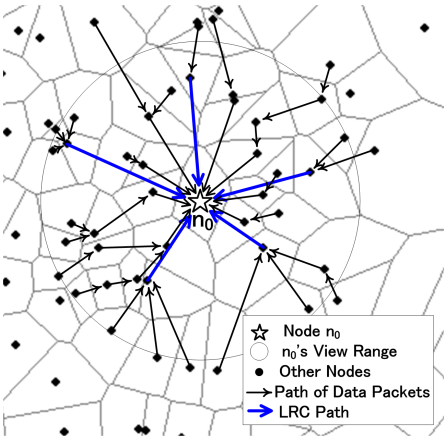


Fig. 4. Aggregation Tree Model for Data Transfer

intersection data to its neighbors in the same way. Fig. 2 right shows n_8, which has received R_8 data from n_0, sends the intersection data of neighbor's $V.Vor$ and R_8 to neighbor nodes n_7 and n_9, respectively. Using this routing scheme, the data can be delivered to the entire nodes within the circular range with radius r.

This method for disseminating data within view range has the following characteristics:

- Data can be sent to every node with an intersection of its Voronoi region and view range
- Data can reach the destination node(s) regardless of the location and shape of view range

Congested Nodes on SDN

In the previous section, we have shown that we can expect an efficient routing scheme for disseminating data to remote nodes by using SDN. We believe that this scheme can provide efficient routing for nodes within the view range. In this section, we describe how to solve the remaining problems for SDN when data concentrate to nodes in the crowded area.

In SDN, the network load can be balanced if nodes have equal size ranges and are uniformly distributed on a plane. However, when nodes are skewed at a particular location on space, the network load can be congested at the skewed location of nodes on space. Specifically, SDN has a characteristic that a particular node possesses many connections if a large number of nodes are within the view range, which increases the risk for data packets to be received from multiple nodes frequently (Fig. 3). Therefore, to deal with such problems, it is necessary to consider an efficient data transfer scheme to avoid network congestion.

When sending data packets, it is obvious that they should be sent with long messages up to the limit of window size, instead of sending numerous short message packets frequently. Otherwise, frequent transfers of short message packets can waste the bandwidth, as several packets may get lost due to buffer overflow, which requires the packets to be sent again. Resultantly, this increases the risk for massive packets to flow in the network, which causes network congestion.

Thus, in order to avoid such situations, we consider a model to cache multiple received data and send to the following neighbor nodes at once. In this way, we can save bandwidth by reducing frequent transfer of data packets, and reduce the number of data packets in the entire network.

C. Data Aggregation Method

In the previous section, we have shown method for nodes to send data using geocast, and explained that skewed distribution of nodes can cause network congestion when performing geocast on SDN. Here, we provide solution to reduce network load and frequent data transfer, using data aggregation tree to transfer data efficiently.

Data aggregation tree is a tree structure built from the paths of multiple data sent to a common destination node. This tree is generated passively from the paths of data sent by geocasting. The node which constructs the view range is the root node of the tree, and each node generates its unique data aggregation tree. Data are sent to its parent node of the tree recursively, until the data reach the destination node.

When data are sent to their parent node, multiple data can intersect at a particular internal node of the tree. From this internal node, the data take the common path to the destination node. We consider that instead of sending the data individually, multiple data should be cached and sent together as a single data to the successor node.

Here, we provide an example of data aggregation tree of node n_0 (Fig. 4). Let n_i, n_j be source nodes to send data d_i, d_j to destination node n_0 respectively. The paths for d_i, d_j have either of the following characteristics:

- d_i and d_j intersect at a particular node n_k, and takes the same path to n_0
- d_i and d_j intersect at n_0

In this tree, n_0 is the root node and intersecting node n_k is the internal node. To reduce frequent data transfer, d_i and d_j are packaged as a single data at n_k and sent to n_0 accordingly.

We believe that data aggregation tree can considerably reduce the data transfer frequency for these nodes. Moreover, the total data transfer frequency can stay low throughout the entire nodes.

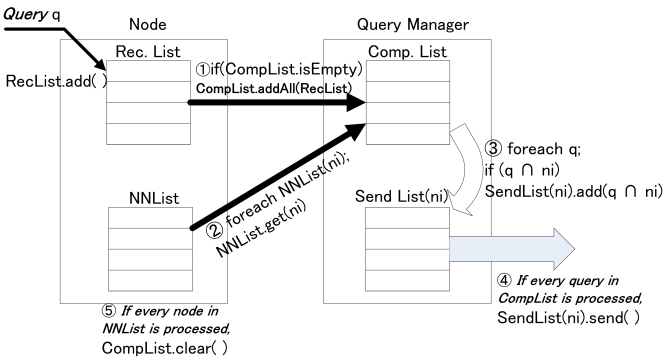


Fig. 5. Data Structure of our Method

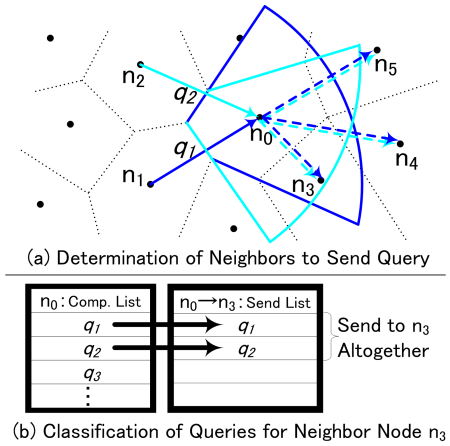


Fig. 6. Data Aggregation and Transfer Method

III. Proposed Method and its Data Structures

Here, we describe the proposed method for generating an aggregation tree. Figure 5 shows the data structure of our method and roles for each list. Initially, each node has three lists, namely Receive List (Rec List), Neighbor Node List (NN List), and Computation List (Comp List).

The queries include [Node ID, Packet ID, Location, Query Range, Time]. All the queries received from other nodes are stored in Rec List.

The details of the processes in Fig. 5 are as follows:

1) When Comp List is empty, move all the queries from Rec List to Comp List, and clear Rec List
2) Extract each neighbor node n_i from NN List
3) Obtain the intersection area with each query range in Comp List and $V.Vor(n_i)$. Generate Send List(n_i) and store intersection area for each query
4) When every query in Comp List is processed, send Send List(n_i) to node n_i
5) Perform step 2) with next neighbor in NN List. Clear the Comp List when every node in NN List is processed

Get Queries from Rec List

In order to determine the next destination node to send range query, we obtain the intersection area of query range and $V.Vor$ of neighbor nodes. We do not use queries in Rec List to obtain the intersection area, as this process should not be interfered by the process of adding queries in Rec List. That is, each node in NN List in turns obtains an intersection area with each query in Comp List, hence the new added query might not be processed by some neighbor nodes, which have already completed their intersection area computation.

Therefore, we use Rec List to store queries received from other nodes, and Comp List to get queries from Rec List. In detail, the Comp List pulls out a set of queries from Rec List and stores in Comp List, when the Comp List is empty. Moreover, the Comp List clears the list every time when the processes have been completed throughout all neighbor nodes.

Selecting Neighbor Nodes to Send Queries

Here, we describe the process to choose the neighbor node to send the queries in Comp List. To choose the neighbor node, the following information is required.

- $V.Vor$ of every neighbor node (in NN List)
- Query range from predecessor node (in Comp List)

To choose the destination node, we obtain the intersecting area of query range and $V.Vor$ for every neighbor node in NN List. Every neighbor node with an intersection area will be the target node to send the query. Instead of sending the view range as query range, we send the intersection area as query range to successor neighbor nodes.

Figure 6 (a) illustrates q_1 and q_2's query ranges intersecting with $\{V.Vor(n_3),\ V.Vor(n_4),\ V.Vor(n_5)\}$. Thus, the next destination nodes for q_1 and q_2 will be n_3, n_4, and n_5. Consequently, the next destination nodes (n_i) will be sent with the intersection area of $V.Vor(n_i)$ and q_1, q_2 respectively.

Send List for Packaging and Sending Queries

When sending packets to a particular neighbor node, it is inefficient to send range query packets one after another. Therefore, we generate a Send List to store every query with an intersection area of neighbor node, and send the queries together to the neighbor node.

The Send List is generated separately for each neighbor node, and each query in Comp List is classified to the Send List of relevant neighbor nodes. To send queries in Send List together, we generate a package with all queries in Send List, and send it to the neighbor node. In this way, we can send multiple queries together to neighbor nodes, avoiding frequent data transfer.

Figure 6 (b) shows an example for generating Send List of n_3, and utilizing it for packaging multiple queries to send to neighbor node n_3. Multiple queries q_1, q_2, q_3, ..., q_i are stored in Comp List. The Send List for n_3 stores q_1 and q_2 as these two queries are to be sent to n_3. Finally, q_1 and q_2 are packaged and sent to n_3.

IV. Evaluation

In this section, we verify the efficiency of our data aggregation method from the effects on node congestion within the view range. We have obtained the number of received queries according to the increase of nodes within the view range.

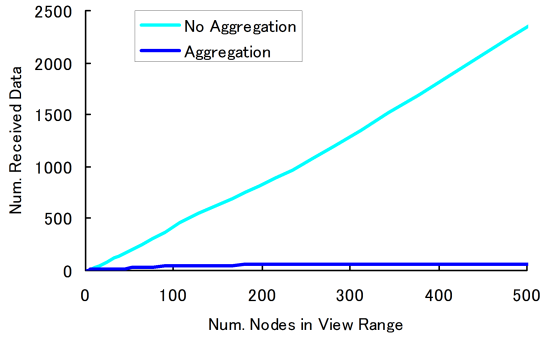


Fig. 7. Node Density and Number of Received Data

In our method, each node has a view range, and multicasts data within the view range. We fix the shape and size of view range for every node, and examine how the node density affects the CPU load for method with/without aggregation.

Settings for Simulation

Under the following settings, we have obtained the number of received data for with/without the data aggregation on SDN.

- **View range**: Nodes have circular view range
- **Node density**: Nodes increase within the view range
- **Data transfer**: Nodes send each data per step
- **Aggregation**: Package data and send to neighbor nodes
- **No aggregation**: Just cast data to neighbor nodes

CPU Load w.r.t. Node Congestion

We compare the method with/without aggregation in terms of congestion of nodes within the view range. In fig. 7, we examine the number of received data w.r.t. the number of nodes within the view range.

The figure shows that number of received data without the aggregation method rises steeply according to nodes increase. From the result, we consider that few increase of nodes can extremely increase the number of received data, as the data sent from a single node can be received from multiple neighbor nodes. On the other hand, the number of received data for aggregation method rises slightly, as multiple data are packed together and sent to the successor nodes. Therefore, the number of received data for aggregation method stays low even in node congestion.

We have performed step-by-step data transfer in this simulation. If we apply our method in real environment, we need to consider the CPU performance and data transfer speed. Thus, it is important to determine an appropriate interval time length for aggregating data, while considering the required threshold of time length for visualization and interaction in space.

V. Related Works

To obtain scalability in respect to number of users, a method to transfer spatial data efficiently is an important issue. Efficient routing mechanism and a scheme to reduce frequent data transfer are some required issues for achieving scalability.

For an efficient routing mechanism, some works on geocast have been proposed. [12] proposes a geocast routing mechanism based on node's location to send messages to nodes within a specified geographical area on mobile ad-hoc networks. [10] proposes method to construct a probabilistic link structure with remote nodes on P2P Delaunay Network, and applies geocast routing mechanism for sending messages to a specific geographical point or range on space.

For reduction of data transfer frequency, a method has also been proposed to construct a tree for aggregating and sending multiple data to its connected node. [13] proposes directed diffusion scheme for data dissemination, to intentionally aggregate multiple data at internal nodes of the tree. On the contrary, our method generates an aggregation tree passively from the path of multiple data sent using geocast.

In our proposed method, we can achieve efficient data transfer from both advantages of an efficient geocast routing mechanism and data aggregation method.

VI. Conclusion

We have proposed method to utilize Data Aggregation Tree for efficient data transfer, and examined that our method works efficiently on SDN through numerical simulation. Furthermore, with Data Aggregation Tree and geometric routing on SDN, we can perform geocast efficiently while avoiding network congestion.

For our plans in future works, additional evaluations are required with/without aggregation method, such as verifying the CPU and network load, acquiring the appropriate interval time lengths for aggregating data, and examining the amount of data loss due to transfer frequency of packets.

References

[1] B. Damer, "Meeting in the ether," ACM interactions, Vol.14 No.5, pp.16–18, 2007.
[2] M. Macedonia, "Generation 3D: Living in Virtual Worlds," IEEE Computer, Vol.40 No.10, pp. 99–101, 2007.
[3] Second Life, http://secondlife.com/
[4] Active Worlds, http://www.activeworlds.com/
[5] S.-Y. Hu, J.-F. Chen and T.-H. Chen, "VON: A scalable peer-to-peer network for virtual environments," IEEE Network, Vol.20 No.4, pp.22–31, 2006.
[6] B. Knutsson, H. Lu, W. Xu, B. Hopkins, "Peer-to-Peer Support for Massively Multiplayer Games," In Joint Conf. IEEE Computer and Communications Societies, Vol.1, pp.107, 2004.
[7] Y. Kawahara, H. Morikawa, T. Aoyama, "A Peer-to-Peer Message Exchange Scheme for Large Scale Networked Virtual Environments," IEEE ICCS, pp. 957-961, 2002.
[8] P. Morillo, J.M. Orduña, M. Fernández, J. Duato, "Improving the Performance of Distributed Virtual Environment Systems," IEEE Trans. on Parallel and Distributed Systems, Vol.16 No.7, pp. 637-649, 2005.
[9] M. Ohnishi, R. Nishide, S. Ueshima, "Incremental construction of delaunay overlaid network for virtual collaborative space," 3-rd Proc. Conf. on Creating, Connecting and Collaborating through Computing (C5'05), (IEEE CS Press), pp.77–84, 2005.
[10] S. Tsuboi, T. Oku, M. Ohnishi, S. Ueshima, "Generating Skip Delaunay Network for P2P Geocasting," 3-rd Proc. Conf. on Creating, Connecting and Collaborating through Computing (C5'08), (IEEE CS Press), 2008.
[11] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, A. Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties", 4th USENIX Symp. on Internet Technologies and Systems (USITS '03), Vol.4, p.9, 2003.
[12] Y. B. Ko, N. H. Vaidya, "Flooding-based geocasting protocols for mobile ad hoc networks", Mobile Networks and Applications, Vol.7 No.6, pp.471–480, 2002.
[13] C. Intanagonwiwat, D. Estrin, R. Govindan, J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks", 22nd Int'l IEEE Conf on Distributed Computing Systems, pp.457–458, 2002.

An Implementation of a First-Person Game on a Hybrid Network

Anthony Steed*, Bingshu Zhu

Department of Computer Science, University College London

ABSTRACT

The majority of current networked virtual environments and networked games use a client-server model of networking. This makes synchronization of the environment simple, but it adds additional latency between any two clients. Peer-to-peer networking is a common alternative, but whilst these systems have potentially lower latency, they are more difficult to scale and lack a single point of synchronization.

In this paper we demonstrate an implementation of a form of hybrid networking, where clients communicate important state to a server, but communicate rapidly changing state peer-to-peer. By using the frontier set concept, we can make this scale since we ensure that only relevant position updates are sent peer-to-peer. The implementation is done as a modification to the game Quake 3 Arena. We show that the latency between clients is indeed reduced significantly for position events, and that this is achieved at a relatively small increase in network traffic.

CR Categories: C.2.4 [Computer Communication Networks]: Distributed Systems; I.3.2 [Computer Graphics]: Graphics Systems

Keywords: frontier sets, latency, synchronization, network scalability, networked virtual environments.

1 INTRODUCTION

Networked virtual environments (NVEs) pose a number of challenges to system designers [22]. The system designer must balance the requirement of ensuring consistent behavior of objects across a possibly widely distributed set of communicating clients, with the requirement of ensuring that any individual client has a continuous and usable experience. This is especially true in networked games such as first person games, where complete consistency between clients is unachievable across wide areas at a rate acceptable to the frenetic pace of the game. Thus in this type of games, the clients commonly run in a partially desynchronized manner. The main decision that a system designer has to make is to choose between a client-server system where synchronization is relatively easily determined by the server but where all messages must go via the server, and peer-to-peer systems where the synchronization problem is exacerbated, but the latency of any particular event is minimized.

In this paper we investigate a hybrid networking model, where communication is a mix of peer-to-peer and client-server, for the game Quake 3 Arena. Normally peer-to-peer communication in such a situation would be prohibitively expensive, however we use frontier sets to partition the world so each peer in the network can independently decide whether to send messages to other peers and thus each peer only communicates their position information with others peers that are likely to be visible. Bypassing the server in this way allows minimal latency between generation and rendering of positions of players in the game.

Reducing some of the sources of latency should provide a better experience for the players. This is because it can remove some discrepancies due to motion extrapolation and rollback that occur in such games where the server determines critical events and communicate these events to clients that are simply extrapolating a previous state. We show that hybrid networking is feasible in a real game scenario, and that, in packet count and throughput terms, it has a relatively small impact on the network load.

*A.Steed@cs.ucl.ac.uk

2 RELATED WORK

Today's NVEs have emerged from technologies developed for military simulators. Two dominant architectures have been pursued: peer-to-peer systems and client-server systems. SIMNET was an early example of a peer-to-peer system where each client simply broadcast information about its state on the network [18]. Such systems require a lot of bandwidth to cope with the volume of messages. There is also a general problem in ensuring consistency and security, problems acknowledged in the early MiMaze system [7].

The main alternative to peer-to-peer systems is client-server systems. In this type of system all clients connect to a server and that server is responsible for computing the state of the game and distributing it to all the clients. Consistency is easily managed since there is one canonical copy of the game state on the server. However a server introduces extra latency because any update to the game state needs to be relayed by the servers. The issue of latency is very important for real-time games, especially first-person shooter games which have become very popular in the last few years [2][10].

2.1 Partitioning

In order to have environments that scale to large numbers of users one common approach is to partition the world and only relay a subset of all events and state to each client [20].

One of the first systems to employ a partitioning scheme was the NPSNET system [16]. NPSNET divides the virtual world into fixed sized hexagonal cells. Each participant sends information (e.g. location updates) to their current local cell but can choose to receive information from potentially many cells that fall within their area of interest. The Spline system [26] divides a virtual world into arbitrary-shaped locales that are stitched together using portals. Each locale defines its own co-ordinate system and participants receive information from their current locale and its immediate neighbors. The ability to use variable-sized locales provides additional flexibility in coping with less predictable entities and is more appropriate for indoor environments.

Many games are built with environment models that have a lot of occlusion between models. One technique that is often used in such games is potentially visible sets (PVS) [27]. A PVS can be used to exclude a pair of entities from consideration for simulation

purposes because they are not mutually visible. However this visibility must be evaluated every time the entities concerned move. The RING system exploits a PVS data structure to increase the scalability of a client-server virtual environment [8]. In the RING system, a server culls messages if it knows that a client can't see the effect of the message.

Other systems propose to partition groups dynamically, depending on local awareness and neighborhood relationships [14][15][19]. These and similar schemes could provide for highly-scalable peer-to-peer networking, but have not as yet been used in practical implementations.

2.2 Latency

Latency in NVEs arises because of the physical transmission of the data and the cumulative processing latency of sender, receiver and router processes. Latency immediately induces inconsistency because an individual client "sees" the behavior of all other clients after a delay. Thus the player reacts to state when at the remote site that state may have already changed. A typical way this manifests in online FPS games is that the client sees and shoots at a target, but that target has moved. The role of the server is to be neutral and resolve such inconsistencies in as fair a way as possible. However, introducing a server causes slightly different problems, where players might think they have dodged a bullet, but because the server "sees" their behavior slightly delayed they subsequently get informed that they were shot in the past. This may require the client to roll-back to a previous state. A thorough survey of latency including a discussion of the inconsistencies that arise and strategies to combat them is given in [5][6].

2.3 Hybrid Networking

Hybrid networking is a term that is commonly used to refer to services that combine client-server and peer-to-peer communications. For example, Chen & Muntz propose a peer-clustering scheme that distributes some server responsibilities amongst peers [4]. In that scheme servers are necessary to support critical tasks, but peer clusters take over the handling of local interactions.

Our use of hybrid networking embodies a similar principle of delegating some responsibilities to the clients, but the goal is simpler: to reduce the latency with which certain events are seen by the clients.

2.4 Frontier Sets

The frontier set was introduced by Steed & Angus [24][25]. It is a concrete example of a more general class of algorithm called update-free regions [9][17]. These algorithms exploit the fact that if any two players know instantaneously where the other is, they might both be independently able to establish that they do not need to communicate until they leave an area. A simple example is two players on either side of a long wall. Until one or the other reaches the end of the wall, they can't possibly see their counterpart: for mutual visibility to be possible, one of them must round the wall.

A frontier set reifies this concept in the specific situation of a world where there is a visibility relationship such as a PVS [27]. If the environment is divided in to regions of space (or *cells*) then for any cell, it will be possible to identify openings (or *portals*) through which other cells can be seen. For any cell, it is possible to explicitly compute which other cells are visible from that cell, because if a cell is visible then there must be a line of sight through all the portals between them.

A single frontier is defined relative to pairs of cells in that subdivision. Given two cells A and B, a frontier comprises two sets of cells F_{AB} and F_{BA} such that no cell in F_{AB} is visible to a cell in F_{BA} and vice-versa. Figure 1 gives an example of a frontier. The complete set of frontiers for a whole environment will be referred to as a frontier set.

2.4.1 Example Usage

Consider two users moving around the environment depicted in Figure 1a. If Anne is in cell A, and Bob is in cell I at time t_0 then a frontier can be established, F_{AI} = {A,B,C,F}, F_{IA} = {G,H,I} as shown in Figure 1a. If Anne remains in the set of cells F_{AI} and Bob remains in the set of cells F_{IA} then they can never see each other. If this were a networked virtual environment this would mean that if Anne and Bob both exchanged location information at t_0 they would not have to send any further updates.

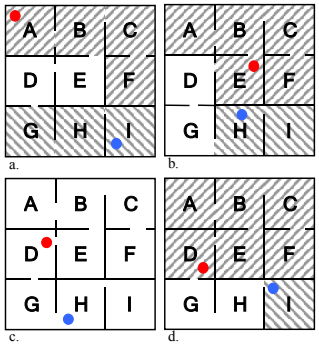


Figure 1. An example of frontiers in use. As users Anne and Bob move between cells, frontiers can sometimes be established. a) Anne and Bob are in cells A, I respectively. A frontier exists F_{AI} = {A,B,C,F}, F_{IA} = {G, H,I}. b) A frontier exists F_{EH} = {A,B,C,E,F}, F_{HE} = {H,I}. c) No frontier exists because cell D can see cell H. d) A frontier exists F_{DI} = {A,B,C,D,E,F} and F_{ID} = {I}.

The work of Steed & Angus was done on simulations using the Quake II code base [24][25]. The work presented in the remainder of this paper is live, playable system, implemented as a modification to the GPL version of the Quake 3 Arena code.

3 Architecture

3.1 Strategy

Even if peer-to-peer networking can be made to scale reliably, for many types of application, it will still be desirable to have a centralized server doing specific, application-critical state. This could be state that must be synchronized quickly so that clients do not diverge, or state that no client can be trusted to compute. In an FPS, examples might be the game score, and individual kills. However, much information is transient and updated frequently. In many systems, and particularly FPSs, this could be positions of players and other entities. In some engines such classes of information might even be separated on to different transport layers. The rationale for this is that if a position information packet is lost, it doesn't matter, and might even be detrimental to performance to retransmit the packet, as would be done with TCP.

The strategy with hybrid networking is thus to classify changing state into that which can be sent peer-to-peer, and that which needs to go client-server. There could be overlap between the two classes; that is information that is sent to both server and peer clients. Because sending directly to peers is low latency, this could be done where there are situation where the client believes that its information is correct and will not be altered by the server. A good example is position information in dynamic simulations. These systems commonly do dead-reckoning of position [18]. When a client observes that it needs to update the models of other

players, it would be best to notify all clients directly, as this will provide for lowest error when the dead-reckoning models adapt to the new information. Similarly in an FPS the information that I fired a weapon is useful to communicate as fast as possible to other players. The server in this FPS still needs to know the position and weapon firing immediately, as it needs to determine the overall effect of this.

A final strategic reason for distributing data both client-server and peer-to-peer is to provide redundancy for time-critical information, in case of loss of either peer-to-peer or client-server or server-client packets. Indeed, in certain situations, it might be necessary to fall-back to sending via a server anyway, because one of the peer-to-peer routes cannot be established due to firewall or other constraints.

3.1.1 Use of Frontier Sets

Although simply reducing the latency may be desirable in some situations, peer-to-peer traffic potentially incurs an overhead on network traffic. A *naïve peer-to-peer* algorithm generates $O(N^2)$ network traffic each frame, where N is the number of clients. A *perfect peer-to-peer* algorithm would send events only when they are necessary because they would be seen by the receiver. Obviously such an algorithm is un-implementable because the client would have to have prior knowledge of where the receiver was in order to know whether to send a packet. We thus propose to use frontier sets as an initial strategy to reduce traffic and make the system scale. It would be eminently feasible to have the server dynamically indicate to each client which other clients it would need to communicate to. The server could use a visibility data structure to determine this. However this introduces some latency in to the set up of communication.

Frontier sets work exclusively peer-to-peer, and in simulations for an FPS game they have been shown in simulation to achieve performance on the network very similar to the perfect peer-to-peer algorithm [25]. Further, in the FPS simulations, where the data sent peer-to-peer did not need to additionally be sent to the server, the frontier set algorithm was more efficient than client-server algorithm in certain situations. Frontier sets are relatively easy to implement, as their calculation does not need any negotiation with the peer or server. Using a peer-to-peer algorithm also reduces the latency of set-up of a particular peer-to-peer data flow: it doesn't require a server computation and thus data is sent from one peer to another as soon as the sending peer detects that it is necessary.

3.2 Impact on Latency

The key advantage of using peer-to-peer information is that it provides for lower latency: the information travels over only one link not two. However, there is another important advantage: even if the server is close to the "mid-point" of the link, that is the peer1-peer2 trip time is close to the sum of the trip time from peer1-server and server-peer2, the fact that we are cutting out a server process means that jitter in packet arrival time can be cut dramatically. Because the peers and server all update at a given rate, the server introduces a new source of process latency.

Once we introduce hybrid networking, each client sees their peers with minimum latency, thus reducing instantaneous inconsistency. However, furthermore, under the assumption that peer-to-peer and peer-server transit times are similar, the server is actually using the same state for calculations as the other peers are displaying at that time.

4 Implementation in Quake 3 Arena

Quake 3 Arena [11] is a first-person shooter that was originally released in 1999. It uses a client-server architecture. A machine can be set up as a dedicated server, or one participant can host a server. The server has an update rate of 20 Hz [1, p. 163]. Typical games have up to 32 participants. Like previous games in the series, Quake 3 Arena uses a cell partitioning of the world and there is PVS structure across this. The source code for Quake 3 Arena is available under the GPL, so it is easily extensible for experiments such as ours.

The first alteration that needs to be made is to set up data structures that allow frontier creation. As discussed in [24] in the context of Quake II, when a game world is loaded, the PVS data structure is converted in to an enhanced-PVS data structure. This takes 10-20 seconds depending on the map, and could be done as a pre-process.

The second alteration is to add a capability to notify each client of all its peers, so that they can make direct connections. There are additional messages to notify of clients leaving and departing the session.

The third alteration is the networking functions. Quake 3 Arena uses UDP distribution, with its own reliability mechanism. Snapshots of data are sent between client and server, and any resend is triggered only if necessary. Each snapshot packet contains a number of data structures, and we add a new data structure which is a peer-to-peer position data structure. The sending peer client simply sends a packet containing this data structure to the UDP port on the receiver client that the server normally connects to, and thus the receiver handles the packet as it would any packet from the server.

The main logic is thus on the sending side. Each client must know whether or not it currently has a frontier with another client. If it doesn't have one, based on the last position packet received, doesn't know, or it has left its half of the frontier then it sends a position packet.

To implement this each client calls a function, *CL_SendNetworkUpdate* each frame. The main job of this function is to establish if, since the last frame, this client or one of other clients have left the agreed frontier. If they have, they get rid of the current frontier. If there is no current frontier then it sends an update to the other client. Finally it tries to establish a new frontier with the current cells for this client and the other client. The client keeps two arrays frontierThis and frontierOther. Each element of these arrays is one of the sets of cells from a frontier. The space for each element much be a binary vector the length of the maximum number of cells in a map. This is not large compared to other static resources that are allocated in memory by the game engine. The process of building frontiers is almost identical to the process discussed in [25] and the implementation borrows heavily from the Quake II code. In essence though, the process requires a single iteration through the list of cells in the map, to construct the two cells lists for the frontier, if they exist. If the two cells are mutually visible, this function returns immediately.

One key aspect of the implementation is that peers send position updates at their frame rate. This can be much more than would be relayed by the server (i.e. 20Hz), but is a reasonable strategy for some situations as it results in much smoother movement. In our tests described in the next section we used fairly modern PCs, and Quake 3 Arena does not stress modern graphics cards. If some of the machines on the network were likely to be more stressed, then rate limiting the peer-to-peer sending would be necessary.

Final changes included logging capabilities to determine the latencies and data usage of the different networking routes. The server process collates all data logged by individual clients.

5 Preliminary Results

Preliminary analysis was done on a small number of two player games to establish throughput and latency characteristics. Latency

TEST	CLIENT	FRAME	SV PACKET	CL PACKET	SV BYTE	CL BYTE
1	A	7997	1795	0	35484	0
	B	7996	1795	0	50235	0
2	A	6949	1565	6957	51980	132541
	B	6957	1565	6949	57605	138111
3	A	9822	2209	582	68620	11659
	B	9817	2209	582	46049	11182

Table 1: Packets and bytes sent by each client, both to the server (SV) and to the other client (CL). The tests are distinguished by the amount of time which the two players can see each other.

is tricky to calculate due to the problems inherent in synchronizing clocks across the network. Figure 2 gives a reference diagram for the timings we have used.

t_{A1} to t_{A4} are client A's local time; t_{B1} to t_{B5} are B's local time and t_{SV1} to t_{SV2} are server's local time. In the game, when client A shoots towards client B at time t_{A1} (meanwhile, B is at time t_{B1}), A will send a packet to B indicating this event. This function is added for testing purpose. A will also send a packet to server and that is an original standard function of Quake 3 Arena. B receives this packet at time t_{B2} and the corresponding time at A is t_{A2}. B processes this event then sends back a response packet to A at time t_{B3}. A gets this response packet at time t_{A4}. The P2P one-way transmission latency Δt_1 is considered to be the average of $t_{A2} - t_{A1}$ and $t_{A4} - t_{A3}$. Since it is hard to synchronize the time on client A and B, the measurable time includes: t_{A1}, t_{B2}, t_{B3}, t_{A4}. Thus the one way P2P transmission is:

$$\Delta t_1 = \frac{(t_{A4} - t_{A1}) - (t_{B3} - t_{B2})}{2}.$$

The P2P overall latency (two-way transmission latency plus processing latency) is simply:

$$\Delta t_2 = t_{A4} - t_{A1}.$$

When A shoots B, another packet is sent to the server. The server gets it at time t_{SV1} then sends a packet to inform B at time t_{SV2}. $t_{SV2} - t_{SV1}$ is the processing latency at the server. B gets this packet from server at time t_{B5}. In the testing plan, overall latency of the message through server is measured and denoted as:

$$\Delta t_3 = t_{B5} - t_{B2} + \Delta t_1.$$

The three latencies were measured by observing 20 packets sent both ways through the network, as shown in Table 2. We can see here that there is a very marked difference between the one-way and client-server communication.

Data rates were also calculated. Rates vary drastically depending on whether the two players can see each other. In Table 1, Test 1 is a situation when clients are in valid frontier sets so they do not need to communicate. In Test 2 valid frontier sets have never been able to be set up. The figures show that clients are sending packets peer-to-peer at the frame rate. In this extreme situation communication between client and server is much less than that between clients. This is because the original Quake 3 Arena server does not send packets to client at the frame rate, but at a fixed lower rate. In addition, Quake 3 Arena does delta-compression on the data packets so packet size is normally small. From the table it could be calculated that the average packet size of client-server Quake 3 Arena is 27.8 bytes and that of peer-to-peer Quake 3 Arena is 19.5 bytes. The new Quake 3 Arena being tested only sends player position information peer-to-peer so 19.5 is quite a high number. In the future the modified Quake 3 Arena could also perform compression on the packets. Test 3 is close to the real situation of playing. In this situation the traffic between the peers is roughly 25% of that to the servers.

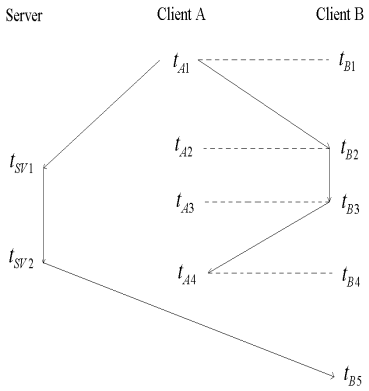


Figure 2. Time stamps used in the calculation of difference in latency for client-server and peer-to-peer communication.

Timing	Δt_1	Δt_2	Δt_3
Mean	6.375	12.95	28.25
Std.dev	2.804	5.633	11.77

Table 2 Timings of peer-to-peer single way, peer-to-peer two way, and client-server communication

6 CONCLUSIONS

We have demonstrated a practical implementation of hybrid networking as a modification to the Quake 3 Arena game. We showed that we could reduce instantaneous inconsistency by reducing client-client position update latency and that this could be done with reasonable increase in network traffic.

Currently this is a proof-of-concept demonstration, though there is no theoretical reason why this should not scale to support larger numbers of players. Although we only presented preliminary tests with two players, the game does function correctly with more players and large scale tests are being planned. The previous simulations on Quake II, [24][25] should indicate that the total packet overhead for 16 or 32 player games would not be onerous .

Future work will look at several issues: delegating more responsibilities to the players, stress testing with more players, subjective and qualitative review of the impact of latency reduction on player experience. We would highlight some expected problems that are common to any peer-to-peer scheme, such as maintaining consistency when there are dense clusters of players. This is could be a problem because at least with a client-server system the peak load on the network, servers and clients, is well known, whereas in peer-to-peer systems peak load is harder to calculate.

The modified Quake 3 Arena code is available on *http://www.cs.ucl.ac.uk/staff/A.Steed/*

REFERENCES

[1] ARMITAGE, G., CLAYPOOL, M., BRANCH, P. 2006 Networking and Online Games: Understanding and Engineering Multiplayer Internet Games, Wiley. k

[2] BEIGBEDER, T., COUGHLAN, R., LUSHER, C., PLUNKETT, J., AGU, E., AND CLAYPOOL, M. 2004. The effects of loss and latency on user performance in unreal tournament 2003®. In Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support For Games (Portland, Oregon, USA, August 30 - 30, 2004). NetGames '04. ACM Press, New York, NY, 144-151.

[3] CECIN, F. R., DE OLIVEIRA JANNONE, R., GEYER, C. F., MARTINS, M. G., BARBOSA, J. L. 2004. FreeMMG: a hybrid peer-to-peer and client-server model for massively multiplayer games. In Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support For Games (Portland, Oregon, USA, August 30 - 30, 2004). NetGames '04. ACM Press, New York, NY, 172-172.

[4] CHEN, A. AND MUNTZ, R. R. 2006. Peer clustering: a hybrid approach to distributed virtual environments. In Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support For Games (Singapore, October 30 - 31, 2006). NetGames '06. ACM, New York, NY, 11.

[5] DELANEY, D , WARD, T., S.MCLOONE 2006. On consistency and network latency in distributed interactive applications: a survey--part I, Presence: Teleoperators and Virtual Environments, v.15 n.2, p.218-234, April 2006.

[6] DELANEY, D , WARD, T., S.MCLOONE 2006. On consistency and network latency in distributed interactive applications: a survey--part II, Presence: Teleoperators and Virtual Environments, v.15 n.4, p.465-482, April 2006.

[7] DIOT, C. GAUTIER, L. 1999. A Distributed Architecture for MultiParticipant Interactive Applications on the Internet. In *IEEE Network*, 13(4), 6-15.

[8] FUNKHOUSER, T. A. 1995. RING: A Client-Server System for Multi-User Virtual Environments. In *1995 Symposium on Interactive 3D Graphics*. 85-92, April 1995.

[9] GOLDIN, A., GOTSMAN, C. 2004. Geometric message-filtering protocols for distributed multiagent environments. Presence: Teleoperators and Virtual Environments, 13(3), 279-295.

[10] HENDERSON, T. 2001. Latency and User Behaviour on a Multiparticipant Game Server. *Networked Group Communication 2001*, Third International COST264 Workshop, London, UK, November 7-9, 2001 1-13

[11] IDSOFTWARE. 1999. Quake 3. http://www.idsoftware.com/games/quake/quake3/

[12] IEEE. 1993. ANSI/IEEE Standard 1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation, March 1993.

[13] KELLER, J., SIMON. G. (2003) Solipsis: A massively multi-participant virtual world. In Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03).

[14] KAWAHARA, Y., MORIKAWA, H., AND AOYAMA, T. 2002. A peer-to-peer message exchange scheme for large scale networked virtual environments. In Proceedings of the the 8th international Conference on Communication Systems - Volume 02 (November 25 - 28, 2002). ICCS. IEEE Computer Society, Washington, DC, 957-961.

[15] KNUTSSON, B. LU, H., XU, W., HOPKINS. B. (2004) Peer-to-peer support for massively multiplayer games. In Proceedings of the 23rd Conference of the IEEE Communications Society (Infocom 2004), Washington, D.C., 2004. IEEE Computer Society.

[16] MACEDONIA, M. R., ZYDA, M. J., PRATT, D. R., BARHAM, P. T., ZESWITZ, S. 1994. NPSNET: A Network Software Architecture for Large Scale Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 3(4): 265-287, MIT Press.

[17] MAKBILI, Y., GOTSMAN, C., BAR-YEHUDA, R. (1999) Geometric Algorithms for Message Filtering in Decentralized Virtual Environments. Proceedings of the ACM Symposium on Interactive 3D Graphics, 39-46.

[18] MILLER, D., AND THORPE, J. 1995. SIMNET: the advent of simulator networking. *Proceedings of IEEE*, 83(8): 1114-1123.

[19] MORILLO, P., MONCHO, W., ORDUÑA, J.M., DUATO, J. 1996. Providing Full Awareness to Distributed Virtual Environments Based on Peer-to-peer Architectures, in Computer Graphics International (CGI'06), volume 4035 of Springer LNCS, pp. 336-347

[20] MORSE, K. L., BIC, L. AND DILLENCOURT, M. 2000. Interest management in large-scale virtual environments. *Presence: Teleoperators and Virtual Environments*, 9(1):52--68, MIT Press.

[21] QuakeWorld, http://en.wikipedia.org/wiki/QuakeWorld

[22] SINGHAL, S. ZYDA, M. 1999. Networked Virtual Environments: Design and Implementation. Addison-Wesley.

[23] SMED, J. KAUKORANTA, T. AND HAKONEN, H. 2001. Aspects of Networking in Multiparticipant Computer Games. In Loo Wai Sing, Wan Hak Man, and Wong Wai (eds.), *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*. Hong Kong SAR, China, Nov. 2001, 74-81.

[24] STEED, A., ANGUS, C. 2005, Supporting Scalable Peer-to-peer Virtual Environments Using Frontier Sets. In Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality (March 12 - 16, 2005). VR. IEEE Computer Society, Washington, DC, 27-34.

[25] STEED, A., ANGUS, C. 2006, Enabling scalability by partitioning virtual environments using frontier sets. Presence: Teleoperators and Virtual Environments, 15 (1). pp. 77-92.

[26] STERNS, I.B., YERAZUNIS, W.S. 1997. Diamond Park and Spline: Social Virtual Reality with 3D Animation, Spoken Interaction and Runtime Extendability. *Presence: Teleoperators and Virtual Environments*, 6(4), 461-481, MIT Press

[27] TELLER, S.J. SEQUIN, C.H. 1991. Visibility Preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91),* 25(4):61-90.

Solipsis: A Decentralized Architecture for Virtual Environments

D. Frey* IRISA
J. Royan† Orange Labs
R. Piegay ‡ Orange Labs
A.-M. Kermarrec IRISA
E. Anceaume IRISA
F. Le Fessant IRISA

ABSTRACT

Lack of scalability is a key issue for virtual-environment technology, and more generally for any large-scale online experience because it prevents the emergence of a truly massive virtual-world infrastructure (Metaverse). The Solipsis project tackles this issue through the use of peer-to-peer technology, and makes it possible to build and manage a world-scale Metaverse in a truly distributed manner. Following a peer-to-peer scheme, entities collaborate to build up a common set of virtual worlds. In this paper, we present a first draft of the Solipsis architecture as well as the communication protocol used to share data between peers. The protocol is based on Raynet, an n-dimensional Voronoi-based overlay network. Its data-dissemination policy takes advantage of the view-depedent representation of 3D contents. Moreover, the protocol effectively distributes the execution of computationally intensive tasks that are usually executed on the server-side, such as collision detection and physics computation. Finally, we also present our web component, a 3D navigator that can easily run on terminals with scarce resources, and that provides solutions for smooth transitions between 3D Web and Web 2.0.

Keywords: Peer-to-peer System, Metaverse, Shared Virtual Worlds, Massively Decentralized System, Adaptative 3D Streaming.

Index Terms: K.6.1 [Management of Computing and Information Systems]: Project and People Management—Life Cycle; K.7.m [The Computing Profession]: Miscellaneous—Ethics

1 INTRODUCTION

The Metaverse concept, first described by Neal Stephenson in his science fiction novel 'Snow Crash' published in 1992, and more generally depicted in the whole cyberpunk writing movement, has deeply influenced generations of virtual reality pioneers, artists, game designers and nowadays virtual worlds enthusiasts. Over the last sixteen years, the notion has evolved toward a synonym for virtual world, loosing progressively the 'Universe' part of the concatenation and the huge and infiniteness feeling emanating from it.

However, for us the Metaverse is a system of numerous, interconnected virtual and typically user-generated worlds (or Metaworlds) all accessible through a single-user interface. According to this strict definition, the only Metaverse existing today is a prehistoric one: the World Wide Web itself. Plenty of virtual worlds flourish these days claiming they are the Metaverse, but they only are a part of it, as websites are the leaves of the worldwide web tree. We need three things to reach this cyberpunk authors' dream. First, a way to sustain the incredible amount of data and MIPS involved, then a set of protocols to provide interoperability and finally new tools to build virtual worlds as easily as a traditional HTML page. These requirements are the cornerstones of our project.

*e-mail: davide.frey@irisa.fr
†e-mail: jerome.royan@orange-ftgroup.com
‡e-mail: romain.piegay@orange-ftgroup.com

The paper is structured as follows. We first present a synthetic overview of the Solipsis research project in Section 2, and an analysis of related work in Section 3. Then we describe how we envision to manage decentralized virtual worlds, describing in details our peer-to-peer architecture, how we manage 3D-model sharing and physics computation, and how to stream 3D contents in an adaptative way. Finally, in Section 5, we present our navigator, which is the user interface used to interact with the Solipsis Metaverse, while in Section 6, we conclude the paper and outline our future directions.

2 PROJECT OVERVIEW

In a word, Solipsis is a platform for massively multi-participant and user-generated virtual worlds. It relies on a peer-to-peer architecture that makes scalability its main characteristic: the universe may thus be inhabited by an unlimited number of participants. As there is no central authority, the virtual universe is by definition public and the inhabitants' freedom, as well as the world builders and developers' imagination, are boundless.

2.1 Expected results

We seek to spark off the emergence of an unbounded public virtual universe that is designed, created, run, but also potentially hosted, by people throughout the world. Freedom can be spotted as the main characteristic of this opensourced system (license GNU/GPL v2+) because the virtual universe does not belong to any organization; it belongs to all the users.

The major deliverable is a new network communication protocol adapted to the strong constraints of self-produced environments, to massively multi-user applications and to virtual reality, which make the system more scalable as the resources its uses are those provided by its users.

We also work on creating an ergonomic and user-friendly interface to allow non-professional agents to easily create 3D scenes and contents (declarative or automatic modeling, tagging..). Nevertheless, we do not want to mark a break with the actual flat 2D web; rather, we aim to start a smooth transition towards an immersive Internet making the most of both 2D and 3D. As we will see in part V, our navigator can be embedded in a regular webpage - in-web world - or map regular webpages as an interactive texture on any 3D surface - in-world web.

The last major expected result is a deep analysis on the behavior (virtual social life, game, education, services..) of metaverse users to give directions for future developments.

3 RELATED WORK

The Web3D consortium originally had the ambition to create a freely navigable online world [14]. Unfortunately, due to technical constraints, such as bandwidth limitations, the VRML standard was only used to encode simple 3D contents in order to visualize them on web sites. Only related research and bandwidth increase have now made it possible to draft the architecture of a Web3D, metaverse or cyberspace.

3.1 Compression and adaptive 3D Streaming

A lot of research work has focused on the compression of 3D models, as well as on adaptive streaming methods that allow a progres-

sive and fast transmission over networks of the required 3D contents visualized from the current viewpoint.

First, existing 3D compression algorithms use both techniques adapted from the 1D and 2D cases (like wavelets, entropy coding, and predictive coding), and completely different approaches that take advantage of the properties of 3D surfaces (like Edgebreaker, Subdivision Surfaces, and triangle strips)[9]. Also, parametric solutions, that provide intuitive modeling tools, are widely used to create avatars, complex creatures, or vehicles in games. More complex 3D contents can be created using procedural modeling solutions that have the drawback of requiring a reconstruction process executed on the fly on the client side. Parametric and procedural modeling provide very good compression rate compared to usual mesh compression algorithms, but no generic solutions exist to model a great variety of objects.

The second solution consists in filtering the 3D contents required to visualize the scene from a given viewpoint. Indeed, a complete download of a huge 3D scene is not necessary to render with optimal details the virtual environment from a given viewpoint. First, the area of a 3D model projected on the screen depends not only on its size, but especially on its distance from the viewpoint. Many solutions have therefore been proposed to adapt the resolution of 3D models to the current viewpoint [8]. Continuous levels of detail allow to transmit refinements progressively as the viewpoint comes closer to 3D objects [10]. Second, most of 3D objects in huge and complex 3D scenes are occulted by other ones during the navigation. Thanks to an offline computation of 3D objects visible from regions resulting from a partitioning of the navigation area, servers can signifanctly reduce the amount of data that has to be sent to the client, without affecting the visual quality [16]. Unfortunately, visibility filtering methods have to be disabled during flying-over navigation, since occultations are clearly limited. In fact, these filtering methods provide multi-resolution functionalities allowing an adaptive 3D streaming of the virtual environments.

3.2 Centralized architectures

Several architectures take advantage of filtering methods presented previously [7, 6]. Commercial platforms such as Google Earth [1] and Second Life [2] have recently known a tremendous craze from the public. The first allows navigation into and over a well-detailed model of the earth, with terrain and buildings. In doing this, it takes advantage of an adaptive streaming of terrain textures [15], associated with a multi-resolution wavelet-based representation for terrain, and static levels of detail for buildings. The second provides an advanced social-network service combined with general aspects of a metaverse. The Second Life client integrates modeling tools that allow users to create new components of the virtual environment.

In addition to the above, more than fifty online virtual worlds have appeared over the last few years, and are usually based on centralized architectures. Huge environments are generally partitioned, according to a grid, in regions that are managed by dedicated servers, called region servers. Unfortunately, in order to synchronize the game states of the world for all connected clients, most processes such as collision detection, physics computation, and animation, are executed on the server side. Moreover, the region server is the only source that can provide clients with the 3D models of the scene for its visualization. Thus, the number of clients that can navigate into a region managed by only one server is clearly limited.

3.3 Peer to Peer Architectures for Virtual Worlds

A centralized architecture cannot lead to a truly self-scalable solution, even with the use of multiple servers. Indeed, client-server architectures lead to prohibitive deployment and maintenance costs when it comes to very large scale applications with thousands of connected clients. On the other hand, thanks to their *self-adaptation* features, P2P network overlays have clearly proved to be an effective alternative to powerful servers.

Based on the fact that if peers have nearly the same viewpoint, they are likely to need the same data, geometric proximity is obviously the main criterion for setting up peer connectivity within virtual environments. However, finding and maintaining the appropriate peer connectivity is a very difficult problem in a dynamic environment and in the presence of churn, that is where peer viewpoints are allowed to move freely and when peers can disconnect or appear at any time. Solipsis [12] and VON [11] are the first P2P layers providing a solution for 2D environments. In these P2P architectures, peers are connected to each other according to their current 2D position. Dedicated algorithms are used to achieve the global stability of the P2P network while fulfilling a global connectivity constraint (i.e. there must exist at least one path between each pair of peers).

Maintaining real-time peer connectivity in a n-dimensionnal space is much more complex. For this reason, Douglas et al. [5] have proposed a region-based approach using a distributed spatial data index in a multidimensional space to find nearby objects with which direct connections can be established. Finally, Cavagna et al. [4] show that server-less P2P networks can efficiently deal with very large environments, first by using well-suited descriptors to specify areas of interest for continuous levels of detail (for on-demand streaming), and second by having peers exchange information about their own serving capabilities (for self-regulating peer-upload bandwidth).

4 Decentralized Virtual Worlds Management

Our response to the demand for an efficient metaverse platform capable of 3D interactions among entities is the new version of Solipsis. Its key characteristic is the ability to distribute the cost associated with the management of its metaverse among the hosts participating in it. This is made possible through the decentralization of heavy processes like collision detection, computation of physics, as well as communication among the large number of nodes that participate in the metaverse. In the following, we present the main characteristics of the architecture and of the protocol enabling such decentralization.

4.1 Definitions

The Solipsis metaverse consists of a set of *entities*, each belonging to one of three categories: *avatars*, *objects*, and *sites*. Avatars are the main actors of the metaverse as they are the only entities that are capable of autonomous movement. In most cases they are virtual representations of the users of the metaverse and are directly controlled by them using the navigator platform described in Section 5. Objects, on the other hand, are virtual representation of entities from the real world such as furniture, books and whatever object may be moved or picked up by an avatar. Avatar and objects may also be robotic-like entities controlled by user-defined software components. Finally, sites constitute the basic building blocks of the metaverse and represents portions of the virtual space that may be occupied by objects or in which avatars can roam.

Avatars, objects, and sites are all associated with 3D-descriptions, each consisting of a mesh- or prims-based model, a set of textures, and, in the case of avatars or objects, also of an animation. Such 3D-descriptions constitute the basis for the rendering of entities in the navigator platform.

The state of an entity in the metaverse is determined by an *entity descriptor*, from now on referred to simply as *descriptor*. An entity's descriptor can contain information regarding its position, its physical properties, and whatever is needed to display the entity and compute its interaction with the rest of the metaverse. For example, a descriptor containing a key frame can be used to synchronize animations, videos, and so on. Moreover, filtering de-

UID	universal identifier of the entity
seqNum	sequence number
owner	identifier of the node managing the entity
type	site, avatar or object
loc	location in the 3D space
ori	orientation in the 3D space
shape	shape from a predefined set
box	bounding box of the object
R_p	perceptibility radius: distance from which the object is visible in the absence of obstacles
R_b	radius of smallest sphere enclosing the entity
$objs_a$	list of entities attached to the current one
f_1	first file of 3d-description
v_1	version number for first file
c_1	list of hosts that have cached v_1 of f_1
...	...
f_n	n-th file of 3d-description
v_n	version number for n-th file
c_n	list of hosts that have cached v_n of f_n
...	additional fields for progressive levels of details

Table 1: Example of a simple descriptor

scriptors defining areas (concentric spheres, hierarchical partitioning, cells) associated with a level of detail or a set of visible objects may be used to achieve efficient on-demand streaming while satisfying view-dependency criteria. Finally, descriptors may contain load-balancing information regarding, for example, the available resources or serving capabilities of a host [4]. Table 1 shows a sample descriptor containing the most relevant information. Although the descriptor is shown as a single object, the Solipsis implementation may split it into several descriptors regarding, for example, physical properties, location, or level of details in order to reduce communication cost.

4.2 Solipsis Hosts and Nodes

From a practical viewpoint, the Solipsis platform is distributed over a set of hosts that maintain information about every entity that is currently present in the metaverse. Each host is associated with a single instance of the Solipsis platform, and throughout its lifetime, it may create new entities or destroy previously created ones according to the requests of the navigator component. Also, similar to an entity, each host is associated with a unique identifier (UID).

Because each host may be responsible for several entities at the same time, we define a *(Solipsis) node* as the set of resources dedicated to the management of a given entity. Each node encapsulates information about the corresponding entity's descriptors as well as the threads of control responsible for managing all the interactions of the node with the rest of the Solipsis metaverse as well as with the navigator platform.

4.3 P2P Architecture

A fundamental aspect of Solipsis is its completely decentralized architecture designed to accommodate large numbers of entities, accessed by large numbers of users distributed over the Internet. The core of this decentralized architecture is a peer-to-peer overlay network, which is essentially a graph where nodes are connected by virtual logical links, each of which may consist of several physical links in the underlying IP network. These logical links may be laid out according to some proximity metric so as to enable efficient storage and retrieval of information.

Recent years have seen the emergence of a large number of peer-to-peer overlay networks, with different structures and capabilities. In the context of Solipsis, we leverage the potentialities of peer-to-peer overlays by building our metaverse on top of RayNet [3]. Raynet is a multi-dimensional overlay network based on the concept of Voronoi tessellation. As shown in Figure 1, each node in the overlay is associated with a position in a multi-dimensional space. Neighborhood relationships between peers are then determined by the distance between the corresponding points in this space. Specifically, two Raynet nodes are neighbors in the overlay if the two corresponding points are neighbors in the Delaunay graph comprising all the points associated with Raynet nodes.

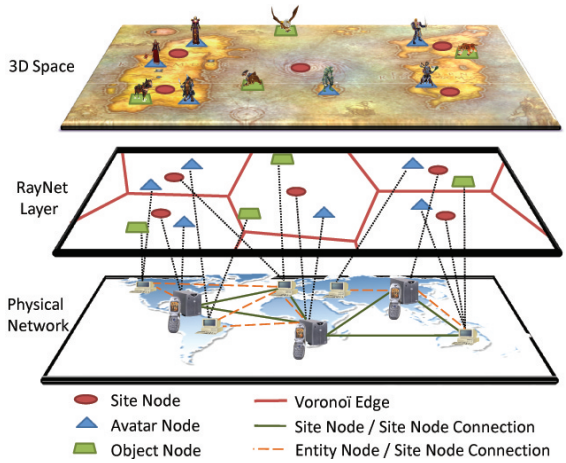


Figure 1: The p2p architecture showing the mapping of the Raynet layer on the virtual world layer according to the proximity of nodes in the virtual space. The host layer shows clients connected into a peer to peer scheme. Note that while the positions of avatars and objects are also shown, for clarity, on the Raynet layer, only site nodes actively join the RayNet and form the corresponding Voronoi diagram.

Given a set of generator points$\{p \in \Re^d\}$, the Delaunay graph is obtained as the adjacency graph of the corresponding Voronoi diagram, which in turn is a tessellation of the d-dimensional space $\Re^d$. Specifically, the cell in the tessellation associated with a point p_x is such that it contains all the points that are closer to p_x than to any other generator point in the set. This property enables Raynet to route to any node in the overlay by means of a simple greedy approach. Moreover, the addition of Kleinberg-like [13] long-range links allows this routing process to converge to its destination node in a polylogarithmic number of hops on average.

Within the context of Solipsis, we use the Raynet structure to organize *site nodes* into a 3-dimensional overlay in which the positions of nodes mimic those of the corresponding sites in the metaverse.[1] This allows our architecture to exploit very simple protocols to manage the interaction between avatars, objects and the sites they are currently located in as we describe in the following.

4.4 Decentralized physics computation

The choice of a peer-to-peer overlay such as Raynet is motivated by our goal to scale to metaverses containing very large numbers of entities, possibly gathered within the same site or within an otherwise small region of space. Organizing sites nodes into such an overlay, however, is only half of the picture. Solipsis also incorporates a fully decentralized protocol that distributes the cost associated with heavy computations such as those regarding collision detection, physics, or animation.

Each avatar node is responsible for computing its own position based on physical criteria like its mass, momentum, and forces applied by the entities in its surroundings. Our decentralized approach allows it to achieve this result by taking into account only a small

[1]Extensions of the metaverse architecture to higher dimensions are naturally supported by the Raynet overlay.

set of 3D models: those associated with the current site and with the entities that are in the avatar's immediate surroundings. For example, in the case of collisions, each avatar may immediately rule out the entities that are at a distance that is greater than the radii, R_b, of the spheres enclosing its and their own bounding boxes plus a configurable safety distance.

The basis for this decentralized computation of physics is a communication protocol that allows nodes to exchange information about entities' positions with three levels of heartbeat messages. Critical information that is required for the computation of collision detection is exchanged directly in a peer-to-peer fashion between the avatars that may potentially collide with each other, based on the size of their bounding boxes. Less critical information that is nonetheless necessary to provide the navigator with a clear snapshot of the current scene state is instead propagated indirectly, in a multi-hop fashion. Finally, information about distant objects or objects that have just joined the metaverse is propagated by site nodes to the avatars within their cells at a significantly lower frequency.

An avatar joining Solipsis contacts the site node responsible for its joining - or latest - location. Although the avatar is not part of the overlay, it can easily do this by using the Raynet to route a message containing its descriptor to the site node that is closest to its own position. This node then reacts by providing the avatar with the descriptors of the entities that are potentially visible from its location. From this point on, the avatar and site nodes exchange this information periodically to implement the low frequency updates described above. An analogous mechanism is used to manage the positions of object nodes. The use of the Raynet always allows avatars and objects to contact the right node as they move from site to site.

In addition, avatar and object nodes interact with the avatars and objects around them in order to exchange the critical up-to-date information about the entities that may collide with them. Also, while sending an update to a nearby node, avatar and object nodes also include information received from other nearby nodes that are not within collision distance of the destination node; thus implementing the second level of peer-to-peer updates.

4.5 Dynamic Object Management

While avatars are the main actors in the Solipsis environment, objects also play an important part as they can interact with avatars, be picked up, be moved from one location to another, or even move freely through space subject to gravity or other forces, or according to a script of animation. This requires our protocol to determine the current position of objects in addition to that of avatars. This is achieved through a combination of three mechanisms that depend on the state the object is currently in.

Let us consider an object that is currently stationary within a site S's cell. The position of the object is maintained by S's site node, which also maintains the main copy of the object's descriptor. As soon as an avatar tries to interact with the object, by moving it or picking it up, its avatar node sends a request to the site node to take over responsibility for the object. The site node grants such responsibility to the first avatar requesting it, according to the order in which requests reach the site node.

When permission to take over the object is granted, the object is dynamically detached from the site and attached to the avatar. The corresponding descriptors are updated, and the avatar nodes starts computing physics and maintaining the descriptor for the object. Note that physics computation need not be performed by site nodes, in that an object can only be attached to a site when it is in static equilibrium.

After interacting with the object, the avatar may pass the object on to another avatar, or leave it in the current or in a different site. In the former case, the recipient of the object takes over its management, and starts computing its physics, and animation, and updating its descriptor. In the latter, on the other hand, the object may be left in a stationary or in a dynamic state. If the object is stationary, then the site may take control of its descriptor and manage it without computing any physics. If, on the other hand, the object is moving, or subject to forces that are not in equilibrium, then a new object node with a corresponding thread is instantiated on the host associated with the avatar to which the object was attached. This new object node takes responsibility for managing the object's movements and updating its descriptor until it is picked up by a new avatar or until it reaches an equilibrium state at some site. It should be noted that a host may keep managing an object, even if the associated avatar is at a distant location. This allows hosts to manage robotic-like entities that are controlled by user-defined software components, e.g. by associating streaming video to objects.

4.6 Decentralized 3D-Model Sharing

The architecture we have described so far assumes that nodes are able to retrieve the 3D-description of entities in order to compute collisions and display the entities after filtering them based on the information in their descriptors. In the following, we describe our mechanism for maintaining and retrieving these 3D-descriptions. As with the rest of the protocol, our goal is to distribute the cost of managing 3D-descriptions among Solipsis participants.

3D-descriptions are managed by the nodes responsible for the corresponding entities, and may be downloaded by any node that needs to display the object or perform collision detection. However, having all hosts download a given 3D-description from the node responsible for its entity would place an unnecessary load on the corresponding host. We address this issue, by having Solipsis nodes cache previously downloaded descriptions. This allows them to offer them for download to other Solipsis nodes running on the same or remote hosts.

All the nodes residing on a host, store their own 3D-descriptions as well as those of other visible objects in a shared repository (depicted in Figure 2) that is accessible by all the nodes running on the same host. Moreover, each node records in its entity's descriptor a list of the hosts that currently hold a copy of any of the files that constitute the associated 3D description. Whenever a node caches a 3D-description, it informs the node responsible for the entity, which then updates the corresponding descriptor to include a reference to the new copy and then propagates it with its subsequent heartbeat messages. Similarly, when a node detects that a host listed in the descriptor does not have the current copy of an entity's description, it sends a message to the corresponding node, which updates the descriptor accordingly.

4.7 Managing Disconnections

The architecture of Solipsis is designed to tolerate unexpected disconnections of hosts throughout its operation. First, site nodes always cache the 3D-descriptions, in addition to the descriptors, of neighboring site nodes. This allows the RayNet to manage the disconnection of a site node by automatically assigning its management to any of the neighboring site nodes. The disconnection of an avatar or object node, on the other hand, is treated by attaching the object or avatar to its current site and by leaving it in a stationary state until the corresponding user reconnects.

5 The navigator

A navigator must be able to create a huge metaverse and democratize its use. For this reason, we base our navigator on the Ogre 3D rendering engine. Its robustness, efficiency, upgradability and openness are, in fact, essential features for a durable system. Moreover, Ogre 3D can be easily embedded in web pages thanks to a Mozilla or ActiveX plugin. Conversely, web pages can be mapped on 3D models as interactive textures thanks to the Navi library

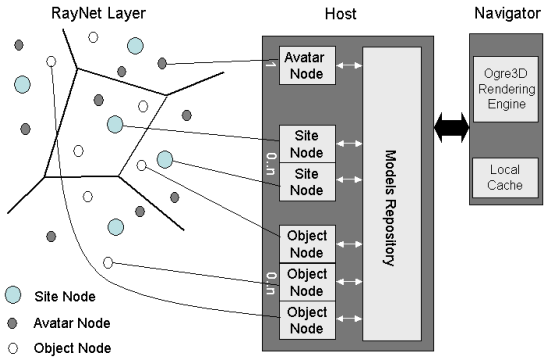


Figure 2: Nodes hosted on a proxy server to allow accesses to the metaverse on terminals with low resources such as mobiles.

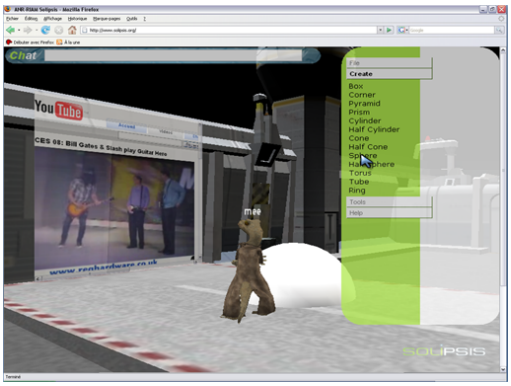


Figure 3: Web based navigator embedding modeling tools, and providing a mapping of web pages as interactive textures.

based on Gecko. Thus, the navigator allows a Web 2.0 navigation inside the Web 3D, as well as a Web 3D navigation inside the Web 2.0. Our view is, in fact, that Web 3D and Web 2.0 should integrate each other (see figure 3). Since the vitual universe belongs to all users, modeling tools are embedded within the navigator in order to create, modify or delete contents. At the moment, we focus on intuitive tools, but procedural, declarative, parametric and sketch-based modeling tools are obviously considered. Finally, to provide access to the virtual universe on mobile devices, nodes can be hosted by proxy servers, to reduce the amount of computation that has to be performed on terminals with scarce resources. Thus, hosts can compute collision detection, physics animation, data-exchange management, and viewpoint-based filtering, and then communicate with the navigator that only needs to render the virtual environment and interact with it (see figure 2).

6 Conclusions and Perspectives

We presented our vision for a decentralized architecture for virtual environments. Our solution is based on an n-dimensional Voronoi-based peer-to-peer network, called RayNet. This allows it to distribute communication and computational cost among the various nodes present in the virtual space. Our architecture enables the decentralization of complex tasks related to physic-realistic modeling. To achieve this, nodes preliminarily filter the set of avatars and objects that may collide with their own and then evaluate collisions and physics for a small set of entities. Our solution also supports dynamic objects that may be picked up and dropped by avatars or controlled by means of user-defined software. Moreover, it enables adaptive streaming of 3D models in a completely decentralized fashion while adapting streamed data to avatars' viewpoints. Also, the use of an n-dimensional overlay allows us to extend the metaverse to support social or semantic proximity between object or avatar nodes. Access to the metaverse is made possible by a navigator that may run as a stand-alone platform or embedded within a web page. The navigator exploits interactive texturing to enable the visualization of Web 2.0 components in the virtual world and it integrates tools for modeling new 3D contents. Moreover, it supports operation on resource-scarce devices such as mobile phones. Our project team is currently working on a fully open-source implementation of the Solipsis architecture, and we hope that the community of users will help us improve Solipsis and enable us to evaluate its effectiveness in a world-wide testbed.

Acknowledgements

First, the authors wish to thank all people who contributes to the Solipsis project as well as all pioneers of the decentralization of virtual environments whithout who this project would not be also advanced: M. Keller, M. Simon, M. Cavagna, M. Bouville and M. Beaumont. This work was supported in part by a French collaborative R&D project (ANR-RIAM) leaded by Orange Labs, funded by ANR and Media & Networks cluster of Brittany, involving IRISA, Rennes 2 University, Archivideo and Artefacto.

References

[1] http://earth.google.com/.

[2] http://www.secondlife.com/.

[3] O. Beaumont, A.-M. Kermarrec, and E. Rivire. Peer to peer multidimensional overlays: Approximating complex structures. In *OPODIS, 11th International conference on principles of distributed systems*, 2007.

[4] R. Cavagna, C. Bouville, and J. Royan. P2p network for very large virtual environment. In *VRST*, pages 269–276. ACM, 2006.

[5] S. Douglas, E. Tanin, and A. Harwood. Enabling massively multi-player online gaming applications on a p2p architecture. In *Proceedings of the IEEE International Conference on Information and Automation*, pages 7–12. IEE, 2005.

[6] E. Frécon and M. Stenius. Dive - a scalable network architecture for distributed virtual environments. *Distributed Systems Engineering Journal (Special issue on Distributed Virtual Environments)*, 5, 1998.

[7] T. A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *Symposium on Interactive 3D Graphics*, pages 85–92, 209, 1995.

[8] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH '93*, pages 247–254, New York, NY, USA, 1993. ACM.

[9] P. Gioia, O. Aubault, and C. Bouville. Real-time reconstruction of wavelet-encoded meshes for view-dependent transmission and visualization. *IEEE Trans. Circuits Syst. Video Techn.*, 14(7):1009–1020, 2004.

[10] H. Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[11] S.-Y. Hu and G.-M. Liao. Scalable peer-to-peer networked virtual environment. In *NetGames '04*, pages 129–133. ACM, 2004.

[12] J. Keller and G. Simon. Solipsis: A massively multi-participant virtual world. In *Int. Conf. Parallel and Distributed Techniques and Applications*, pages 262–268, 2003.

[13] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[14] M. Pesce, P. Kennard, and A. Parisi. Cyberspace. In *First International Conference on WWW*, 1994.

[15] C. C. Tanner, C. J. Migdal, and M. T. Jones. The clipmap: a virtual mipmap. In *SIGGRAPH '98*, pages 151–158. ACM, 1998.

[16] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. *SIGGRAPH Comput. Graph.*, 25(4):61–70, 1991.

The HyperVerse - Concepts for a Federated and Torrent-Based "3D Web"

Jean Botev, Alexander Höhfeld,
Hermann Schloss, Ingo Scholtes
Systemsoftware and Distributed Systems
Computer Science Department
University of Trier
Email: {botev, hoehfeld, schloss, scholtes}@syssoft.uni-trier.de

Markus Esch
Faculté des Sciences, de la Technologie
et de la Communication
University of Luxemburg
Email: markus.esch@uni.lu

***Abstract*— The vision of a "3D Web" as a combination of massive online virtual environments and today's WWW currently attracts a lot of attention. While it provides a multitude of opportunities, the realization of this vision on a global scale poses severe technical challenges. This work-in-progress paper intends to point out some of the major challenges and highlights key concepts of an infrastructure that is being developed in order to meet them. Among these concepts, special emphasis is put on the usage of a two-tier Peer-to-Peer approach, the implementation of Torrent-based data distribution and the development of a graded consistency notion. The paper also briefly presents the current state of a prototype implementation that is being developed in order to validate these concepts and evaluate alternative approaches.**

I. Introduction

Although various forms of distributed virtual environments currently attract a lot of attention, most of today's representatives are proprietary worlds that are hosted in a centralized fashion. When considering the vision of something one might call the "3D Web", in section II we find that - in contrast to today's precursors - decentralized approaches are required in order to provide a distributed persistent virtual environment on a global scale. With the HyperVerse project we intend to investigate technologies that are appropriate to realize such a scenario.

A global scale "3D Web" offers a variety of obvious interesting opportunities, like immersive mass events and the facilitation of real-time interaction between users based on advances in human interface technologies. Another interesting scenario arises from the observation that mobile devices are becoming increasingly location-aware and network-capable while at the same time getting smaller and cheaper. Thus it seems reasonable that in the future more and more everyday objects will feature such capabilities as well as all kinds of sensor technology. Combining these developments with a "3D Web" would allow more and more real-world objects to possess a real-time virtual representation, giving users intuitive means to remotely access various kinds of information on their state. In this respect, the current surge of geo-referenced information accessible via virtual globes like "Google Earth"[1] or "Virtual Earth"[2] is a first glimpse of what is yet to come. In the course of this paper, a persistent virtual environment which provides these opportunities at a global scale will henceforth be called "HyperVerse".

While its opportunities sound alluring, it is obvious that the realization of a HyperVerse scenario poses severe technical challenges. In the context of this paper, we restrict ourselves to briefly mentioning some of the - according to our opinion - most important questions:

- How can scalability suitable for the provision of a global scale HyperVerse scenario be achieved?
- How can such a scalability be combined with interactivity, consistency and persistency?
- How can client resources be utilized in a way that unburdens core network resources?
- Which cross-layer aspects can be identified that are induced by HyperVerse-specific communication patterns?
- How can the client-side fan-in problem in densely populated regions be resolved?

The HyperVerse project aims at the creation of a federated infrastructure meeting these requirements. In section II we will present some of the key concepts the project relies on. Section III will give a brief description of the current state of a prototype implementation that is being developed in order to evaluate these concepts.

II. Concepts of a Federated HyperVerse Infrastructure

In order to retain the decentralized nature, scalability, independence and reliability of the WWW and the Internet in general, for the provision of a HyperVerse scenario it is not eligible to rely on centralized server farms that are controlled by a single instance. Thus we embrace Peer-to-Peer (P2P) technologies in order to support the targeted global scale. For a multitude of reasons we do not aim at a pure P2P approach but rather use a two-tier architecture consisting of a highly structured federated backbone and a loosely structured P2P client overlay. The main reason for this is the expected churn rates in the HyperVerse scenario. Since we envision

[1] http://earth.google.com

[2] http://www.microsoft.com/virtualearth

a lightweight client software that is used in a way that is similar to today's Web browsers, clients will most likely exhibit exceedingly high churn rates as well as heterogeneous capabilities. Accordingly the P2P topology used for clients needs to be highly churn resilient in the face of global-scale user numbers. At the same time HyperVerse scenarios require massive amounts of persistent data to be reliably and efficiently accessible. Unlike in today's Massive Multiplayer Online Games (MMOGs), this data is required to be totally dynamic and cannot be predistributed with clients. We tackle this problem by using a massive amount of public servers that are supposed to be comparably reliable and host data in a federated manner. Public servers resembling today's Web Servers, we do not require them to be under control of any centralized authority. For their provision we rely on the incentive of being able to publish information.

In accordance with [11], by explicitly distinguishing between these two classes of peer participants we exploit their different properties in order to provide better reliability, availability and scalability of the whole system. The comparative low churn rate of public servers can be utilized by using highly structured P2P overlay networks. This provides for an efficient and reliable data retrieval. For clients, less structured topologies seem to be appropriate (see figure 1). The BitTorrent protocol [5] has proven to be valuable for the scalable distribution of huge files and is extremely resilient against churn [2]. In particular this resilience does not depend on the number of peers. The following section will provide more details on the application of similar approaches to distributed virtual environments.

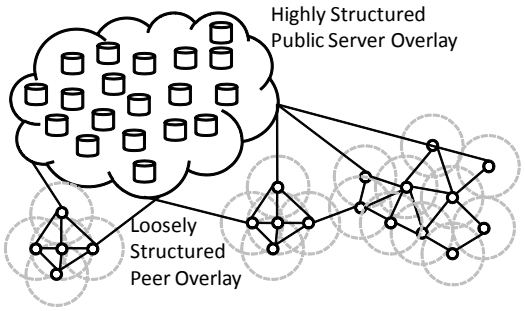


Fig. 1. Two Tier HyperVerse Infrastructure

A. Torrent Based Data Distribution

In order to be able to describe the Torrent-based distribution of information, we give a short description of the dynamic space-based interest management model implemented in the current HyperVerse prototype in so far as it affects concepts described in the following paragraphs. For the same reason, we also give a short description of the caching scheme that has been implemented in order to exploit data locality inherent to virtual worlds. Throughout the paper we refer to the terms object and terrain data as mesh, texture and meta information that are associated with dynamic 3D objects as well as comparably static world terrain.

a) Interest Management: Given an avatar's 3D position p in the virtual world, we differentiate between its Field of View (FoV) and Area of Interest (AoI). Assuming a maximum view distance d, an avatar's FoV is described by a sphere with radius d around p. The AoI is another sphere with radius $d + \Delta$ $(\Delta \geq 0)$ around p. Initially all objects and terrain data within the AoI sphere around p are retrieved. In order to mitigate the effects of retrieval latency, we use another sphere with radius $d + \Lambda$ $(\Lambda < \Delta)$ around p. The user can move within a distance Λ around p (see e.g. position p' in figure 2) without requiring further object retrieval. Whenever the avatar has moved more than Λ away from the position p, around which the AoI has been retrieved (see e.g. position p'' in figure 2) - a new AoI centered around the current position will be set. At this point, information on all objects and terrain within the new AoI need to be retrieved, more precisely only on those that haven't been in the AoI before. The introduction of the threshold Λ allows for more time for requesting these data since the users FoV is still $\Delta - \Lambda$ away from regions for which no information has been prefetched. Accordingly, the choice of the parameters Λ and Δ influences retrieval frequency, the amount of data present within the AoI as well as the retrieval latency that can be tolerated without having visual effects. Both parameters can be chosen by clients according to their individual capabilities.

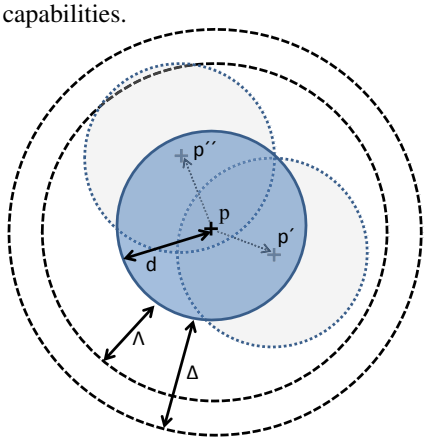


Fig. 2. A 2D projection of Area of Interest (AoI) and Field of View (FoV)

b) Data Locality in Virtual Environments: Looking at the pattern of access to object and terrain data induced by a supposed primary continuous movement through virtual worlds, one recognizes both temporal and spatial locality of reference with respect to the world's geography. Due to the avatar's movement, there is a higher probability for objects near the avatar's FoV to be accessed in the future - a fact that is allowed for by prefetching data from within the AoI as described above. One aspect of temporal locality in virtual worlds refers to the fact that recently accessed objects remain in the users FoV for some time and therefore will be accessed repeatedly. Another aspect is caused by frequent visits to the user's favorite venues. A certain user exhibits e.g. a higher probability of frequently accessing information residing in distinct areas within the virtual world. Even in today's WWW

one recognizes that most users repeatedly check an individual working set of favorite information resources, be it subscribed feeds, news portals, Blogs or community Websites. In today's WWW, in order to save redundant transmission, these locality aspects are allowed for by caches at various stages like user agent caches or caching proxy agents. In distributed virtual environments, data locality can be exploited by applying caching techniques as well. The rendering process involved in clients implicitly requires objects and terrain data to be cached locally at least as long as these data concern objects in the user's FoV. While such a cache can be based on a simple LRU strategy, the further exploitation of locality and optimization of the cache hit rate requires more advanced concepts. The current prototype HyperVerse browser (being described in section III) uses a multi-tier caching strategy in which object information within the FoV are held in graphics memory. This is backed up by a fixed-size in-memory cache that uses a combined LRU, geographical distance and object size metric as a replacement heuristic. It is especially important to consider the in-memory size of cached objects since it might involve huge variations and discarding big objects bears a greater potential for wasting network resources: Wrongly ejecting e.g. a few Kilobyte-sized object is doubtlessly less troublesome than spuriously discarding one being several Megabytes in size. The in-memory cache is backed up by a permanent storage cache file with configurable size. The maximum size of this cache can most likely be amply chosen in order to optimize performance and unburden network resources.

Caching is complicated by the fact that objects must be allowed to dynamically change at any time. This may occur following a user interaction with an object or abruptly in a scenario of augmented virtual real-world object representations that actively push state updates triggered e.g. by sensors. Due to the usage of caching, at a given time several copies of data may reside in the caches of different HyperVerse browsers. In order to keep these cached data coherent, a Publish/Subscribe paradigm maintaining subscriptions to objects residing in a client's cache is used. Dynamic and asynchronous changes of objects are actively pushed to subscribers in order to maintain cache coherency. Depending on the cache-tier, different update strategies are used. Updates of objects that reside in graphics or system memory (i.e. especially those in the FoV) are pushed to subscribers instantly. For objects residing in the disk storage cache file, the first update of an object sets a dirty-bit and the subscription is canceled. Dirty-marked objects are actively refreshed by clients as soon as they enter the AoI, preventing needless communication on objects that cannot instantaneously enter the FoV. Rather than requiring information sources to send unicast messages to all subscribers or relying on multicast facilities of lower protocol layers, the Torrent-like scheme that will be described in the next section can be used for a scalable dissemination of update messages.

c) Application of Torrent Principles: The number of users residing in a certain region of the virtual world and thus requesting the same object and terrain data is hard to predict and possibly massive. Thus it is clear that their distribution as well as the maintenance of subscriptions poses scalability problems. We particularly address a future scenario in which clients at the network's edges have high uplink bandwidths. Due to the multi-tier caching scheme in combination with the Publish/Subscribe pattern used, objects in memory as well as unmarked objects in the disk cache are known to be up-to-date. Furthermore, within highly populated regions this information is available redundantly. In order to utilize these resources we apply a distribution scheme similar to the BitTorrent protocol. The basic idea is that each HyperVerse client makes accessible cached information in a Torrent-like manner, i.e. data are split into individually addressable pieces. When a user requests data of a certain region, a number of other clients can be used to concurrently transfer pieces of data that are already present in their caches. For this the distributed backbone consisting of public servers (being described in section II-B) keeps track of the clients' AoI. According to the interest management model described above and by choosing an appropriate value for Λ and using an adequate federation scheme for the backbone service, the update frequency in individual servers can be kept manageable.

Clients use the backbone service in order to retrieve a (size-constrained) subset of nearby peers along with their AoIs. Knowing that peers at least contain an up-to-date version of objects within their AoI, a client uses this information in order to compute the coverage of its own AoI by peer AoIs. In a first step peers are used to retrieve meta-information on all objects that reside in areas that are covered by peers' AoIs. From this information the redundancy degree of objects within different portions of the covered area is computed based on the peers' AoI. Pieces of objects are then retrieved concurrently from all peers whose AoI contains the object's position. AoIs in low-density regions may contain portions uncovered by peers. Objects from within such portions will be retrieved from the backbone service which at the same time serves as initial seed for the Torrent-based distribution of data. An optimum peer selection strategy is still being investigated. In the current version random peers are used although ideally the peer subset should be chosen in a way that optimizes AoI coverage as well as object redundancy. Without going into further detail, a Publish/Subscribe extension to this Torrent-based scheme can be realized by clients propagating object piece updates to peers that have recently requested a given object. Since objects consist of many pieces, the probability of not receiving any piece update is comparably low and can only occur if all peers of which the object pieces have been received are offline. If at least some piece updates have been received, any remaining pieces can be actively requested by updating the peer set after a certain threshold.

B. Federated Backbone Service

As motivated earlier in this section, a two-tier P2P approach has been chosen in order to allow for lightweight clients and handle varying churn rates. Public HyperVerse servers can be thought of as a kind of federated "3D Web Servers". Objects can be published by adding a new public server to the

backbone federation or by uploading them to existing servers - their providers possibly demanding a fee for this service. By this means, public server providers are incentivized in a way that is similar to today's WWW and abides existing business models. The responsibilities of public servers consist in tracking client AoIs and thus connecting nearby peers for a Torrent-based distribution and a P2P exchange of movement information. Public servers also serve as initial seeds for all objects they contain. Initial seeds of world-specific terrain data are redundantly and equally distributed among all servers.

It is clear that the federation scheme underlying the public server backbone needs to be organized in a way that balances load between them. Apart from this it must provide simple facilities to perform range queries in order to efficiently retrieve the peer sets of client AoIs. Although to date it is not yet clear which of them will be actually chosen, several promising candidate technologies have been identified. Among them the *P-Grid* [1] system represents an efficient structured overlay network based on the concept of a distributed trie. It achieves highly efficient lookup operations and guarantees load-balancing of storage and query load. By preserving key order it furthermore supports range queries. Another interesting fact is that approaches relying on a space-based federation of public servers can capitalize on research stemming from the field of mobile ad hoc networks. One federation scheme we are currently testing for applicability is e.g. loosely based on the distributed and scalable GRID location service [13]. Moreover we investigate in how far techniques and methods of swarm intelligence are applicable in order to realize a decentralized and self-organized network of public servers. This includes the evaluation whether decentralized control algorithms for public servers can be combined with efficient routing protocols in order to support the requirements mentioned above.

C. Individual Dynamic Instantiation

The Torrent-based distribution of 3D data described above is suitable to mitigate the problem of distributing the same object or world data to a massive number of clients resulting from high avatar densities in confined virtual regions. Under such circumstances, the problem of mutual visibility and thus exchange of motion information remains. Apart from exhibiting a fan-in problem at clients, it also hampers usability since from a user's perspective it is not reasonable to visualize an unlimited number of nearby avatars. This problem already occurs in today's MMOGs, so the countermeasure of so-called instantiated sessions has been developed in this domain. Using this technique, several separated and dynamically instantiated "copies" of the same region are created, each copy allowing a certain maximum number of users. This not only improves usability but also scalability since instances can be handled separately on machines in the provider's server farm. Within the HyperVerse, this is not a desirable solution since it precludes interaction between users residing in different instantiated sessions. Accordingly, we investigate how a dynamic and individual instantiation based on an avatar's *social bias* can be provided. In this context, the social bias is auto-generated information on an avatar's position in the social network based on its history of interaction with other avatars. Using this information, rather than separating instances, it is necessary to provide a dynamic *individual instance* for each user in a scalable way whenever avatar density in a certain region exceeds a certain threshold. In each individual instance, only relevant avatars shall be visualized, thus separating out and/or coarsely summarizing dispensable information. Interaction across these instances is possible since each avatar is present in individual instances of all user's that share a similar social bias. Non-overlapping transitive relations are not approached at this stage though.

Extending 3D space by additional dimensions which allow an avatar's social bias to be encoded as position in a multidimensional space seems alluring. Using an appropriate encoding, a simple Euclidean distance between these positions could be interpreted as "social connectedness". The interest management scheme depicted above could then be extended to use higher dimensional FoV and AoI sphere abstractions. Unfortunately it is easy to see that social relationships between an unlimited number of users cannot be represented by positions in any finite-dimensional metric space, since it would require encoding a potentially unlimited amount of information in a fixed-size position vector. The usage of non-metric topological spaces and an appropriate definition of closeness appears to hold the key for a scalable implementation but requires further research.

D. Multilevel Consistency

The consistency of distributed data and state is an important issue which needs to be addressed at the design stage of any distributed environment. We classify consistency problems into *data consistency problems* arising from data replication and *update propagation problems* resulting from the distribution of global state representations among adjacent users.

Consistency in centralized environments is comparably easy to achieve because only a single copy of the data or state representation exists on a central instance (server). As motivated in section II, there are several arguments which suggest that centralized approaches are not suitable for a global scale virtual environment. Accordingly, we expect such environments to be based on decentralized federated infrastructures. For the sake of scalability it is hardly possible to provide strict consistency for the whole virtual world targeting global scale user numbers. Therefore we introduce the term *world partition* representing a designated virtual region along with all users within. Hereby the virtual world can be arbitrarily subdivided in several world partitions which are not necessarily disjoint. We define the *weight* of a world partition as a function $w : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$ of the region's area and number of users within. In order to guarantee consistency within these world partitions, we use a multilevel consistency model. Besides the given application scenario, the weight of a world partition is decisive for the guaranteed consistency level. The maximum consistency degree that can be provided is reciprocally proportional to the weight of the a world partition. That is, in

a "lightweight" world partition we ensure higher consistency levels than in a "heavier" one. We are confident that this relaxation of the consistency notion is expedient for massive virtual environments since e.g. small consistency variations of avatars in a user's FoV become less perceptible and thus crucial the more avatars are visible.

III. A Prototype HyperVerse Implementation

In this section we will briefly describe the current state of a HyperVerse prototype that has been implemented based on the aforementioned concepts. It consists of a lightweight 3D browser which is based on a network-aware engine capable of retrieving, caching and rendering 3D terrain and object data from a Web Service based backbone service. The HyperVerse browser as well as the underlying engine serve as a prototype reference implementation which we use to evaluate different approaches. In order to perform "simulations" using the actual implementation while at the same time saving resources, the actual 3D visualization can be detached from the browser. In this mode, the client is remotely controllable via a Web Service. This can be used in order to create a number of client instances on a test cluster, manage them and retrieve measurands via a centralized controller based e.g. on real-world mobility models that are available from the MANET research community [12].

a) HyperVerse Browser: The HyperVerse browser is a thin-client interactive 3D application combining concepts known from today's Web browsers (like e.g. Bookmarks) and virtual globes (like e.g. the use of geo-referenced objects, geography-based navigation, satellite imagery and topographic height data) with an avatar-based interaction and navigation known from MMOGs. Based on widespread and cheap game console hardware - namely the Bluetooth-connected Nintendo Wii controller, head-tracking facilities have been implemented. By this means, an immersive 3D experience and fine-grained interaction between avatars can be provided. Figure 3 shows the user interface of the HyperVerse browser. Although we currently use a virtual world that is based on real world topographic and imagery data, the browser does not depend on this. The backbone service providing appropriate data, any other virtual world that uses either a spherical or cartesian coordinate system can be used. Communication between the browser and the backbone service as well as other clients is based on Web Services and defined by descriptive interfaces, thus making the used protocol traceable and furthering interoperability. The HyperVerse browser is available at[3].

b) HyperVerse Engine: The network-aware HyperVerse engine is based on Microsoft DirectX and the .NET framework and is capable of rendering XML-based Collada[4] models. Being a powerful and wide-spread intermediate format, Collada is most prominently used as a 3D inlet of the Google Earth KMZ4 format with a large pool of available models. Furthermore the engine supports the rendering of SRTM[5] terrain, Blue Marble[6] as well as Landsat[7] topographic imagery data. It contains basic dead reckoning technologies for real-time rendering of object and avatar movement in the face of network delay, basic geodetic mathematics utilities as well as an implementation of the space-based interest management model and the caching scheme described in section II-A.

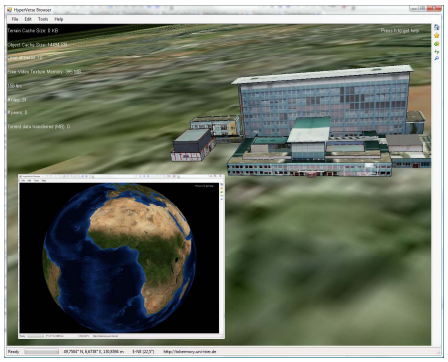


Fig. 3. HyperVerse Browser Application

c) HyperVerse Backbone Service: Since candidate technologies for the federated backbone service are currently under investigation, for the time being only a simple Web Service based Tracker and initial seed service has been implemented. It serves real-world topographic and imagery data in different levels of detail as well as object data. According to the Torrent-scheme described in section II it also keeps track of client AoIs and is used by the HyperVerse browser in order to retrieve peers whose avatar's AoIs subtend that of the local avatar.

IV. Related Work

Similar to our notion, [14] claims that peer-to-peer architectures are suitable for supporting distributed virtual environments (DVE). In order to investigate this type of architectures and to simulate large-scale DVEs in an efficient way, the authors propose a distributed simulation platform that provides appropriate performance metrics and contains all the elements involved in real DVE simulations. Techniques resembling our Torrent-based data distribution have been introduced to 3D virtual environments by [8]. The authors propose the P2P *3D Streaming* framework *FLoD*. It allows clients within a virtual environment to retrieve relevant data from nearby clients while minimizing perceived transmission delay. An important contribution of FloD is its support for progressive meshes and textures by defining so-called base and refinement pieces. By prioritizing base pieces, the rendering process can start before all object pieces have been received. The overlay topology of FloD is based on the Voronoi-based *VON* [7] scheme. Evaluations of FloD have shown that P2P 3D streaming is much more scalable than client-server approaches. Solipsis [9] is another massively shared P2P virtual reality system that addresses global-scale user numbers. In order to tackle the scalability problem and heterogeneous access device

[3]http://hyperverse.informatik.uni-trier.de
[4]http://collada.org
[5]http://www2.jpl.nasa.gov/srtm
[6]http://earthobservatory.nasa.gov/Newsroom/BlueMarble/
[7]http://landsat.gsfc.nasa.gov/

capabilities, it provides adjustable data flows based on varying awareness radii.

Similar to our notion of providing consistency within virtual environments, Myriad [15] permits transient inconsistency, thus relaxing resource requirements in collaborative virtual environments. [10] considers consistency aspects in distributed virtual environments and introduces an approach based on global timestamps. In [6] multilevel consistency addressing the replication techniques of world data is introduced. *CyberWalk* [4] is a DVE using an on-demand transmission approach for the distribution of the virtual environment to the clients. Similar to FloD, CyberWalk uses a multiresolution caching mechanism that reduces model transmission and rendering times by employing progressive models. Network delay is mitigated by providing a caching and prefetching mechanism. Moreover, it allows a client to continue to operate, at least partially, when a network connection is unavailable.

The paper [3] examines an architecture that supports persistent game state in public server based multiplayer games. In opposite to our notion, public servers do not provide a single virtual environment but are separated in the sense that they provide local per-server virtual worlds for a very limited number of users. All servers share certain persistent game items and character capabilities which are contributed and controlled by the game publisher in a centralized fashion.

V. Conclusion and Future Work

In the course of this paper we presented key concepts of the HyperVerse project in which we investigate how global-scale persistent virtual environments can be provided. In order to guarantee persistent information without putting scalability at stake, we have chosen to apply a two-tier hybrid P2P approach by combining a WWW-like federated public server backbone with a scalable Torrent-based data distribution. While our approach resembles the one described in [8], the main difference is the abdication of highly structured client overlay topologies. Since we explicitly address a scenario with Browser-like clients, we assume churn rates to be much higher than that of any prevalent Peer-to-Peer applications. In order to mitigate this problem, we use a highly structured federated backbone acting as Torrent Tracker and interconnecting clients to a loosely structured and highly churn resilient overlay. Another noticeable difference is that due to the propagation of peers' AoIs information, clients are able to locally decide which peers contain required data pieces without having to actively send requests to nearby clients.

The main contribution of introducing a Torrent-based approach to massive virtual environments is the mitigation of Flash Crowds - a spontaneous surge of interest in a certain region of the HyperVerse. The BitTorrent protocol has proven to successfully address this problem in the context of distributing large files in today's WWW. By means of considering virtual geography and locality aspects that are inherent to virtual worlds, we argued that massive virtual environments can benefit from a Torrent-based data distribution much in the same way. In order to overcome the Flash Crowd related client-side fan-in problem, we intend to use a dynamic instantiation approach that is based on the relationship between users and does not preclude interaction across instances.

While we cannot yet provide a definite answer to the question which federation scheme shall be used for the massively distributed public server backbone service presented in section II-B, we identified some candidate technologies as well as general directions of research. As a next step, some of these technologies need to be evaluated. Being a promising approach, a P-Grid-based backbone service is being implemented and will be available soon for further evaluations. In the course of this paper we have also given a brief overview of our approach to consistency handling which is characterized by a twofold relaxation of the consistency notion. We argue that this relaxation is suitable for massive virtual environments in the sense that it allows a minimization of sensible effects while being conducive to scalability.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Sixth International Conference on Cooperative Information Systems (CoopIS 2001)*, 2172:179–194, 2001.

[2] A. Al-Hamra, A. Legout, and C. Barakat. Understanding the properties of the bittorrent overlay. Technical report, INRIA Sophia Antipolis, France, 2007.

[3] C. Chambers, W. chang Feng, and W. chi Feng. Towards public server mmos. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames '06)*, page 3, New York, USA, 2006.

[4] J. H. P. Chim, R. W. H. Lau, H. V. Leong, and A. Si. Cyberwalk: a web-based distributed virtual walkthrough environment. *IEEE Transactions on Multimedia*, 5(4):503–515, 2003.

[5] B. Cohen. Incentives build robustness in bittorrent, 2003. citeseer.ist.psu.edu/cohen03incentives.html.

[6] J. C. de Oliveira. Issues in large scale collaborative virtual environments. http://citeseer.ist.psu.edu/oliveira01issues.html.

[7] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. Von: a scalable peer-to-peer network for virtual environments. *Network, IEEE*, 20(4):22–31, July-Aug. 2006.

[8] S.-Y. Hu, T.-H. Huang, S.-C. Chang, W.-L. Sung, J.-R. Jiang, and B.-Y. Chen. Flod: A framework for peer-to-peer 3d streaming. In *The 27th Conference on Computer Communications (IEEE INFOCOM '08)*, 2008.

[9] J. Keller and G. Simon. Solipsis: A massively multi-participant virtual world. In *PDPTA*, pages 262–268, 2003.

[10] S.-J. Kim, F. Kuester, and K. H. Kim. Towards enhanced data consistency in distributed virtual environments. volume 5006, pages 436–444. SPIE, 2003.

[11] J. Kubiatowicz. Extracting guarantees from chaos. *Commun. ACM*, 46(2):33–38, 2003.

[12] J.-Y. Le Boudec, S. PalChaudhuri, and M. Vojnovic. Perfect simulations for random trip mobility models. *Proceedings of the 38th Annual Simulation Symposium 2005*.

[13] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom '00)*, pages 120–130, New York, USA, 2000.

[14] S. Rueda, P. Morillo, and J. M. Orduna. A Peer-To-Peer Platform for Simulating Distributed Virtual Environments. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems (ICPADS '07)*, 2007.

[15] B. Schaeffer, P. Brinkmann, G. Francis, C. Goudeseune, J. Crowell, and H. Kaczmarski. Myriad: scalable vr via peer-to-peer connectivity, pc clustering, and transient inconsistency. In *Proceedings of the ACM symposium on Virtual reality software and technology (VRST '05)*, pages 68–77, New York, NY, USA, 2005. ACM.

Towards an Authentication Service for Peer-to-Peer based Massively Multiuser Virtual Environments

Arno Wacker*, Gregor Schiele†, Sebastian Schuster*, and Torben Weis*

*University of Duisburg-Essen
Duisburg, Germany
{arno.wacker|sebastian.schuster|torben.weis}@uni-due.de

†University of Mannheim
Mannheim, Germany
gregor.schiele@uni-mannheim.de

***Abstract*— In this paper we propose a distributed authentication service for peer-to-peer (P2P) based massively multiuser virtual environments. Such a service is necessary to provide security, e.g. preventing a user's account being stolen or the user being impersonated. We describe two variants of our authentication service. The first uses certificates and a central certification authority to ensure the validity of user-generated public keys. These keys are then used to sign messages sent by the users' peers. The second variant distributes the users' public keys in the P2P network and uses quorums to verify them.**

I. Introduction

Massively multiuser virtual environments (MMVEs) allow a large number of users to participate in a shared virtual environment via the Internet. Security is a crucial requirement for such systems, to guarantee their smooth operation [1]. Otherwise, users may e.g. pose as somebody else or steal other users' data. This is especially true since the participants of a large scale system typically do not know each other and therefore cannot trust each other.

To provide security, we first must guarantee message authenticity, i.e. a user must be able to identify the sender of a message reliably. Building upon this, other goals can be realized, e.g. data confidentiality or integrity.

In this paper we discuss how to provide message authenticity in MMVEs. We restrict our discussion to peer-to-peer (P2P) based systems. In such systems, the virtual environment is provided and managed by the participating users' computers themselves, instead of a centralized server, as in a client/server-based MMVE.

Our approach is based on certificates and signed messages. We depict two variants. First we discuss a straight forward approach using a Certification Authority (CA). This approach builds on well known techniques. Secondly, we propose a novel approach that is based on replicating public keys in the P2P network. This approach is still work in progress. However, in our opinion it offers a lot of potential for future research activities.

The paper is structured as follows. First, we present our system model and introduce two types of MMVEs, which need to be distinguished when developing an authentication service. After that we discuss requirements for our authentication service. We then provide an overview of related work and present our approach. Finally, we offer a short conclusion and some thoughts about future work.

II. System Model

We define an MMVE as a persistent virtual environment that is shared by a large number of users worldwide. The number of users is a priori undetermined and may change dynamically. A P2P-based MMVE is a special kind of MMVE that is executed cooperatively by all users' computing devices, called peers. Each device is connected to a common communication network, e.g., the Internet. Using this network, the peers form a connected overlay network, the so-called P2P (overlay) network. To participate in the MMVE, a user activates his device and starts the preinstalled MMVE software. The software logs into the P2P network and the user can start operating in the MMVE. Each user is represented in the MMVE by a special character, called his avatar. Conceptually, our approach is able to handle multiple avatars per user. However, in this paper we restrict each user to one avatar and use both terms interchangeably. This makes the description of our approach easier to understand. To operate in the MMVE, the user directs his avatar to perform actions for him, e.g. moving or interacting with other users' avatars. Each activity is distributed to other peers and processed by them, e.g. by updating the state of the MMVE. After the user is done, he stops the software and deactivates the device.

Our approach assumes the existence of an underlying structured P2P network that can be used to store and retrieve data, e.g. a distributed hash table (DHT). The P2P network must support four operations, discussed below. The operation `storeObject(pos, data)` stores a data item at a given (logical) position in the P2P network. Using the position, the P2P network determines a set of peers and stores the data item at them. The specific algorithm to do so depends on the used P2P network. Different positions are mapped to different peer sets, if possible, to distribute the data evenly. Due to peer fluctuations, the peers responsible for a given position may change over time. We assume that the P2P network detects this and relocates the involved data items automatically. The reason for using a replicated storage is to ensure that data items are persistent. Otherwise, when a peer is lost unexpectedly, data may be lost. We assume that the

P2P network contains consistency between all replicates of a given data item. With `retrieveObject(pos)` we can retrieve a data item from the P2P network. The parameter `pos` specifies the logical position of this item in the P2P network. The network automatically resolves the position to a set of peers and retrieves the data item from them. To delete a data item stored at a given logical position, the operation `deleteObject(pos)` can be used. It determines all peers saving the data item and instructs them to delete it. These three operations realize a simple distributed data space. In addition, the P2P network allows peers to send data messages to each other, using the `send(peer, data)` operation. It sends a given data item reliably to the specified peer. In the peers@play project we are developing a P2P network that fulfills all properties discussed above. This network can be used to realize our approach. Its exact implementation is beyond the scope of this paper.

III. Open vs. Moderated MMVEs

We distinguish two types of MMVEs, moderated and open ones. A moderated MMVE is operated by a specific system operator. The operator is responsible for managing the MMVE and its participants. As an example, the operator decides which user may participate in the MMVE. To do so, users are normally expected to register with the operator before entering the MMVE. The operator may also decide to remove a user from the MMVE, e.g. in case of the user violating the license agreement. Clearly, all users have to trust the operator. A typical example for this type of MMVE is a commercial system, e.g. an online game.

An open MMVE works without a specific operator, i.e. the system is operated cooperatively by its participants. There is no single entity responsible for the system or trusted by all users. Access to the system is free for all. A typical example for such an MMVE is a non-commercial online social network.

IV. Requirements

In the following section we analyze the requirements that an authentication service (AS) for P2P-based MMVEs must fulfill.

1) Decentralized operation: The first requirement for authentication in P2P-based MMVEs is decentralization. The AS must allow users to login into the MMVE and to operate within it without accessing a centralized server. This server could become a bottleneck for the system performance and a single point of failure. In addition, someone would have to operate the server, making this approach unsuitable for open MMVEs.

2) Privacy: With respect to privacy, the authentication must make sure that other users of the MMVE are not able to derive knowledge about the identity of other users. Note that an MMVE developer/operator may decide to make the identity of its users available to others. This is an MMVE-specific design decision and does not influence the AS requirements.

3) Availability: Clearly, authentication is a crucial service for most MMVE. If the AS is not available, no user can log into the MMVE. This may lead to users getting frustrated with the MMVE and eventually abandoning it and must therefore be avoided.

V. Related Work

Security has been widely recognized as a major concern in MMVEs. However, most researchers concentrate on designing cheat-proof algorithms, e.g. [2], [3]. All of these approaches make heavy use of cryptographic functions. They implicitly assume a public key infrastructure to be present delivering means to authenticate the a priori unknown communication partners. For example Rieche et al. [4] explicitly state that they intend to use an existing server for accounting in their P2P MMVE infrastructure.

Fully distributed key management infrastructures can be found in the area of P2P and ad-hoc networks. Threshold cryptography [5] is a way to realize such a distributed CA. Out of a group of n nodes, at least k are necessary to authenticate other nodes. It shares some similarities with our approach basing decisions on trustworthiness of multiple nodes. However, there is no guarantee that k out of the n nodes are present at a specific time. Furthermore, it is not clear how to choose n and k and how to form subgroups. Choosing n as all nodes of the network to fully distribute the CA is unfeasible, due to the huge network size and its dynamic change at runtime.

Another approach to distribute authentication in an ad-hoc network is forming a web of trust [6], similar to PGP. Here, no single trusted authority exists. Instead, everybody can issue certificates showing that he believes in the identity of someone else. Thus, trust chains can be built to verify identities. Metrics like counting trust chains can be used to further strengthen the belief in one's identity. However there is no guarantees that a trust chain exists at all. At the same time, the verification of a nodes identity is mixed with its ability to authenticate other nodes. Consequently, misbehaving nodes can hurt the whole system by issuing lots of false certificates.

Finally, multiple approaches in P2P systems are based on reputation measuring the trustworthiness of nodes e.g. [7]. Reputation of a node changes according to other nodes' experiences with that node. This is a distributed voting mechanism, and could be used to build up trusted peer-based CAs. However, it relies on authentication of a node's identity and binding its identity to its reputation in the first place.

VI. Authentication in Moderated MMVEs

After discussing our requirements, we now present our approach for authentication in MMVEs. We start with our first variant, which is tailored towards moderated MMVEs. In Section VII we propose another variant for open MMVEs.

Our first approach is based on three parts: (1) peers signing their messages, (2) certificates to validate signatures, and (3) the MMVE operator issuing certificates to users at registration time.

Before any user is able to access the MMVE, the MMVE operator sets up a CA that is accessible for all potential users, e.g. using the WWW. A CA is a trusted system service for generating certificates. The operator can use any existing CA software for this. He creates an asymmetric key pair for the CA, i.e. a public (K^+_{CA}) and a private key (K^-_{CA}). The public key is then embedded into the MMVE software and distributed with it. Thus, the CA's public key is well known to every peer in the system. When a user wants to register for the MMVE, he contacts the operator's CA and registers himself for the MMVE. During this registration, the user creates an MMVE identifier (ID_U) and generates an asymmetric key pair (K^+_U, K^-_U) for it. The identifier can be any arbitrary and sufficiently long string or number, which cannot be correlated with the real identity of the user. One way to create it is to hash the email address of the user with a secure hash function, i.e. $ID_U = Hash(user@domain.com)$. Other users in the P2P network will only see ID_U and never the real identity of the user. Hence, with this step the privacy requirement is fulfilled.

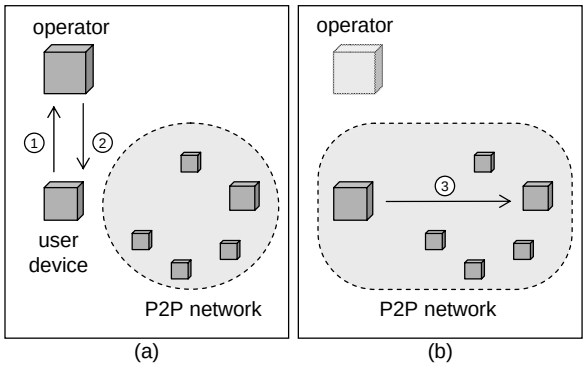


Fig. 1. Authentication in a moderated P2P-based MMVE

After the generation of these values the user finishes his registration by sending a signature request to the CA, i.e. message (1) in Fig. 1(a):

$$Message1 : U \rightarrow CA : \{ID_U, K^+_U\}_{K^+_{CA}} \quad (1)$$

Clearly, the system operator (i.e. his CA) can choose any appropriate means to identify the user. If the system operator needs to guarantee the identity of the user, a valid external certificate on the email address of the user could be used. When the user fulfills the registration requirements (e.g. payment was received) the operator issues a certificate for the new MMVE identity, i.e. message (2) in Fig. 1(a):

$$Message2 : CA \rightarrow U : \{ID_U, K^+_U, t_s, t_v\}_{sign(CA)} \quad (2)$$

where $\{m\}_{sign(X)} = m|\{Hash(m)\}_{K^-_X}$ for any message m. This signed message includes the MMVE identity (ID_U), the corresponding public key (K^+_U) and a period of validity given with the two time marks t_s and t_v. Clearly, the CA could include any additional data in the certificate – e.g. a serial number – if this is required by the MMVE.

After that, the user can log into the MMVE. To do so, no additional communication is needed. Instead, the user's peer signs all its messages with the user's private key, i.e. message (3) in Fig. 1(b):

$$Message3 : U \rightarrow peer : \{m\}_{sign(U)} \quad (3)$$

Upon receiving such a message, each peer checks if it knows the sender's certificate. Otherwise, it requests the certificate at the sending peer. Once the receiver knows the certificate, it validates the message signature using the sender's public key. If the signature is valid, the peer accepts the message. Otherwise the message is simply ignored, denying that peer the participation in the network.

Note that this approach resembles a classical certificate-based approach very closely. The main differences are that the MMVE operator is used to provide the CA instead of an external CA, e.g. VeriSign. In addition, each certificate is also a ticket, i.e. it allows the owner of the certificate to enter the network and participate in the MMVE. Our certificates have a restricted lifetime, similar to most other certification-based approaches. The duration of the certificate lifetime depends on the provider's payment model. If users have to pay to enter the MMVE, e.g. using a monthly fee, the certificate should be valid for the paid duration. After that, a new certificate must be issued. Clearly, this should be transparent for the user, i.e. the certification renewal must be done automatically.

In certain cases it may be necessary to remove a user from the MMVE while his certificate is still valid. This may be the case if the user violates the usage terms or if his account is stolen. To remove a user, his certificate is revoked. A possible approach for revocation is to let each peer check with the CA if the certificate is still valid. However, this would put additional load on the CA and would require it to be available to perform the check. Therefore we propose the usage of *revocation list messages*, containing the current revocation list. The system operator creates this list and signs it with his private key. Then the list is send to the P2P network e.g. via (authenticated) flooding. Each (non-compromised) peer needs to store the revocation list locally. To identify updates a simple version counter could be used. With this kind of revocation list we just need to ensure, that the list is never lost, i.e. it needs to be replicated consistently. This is done automatically by our assumed underlying P2P network (see Section II).

This approach fulfills the requirements given in Section IV. It is decentralized since peers can authenticate themselves against other peers at runtime without contacting a centralized system component. To do so, they only have to sign their messages. The CA is only needed at registration time to create a new certificate. If the CA fails, no new users can register for the MMVE. However, already registered users can keep logging into the MMVE and use it. As long as two peers are able to communicate with each other, they can authenticate against each other. Thus, the approach does also fulfill our third requirement, availability. Finally, privacy is provided by using a secure hash function to create the user's identifier ID_U. For authentication, only ID_U is send to other peers and only the MMVE provider can link ID_U to the user's identity.

VII. Authentication in Open MMVEs

Our approach for authentication in moderated MMVEs is based on the system operator providing a trusted CA to validate public keys. In an open MMVE, no system operator exists. Therefore, another approach is needed. An easy modification is to use an external globally available CA, e.g. VeriSign, to provide the needed certificates. In the future, many users may have such a certificate anyway, e.g. for electronic commerce. However, such CAs usually include the user's identity into the certificate. This violates our privacy requirement. In addition, all users would need a certificate issued by the external CA, resulting in addition costs.

We omit using a central CA and propose a different approach to validate public keys. The basic idea is to adapt the registration process such that instead of a certificate being created, the public key is stored at a number of different peers in the P2P network.

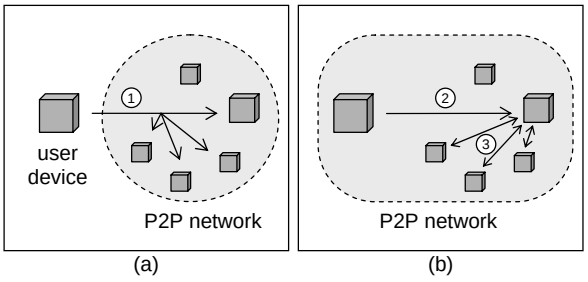


Fig. 2. Authentication in an open P2P-based MMVE

Similar to the registration in moderated MMVEs, a new user first has to create an MMVE identifier ID_U and generate an asymmetric key pair (K_U^+, K_U^-). The public key K_U^+ must be stored in a way such that (1) it cannot be tampered with and (2) all other peers can retrieve it when needed. We cannot assume that any single peer is secure. Therefore, we propose to store K_U^+ on multiple peers in the P2P network. When we later want to retrieve the public key, we use a majority voting mechanism to identify modified entries. We introduce the security level s to denote the number of manipulated entries that are tolerated by our approach. To tolerate s such manipulated entries, we need to store the data on at least $(2s+1)$ different peers.

An overview of this approach is given in Fig. 2. Initially, the new peer stores its user's public key at $(2s+1)$ different positions (see (1) in Fig. 2(a)). This registers the user in the MMVE. After that, the user can log into the MMVE and operate in it. Whenever his peer has to send a message to another peer, it signs the message with the user's private key (see (2) in Fig. 2(b)). To check the signature, the receiver contacts the P2P network and retrieves the corresponding public key from all $(2s+1)$ different positions in the network (see (3) in Fig. 2(b)).

In the following we describe our approach in more detail and provide some pseudo code for it.

A. Registration

The algorithm for registering a new user is given in Algorithm 1. It must be executed before the user logs into the MMVE for the first time.

Algorithm 1 Registering in open MMVEs

```
1: ID_U = Hash(user@domain.com);
2: (K_U^+, K_U^-) = new Key(); // Generate a new key-pair
3: for i = 1 to (2s + 1) do
4:     pos_i = Hash(ID_U|i); // Calculate DHT position
5:     storeObject(pos_i,(ID_U, K_U^+));
6: end for
```

Lines 1–2 show the creation of a new MMVE identity and the generation of a new asymmetric key pair. After that, the for-loop (see Line 3–6) stores the public key at $(2s+1)$ different positions in the P2P network. Each individual position is calculated by hashing the MMVE identity ID_U concatenated with a unique number (see Line 4). After that, we store the public key K_U^+ and the corresponding ID_U using the operation `storeObject` (Line 5).

Note that if the P2P network stores different positions pos_i on the same peer, the security of our approach decreases. By compromising this peer, an attacker can gain control over multiple copies of the public key. In our system model, we assumed the P2P network to distribute positions evenly on the available peers. Therefore, the probability for such a situation to occur decreases with the number of peers in the network.

B. Key Retrieval

When a peer sends a signed message to another peer , the receiver has to check the message signature. To do so, it first checks if it already knows the sender's public key. If not, the receiver executes Algorithm 2.

Algorithm 2 Retrieving a public key

```
1: // Retrieves the public key for a given ID
2: items[] = new array[2s + 1];
3: for i = 1 to (2s + 1) do
4:     pos_i = Hash(ID_U|i); // Calculate DHT position
5:     items[i] = retrieveObject(pos_i);
6: end for
7: item = majority(s,items);
8: return item.K_U^+;
```

Since we cannot rely on a single peer to provide the valid public key of a user, we have to collect the public key from all $(2s+1)$ positions and perform a majority voting. With the for-loop (Line 3–6) we retrieve each copy of the previously stored public key and store it in the local array $items[]$ (Line 5). In case there are different answers, a simple majority is used (Line 7), i.e. at least s keys need to be equal in order to return the public key (Line 8). If no majority of s can be achieved, we cannot decide which public key is correct. In this case, the received message is discarded.

C. Key Updates

In certain cases, a user might want to update his public key, e.g. because he suspects that it might be compromised. In addition, the MMVE might use relatively short keys to achieve higher encryption efficiency. In this case, the keys should be exchanged regularly. Finally, if the user decides that he will not use the MMVE anymore, he should be able to remove his registration from the P2P network. To update a registered key, Algorithm 3 can be used.

Algorithm 3 Updating a public key item

```
1: // Received a message m = {ID_U, K'^+_U}_sign(U)
2: // K'^+_U: new (updated) public key
3: // To be stored at position pos
4: // Note: {m}_sign(X) = m|{Hash(m)}_K^-_X
5: // Note: m.sig(X) = {Hash(m)}_K^-_X
6: if localStore[pos] != null) then
7:    kpub = localStore[pos].K^+_U;
8:    if (Hash(ID_U, K'^+_U) == {m.sig(U)}_kpub) then
9:       if K'^+_U != null then
10:         localStore[pos] = (ID_U, K'^+_U);
11:      else
12:         localStore[pos] = null;
13:      end if
14:   end if
15: end if
```

When the user wants to update his public key, he prepares an update message and sends it to all peers holding a copy of his public key. These peers are determined similarly to the original registration (see Algorithm 1). An update message contains the user's MMVE identifier ID_U and the new public key K_U^+. It is signed using the old public key. Upon receiving an update message, each peer first checks if it has an entry at the specified position (Line 6). If this is the case, the stored public key is used to validate the signature of the message (Line 8). If the signature is valid, the peer updates the entry (Line 10). Otherwise, the update is denied. To allow deleting entries, the peer checks if the provided new key equals null (Line 9). In that case, instead of updating the key, the peer removes the entry from its local storage (Line 12).

Note that while this algorithm is executed, peers trying to retrieve the public key may not be able to constitute a valid quorum, if some copies are compromised. This is resolved once the algorithm terminates for all copies.

D. Discussion

Our approach for authentication in open MMVEs fulfills all three requirements. It is completely decentralized, both at registration time and during the MMVE execution. Similar to our approach for moderated MMVEs, privacy is provided by using only hashed user identities. Hence, the identity of other users is kept hidden. Since the public key is stored in the P2P network, it is always accessible. Thus, the AS is always available, fulfilling our third and last requirement.

For our approach to work, a peer must be connected to at least $(2s + 1)$ other peers in the P2P network. Otherwise, during the first connection of a new user, the neighboring peers could cheat about its identity, i.e. mount a so called man-in-the-middle attack. One way to guarantee this, is to organize the P2P network such that it forms a $(2s+1)$-connected graph. In such a graph, there are always $(2s+1)$ paths between each peer, making attacks impossible if less than $(s+1)$ peers are compromised. We already applied this approach successfully in wireless sensor networks [8] and are planning to transfer these results to MMVEs.

VIII. Conclusion and Future Work

In this paper we proposed an AS for P2P-based MMVEs. We provided two variants. The first is tailored towards moderated MMVEs. It relies on the MMVE operator offering a CA to issue certificates to users of the MMVE. At runtime, messages are signed with the users' public keys and validated using their certificates. This approach is relatively straight forward and can be applied with little effort. However, it is not applicable to open MMVEs, since they do not have an operator. Our second variant is able to operate without an operator. Instead of a CA issuing certificates, public keys are stored redundantly in the P2P network and checked using quorums. This approach is able to tolerate up to s compromised copies of the stored public key. If an attacker is able to gain control over more copies, he can modify the public key and impersonate another user. Therefore, the MMVE must use a sufficiently high security level s.

Clearly, the usage of asymmetric cryptography on each message is very resource intensive. The straight forward improvement is to establish a symmetric session key at the beginning of the communication. This key can then be used for creating message authentication codes for each message. With the same method also confidentiality can be achieved.

At the moment we are developing a prototypical implementation of our approach for moderated MMVEs. Our approach for open MMVEs is still work in progress and must be refined and analyzed in more detail with respect to different possible attacks. As an example, until now we assumed that the P2P network is able to relocate data items securely between peers, when the responsibility for a given logical position in the network changes. We plan to analyze this assumption and investigate adequate mechanisms to achieve it.

References

[1] G. Schiele, R. Sueselbeck, A. Wacker, J. Haehner, C. Becker, and T. Weis, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proceedings of the Seventh International Workshop on Global and Peer-to-Peer Computing*, 2007.

[2] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr, "Low latency and cheat-proof event ordering for peer-to-peer games," in *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2004, pp. 134–139.

[3] A. B. Corman, S. Douglas, P. Schachte, and V. Teague, "A secure event agreement (SEA) protocol for peer-to-peer games," in *The First International Conference on Availability, Reliability and Security (ARES)*. IEEE Computer Society, 2006, pp. 34–41.

[4] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle, "Peer-to-peer-based infrastructure support for massively multiplayer online games," in *4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007)*, Las Vegas, Jan. 2007.

[5] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1990, pp. 307–315.

[6] J.-P. Hubaux, L. Buttyn, and S. Capkun, "The quest for security in mobile ad hoc networks," in *Proceeding of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, 2001.

[7] A. Singh and L. Liu, "Trustme: Anonymous management of trust relationships in decentralized P2P systems," in *Peer-to-Peer Computing*, N. Shahmehri, R. L. Graham, and G. Caronni, Eds. IEEE Computer Society, 2003, pp. 142–149.

[8] A. Wacker, T. Heiber, and H. Cermann, "A key-distribution scheme for wireless home automation networks," in *Proceedings of IEEE CCNC 2004*, IEEE Communications Society. Las Vegas, Nevada, USA: IEEE, January, 5-8 2004.

Index of authors

Anceaume, Emmanuelle, 29

Becker, Christian, 14
Botev, Jean, 34

Carle, Georg, 9

Esch, Markus, 34

Fouquet, Marc, 9
Frey, Davide, 29

Hu, Shun-Yun, 3
Huang, Guan-Yu, 3
Höhfeld, Alexander, 34

Ito, Dai, 19

Jiang, Jehn-Ruey, 3

Kermarrec, Anne-Marie, 29

Le Fessant, Fabrice, 29

Niedermayer, Heiko, 9
Nishide, Ryo, 19

Ohnishi, Masaaki, 19

Piegay, Romain, 29

Rieche, Simon, 9
Royan, Jérôme, 29

Schiele, Gregor, 14, 40
Schloss, Hermann, 34
Scholtes, Ingo, 34
Schuster, Sebastian, 40
Steed, Anthony, 24
Sueselbeck, Richard, 14

Teifel, Timo, 9
Triebel, Tonio, 14

Ueshima, Shinichi, 19

Wacker, Arno, 14, 40
Wehrle, Klaus, 9
Weis, Torben, 40

Zhu, Bingshu, 24