© 2012 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Generation of Synthetic Workloads for Multiplayer Online Gaming Benchmarks

Tonio Triebel*, Max Lehn†, Robert Rehner†, Benjamin Guthier*, Stephan Kopf* and Wolfgang Effelsberg*
*Department of Computer Science IV
Universität Mannheim, Mannheim, Germany
Email: {triebel, guthier, kopf, effelsberg}@informatik.uni-mannheim.de
†Databases and Distributed Systems
Technische Universität Darmstadt, Darmstadt, Germany
Email: {mlehn, rehner}@dvs.tu-darmstadt.de

***Abstract*—We present an approach to the generation of realistic synthetic workloads for use in benchmarking of (massively) multiplayer online gaming infrastructures. Existing techniques are either too simple to be realistic or are too specific to a particular network structure to be used for comparing different networks with each other. Desirable properties of a workload are reproducibility, realism and scalability to any number of players. We achieve this by simulating a gaming session with AI players that are based on behavior trees. The requirements for the AI as well as its parameters are derived from a real gaming session with 16 players. We implemented the evaluation platform including the prototype game Planet PI4. A novel metric is used to measure the similarity between real and synthetic traces with respect to neighborhood characteristics. In our experiments, we compare real trace files, workload generated by two mobility models and two versions of our AI player. We found that our AI players recreate the real workload characteristics more accurately than the mobility models.**

I. Introduction

In the past decade, a number of researchers focused their work on using peer-to-peer (P2P) technologies for networked multi-player games [1], [2], [3], [4], [5]. When taking a closer look at them, it becomes apparent that each research group employs an individual evaluation technique. Specific test setups are used, different workloads are generated and numerous metrics are defined in order to evaluate the quality of the proposed overlays. This variety impedes a comparative performance study of the different architectures. The experimental results provided in the existing works cannot be mapped onto each other for comparison. To perform an objective evaluation that spans a multitude of gaming infrastructures, it is necessary to implement a single common benchmark for these systems.

In this paper, we propose a method for generating synthetic workload to be used in benchmarks for online gaming infrastructures. Although originally developed for P2P gaming benchmarks, the workload generation as such is network-agnostic and can thus be used independently from the particular network infrastructure. It fulfills the requirements of being reproducible, scalable and realistic. We achieve this by first analyzing which aspects of the workload have significant effects on network communication. Second, we gather traces of a real gaming session in a shooter game prototype. Third, the knowledge obtained from the real gaming session forms the basis for determining the parameters of a goal-oriented artificial intelligence (AI) player we have implemented. Finally, we define a similarity metric to compare traces. We show that the AI player behaves similar to the humans according to the metric.

Any number of AI players can then be used to generate reproducible and scalable workload. In order to evaluate the degree of realism of our workload, we define a metric that expresses the similarity between real and synthetic traces. It is based on the characteristics of the number of neighbors a participant has in the game world at any point in time.

The rest of the paper is structured as follows. In the next section, we make considerations for implementing the network engine of a P2P game. Section III describes the properties of a good benchmarking workload and discusses approaches to creating it. Our evaluation framework and the AI player we implemented are outlined in Section IV. Experimental results are provided in Section V. Section VI concludes the paper.

II. P2P Games

Besides graphics, sound, and game mechanics, the implementation of the network infrastructure of a multiplayer game also plays a major role in its perceived quality of experience. The network is responsible for communicating interactions with the game world to the other players. Imperfections in the communication process can lead to noticeable delays, loss of events, or general inconsistency in the game state among the peers. Claypool and Claypool show that required capabilities of the network strongly vary with the type of game under consideration and the tasks within the game world. [6] It can be generally observed that shooter games require a high game state accuracy and low latency whereas strategy games are more tolerant in this regard.

The game we consider in our work is the 3D shooter game Planet PI4 (Fig. 1). For the most part, these requirements

This work was partially funded by the DFG research group FOR 733 and the DFG research training group GRK 1343.

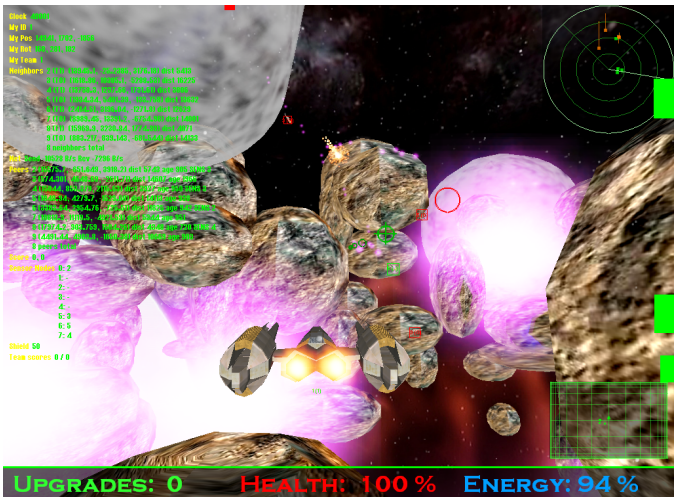

Fig. 1. Screenshot of the game *Planet PI4*

apply to traditional client/server games as well as to P2P games. On the networking level, there are tasks a network must perform that are specific to P2P games. Of the P2P issues described by Fan et al. [7], interest management and game event dissemination are the two most relevant in this context. Generally, however, these aspects are not P2P-specific and need to be solved in client/server-based approaches as well.

In an MMOG, every player has their own personal view of the game world, depending on their current state. This view determines which parts of the world the player can see and which events can be perceived. *Interest management* is the process of distinguishing between information that is essential for a player to build the personal view of the world and information that is not. The area of interest (AOI), typically centered at the player's position and bounded by the vision range, defines the region within which the player needs to receive game event information. All other players that are inside the AOI are considered to be the neighbors. In a P2P gaming overlay, they are the peers one communicates with directly. Maintaining an accurate and up-to-date neighbor list is the main objective of interest management.

The *game event dissemination* system ensures that each player receives all relevant game events within their AOI. Real-time games require low latencies for event dissemination to keep the players' views up-to-date as much as possible. Since the AOI is bound to game world positions, the dissemination systems are typically based on game world proximity. The task can thus also be formulated as a spatial publish/subscribe model. The way data is disseminated in a client/server game differs from the way it is done in a P2P game. Data aggregation and filtering can be done centrally in client/server systems, whereas in P2P systems it must be done at the peers.

Quality of experience of a multiplayer online game is a highly subjective concept and cannot be measured directly. However, it strongly depends on the correctness of the neighbor lists and the responsiveness of the game events, which in turn depend on the capabilities of the chosen overlay. Benchmarks for gaming infrastructures thus estimate quality of experience by objectively measuring neighbor list accuracy and responsiveness. The former can be evaluated by calculating precision and recall of the neighbor list management. The latter is measured as latency and game state update frequency.

III. Gaming Workload

Following our benchmarking methodology [8], four main components are required for benchmarking P2P gaming overlays. The first is the definition of the *common functionality* each candidate system must provide. Next, it is necessary to identify *quality metrics* to be measured for each overlay. Typical examples are the precision and recall of the list of neighbors in the game world or the accuracy of the game state each peer maintains. In order to actually perform a benchmark, a *test environment* must be implemented. In our work, the test environment is the P2P game *Planet PI4* running in a simulator mode with computer-controlled players. Both the environment and the metrics have already been published in previous work. [9], [10]

The fourth component of a benchmark is a realistic *workload* for the gaming overlays to be evaluated. Workload generation is the focus of this paper. The goal is to generate synthetic workloads, that is, to simulate the behavior of players playing the game. Such a workload needs to fulfill three fundamental requirements to be usable for benchmarking. The workload must be *reproducible* so that the test scenario is exactly the same for each evaluated overlay and the test can be repeated any number of times. It must be *scalable*, so that it allows to provide an arbitrary number of players. And lastly, the workload used for benchmarking should be as *realistic* as possible in order to allow to make meaningful statements about the quality of an overlay.

There have been several studies on traffic patterns and models for client/server-based (massively) multiplayer online gaming. [11], [12] The typical approach to doing this is to collect data from real gaming sessions and to fit a traffic model to them to simulate simple characteristics like data rate distributions over time. Those approaches, however, are determined to a specific network structure, typically client/server. Such a coarse simulation is not sufficient for benchmarking P2P gaming overlays. In order to maintain the players' AOI and to decide which peers to connect to, more high level knowledge like player positions is required. A synthetic workload for benchmarking a gaming overlay thus needs to include player positions and interactions, and the players' behavior must be simulated.

The behavior of a player is composed of two basic aspects: The first aspect are the *static constraints* dictated by the game itself. For example, they limit how fast players can move, where they can go and how they can interact. These constraints are mostly invariant. The second aspect are the *natural attributes* of the players. Some players may be playing more aggressively or defensively, or they can be highly skilled or play the game for the first time. It is obvious that the former

aspect is much easier to reproduce, but both aspects need to be simulated to create a truly realistic workload.

A. Workload Generation Models

Three approaches to the creation of workload are: static traces, context-insensitive mobility models and context-sensitive AI players. By "traces", we mean complete records of all actions, e.g., movement or interaction, performed by all players in a real gaming session. Such traces are clearly not scalable to any number of players other than the actual number of participants when they were created. On the positive side, they provide a reproducible workload which is realistic per se.

Mobility models on the other hand coarsely approximate player behavior. Beside their use for testing of extreme cases (e.g., massive crashes, extreme high player density), their biggest advantages are their simplicity and scalability. A random walk model only requires a few lines of code and already allows to model a large portion of the static constraints of the game and can be applied to an arbitrary number of players. Such a model can be gradually extended to include interactions like random shooting or random dying and respawning at a different location. Mobility models are deterministic and thus reproducible, and it is easy to simulate even large numbers of players. However, their degree of realism is questionable and difficult to substantiate. This is due to their insensitivity to the gaming context, i.e., they enable players to act, but not to react and interact. They may be capable of modeling characteristics of moving, shooting and dying/respawning individually, but fail to emulate the interrelation between them. Consider the following: A player may be more likely to shoot, the more hostile players are within the AOI. The more a player shoots, the more likely it may be for other players to die and respawn somewhere else. This relationship may often lead to situations where many messages (one for each shot) need to be sent to a large number of participants (crowded AOI) while neighbor lists constantly change (dying and respawning in a different location). This is an example for a complex interrelationship which is relevant to benchmarking, but cannot be modeled without understanding the game context.

AI players are a more context-aware way to generate workload. They are sensitive to the situations as they occur in the game and are able to react to them. The player behavior is recreated much more realistically than in the case of mobility models. In particular, they also allow modeling the natural attributes of the players. If implemented well, adjusting the parameters of the AI allows imitating even higher level patterns like aggressiveness or skill level. The ultimate goal for an AI player is to behave in a way that initiates the transmission of network messages in the same way a real player would. Looking at multiplayer games like Planet PI4, we find that the following messages are being sent. Updates of the player's position are sent periodically to the in-world neighbors. The frequency of these messages is assumed to be constant. Game events like shooting, hitting somebody or capturing a base all trigger messages as well. Their rate can be indirectly steered by adjusting the AI parameters. All situations where messages are disseminated have in common that the set of receivers strongly depends on the neighbors in the current AOI of the player. We thus need to implement the AI in a way so that the characteristics of the AI player's neighbors over time approximates the real situation as closely as possible. Our neighborhood metric defined in Section V allows us to measure the similarity of the neighbor list characteristics between the AI and the real player. We use this metric to fine tune the parameters of our AI implementation and also to evaluate the quality of different workload generation techniques.

TABLE I
PREVIOUS P2P GAMING WORKLOADS

	Mobility Models				
	Random Walk	Random Waypoint	Hot Spot	AI Players	Human Players
MOPAR [1]		×			
Colyseus [2]				×	
VON [3]	×		×		
Donnybrook [4]				×	×
pSense [5]		×	×		
Gross et al. [9]		×	×		

B. Previous P2P Gaming Workloads

This section provides a brief overview of the workload generation methods used in selected publications on P2P gaming overlays (see Table I). The authors of VON [3] use a simulation of discrete time steps and two mobility models. The first is a random walk where each node moves in a certain direction which changes with a certain probability. The second is a hotspot mode: Each node performs a random walk in the region of one of several hotspots and switches to another hotspot after a random interval. Similarly, the pSense [5] evaluation employs a (not further specified) random movement mode as well as a hotspot mode. MOPAR [1] is also evaluated in a simple simulator using a random mobility model which is not further specified. For the evaluation of Colyseus [2], its authors use an Emulab testbed with up to 50 hosts running modified Quake III instances. The game is then played by Quake III bots that are using an obstacle-sensitive mobility model based on Voronoi diagrams. The authors of Donnybrook [4] apply a larger-scale simulation using a behavior generator based on the same Quake III bots that where already used for Colyseus. In addition, they use a 32-player game played by humans for validation. An earlier approach to benchmarking of P2P overlays for interest management and spatial event dissemination has been proposed by Gross et al. [9] In their work, the authors focus on evaluation metrics and user churn modeling, but only use a simple mobility model (random waypoint and single hotspot) to generate the workload.

IV. Implementation

We implemented a comprehensive evaluation framework to be able to perform a complete benchmark for different network infrastructures. [10] Our framework allows to conduct real multi-player gaming sessions with humans and to create detailed trace files. In simulator mode, synthetically created sessions can be carried out in a controlled environment. Special care was taken to ensure that all processes in the simulation are reproducible. This was mainly achieved by explicitly setting the seed values wherever random numbers are generated.

A. Evaluation Framework

The evaluation framework is composed of three major components: the game Planet PI4, an integrated simulation environment, and an implementation of a monitoring server.

Planet PI4: Planet PI4 is fully implemented third-person 3D space shooter game for multiple players connected by an exchangeable P2P overlay network. The players can fly through a virtual asteroid field with space ships and shoot at each other. The game world contains several points of interest like bases or repair points as incentives for players to gather at certain locations. Players can either compete in a free-for-all fashion or as opposing teams. The game software has a modular architecture so that the implementation of the gaming overlay can be exchanged as needed with little effort. It currently runs with one of two implemented overlay networks: pSense [5], BubbleStorm [13], and a simple client/server implementation.

Discrete Event Game Simulator: This mode provides a reproducible environment that is able to simulate peers playing the game as well as the underlay network. Real-time game events are mapped to a discrete-event queue and the rendering of the virtual world can be disabled. In addition, the simulation environment maintains a global view of all peers. [10]

Monitoring Server: This server is used to monitor and trace all the game data from the human players as the game progresses. The server's clock is used as a global time reference in the created traces.

B. AI Player

The main goal of implementing a game AI is to enable a purposeful behavior of the computer-controlled players. There exist many different actions in a game that trigger network messages. The progression of these actions reflects the characteristic of the player's gaming behavior. This includes simple reactions to game events as well as behaviors with a more high-level motivation like strategies and team play. This is best modeled by a goal-oriented AI. We implemented a Behavior Tree AI for workload generation and adjusted the parameters of the goals in order to achieve a maximum amount of realism. The goals can be simple or complex. Complex goals are composed of a sequence of simple sub-goals where each sub-goal is mandatory for the success of the goal. The leaf goals of the tree form the interface to the game world. They can gather information about the current game state and interact with the world using concrete actions. Combining goals in such a way allows for an intuitive modeling of simple and complex behaviors. The desirability of each goal is periodically evaluated based on the current game state. The goal with the highest desirability score gets executed. The desirability functions are shown in Table II. In our implementation we created 4 complex goals that are based on 6 sub-goals. During the execution of every goal, we permanently run obstacle avoidance to prevent collisions with other players or objects. The goals are as follows.

Go To Position (sub-goal): This goal sets the current speed of the ship to the maximum and steers towards the destination.

Find Highest Threat (sub-goal): This goal analyzes the enemies that are inside the area of interest. It determines the opponent that poses the highest threat based on distance, angle and shooting frequency.

Attack Opponent (sub-goal): Follows the enemy target to take it down. Since an appropriate strategy depends on the distance to the target, we implemented the following different strategies: If the target is out of firing range, approach the target at full speed. If the target is in range, decrease speed, keep following the target and start shooting. If the target is too close, try to flank it by applying lateral thrust to fly around the enemy ship and keep shooting.

Combat(complex): This goal is a sequence of the goals "Find Highest Threat" and "Attack Opponent".

Find Repair Point (sub-goal): Selects the closest repair point among all repair points inside the AOI.

Repair Ship (complex): This goal is a sequence of the goals "Find Repair Point" and "Go To Position".

Find Base (sub-goal): The goal checks all bases in the AOI and determines the one that is most desirable to capture. The decision depends on the distance to the base and its current state. Bases that are controlled by the enemy are more preferable than neutral ones.

Capture Base (complex): This goal is a sequence of the goals "Find Base" and "Go To Position".

Find Waypoint (sub-goal): Determines an interesting area for exploration. This is done either by selecting a uniformly distributed random waypoint or by selecting a random point of interest (e.g., bases and repair points). Both versions are implemented and evaluated in Section V.

Exploration (complex): This goal acts as the default behavior. It explores the map until a goal with a higher desirability arises. It is a sequence of the goals "Find Waypoint" and "Go To Position".

V. Evaluation

This section evaluates the synthetic workload created by our AI players. The results are based on a real gaming session of the game Planet PI4. 16 players ranging from novices to experienced shooter game players were divided into two teams and played the game for 20 minutes. All peers sent their application data to a central monitoring server where a trace file was created. The trace file contains a time-stamp, a unique player ID, the number of neighbors each player has,

TABLE II
DESIRABILITY FUNCTIONS FOR THE COMPLEX GOALS. THE T ARE ADJUSTABLE PARAMETERS TO TWEAK THE AI.

Goal	Desirability Function
Combat	$(\frac{CSV}{MSV}) * (\frac{VR}{DTE}) * T_{Combat}$
Repair Ship	$(\frac{MSV-CSV}{MSV}) * (\frac{VR}{DTR}) * T_{Repair}$
Capture Base	$(\frac{BVR}{CB}) * (\frac{VR}{DTB}) * T_{Capture}$
Exploration	$T_{Explore}$

CSV : Current Shield Value
MSV : Maximum Shield Value
DTE : Distance to next Enemy
DTB : Distance to next Base
DTR : Distance to next Repair Point
VR : Vision Range
CB : Captured Bases
BVR : Bases in Vision Range

the number of shots fired, number of hits and the number of deaths. Each value is accumulated over one second and written to the trace file as one entry per second per player. The same experimental setup was then repeated for four artificial gaming sessions by using the following workload generation techniques:

Random Waypoint (RWP): Mobility model that makes the artificial players move to completely randomly selected positions on the map.

Random Point of Interest (RPOI): Mobility model that moves to randomly selected points of interest (bases, repair points).

AI Player with RWP Exploration (AI-RWP): AI player with a random waypoint exploration mode.

AI Player with RPOI Exploration (AI-RPOI): AI player with increased interest in exploring bases and repair points.

We denote the session of real players by **REAL**.

We argue that closely approximating the real characteristic of the number of neighbors in a player's AOI is the most important feature for synthetically generated workloads. Adding neighbors or removing them from the neighbor list is the main task of an overlay's interest management. AOI neighbors also represent the receivers of most of the sent messages and thus strongly influence how many messages are communicated over the network.

In order to express the characteristics of a player's neighbors over time, we use a simple Markov chain. We assume that the number of neighbors at one point in time only depends on the number of neighbors in the previous time step. Higher order dependencies are ignored. We thus count how often the number of neighbors changed from i to j over the entire duration of the gaming session. This results in a neighborhood transition matrix $A = (a_{i,j})$, $i = 0, \ldots, n$, $j = 0, \ldots, n$ where each coefficient $a_{i,j}$ counts the number of transitions from i neighbors to j neighbors from one time step to the next (1 second). The matrix is then normalized such that $\sum_{j=0}^{n} a_{i,j} = 1$ for all i, that is, all rows sum up to 1. Each row i then represents a probability distribution for the number of neighbors j in the next time step. Like this, a matrix $A_{s,p}$ is created for each of the 5 sessions s and each of the $n = 16$ players p. Additionally, a matrix $\tilde{A}_s$ is created for each session by averaging the neighborhood transition matrices over all players in the session. Fig. 2 shows a plot of the average matrix $\tilde{A}_{REAL}$ of the real gaming session.

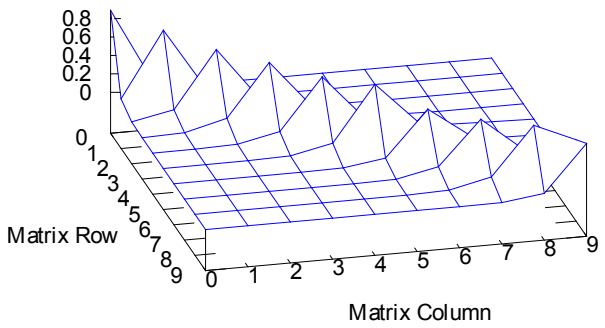

Fig. 2. Visualization of the transition matrix between the number of neighbors in two subsequent time steps. The matrix is diagonally dominant.

We can now measure the difference $d_1(\tilde{A}, \tilde{B})$ between the neighborhood characteristics of two sessions by simply computing the L1 norm between the respective averaged neighborhood transition matrices $\tilde{A}$ and $\tilde{B}$:

$$d_1(\tilde{A}, \tilde{B}) = \sum_{i=0}^{n} \sum_{j=0}^{n} |\tilde{a}_{i,j} - \tilde{b}_{i,j}|. \quad (1)$$

We evaluate d_1 in two different ways. First, we compare the averaged transition matrix $\tilde{A}_{REAL}$ to the transition matrices $\tilde{A}_s$ of the four synthetic sessions. We do this by calculating $d_1(\tilde{A}_{REAL}, \tilde{A}_s)$ for $s \in \{\text{REAL, RWP, RPOI, AI-C, AI-S}\}$. This is shown in the second column of Table III. Second, we compare the matrices $A_{s,p}, p = 1, \ldots, n$ of all players of a particular session s with the average $\tilde{A}_s$ of the respective session and average over all players:

$$d_{var}(s) = \frac{1}{n} \sum_{p=1}^{n} d_1(\tilde{A}_s, A_{s,p}) \quad (2)$$

for each session s. This gives an indication of how much the metric d_1 varies among the players of a session. This is shown in the third column of Table III. It can be seen from the table that the neighborhood characteristic is better modeled by the AI players than the two mobility models. Also, the AI player that prefers to explore points of interest when there is no other goal to follow, achieves a higher similarity than the random explorer AI. This is due to the fact that human players also tend to gather at interesting points of the world. The small variation of the players around their average matrix indicates that the metric we defined is reliable property.

The second goal of creating realistic gaming workload is to accurately mimic the interactions of the real players. The

TABLE III
DIFFERENCE BETWEEN THE NEIGHBORHOOD TRANSITION MATRICES OF THE FOUR WORKLOAD GENERATION TECHNIQUES AND THE REAL SESSION.

Session s	$d_1(\tilde{A}_{REAL}, \tilde{A}_s)$	$d_{var}(s)$
REAL	0	0.65
RWP	7.49	2.97
RPOI	2.74	0.59
AI-RWP	2.26	0.75
AI-RPOI	0.81	0.40

interactions shooting, hitting, and dying all produce messages that are sent over the network. We counted the number of shots fired, the number of hits and the number of kills for the real session and the two AI players and compare them to each other in Table IV. All values are given per minute and per player. Note that the mobility models are omitted from the table, because they are insensitive to the game context and thus unable to shoot other players. The table reveals that our AI players are more aggressive than real players and shoot more frequently. Together with their increased accuracy, they yield a higher kill rate per minute. Unfortunately, it is not possible to adjust shooting rate and accuracy directly. They are implicit effects of adjusting the desirability of the Combat goal, because an AI player always shoots when it is in Combat mode and the opponent is in range. However, adjusting the desirability of Combat also negatively affects the neighborhood characteristic, which is our main focus. Making the bots less aggressive while maintaining a good neighborhood characteristic is left for future work.

VI. CONCLUSIONS AND FUTURE WORK

We presented an approach to generating realistic workload for the use in the benchmarking of multiplayer online gaming infrastructures. It is based on goal-oriented AI players playing the 3D shooter game Planet PI4. Unlike mobility models, AI players take the context of a game into account. By examining the types of messages used in such a game, it became apparent that the number of AOI neighbors play an important role in the generation of network messages. We thus defined a metric that allows to compare neighborhood characteristics of different workloads. This metric was applied in an experiment with real workload, two mobility models and two versions of our AI player. The experiment showed that the metric is a reliable property and that the AI players modeled real player behavior more accurately with respect to the number of neighbors. We also found that the current AI implementation models the number of shots, hits and deaths inexactly. These values can only be adjusted indirectly, which leads to a change in the neighborhood characteristic at the same time.

In future work, we would like to investigate the parameters of our AI more thoroughly to find a setting that meets both requirements simultaneously. The impact of a further increased AI complexity to the generated workload also needs to be investigated. Furthermore, we want to compare the results

TABLE IV
COMPARISON OF THE SHOTS/HITS/KILLS PER MINUTE AND PER PLAYER AND THE ACCURACY FOR THE REAL GAMING SESSION AND THE TWO AI PLAYER WORKLOADS.

Session	Shots	Hits	Kills	Accuracy
REAL	148.09	41.76	0.41	28.2%
AI-RWP	161.19	71.49	0.74	44.3%
AI-RPOI	199.48	70.86	0.74	35.5%

with other real gaming session with more participants. For this purpose, it is desirable to extend the metric d_1 so that the neighborhood characteristics of two sessions with differing numbers of players can be compared.

REFERENCES

[1] A. P. Yu and S. T. Vuong, "Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games," in *Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV'05)*. New York, NY, USA: ACM, 2005, pp. 99–104.

[2] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *Proceedings of the 3rd conference on Networked Systems Design & Implementation (NSDI'06)*. Berkeley, CA, USA: USENIX Association, 2006, pp. 12–12.

[3] S.-Y. Hu and G.-M. Liao, "VON: A Scalable Peer-to-Peer Network for Virtual Environments," in *IEEE Network, vol. 20, no. 4, Jul./Aug. 2006*, 2006, pp. 22–31.

[4] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: enabling large-scale, high-speed, peer-to-peer games," in *Proceedings of the ACM SIGCOMM conference on Data communication (SIGCOMM'08)*. New York, NY, USA: ACM, 2008, pp. 389–400.

[5] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. Buchmann, "pSense - Maintaining a Dynamic Localized Peer-to-Peer Structure for Position Based Multicast in Games," in *IEEE International Conference on Peer-to-Peer Computing (P2P'08)*, 2008.

[6] M. Claypool and K. Claypool, "Latency can kill: precision and deadline in online games," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems (MMSys'10)*. New York, NY, USA: ACM, 2010, pp. 215–222.

[7] L. Fan, P. Trinder, and H. Taylor, "Design issues for peer-to-peer massively multiplayer online games," *International Journal of Advanced Media and Communication*, vol. 4, no. 2, pp. 108–125, Mar. 2010.

[8] M. Lehn, T. Triebel, C. Gross, D. Stingl, K. Saller, W. Effelsberg, A. Kovacevic, and R. Steinmetz, *From Active Data Management to Event-Based Systems and More*, ser. Lecture Notes in Computer Science. Springer, nov 2010, vol. 6462, ch. Designing Benchmarks for P2P Systems, pp. 209–229.

[9] C. Gross, M. Lehn, C. Muenker, A. Buchmann, and R. Steinmetz, "Towards a comparative performance evaluation of overlays for networked virtual environments," in *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P'11)*. IEEE, Sep 2011, pp. 34–43.

[10] M. Lehn, C. Leng, R. Rehner, T. Triebel, and A. Buchmann, "An online gaming testbed for peer-to-peer architectures," in *Proceedings of the ACM SIGCOMM 2011 conference (SIGCOMM'11)*. ACM, August 2011, demo.

[11] T. Lang, P. Branch, and G. Armitage, "A synthetic traffic model for Quake3," in *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, no. cycle 1. ACM, 2004, p. 238.

[12] P. Svoboda, W. Karner, and M. Rupp, "Traffic Analysis and Modeling for World of Warcraft," *IEEE International Conference on Communications (ICC'07)*, pp. 1612–1617, Jun. 2007.

[13] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann, "Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'07)*. New York, NY, USA: ACM, 2007, pp. 49–60.