Multimedia Systems (2014) 20:609–620
DOI 10.1007/s00530-014-0359-z

SPECIAL ISSUE PAPER

On synthetic workloads for multiplayer online games: a methodology for generating representative shooter game workloads

Max Lehn · Tonio Triebel · Robert Rehner · Benjamin Guthier · Stephan Kopf · Alejandro Buchmann · Wolfgang Effelsberg

Published online: 4 February 2014

Abstract We present approaches to the generation of synthetic workloads for benchmarking multiplayer online gaming infrastructures. Existing techniques, such as mobility or traffic models, are often either too simple to be representative for this purpose or too specific for a particular network structure. Desirable properties of a workload are reproducibility, representativeness, and scalability to any number of players. We analyze different mobility models and AI-based workload generators. Real gaming sessions with human players using the prototype game Planet PI4 serve as a reference workload. Novel metrics are used to measure the similarity between real and synthetic traces with respect to neighborhood characteristics. We found that, although more complicated to handle, AI players reproduce real workload characteristics more accurately than mobility models.

This work has been co-funded by the German Research Foundation (DFG) in the collaborative research center (SFB) 1053 "MAKI: Multi-Mechanism Adaptation for the Future Internet" and the research training group (GRK) 1343 "Topology of Technology".

M. Lehn · R. Rehner (✉) · A. Buchmann
Databases and Distributed Systems, Technische Universität Darmstadt, Darmstadt, Germany
e-mail: rehner@dvs.tu-darmstadt.de

M. Lehn
e-mail: mlehn@dvs.tu-darmstadt.de

A. Buchmann
e-mail: buchmann@dvs.tu-darmstadt.de

T. Triebel · B. Guthier · S. Kopf · W. Effelsberg
Department of Computer Science IV, Universität Mannheim, Mannheim, Germany
e-mail: triebel@pi4.informatik.uni-mannheim.de

B. Guthier
e-mail: guthier@informatik.uni-mannheim.de

S. Kopf
e-mail: kopf@informatik.uni-mannheim.de

W. Effelsberg
e-mail: effelsberg@informatik.uni-mannheim.de

1 Introduction

In the past decade, several researchers have focused their work on using peer-to-peer (P2P) technologies for networked multi-player games [2, 3, 9, 14, 19]. When taking a closer look at these approaches, it becomes apparent that each research group employs their individual evaluation technique. Specific test setups are used, different workloads are generated, and numerous metrics are defined for the evaluation of the proposed overlays. This variety impedes a comparative performance study of the different architectures. To perform an objective evaluation that spans a multitude of gaming infrastructures, it is necessary to implement a common benchmark applicable for these systems.

In this article, we present and discuss methods for the generation of synthetic workloads to be used in benchmarks for online gaming infrastructures. The proposed workload generation is network-agnostic and can thus be used independently from a particular network infrastructure. We achieve this using only game logic inputs (e.g., steering, throttling, shooting), instead of a direct communication with the network engine or the underlying network.

A good workload needs to fulfill the requirements of being reproducible, scalable and representative for real applications. While the first two requirements can be fulfilled with synthetic workloads, the third is particularly challenging for online games. We tackle this challenge using application-level workload similarity metrics that

allows us to tune artificial gaming workloads based on reference traces from real games. We present and compare four workload generation methods: two mobility models and two AI-based workloads. With these methods, any number of players can be employed to generate reproducible and scalable workload. First, results have been published in previous work [7].

The rest of the article is structured as follows. In the next section, we make considerations on the implementation of multiplayer game network engines and discuss what is needed for benchmarking. Section 3 discusses the properties of gaming workloads and approaches to the generation of artificial workloads for benchmarking. In Sect. 4, we present metrics that can be used to measure the similarity of generated workloads with reference workloads. Our evaluation framework and the AI player we implemented are outlined in Sect. 5. Experimental results are provided in Sects. 6 and 7 concludes the paper.

2 Benchmarking considerations

Besides graphics, sound, and game mechanics, the network infrastructure of a multiplayer game plays a major role in its perceived quality of experience. The network is responsible for synchronizing the game state among the players. Flaws in the communication process can lead to additional delays, loss of events, or general inconsistencies in the game state.

An unbiased performance evaluation of such a multiplayer network infrastructure is not a trivial task. Comparable to hardware benchmarks, it is necessary to create a test procedure that is able to stress the network infrastructure with a realistic workload, to get evidence of its performance. The first step of the creation of such a benchmark is the definition of the relevant aspects that need to be evaluated. In the case of online games this includes two major aspects. First, as Claypool and Claypool [6] have shown, the required performance of a network infrastructure strongly varies with the type of game under consideration and the tasks in the game world. We have chosen to create a benchmark for shooter games, since they have been identified as the most demanding type [5]. They require a high game state accuracy and low latency for all game events. Although shooter games are technically similar to role-play games (RPG), there are only a few that support a 'massive' amount of players. We believe one reason for this is the lower performance demand of a massively multiplayer online RPG (MMORPG) that can be fulfilled more easily with today's client/server architectures.

Second, to specify the tasks in the game world, it is necessary to take a closer look at the structure of a network game. The tasks that a network infrastructure has to deal with can be divided into: Interest Management, Game Event Dissemination, NPC Computation, Game State Persistence and Cheating Mitigation.[1] In this paper we will focus on:

Interest management (IM) IM is the process of distinguishing between information that is essential for a player to build her 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 in which the player needs to receive game event information. All other players that are inside the AOI are considered to be her neighbors. In a P2P gaming overlay, these are the peers that communicate with each other mostly directly. Maintaining an accurate and up-to-date neighbor list is the main objective of interest management.

Game event dissemination (GED) GED 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. Since the AOI is bound to positions in the game world, the dissemination systems are typically based on game world proximity. The task can therefore also be formulated as a spatial publish/subscribe model. The way data are 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 has to be done cooperatively by the peers.

We picked these issues, because

- they are mandatory for every shooter game,
- they have the highest timing demands on the network, and
- most of the traffic is created by these two tasks.

Our benchmarking concept for shooter game infrastructures objectively measures the neighbor list accuracy and responsiveness of game events, which serve as an indicator for the game's quality of experience.[2]

Following our methodology [11], four main components are required for a P2P gaming benchmark. 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 player perceives. To actually perform a benchmark, a

[1] This categorization is inspired by Fan et al. [7], who divide the tasks of a P2P game into six issues, of which five also apply to client/server infrastructures.

[2] The relation between quality of service and quality of experience is far from trivial and a field of research on its own. Specifically for games, there has been research on player performance depending on network properties [1].

test environment must be implemented. In our work, the test environment is the game prototype Planet PI4 running in a simulation mode with computer-controlled players. Both the environment and the metrics have already been published in previous work [8, 12].

The fourth component of a benchmark is a representative workload for the gaming overlays to be evaluated. Workload generation is the focus of this paper.

3 Gaming workloads

The goal of the generation of synthetic workloads is to reproduce the activity that has to be processed by the networking component. Such a workload needs to fulfill three fundamental requirements to be useful for benchmarking: the workload must be reproducible so that the test scenario is equivalent for all tested systems, and the test can be repeated any number of times. It must be scalable, so that it allows simulating an arbitrary number of players. And lastly, the workload used for benchmarking must be representative to real workloads to make meaningful statements about the performance of an infrastructure.

There have been several studies on traffic patterns and models for client-/server-based (massively) multiplayer online gaming [10, 15]. The most common approach is to collect data from real gaming sessions and to fit a traffic model to the measured data. Such models are able to reproduce characteristics like the data rate distributions over time. They are widely used to estimate the network load for a game server or an ISP network. They have, however, two major limitations that make them inappropriate for our purpose. First, they are specific to a certain architecture. While for client/server architectures there are only a few possible topologies, there is a plethora of architectures for P2P-based MMOGs [18]. Creating an empirical model for each topology is not only laborious, but also also hinders research on new architectures such as hybrid P2P with cloud support. Second, a simulation on the network or transport layer is not sufficient for benchmarking P2P- or cloud-based gaming overlays, since many overlays are using application-specific information to create connections. For example, to maintain the players' AOI and to decide which peers to connect to, player positions are required. A synthetic workload for benchmarking a gaming overlay thus needs to include player positions and interactions.

3.1 Workload generation models

Figure 1 shows a schematic overview of the layers of a network game and the different options for the generation of workload. There are three basic approaches to the creation of workloads that employ the network engine: static traces, context-insensitive mobility models, and context-sensitive AI players.

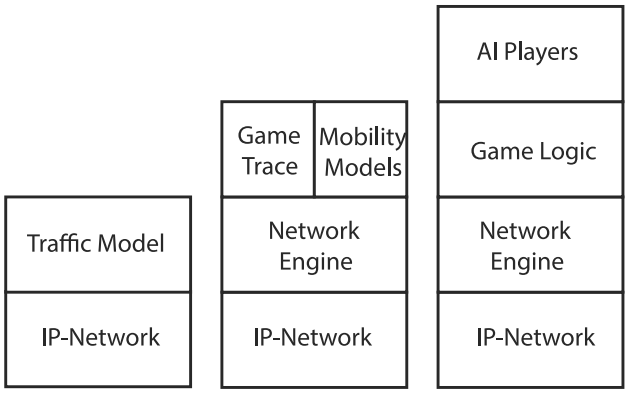


Fig. 1 Layers of a network game with different workload generation methods. Traffic models do not incorporate the game's network engine. Only AI players allow simulating game logic-dependent network traffic

Traces are complete records of all actions, e.g., movement and interaction, performed by all players in a real gaming session. Such traces can be used to replay the respective games. Scaling the number of players in the traces, however, is hardly possible. Although a reduction could be achieved by omitting a subset of the players from the trace and an increase by duplicating players, this will likely break interaction schemes. Interaction partners with will be missing, and player duplication requires breaking up the original interactions. On the positive side, traces 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 models a large portion of the static constraints of a game. Such a model can be gradually extended to include interactions like random shooting or random respawning at different locations. 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

exchanged with a large number of participants (crowded AOI), while neighbor lists constantly change (dying and respawning in a different location).

AI players have a context-aware approach to workload generation. They are sensitive to situations as they occur in the game and are programmed to react to them. Player behavior can potentially be recreated much more accurately than by mobility models. In particular, AI players allow modeling the natural attributes of the players. The goal of an AI player is to simulate the human gaming behavior, which is composed of two basic aspects. The first aspect is 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 is 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 need to be taken into account to create a sufficiently realistic workload. If implemented well, adjusting the parameters of the AI allows imitating even high-level patterns like aggressiveness or skill level.

Looking at multiplayer games like Planet PI4 (see Sect. 5), we find that the following messages are sent for IM and GED. Updates of the player's position are sent periodically to the player's 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 adjusted by 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 configure the AI in a way so that the characteristics of the AI player's neighbors over time approximate the real situation. Our neighborhood metrics defined in Sect. 4 allow us to measure the similarity of neighbor list characteristics between AI and real players.

3.2 Related P2P gaming evaluations

This section gives a brief overview of the workload generation methods used in selected publications on P2P gaming overlays (see Table 1). The authors of VON [9] 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 proximity of one of several hotspots and switches to another hotspot after a random interval. Similarly, the pSense's [14] evaluation employs a (not further specified) random movement mode as well as a hotspot mode. MOPAR [19] is also evaluated in a simple simulator using a random mobility model which is not further specified.

Table 1 P2P gaming workloads used in previous work

	Mobility models			AI players	Human players
	Random walk	Random waypoint	Hot spot		
MOPAR [19]		×			
Colyseus [2]				×	
VON [9]	×		×		
Donnybrook [3]				×	×
pSense [14]		×	×		
Gross et al. [8]		×	×		

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 [3] 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 [8]. 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.

4 Workload similarity metrics

To gain evidence of the quality of synthetically generated workload, it is necessary to compare it with real reference workloads. This can be achieved using a metric that reflects the similarity between different workloads. Such a metric should be as simple as possible but at the same time cover the aspects that have a significant effect on the tested systems' load. In this section, we discuss several options for workload comparison metrics, from very simple to more sophisticated approaches.

The messages transmitted among participants of an online game can be coarsely classified into two categories: regular status update messages, especially position updates, and irregular messages instantly arising from certain player actions, such as firing a missile.[3] Since these two categories

[3] Using dead reckoning techniques [13], position updates might not be sent in a precisely fixed frequency, but instead to a certain degree depend on the players' activities. Still, there is usually a minimum and maximum rate at which these updates are transmitted. On average, they are thus expected to show a more regular behavior.

show very different characteristics, we will discuss them separately.

4.1 Regular events

Regular events usually contain status updates that are of interest to the surrounding players in the virtual world. Accordingly, these events need to be disseminated to all interested neighbors.

Game session model For the definition and discussion of possible metrics, we define a simple formal model for game sessions created using a given workload model.

$$\mathcal{G} = (P, T, \mathcal{N})$$

denotes a game session trace. It consists of a set of players

$$P = \{p_1, p_2, p_3, \ldots\},$$

which, for the following discussion, is assumed to be constant over the whole game session.

$$T = \{\tau_1, \tau_2, \tau_3, \ldots\}$$

is the set of sampling timestamps used for the trace. The sampling intervals should have fixed lengths; we will use intervals of 1 s. The selection of appropriate values depends on the type of game and has an influence on the results.

Finally, the neighbor sets of all players p at each sample τ are defined as

$$\mathcal{N} = \left(N_{p,\tau}\right); p = 1, \ldots, |P|; \tau = 1, \ldots, |T|,$$

where

$$N_{p,\tau} = \{q \in P \mid q \text{ is interested in } p\text{'s events at time } \tau\}$$

First approach: average size of neighbor sets Assuming that the frequency of regular updates is a parameter of the system under test, and thus not a workload parameter, the most significant workload factor is the size of the interest neighbor set $|N_{p,\tau}|$ defined above. AOI neighbors represent the receivers of most of the sent messages and thus strongly influence how many messages are sent over the network. We argue that closely approximating the characteristics of the number of neighbors in a player's AOI is the most important feature for synthetically generated workloads. The easiest approach would be to use the mean neighbor set size

$$\overline{N} = \frac{1}{|P||T|} \sum_{p \in P} \sum_{\tau \in T} |N_{p,\tau}|.$$

For two given game sessions $\mathcal{G} = (P, T, \mathcal{N})$ and $\mathcal{G}' = (P', T', \mathcal{N}')$, we can then define the metric

$$\mathcal{M}_{\text{mean}}(\mathcal{G}, \mathcal{G}') = |\overline{N} - \overline{N}'|.$$

This metric is very simple and intuitive, but it neglects the variance in the neighbor set size distribution. Therefore, one could also consider the sample variance

$$s^2 = \frac{1}{|P||T| - 1} \sum_{p \in P} \sum_{\tau \in T} (|N_{p,\tau}| - \overline{N})^2.$$

This, however, results in two separate values for the metric, making a comparison difficult. We thus follow a different approach.

Second approach: neighbor set size distribution To reflect the distribution of the neighbor set size even further, we can compare histograms of their distributions.

$$h_x(\mathcal{G}) = \left|\left\{(p, \tau) \in P \times T \mid |N_{p,\tau}| = x\right\}\right|, \quad x \in \mathbb{N}^0$$

counts the occurrences of a certain number x of neighbors. To be able to compare different session lengths and numbers of players, the histogram should be normalized, resulting in an empirical probability function:

$$\widetilde{h}_x(\mathcal{G}) = \frac{h_x(\mathcal{G})}{\sum_{z \in \mathbb{N}^0} h_z(\mathcal{G})}$$

The metric based on such a histogram is then defined as

$$\mathcal{M}_{\text{hist}}(\mathcal{G}, \mathcal{G}') = \sum_{x \in \mathbb{N}^0} \left|\widetilde{h}_x(\mathcal{G}) - \widetilde{h}_x(\mathcal{G}')\right|.$$

Third approach: transitions Another factor that has not been considered yet is the dynamism of the neighbor sets. Adding neighbors or removing them from the neighbor list is the main task of an overlay's interest management. Overlays that build their topology based on the players' proximity generally need to perform restructuring operations when neighbor sets change, which are associated with costs and thus should be reflected in the workload.

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. We define the transition matrix

$$A(\mathcal{G}) = \begin{pmatrix} a_{0,0} & \cdots & a_{1,|P|} \\ \vdots & \ddots & \vdots \\ a_{|P|,1} & \cdots & a_{|P|,|P|} \end{pmatrix},$$

where each coefficient $a_{i,j}$ counts the number of transitions from i neighbors to j neighbors from one time step to the next:

$$a_{i,j} = \left|\left\{(p, t) \in P \times \widehat{T} \mid |N_{p,\tau_t}| = i \wedge |N_{p,\tau_{t+1}}| = j\right\}\right|;$$
$$\widehat{T} = \{1, , |T| - 1\}.$$

To be able to compare sessions with a different duration, the matrix is normalized:

$$\widetilde{A}(\mathcal{G}) = \left(\widetilde{a}_{i,j}\right), \text{ with } \widetilde{a}_{i,j} = \frac{a_{i,j}}{\sum_{k=0}^{|P|} a_{i,k}}$$

so that $\sum_j \widetilde{a}_{i,j} = 1$ for all $i \in \{0, \ldots, |P|\}$. $\widetilde{a}_{i,j}$ is thus the empiric probability of a transition from i to j neighbors.

We can now measure the difference $d_1(\widetilde{A}(\mathcal{G}), \widetilde{A}(\mathcal{G}'))$ between the neighborhood characteristics of two sessions by simply computing the L_1 norm between the respective averaged neighborhood transition matrices $\widetilde{A}(\mathcal{G})$ and $\widetilde{A}(\mathcal{G}')$:

$$\mathcal{M}_{\text{trans1}}(\mathcal{G}, \mathcal{G}') = d_1(\widetilde{A}(\mathcal{G}), \widetilde{A}(\mathcal{G}')) = \sum_{i=0}^{|P|} \sum_{j=0}^{|P|} |\widetilde{a}_{i,j} - \widetilde{a}'_{i,j}|.$$

This metric, however, has two problems: first, there might be rows in the matrix A, where all coefficients are zero. In such a case, normalization does not work. A solution would be to set all entries of these rows of $\widetilde{A}$ to $|P|^{-1}$, which corresponds to a uniform probability distribution. Second, matrices of different size, i.e., from traces with a different number of total players, cannot be effectively compared.

Weighted transition matrix To deal with the latter two problems, we combine the histogram-based approach with the Markov transition model. By weighting the rows of $\widetilde{A}$ with the empirical probability $\widetilde{h}$ of being in the corresponding state, we eliminate both problems at once. All-zero rows remain zero, because the probability of being in that state is zero. Matrices of different size can then be compared by just extending the smaller one with zeros.

We therefore define $\widehat{A}(\mathcal{G}) = \left(\widehat{a}_{i,j}\right)$, with

$$\widehat{a}_{i,j} = \widetilde{a}_{i,j}\widetilde{h}_i = \frac{a_{i,j}}{\sum_{k=0}^{|P|}\sum_{l=0}^{|P|} a_{k,l}}.$$

Finally, the metric is defined accordingly:

$$\mathcal{M}_{\text{trans}}(\mathcal{G}, \mathcal{G}') = d_1(\widehat{A}(\mathcal{G}), \widehat{A}(\mathcal{G}')) = \sum_{i=0}^{|P|} \sum_{j=0}^{|P|} |\widehat{a}_{i,j} - \widehat{a}'_{i,j}|,$$

with d_1 being extended so that it fills the smaller one of the two matrices with zeros to fit.

4.2 Irregular events: interactions

Now, we will have a look at the irregular events which are induced by player (inter)actions.

Game session model The game session model is extended with the irregular events $\mathcal{E}$ of type classes $\mathcal{C}$.

$$\mathcal{G} = (P, T, \mathcal{N}, \mathcal{C}, \mathcal{E})$$

with

$$\mathcal{E}_c = \{(t, e) \mid \text{event } e \text{ of type } c \in \mathcal{C} \text{ was fired at time } t\}$$

Type classes can be, for instance, shooting, hitting another ship, being hit, and dying. We assume that the events of one type show the same characteristics and that their messages have about the same sizes.

As an alternative representation, the events can be sorted into buckets according the sampling intervals T from the regular events:

$$E_{c,\tau_i} = \{e \mid (t, e) \in \mathcal{E}_c \wedge \tau_i \leq t < \tau_{i+1}\}, i = 1, \ldots, |T|$$

First approach: averages Again, we start with the simplest approach, which is to take the average event rate per player for each event type:

$$\overline{E}_c = \frac{1}{|P||T|} \sum_{\tau \in T} |E_{c,\tau}|,$$

$$\mathcal{M}_{\text{emean},c}(\mathcal{G}, \mathcal{G}') = |\overline{E}_c - \overline{E}'_c|$$

Second approach: correlation with neighbor sets Since the event messages have to be disseminated among all neighbors, it is of relevance how the occurrence of events correlates with the neighbor set size at the same time. We can use the correlation coefficient to quantify this correlation.

$$r_c = \frac{\sum_{\tau \in T}(N_\tau - \overline{N})(E_{c,\tau} - \overline{E}_c)}{\sqrt{\sum_{\tau \in T}(N_\tau - \overline{N})^2 \cdot \sum_{\tau \in T}(E_{c,\tau} - \overline{E}_c)^2}},$$

$$\mathcal{M}_{\text{ecorr},c}(\mathcal{G}, \mathcal{G}') = |r_c - r'_c|$$

The presented set of metrics serves as a basis for the comparison of new artificial workloads with reference traces from real game sessions.

5 Implementation

To obtain a complete benchmark for different network infrastructures, we have implemented a comprehensive evaluation framework [12]. It allows to conduct real multi-player gaming sessions with humans and to create detailed trace files. In a 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.

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

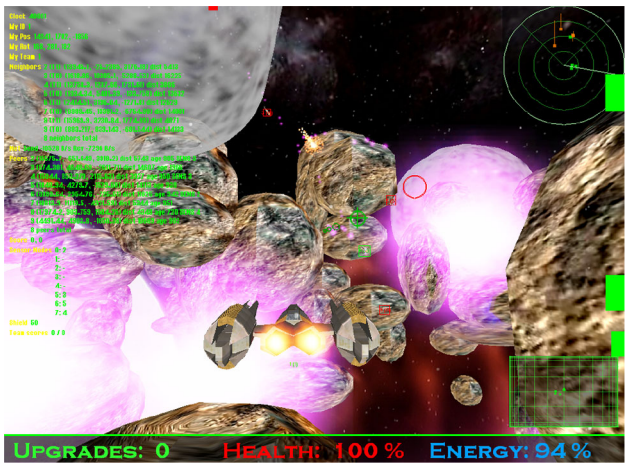


Fig. 2 Screenshot of the game Planet PI4

Planet PI4 Planet PI4 (Fig. 2) is a third-person 3D space shooter for multiple players connected via an exchangeable P2P overlay network. The players can fly space ships through a virtual asteroid field and shoot 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 with little effort. It currently runs with one of two implemented overlay networks, pSense [14] and BubbleStorm [16], or with 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 underlying network. Real-time game events are mapped to a discrete-event queue and the rendering of the graphics can be disabled. In addition, the simulation environment maintains a global view of all peers [12].

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.

The workload can be generated by human players in a real gaming session or synthetically, either by simple mobility models or by AI players.

5.1 Mobility models

Mobility models (MM) are often used because they are easy to implement and mostly independent of a particular game type. An MM simulates agents moving within a simple, usually void, virtual world. Agents can also generate events according to statistical models, but they do not interact with each other. For this paper, we use two common mobility models: random waypoint (RWP): an agent picks a random point in the virtual world and moves towards it. When it reaches the point, it repeats the procedure. Random point of interest (RPOI): first, the points of interest (POI) set is determined by randomly choosing n coordinates within the virtual world. Then, each agent draws a random number $m \in \{1 \ldots n\}$ and moves towards the m-th POI. When it reaches the POI, it repeats the procedure. For our experiments we used a world with 64 POIs.

5.2 AI players

The main goal of implementing a game AI is to enable a purposeful behavior of the computer-controlled players. There exist many 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. Both can be modeled by different types of game AIs such as finite-state machines (FSM), planning-based AIs like hierarchical task networks (HTN), or goal-oriented AIs like behavior trees (BT). We decided to use a goal-oriented AI because they are flexible and scalable, and they provide an intuitive way to model different gaming behaviors [4]. For the concrete implementation we use a behavior tree AI. Its 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 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 2. In our implementation, we created four complex goals that are based on six sub-goals. During the execution of every goal, the AI permanently runs 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

Table 2 Desirability functions for the complex goals

Goal	Desirability function
Combat	$\left(\frac{\text{CSV}}{\text{MSV}}\right) * \left(\frac{\text{VR}}{\text{DTE}}\right) * T_{\text{Combat}}$
Repair ship	$\left(\frac{\text{MSV}-\text{CSV}}{\text{MSV}}\right) * \left(\frac{\text{VR}}{\text{DTR}}\right) * T_{\text{Repair}}$
Capture base	$\left(\frac{\text{BVR}}{\text{CB}}\right) * \left(\frac{\text{VR}}{\text{DTB}}\right) * T_{\text{Capture}}$
Exploration	T_{Explore}

The T_* are adjustable parameters to tweak the AI

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

distance to the target, we implemented the following 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 by selecting either a uniformly distributed random waypoint or a random point of interest (e.g., bases and repair points). Both versions are implemented and evaluated in Sect. 6.

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".

6 Evaluation

In this section, we evaluate the synthetic workloads based on the presented metrics.

6.1 Experimental setup

As the reference workload, we use traces from two real gaming sessions of the game Planet PI4 with 16 and 8 players, respectively. The players, ranging from novices to experienced shooter game players, were divided into two teams and played the game for about 30 min. The trace files contain a timestamp, a unique player ID, the number of neighbors of each player, the number of shots fired, number of hits, and the number of deaths. All values are sampled once per second for each player.

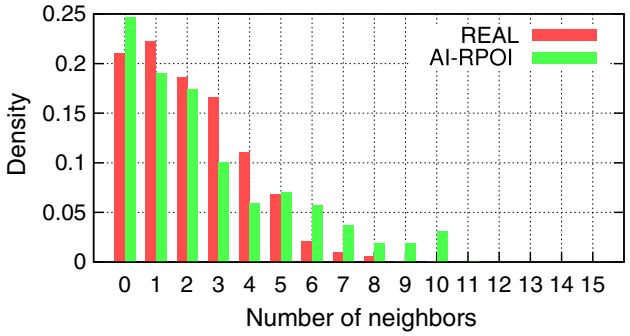


Fig. 3 Visualization of two normalized neighborhood size histograms ($\widetilde{h}$) of a 16-player game

The same experimental setup was then repeated for different numbers players using the different artificial workload generation techniques as described in Sect. 5:

- *RWP* Random waypoint
- *RPOI* Random point of interest
- *AI-RWP* AI player with RWP exploration
- *AI-RPOI* AI player with RPOI exploration
- *REAL* the session of real players.

6.2 Comparing workload generation models

First, we briefly illustrate the different measures for regular events defined in Sect. 4.1. Figure 3 shows exemplary histograms ($\widetilde{h}$, used to calculate $\mathcal{M}_{\text{hist}}$) of the neighbor count distributions of two games. Figure 4 shows a plot of the neighbor count transition matrix $\widehat{A}$ (with all coefficients normalized to 1, used to calculate $\mathcal{M}_{\text{trans}}$) of the real gaming session.

The synthetic workload models were tuned to fit the average number of neighbors of the real game with 16 players, i.e., to a low value of the metric $\mathcal{M}_{\text{mean}}$. For the mobility models, this can be easily achieved by adjusting the effective world size. The AI players show a less linear behavior, making it more difficult to adjust their behavior to a given target.

Figure 5 shows the differences of the games generated by the four workload generators according to the metrics $\mathcal{M}_{\text{mean}}, \mathcal{M}_{\text{hist}}$, and $\mathcal{M}_{\text{trans}}$. The mobility models RWP and RPOI are very similar to the real game with $\mathcal{M}_{\text{mean}}$, but the more sophisticated metrics $\mathcal{M}_{\text{hist}}$ and $\mathcal{M}_{\text{trans}}$ show a

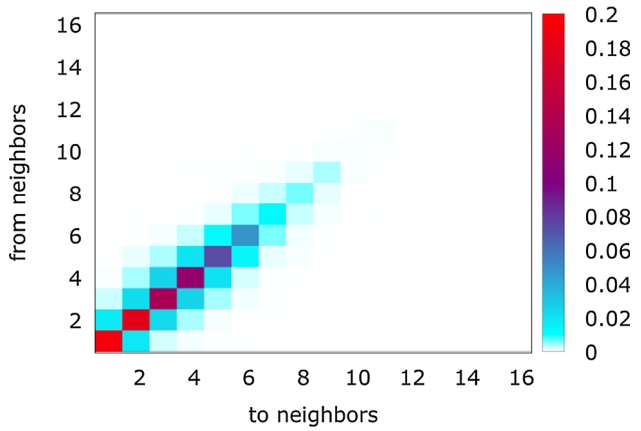


Fig. 4 Visualization of the transition matrix (fully normalized, $\widehat{A}$) between the number of neighbors in two subsequent time steps of a 16-player game (REAL)

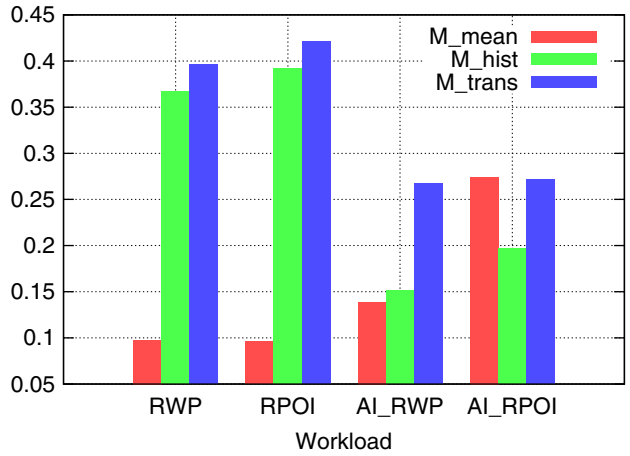


Fig. 5 Differences of the artificial workloads in 16-player games to the real 16-player game, measured using the tree metrics $\mathcal{M}_{\text{mean}}$, $\mathcal{M}_{\text{hist}}$, and $\mathcal{M}_{\text{trans}}$

significant difference. In contrast, the AI models have a higher difference in $\mathcal{M}_{\text{mean}}$ because their tuning is less efficient, but $\mathcal{M}_{\text{hist}}$ and $\mathcal{M}_{\text{trans}}$ show a much higher similarity to the human game.

Figure 6 provides more detail on the behavior of the three metrics. It shows the cross-comparison among human, mobility model, and AI gaming sessions, with three repetitions each. $\mathcal{M}_{\text{mean}}$, visualized in Fig. 6a, shows rather homogeneous differences, except for one outlier with AI-RPOI (workload #3). $\mathcal{M}_{\text{hist}}$ and $\mathcal{M}_{\text{trans}}$, however, show a clear separation between AI and real game on the one hand and mobility models on the other. Within the two groups (bottom left and top right block), the difference is low. With $\mathcal{M}_{\text{trans}}$, the outlier workload #3 is less significant. This metric thus appears to be the best suited for distinguishing the workload classes.

To test the sensitivity of the metrics to changes in the total number of players, Fig. 7 cross-compares gaming sessions with 8–128 players. Obviously, with a change in the total number of players, the average density and the potential neighborhood sizes change as well. But as the figures show, $\mathcal{M}_{\text{mean}}$ is much more sensitive to this than $\mathcal{M}_{\text{hist}}$ and $\mathcal{M}_{\text{trans}}$. The latter two can—to a certain extent—be used to compare traces with varying numbers of players.

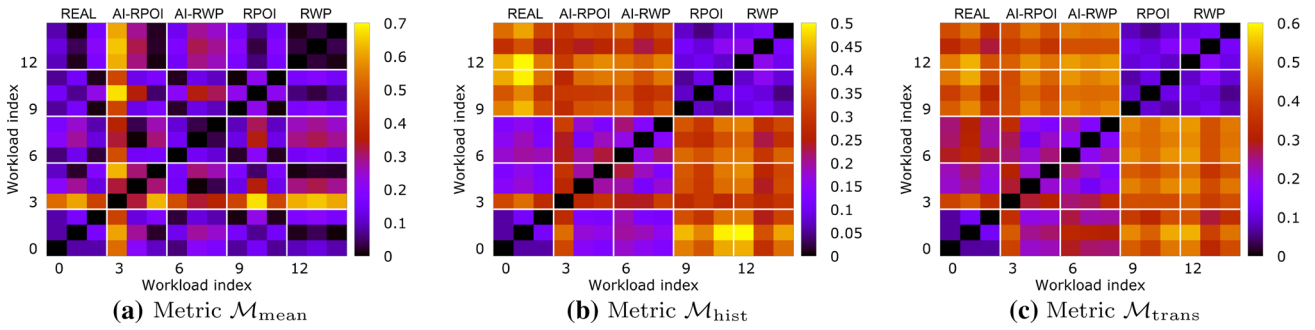
Finally, we analyze the scaling of the workload generation techniques. Figure 8 shows the average neighbor set size depending on the total number of players in the game. Interestingly, with AI players, the neighbor sets grow much slower with the total number of players than with the mobility models. The AI players show the same slope as the human games between 8 and 16 players. For a validation with more players, however, we lack data from a gaming session with a large number of human players.

6.3 Interactions

The second goal of creating representative gaming workloads is to accurately mimic the interactions of the real players. The (inter)actions shooting, hitting, and dying 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 3. 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 generate any interaction events. 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 AI less aggressive while maintaining a good neighborhood characteristic is left for future work.

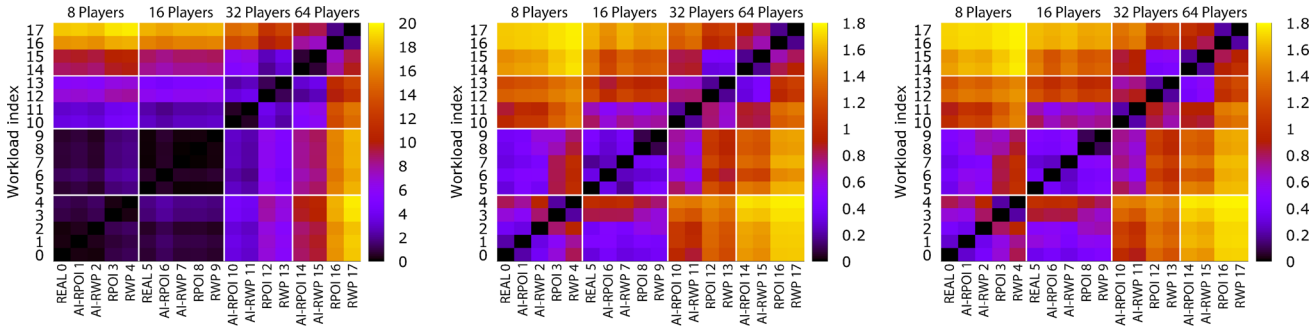
6.4 Network load correlation

Finally, we evaluate how well the workload metrics reflect the actual load induced on the systems. To do so, we measure the network traffic of both a simple client/server and a pSense (P2P) implementation in Planet PI4. We use the network traffic, since this is often the bottleneck in multiplayer online games. Other load factors such as CPU and memory consumption are harder to compare across different types of systems (e.g., client/server vs. P2P). We



(a) Metric $\mathcal{M}_{\text{mean}}$ **(b)** Metric $\mathcal{M}_{\text{hist}}$ **(c)** Metric $\mathcal{M}_{\text{trans}}$

Fig. 6 Cross-comparison between different workload generation techniques for 16-player games with three repetitions each. $\mathcal{M}_{\text{mean}}$ **a** does not show a clear separation between the different techniques. $\mathcal{M}_{\text{hist}}$ **b** and $\mathcal{M}_{\text{trans}}$ **c** separate the mobility models clearly from human and AI workload



(a) Metric $\mathcal{M}_{\text{mean}}$ **(b)** Metric $\mathcal{M}_{\text{hist}}$ **(c)** Metric $\mathcal{M}_{\text{trans}}$

Fig. 7 Cross-comparison between different workload generation techniques and different numbers of players. $\mathcal{M}_{\text{mean}}$ **a** shows a high sensitivity to the number of players. $\mathcal{M}_{\text{hist}}$ **b** and $\mathcal{M}_{\text{trans}}$ **c** are less sensitive and therefore better suited for comparing workloads with different numbers of players

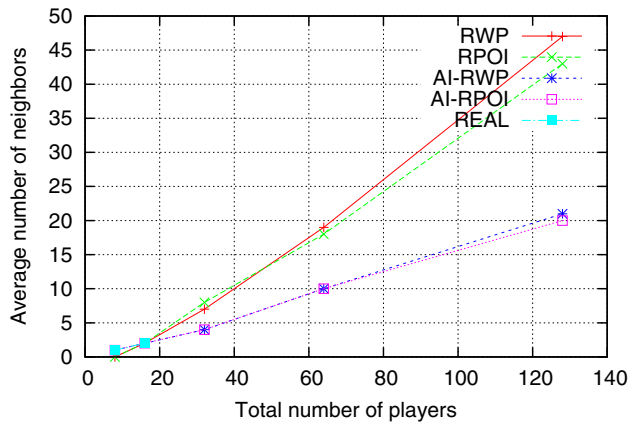


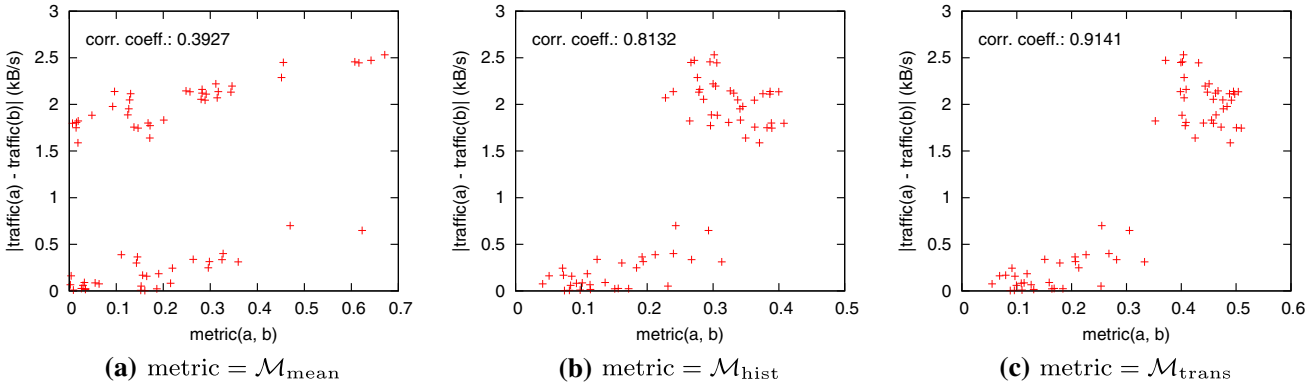
Fig. 8 Average vision range neighbor set size over total number of players. The world size and POI count are constant

measure the total traffic at all nodes in the system, including the server in the client/server scenario. Using traces from various workload configurations as described above as well as real games, we correlate the difference according to the metrics with the difference in network traffic for each pair of configurations.

Table 3 Comparison of the shots/hits/kills per minute and per player as well as the accuracy for the real gaming session and the two AI player workloads for 16 players

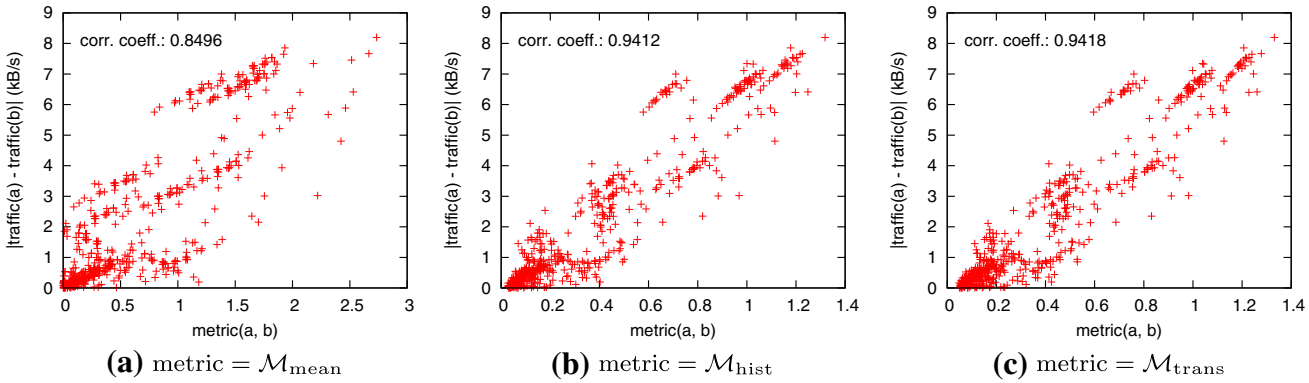
Session	Shots	Hits	Kills	Accuracy (%)
REAL	148.09	41.76	0.41	28.2
AI-RWP	161.19	71.49	0.74	44.3
AI-RPOI	199.48	70.86	0.74	35.5

Figure 9 shows the correlations for the client/server network. The plots' X axes represent the results of the metrics for each pair of traces; the Y axes are the differences in total network traffic. The separation in two clusters in the client/server traces originates from the traffic differences between interactive (i.e., including shooting; human and AI) and non-interactive games (mobility models). Despite these differences, $\mathcal{M}_{\text{hist}}$ (correlation coefficient: 0.81) and even more $\mathcal{M}_{\text{trans}}$ (0.91) show a much better correlation to the generated network traffic than $\mathcal{M}_{\text{mean}}$ (0.39), indicating a better approximation of the actual load. A similar picture is shown for the P2P network measurements (Fig. 10), only with a less apparent clustering. Again, $\mathcal{M}_{\text{trans}}$ and $\mathcal{M}_{\text{hist}}$ (both close to 0.94) are



(a) metric $= \mathcal{M}_{\text{mean}}$ **(b)** metric $= \mathcal{M}_{\text{hist}}$ **(c)** metric $= \mathcal{M}_{\text{trans}}$

Fig. 9 Correlation between metric differences and traffic differences for client/server



(a) metric $= \mathcal{M}_{\text{mean}}$ **(b)** metric $= \mathcal{M}_{\text{hist}}$ **(c)** metric $= \mathcal{M}_{\text{trans}}$

Fig. 10 Correlation between metric differences and traffic differences for pSense (P2P)

clearly better than $\mathcal{M}_{\text{mean}}$ with a correlation coefficient of 0.39. Concluding, $\mathcal{M}_{\text{trans}}$ is slightly better than $\mathcal{M}_{\text{hist}}$, and both are far above the simple $\mathcal{M}_{\text{mean}}$.

7 Conclusions and future work

In this paper, we have discussed and evaluated approaches for generating representative workloads for the benchmarking of multiplayer online gaming infrastructures. For the purpose of assessing the similarity of two given workloads, we developed and evaluated a set of metrics. These allow estimating the similarity to real gaming workloads as well as tuning synthetic workloads according to real workloads. Four workload generators, two mobility models and two AI configurations, were implemented and evaluated in the 3D shooter game Planet PI4.

The tests have shown that mobility models can be tuned well to fit basic properties such as the average neighbor count of a real game session. A realistic density distribution, however, is much harder to achieve with these simple models. In contrast, the AI implementations are less tunable to a certain target value, because they do not have a linear behavior. They are distinctively better in mimicking the density distributions of the human gameplay. This also affects the scaling behavior with the total numbers of players in the game. To validate this scaling behavior further, it will be necessary to perform larger scale games with human players. Finally, interaction between players can only be reproduced by AI workloads.

In our future work, we want to investigate the parameters of our AI more thoroughly to find a setting that meets both requirements simultaneously. The impact of the AI complexity to the generated workload also needs to be investigated. Furthermore, we want to compare the results with other real gaming sessions with more participants.

References

1. Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E., Claypool, M.: 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 (NetGames'04), ACM, pp. 144–151 (2004)
2. Bharambe, A., Pang, J., Seshan, S.: Colyseus: A distributed architecture for online multiplayer games. In: Proceedings of the

3rd conference on Networked Systems Design & Implementation (NSDI'06), USENIX Association, Berkeley, pp. 155–168 (2006)
3. Bharambe, A., Douceur, J.R., Lorch, J.R., Moscibroda, T., Pang, J., Seshan, S., Zhuang, X.: Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In: Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM'08), ACM, pp. 389–400 (2008)
4. Champandard, A.: Behaviour trees for next gen game ai (video part1). http://aigamedev.com/open/articles/behavior-trees-part1/ (2007)
5. Claypool, M., Claypool, K.: Latency and player actions in online games. Commun. ACM **49**(11), 40–45 (2006)
6. Claypool, M., Claypool, K.: Latency can kill: precision and deadline in online games. In: Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems (MMSys'10), ACM, pp. 215–222 (2010)
7. Fan, L., Trinder, P., Taylor, H.: Design issues for peer-to-peer massively multiplayer online games. Int. J. Adv. Media Commun. **4**(2), 108–125 (2010)
8. Gross, C., Lehn, M., Muenker, C., Buchmann, A., Steinmetz, R.: 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, pp. 34–43 (2011)
9. Hu, S.Y., Liao, G.M.:Von: A scalable peer-to-peer network for virtual environments. IEEE Netw. **20**(4), 22–31 (2006)
10. Lang, T., Branch, P., Armitage, G.: A synthetic traffic model for Quake3. In: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE'04), ACM, pp. 233–238 (2004)
11. Lehn, M., Triebel, T., Gross, C., Stingl, D., Saller, K., Effelsberg, W., Kovacevic, A., Steinmetz, R.: Designing benchmarks for p2p systems. In: Sachs K, Petrov I, Guerrero P (eds) From Active Data Management to Event-Based Systems and More, Lecture Notes in Computer Science, vol 6462, Springer, pp. 209–229 (2010)
12. Lehn, M., Leng, C., Rehner, R., Triebel, T., Buchmann, A.: An online gaming testbed for peer-to-peer architectures. In: Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM'11), ACM, demo (2011)
13. Pantel, L., Wolf, L.: On the suitability of dead reckoning schemes for games. In: Proceedings of the 1st Workshop on Network and System Support for Games, ACM, pp. 79–84 (2002)
14. Schmieg, A., Stieler, M., Jeckel, S., Kabus, P., Kemme, B., Buchmann, A.: 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)
15. Svoboda, P., Karner, W., Rupp, M.: Traffic analysis and modeling for World of Warcraft. In: Proceedings of the IEEE International Conference on Communications (ICC'07), pp. 1612–1617 (2007)
16. Terpstra, W.W., Kangasharju, J., Leng, C., Buchmann, A.P.: 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), ACM, pp. 49–60 (2007)
17. Triebel, T., Lehn, M., Rehner, R., Guthier, B., Kopf, S., Effelsberg, W.: Generation of synthetic workloads for multiplayer online gaming benchmarks. In: International Workshop on Network and Systems Support for Games (NetGames'12), IEEE (2012)
18. Yahyavi, A., Kemme, B.: Peer-to-peer architectures for massively multiplayer online games: a survey. ACM Comput. Surv. **46**(1), 9:1–9:51 (2013). doi:10.1145/2522968.2522977
19. Yu, A.P., Vuong, S.T.: 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), ACM, pp. 99–104 (2005)