CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 20xx; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Model driven generation of mobility traces for Distributed Virtual Environments with TRACE

Emanuele Carlini[2], Alessandro Lulli[3], Laura Ricci[1,2]

[1] *Department of Computer Science, Largo B. Pontecorvo, University of Pisa,* [2] *ISTI, CNR, Via Moruzzi, Pisa,*
[3] *DIBRIS - University of Genoa, Genova, Italy*

SUMMARY

Avatars' mobility is an essential element to design, validate and compare different distributed virtual environment architectures. It has a direct impact on the management of such systems because it defines the workload associated with the areas in the virtual world. Currently, a relevant part of this evaluation is conducted by means of synthetic traces generated through mobility models. Despite that, in the last decade, several models have been proposed in literature to describe avatars mobility. However, a standard methodology that drives researchers in their evaluation does not yet exist. In order to alleviate this issue, we present TRACE, an open source tool supporting the generation and analysis of traces by means of embedded mobility models. TRACE's ultimate aim is to facilitate the evaluation and comparison of virtual environments and allow researchers to focus on developing their solution rather than spend time to code and test custom mobility traces. TRACE provides a unified format to describe the traces. It enables scalable and efficient trace generation and analysis for thousands of avatars with seven built-in models. Also, it defines APIs enabling the integration of additional models, different configurations of the environment and several built-in metrics to analyze the generated traces.

Received . . .

KEY WORDS: Mobility model, distributed virtual environment, open source, multiplayer games

1. INTRODUCTION

On-line gaming entertainment has received lots of popularity in the last decade from both industry and research communities. This attention is justified by the economic growth of the sector, in which Massively Multi-player Online Games (MMOGs) such as World of Warcraft [1] or Second Life [2], represent a remarkable member. From a practical point of view, MMOGs are large-scale distributed applications are a synchronous, persistent and interactive virtual environment that allows a huge amount of users worldwide to share a real-time virtual environment. In these applications each individual is in control of a virtual character called *avatar*. The avatar is free to move in a virtual world, perform actions, activities relative to the game and meet other avatars.

One of the largest area of research in this context is to foster the transition of virtual environments from client-server to distributed applications. Such approaches, broadly referred to as Distributed Virtual Environments (DVEs) [30], aim at achieving scalability and cost-effectiveness, by orchestrating the support of the virtual environment to exploit the computational and network resources of the users of the DVE. Common approaches for the definition of the distributed support are based on unstructured [19, 31] and structured [5, 21] peer-to-peer technologies, as well as more centralized technologies like cloud computing [29].

*Correspondence to: ISTI, CNR; Pisa

In such distributed approaches, a user resource (i.e. peer) concurrently works both as client for a (usually single) avatar and as server for a part of the virtual world. This organization allows to exploit the inherent scalability of P2P systems, but it raises a number of challenging issues. For example, there is the need to find a proper way to partition the virtual environment among the user resources, both in terms of number and size of partitions. Further, an infrastructure supporting DVEs has to implement proper consistency preserving mechanisms in order to make sure that all clients have a consistent view of the virtual world. This problem is exacerbated by the fact that all communication largely takes place on the Internet, consequently suffering from unpredictable jitter and delays, and hence making perfect consistency hard to achieve. In fact this problem is tackled by tuning the tradeoff between consistency and interactivity [7, 12], whose values depend mostly on the type of virtual world running on the DVE, including the movement of the avatars.

Another typical issue in DVE is the propagation of the relevant events generated by the participants to the set of interested avatars, which is typically a subset of the whole set of participants. This issue, commonly referred as Interest Management (IM) [18] poses well recognized scalability challenges, due to the high number of participants and to the distributed nature of the DVE. Nevertheless IM is mostly affected by the movements of the avatars, and by their interactions which, in turn, depend on their position and surroundings. Moreover, the position of the avatars can affect the distribution of the DVE. For example, in Voronoi-based DVE approaches, the management of the DVE is assigned to the hosts of the avatars according to a tessellation of the virtual world, which depends on the position of the avatars [19, 31]. When avatars move, the assignments change accordingly, triggering a reconstruction in the distribution of the DVE.

As a matter of fact, the representation of avatar movements is an essential feature to properly design, validate and compare different DVE architectures. Normally, two widespread ways are exploited to represent avatars' movements (often referred to as *traces* [27]): *real traces* taken from an instance of a running virtual environment application, or *synthetic traces* generated from mobility models, that try to simulate the behavior of the avatars. Real traces are usually a good mean of validation, as they provide the actual behavior of avatars in virtual environments. However, obtaining real traces can be difficult, as it often requires to have access to the data of a commercial application. A few online repositories allow access to such data. For instance, *The Game Trace Archive*[†] [16] to date makes available 14 different traces of popular virtual environment. However, it can be difficult to collect a large and heterogeneous number of real traces and, especially, they may be not suitable to validate the DVE on extreme and specific scenarios. On the other hand, synthetic traces generation provides a simulation of avatars' movements, and present many advantages with respect to real traces [37]. First, they provide good means to test the scalability in terms of avatars number, often going beyond the actual capacity of real world virtual environments; second, synthetic traces are reproducible and easily embeddable in different applications and configurations, providing a common ground toward comparison; third, they aim to realism, with a number of mobility models released to capture the peculiar behavior of different kind of avatars and virtual environments (see Section 2).

However, in the majority of cases, synthetic traces are generated with custom ad-hoc solutions, which raise several drawbacks: (i) it is hard to compare different systems on the same scenario, as exact details on how the traces are generated are usually not released; (ii) researchers spend time to code and test traces and trace generators, rather than improving their solutions; (iii) there are no clear reference mobility models that are targeted by the DVE community; (iv) it is time-consuming to reuse traces because, usually, they are encoded using specific formats.

In order to overcome these drawbacks, we have developed TRACE, a Java software library for the generation of avatar movement traces aimed to an easy integration and portability among different systems and approaches. TRACE is a tool for the fast and easy embedding of already designed mobility models in DVE architectures, that promises to follow the "write once, run anywhere" philosophy. Apart from mobility traces generation, TRACE can be used to support the creation of

[†] `http://gta.st.ewi.tudelft.nl/`

DOI: 10.1002/cpe

new mobility models from real traces due to its flexibility, however in this presented version of TRACE we do not claim to tackle directly such challenge.

TRACE presents the following characteristics: (i) is able to generate mobility traces for a wide variety of included DVE-based and human-based mobility models, and provides interfaces for the definition of personalized mobility models; (ii) exports traces in a simple, unified and reusable format, so to foster further uses and comparisons; (iii) is designed to be fully configurable, in order to adapt to heterogeneous approaches and scenarios; (iv) comes with many additional tools (e.g. trace analyzer and visualizer) to help creating the right trace and analyze its features. Initially, we have used TRACE for internal research (such as in [12, 9]) but eventually we have made it available for the whole DVE community[‡].

1.1. Contribution

In this paper we present the main features and characteristics of TRACE, unraveling the relevant under-the-hood that makes it easy and practical to use. We provide an overview of the API for the definition of personalized mobility models, corroborated by two use cases: (i) BLUEBANANA, a mobility model for Second Life [25], and (ii) SYMPATHY, an original mobility model for Multiplayer Online Battle Arena (MOBA) based on attraction forces. We also provide an evaluation of TRACE that covers generation performances and trace analysis.

This paper is an improved version of a previous paper from the same authors [10]. With respect to that paper, we extended our work with the following new contributions:

- an improved version of TRACE, which includes trace analysis, the possibility to combine multiple models in the same trace, and performance improvements;
- a comprehensive related work study on mobility models and mobility traces analysis for DVE;
- the implementation in TRACE of a new mobility model for MOBA called SYMPATHY.

The paper is structured as follows. Section 2 discusses the related work in the field of mobility analysis and mobility models for DVEs. Section 3 describes TRACE from a design and implementation viewpoint. Section 4 introduces SYMPATHY, a novel mobility model for DVEs. Section 5 presents the evaluation of TRACE and the analysis and comparison of a number of mobility traces. Finally, Section 6 concludes the paper.

2. RELATED WORK

When it comes the need for evaluating DVEs, the universal strategy is to check the DVE support against the behavior of the avatars. Common ways to perform this operation are to exploit directly the (network) load of a realistic DVE, or to derive the load for the DVE support using avatars mobility traces. Suznjevic et al. [35] employ the former approach by proposing a network traffic generator based on the action undertaken by the avatars in a large DVE. Simulating a realistic traffic footprint from avatars' behaviors, although useful to evaluate the network support, can skip relevant issues related to other aspects of a DVE framework. Indeed, embedding mobility inside the DVE test environment can help to evaluate and compare additional aspects of the support, such as state consistency and distributed interest management. Generally, two approaches are followed when exploiting mobility traces, i.e. real and synthetic traces. In case of real traces, the Game Trace Archive [16] collects different real traces (14 to date) defining a common format so that these can be easily used.

As outlined in the introduction, the motivation of using synthetic traces to complement real traces for evaluating DVE have many advantages, such as reproducibility, testing particular scenarios and comparing different solutions. Many approaches such as VON [19], Mopar [40], pSense [32] and

[‡]publicly available at: `https://github.com/hpclab/trace`

DOI: 10.1002/cpe

Gross *et al.* [15], have employed random based walker models or random walks between hotspots. Those models are popular thanks to their simplicity: a random walk model only requires a few lines of code and the community has generally accepted it as a model able to describe several simple gaming scenarios.

Over the years, mobility models for virtual environment of increased complexity have been defined thanks to the effort put in the task of analyzing mobility traces. Such analysis borrowed tools from human mobility analysis and opened the road for the creation of more complex mobility models. In [26] authors propose a statistical analysis of Second Life traces as well as a discussion on the implication about the design of a DVE framework. The analysis is performed characterizing both the mobility (Avatar speed, pause time etc..) and contact patterns (AOI sizes, etc..). We equipped TRACE with a statistical analyzer for newly generated traces that provides such statistical features, whose description is found in Section 3.3. In [27] and [28] authors provide an analysis of avatar mobility for World of Warcraft. The analysis is focused on a particular area of the DVE where avatars battle for the control of several objectives. The paper presents a modeling of the avatars' behaviors in terms of hotspots, grouping and waypoints. In [20] authors advocate synthetic traces as a necessary complement to real world traces for the evaluation of DVEs. They identify five features characterizing avatar mobility, namely: distribution, pause time, distance, grouping and mobility constraints. They compare several mobility models against such features. Instead, in [33], authors characterize mobility models in DVE according to several features (including: flight lengths, pause duration, area popularity, invisible boundary of human movements, personal area preferences) and compare human traces with DVE traces. An interesting found is that personal preferences are more pronounced in human traces.

Recently, mobility models try to capture many of the features described above. BlueBanana [25] is inspired by the virtual world defined by Second Life. In this world, players gather around a set of hotspots, which usually correspond to towns, or, in general, to points of interest of the virtual world. Similar to the previous, a subset of the authors of this work defined a mobility model called WOW [8]. WOW considers also hotspots where players are placed at the start of the game and spawn after death. This model takes into account the team-oriented nature of the scenario, where moving in group is encouraged by the game semantics. However, an avatar may decide to move alone by itself, for instance to take the enemy by surprise. In [36] authors propose an enhancement of the random way point mobility model to better fit the behavior of players in a first person shooting game (i.e. Quake 2). Among the changes, they added different conditions for an avatar to be stationary, an hotspot popularity and non-straight movement paths. In [34] authors develop a DVE mobility model based on WOW and Second Life. Each avatar has a personalized path of movement among the set of hotspots.

Also, using the Least Action Planning trip (LAPT) [24] the avatars select hotspots in close proximity with higher probability. When an avatar visits an hotspot, it stays there for a time drawn from a truncated-Pareto distribution and then moves to another hotspot. In RPGM [17], each player belongs to a group and it moves by following the movements of its group, in order to model the players habit to gather in teams. All the above models have been defined and used on specific scenarios. However, we think it is important to unify how the models are generated and how they are used.

Finally, Triebel *et al.* [37] study both the mobility of avatars and their interactions. They compare the movement of avatars guided by mobility models versus movements generated by artificial intelligence techniques. Although the latter provides better results, using simple mobility models such as random way point and a random model based on hot spots, give close results, in particular the one based on hot spots. Artificial intelligence movements take into account also the context of the game, must be built specifically for each game and they base their movement on the mobility models. For all these reasons, although specific solutions may get marginal improvements on the validation, we think that the generality of the mobility models is an important way to validate games.

As demonstrated by the above works, many researchers provided practical analysis of mobility traces. Unfortunately, often these analysis are limited to a specific evaluation of DVE and are not shared and/or easily embeddable in other systems. Table I shows a list of selected popular and

DOI: 10.1002/cpe

Approach	Year	Mobility Strategy	Public Traces
VON [19]	2006	Avatars move randomly in the virtual world	No
Donnybrook [5]	2008	Quake III simulated movements feed a workload generator [6]	No
Badumna [23]	2010	Avatars move randomly in the virtual world	No
Watchmen [39]	2013	Quake III real traces with proprietary trace-based simulation environment	No
Kiwano [13]	2016	Movements inspired to BlueBanana [25]	No

Table I. Mobility models and trace availability of popular and recent DVE approaches

recent works on DVEs and their approaches towards mobility. Such proposals can be classified in two categories: (i) the embedding of traces in the system, (ii) the use of self made workload generators feed with some traces. A common methodology based on the same mobility models is clearly missing. Even recent approaches find more convenient to base their evaluation on custom implementation, suggesting that there is no easy way to embed traces prepared by somebody else in one's own DVE. The ultimate aim of TRACE is to cope with such very issue by providing the community a tool that goes in the direction of make as easier as possible to create, embed and share mobility traces for DVE evaluation.

3. THE TRACE TOOL-KIT

TRACE is an open-source Java library to generate mobility traces in a two-dimensional space. Although it has been originally conceived for the movements of avatars in a virtual environment, it is flexible enough to be used in other contexts in which a number of entities move across a 2D area. TRACE has been primarily designed with the idea of focusing on experimentation and evaluation of distributed virtual environments, therefore most of the terminology used in this section refers to such field. Avatars are the digital agents of the users in the virtual environment and are associated with a position in the virtual world. They are the moving unit considered in TRACE. Other than avatars, TRACE gives the possibility to specify *static* entities, namely passive objects and hotspots. Passive objects are entities that have a state and can be interacted by avatars (e.g. doors), but unlike avatars are not controlled by an human user. The hotspots are those areas of the virtual environment corresponding to places of interest and where usually is present an higher density of passive and active entities.

The main design characteristics of TRACE are the following:

Easy to use and extend. TRACE has been designed with the idea of being usable right off the box. It comes with 7 mobility models already implemented (see Section 3.2 for more details), of which 5 are specific for DVEs. Furthermore, TRACE provides an API to facilitate the creation of personalized mobility models. All the models, including the additional ones, can be combined such that each avatar may follows a different mobility model inside a single trace.

Reusable. TRACE provides a neat separation between the mobility models and the virtual area (i.e. the map). This allows for a high degree of reusability, as the same models can be used in different DVE scenarios. Also, TRACE unifies the output of the models to facilitate the adoption and testing of different trace models.

Scalable and efficient. TRACE makes use of specific solutions to generate traces in a resources friendly fashion. In order to speed up the computation, the generation of traces is done in parallel when multi-core processors are available. Moreover it minimizes the memory footprint, by keeping in memory only the current state of avatars. These aspect allow TRACE to scale to thousands of avatars.

DOI: 10.1002/cpe

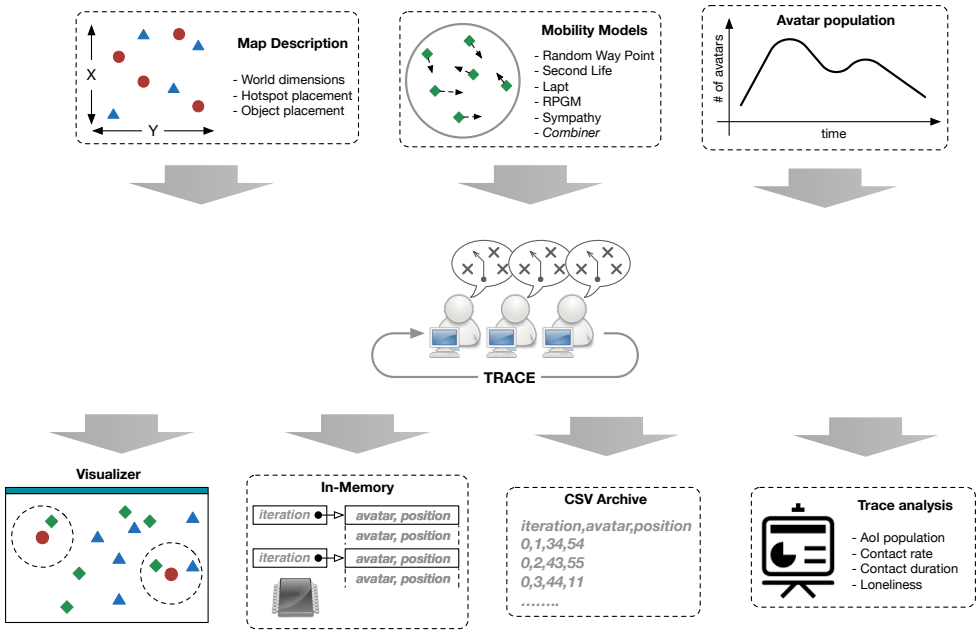


Figure 1. TRACE overview: inputs and outputs

3.1. TRACE *overview*

Figure 1 shows a high level overview of TRACE, with a focus on its inputs and outputs. Table II provides a list of the most relevant interfaces for the personalization of TRACE. All the inputs are defined by means of a *configuration* file composed by a list of key-value tuples that contains all the necessary information for the generation of the traces. Inputs can be grouped as map description elements, mobility models and avatar population definitions.

The *map description* defines a rectangular area representing the virtual environment two-dimensional space. It includes the position of hotspots and how to assign the position of the passive objects. Note, both hotspots and passive objects can influence the movements of the avatars, but they are not essential for the generation of the traces. By default, hotspots and passive objects are uniformly randomly placed in the virtual world. Nevertheless, their placements is totally configurable via the class `AStaticPlacement`, which can be extended to place static entities according to a user-defined function.

The specification of the *mobility model* is the core part of the configuration. TRACE gives the possibility to choose one of the available mobility models, create a new one, or any combination of them. It is also possible to define personalized mobility models by implementing the `AMobilityModel` interface. Additional details regarding mobility models are given in Section 3.2. TRACE comes bundled with the following mobility models already implemented and ready to be used (more mobility models are under development and will be added in the

Table II. Relevant interfaces and classes of TRACE.

`AMobilityModel`	Abstract class to define mobility models. The core method is `move` in which the movement of the generic avatar is defined according to the iteration
`AStaticPlacement`	Abstract class to define placement function for static entities, such hotspots and passive objects. This class is called once during the initialization of the virtual environment
`IAvatarNumberFunction`	Interface to define the amount of avatars at any iteration. It is called by the engine before the computation of each iteration to adjust the avatar population

DOI: 10.1002/cpe

Name	Movement of Avatar a at time t	Hotspot	Avatar	DVE	Original
RW	a moves to a random direction with a constant speed.				
RWP [3]	a moves toward a random point p in the map with a constant speed. When a reaches p, it starts moving to another random point p'.				
LAPT [24]	a moves toward a random hotpsot h. When a reaches h, it starts moving to another hotspot h'.	✓		✓	
BLUEBANANA [25]	a moves to a random point in the map. Points in hotspots have a different probability to be reached. a may be in three state: halt, moving, exploring. a moves differently in the three states.	✓		✓	
RPGM [17]	a forms a group with other avatars having a leader l. l choose a random point in the map. All the avatars, in the group of l, moves to the same destination.		✓	✓	
SYMPATHY	In different moments a is attracted / repelled / stable with respect to other avatars. Such attractions follow the Coulomb laws, where a is an electrical charge.	✓	✓	✓	✓
COMBINER	a may move according to one of the models present in TRACE. The models and the subsets of the avatars moving according to a model is configurable.				✓

Table III. The mobility models built-in in TRACE with a brief description about how an avatar a moves according to each model. Hotspot column identifies the models taking into consideration the hotspots, during the movements. Avatar column identifies models considering other avatars when defining the movement of a. DVE column identifies models originally conceived for DVEs, while original those presented in this work.

future): *(i)* Random Way Point (RWP) [3], *(ii)* Random Walk (RW), *(iii)* Least Action Planning Trip (LAPT) [24], *(iv)* BLUEBANANA [25], *(v)* Reference Point Group Mobility (RPGM) [17], *(vi)* SYMPATHY (see Section 4, *(vii)* COMBINER. Table III presents a brief description of the main characteristics of the different models implemented in TRACE.

Avatar Population defines the amount of avatars in the virtual environment over time. The default behavior is to have the same number of avatars for the entire trace. However, usually DVEs exhibit a scaling up and down of the number of avatars due to the churn of the players. Due to this, it is possible, by implementing the `IAvatarNumberFunction` interface, to control the number of avatars present in the DVE at each iteration. Note, TRACE allows a fine grained control of the churn, as it is possible to understand which avatars leave (or enter) the DVE. To make this possible, the avatars' identifiers are kept consistent across iterations.

When TRACE is launched it creates the mobility traces, iteration by iteration, completing each avatar movement before dealing with the next iteration. The `MapVirtualEnvironment` object stores *in-memory* all the information about movements of the avatar, hotspots and passive objects. This class can be accessed in a read-only fashion to be used right away when the generation of the traces is done contextually to the DVE evaluation. With the *visualizer* option activated, TRACE opens a windows with the graphical representation of the avatars moving across the map. Although this option may slow down the generation of the traces, it results very useful to tune the parameters of a mobility model in order to obtain specific behavior from avatars. If the corresponding option is active, a file containing all the information relative to the traces is saved on disk. Regardless of the model used, TRACE builds a compressed archive consisting of the following files: *(i) configuration*, which contains all the variables to replicate the scenario; *(ii) avatars*, which stores the movement of the avatars; *(iii) hotspots*, which stores the position of the hotspots; *(iv) objects*, which stores the position of the passive objects in the game; *(v) bandwith*, which provides statistics regarding the number of passive objects in the Area Of Interest (AoI) of each avatar; *(vi) aoiStatAVG*, which stores statistics regarding the AoI of avatars (see Section 3.3). In particular, the *avatars* file contains a snapshot of the position of all the avatars in each time step in a CSV format containing the following values: time step, unique avatar identifier, position of the avatar in the map as a couple (x, y). The resulting file can be loaded at a later time to be used in different experimental evaluation.

DOI: 10.1002/cpe

Algorithm 1: *AMobilityModel.move()* implementation: BlueBanana

```
Input  : map: a Map representing the virtual world
         t: the current time
         avatarList: the avatars position at time t − 1
Output: the position of the avatars at time t
List next = avatarList
forall Avatar a∈avatarList do
    State nextState = markovChain.getNextState(a, markovChain.getState(a))
    if nextState = E then
        Point current = a.getPosition()
        next(a) = current.explore()
    else if nextState = T then
        Point t = map.getRandomPoint()
        Point current = a.getPosition()
        next(a) = current.moveToward(t)
    else
        do nothing
    end
end
return next
```

3.2. Integrating mobility models

The definition of a mobility model is one the core parts of TRACE, and can be done by extending the `AMobilityModel` interface. A mobility model defines how a generic avatar moves within the boundaries of the virtual environment, and this behavior is then replicated for all avatars in the DVE. TRACE considers discrete time iterations, and at each iteration avatars move according to the mobility model specified. In particular, during iteration t avatars move independently without the knowledge of each other position at iteration t; however they can have a read-only access to the positions of the avatars at iteration $t-1$.

A common issue when generalizing the generation of traces is that any mobility model can have its own configuration with specific parameter. TRACE resolves this issue by allowing a free definition of the parameters inside the configuration file, leaving to the developer the responsibility of matching the correct parameter within the implementation of the mobility model. For example, the Blue Banana mobility model (whose implementation is described in detail in Section 3.2.1) is heavily focused on hotspots and therefore define specific properties such as the probability for an avatar of being inside the area of an hotspot.

In the remaining of this Section we provide an in-depth description of how it is possible to implement a mobility model within TRACE. In particular, we report, in Section 3.2.1, how to implement BLUEBANANA. Next, we describe two novel mobility models introduced for the first time in this work, namely, SYMPATHY (Section 4) and COMBINER (Section 3.2.2). These are respectively modeling Multiplayer Online Battle Arena games taking inspiration by the electrical attraction between electrical charges according to the Coulomb law and the idea of combining multiple models to provide different movements to different subsets of avatars.

3.2.1. Case Study: Blue Banana BLUEBANANA is a mobility model presented by Legtchenko et al. [25], which simulates avatars movement in a commercial DVE, Second Life [2]. In their paper, authors of BLUEBANANA provide details on the model characteristics and behaviours, but no actual implementation. We have provided a preliminary implementation of this mobility model, as well as a comparison with other mobility models in [8]. Here, we give a description of the model (taking inspiration from [7]) and subsequently present the details of its implementation within TRACE.

In BLUEBANANA, avatars gather around a set of *hotspots*, which usually correspond to towns, or in general to points of interest in the virtual environment of Second Life. Each hotspot is modelled as a circular area characterized by a center and a radius. Traces generation goes through two phases: *initialization* and *running*.

DOI: 10.1002/cpe

In the initialization phase, the area of the virtual environment is divided in *hotspot areas* and *outland areas*. The percentage of the hotspot area is defined by p_{hot} and, consequently $1 - p_{hot}$ represents the outland area. The hotspots are placed randomly in the virtual environment. Their radius is computed such that the total area covered by the hotspots is in accordance to p_{hot}. The parameter p_{den} defines the probability that an avatar would be initially placed in an hotspot, whereas $1 - p_{den}$ defines the probability for an avatar to be initially placed in outland. If the avatar is placed in the outland, its position is chosen uniformly at random on the whole map. Otherwise, an hotspot for the avatar is randomly selected. The position inside the hotspot is chosen by considering a Zipfian distribution, so to ensure an higher density of avatars near the center of the hotspot.

The running phase moves the avatars across the virtual environment. The movements are driven by a Markov chain, one for each avatar, whose states are the following:

- *Halt*(H): the avatar remains in place;
- *Exploration*(E): the avatar explores a specific area. If the avatar is moving inside an hotspot, the new position is chosen according to a power law distribution. Otherwise, the new position is chosen at random;
- *Travelling*(T): the avatar moves straight toward another point in the virtual environment. The new point is chosen in accordance with p_{den}.

Initially every avatar is in state H. At each step t, the model decides the new state according to the probability of moving between states defined in the Markov chain. This mobility model exposes a fair balance between the time spent by avatars in hotspots and outland.

To integrate such model in TRACE the following steps are required:

- *configuration*: it is required to load all the model specific configuration variables such as p_{hot}, H_{num} and p_{den};
- *additional functionalities*: since this model requires a Markov Chain to move the avatars between different states, i.e. (H, E, T), we implemented an utility class to easily know, given a state, which is the next state of the avatar;
- *AMobilityModel*: the core of the model is the implementation of the *AMobilityModel* interface. Specifically, it is required to implement the *move* method where TRACE provides the position of the avatars at time $t - 1$ as well as an object describing the virtual environment where is possible to find the position and size of the hotspots and objects. The model must return the position of the avatars at time t.

Algorithm 1 is an example of the *move* method's definition for BLUEBANANA. For what concerns the initialization phase, our implementation at time 0 follows the specification of the initialization phase provided in the original paper. During the running phase, we generate a new position for each avatar (Line 2) and the new state of the avatar according to its previous state (Line 3). Based on the next state, we follow the specification of the model for the *Travelling* state (Line 7), *Exploration* state (Line 4) and *Halt* state (Line 11). We collect all the new positions in a list and we return all the new positions (Line 8).

Finally, to use the new implemented model, it is required to modify the configuration of TRACE, giving a name to the new model, for instance "BlueBanana", and providing the package and class name where it is implemented. Next, it is necessary to set, in the configuration, the property "model BlueBanana", as well as all the configuration parameters required by the model. The execution will use the selected model and generate the traces accordingly.

3.2.2. Combining mobility model togheter The last mobility model introduced is called *Combiner*. The idea is that, commonly, avatars can move according to different behaviors, according to their role in the DVE [36]. For example, let us consider a popular DVE like World of Warcraft, in which coexist avatars that like to discover the virtual world, others willing to search other avatars for

DOI: 10.1002/cpe

combat and others just improving their skills. In such scenario the avatars move according to different mobility models.

Combiner aims at modeling such scenario and permits to choose different mobility models from the ones available in TRACE and the percentage of avatars moving according to each of the chosen models. For instance, an example of configuration required by Combiner, is the following:

```
MODEL combiner
COMBINER_MODELS randomwalk,lapt,bluebanana
COMBINER_PROBABILITY 0.2,0.4,0.4
```

In this case, Combiner is configured setting the 20% of the avatars moving according to Random Walk, the 40% with Lapt and the others (40%) with BLUEBANANA.

3.3. Trace statstics

TRACE provides tools to analyze traces in terms of the relationships of avatars among each other by exploiting the concepts of AOIs and contact. Several of the measures taken by TRACE can be found in researches related to ad-hoc networks, especially in terms of *contacts* among entities [22]. In such context, contacts are important as they represent the moment when two entities can communicate and exchange data. Rather differently, from a DVE perspective, contacts among avatars are important, because two contacting avatars share the same spatial interest, and their knowledge can be useful to each other. Therefore, the rate and the duration of contacts can impact both on the design and the behavior of a DVE architecture. For example, in scenarios in which avatars work as points of centralization for their AOI [11], the analysis of contacts and AOIs are crucial measures. In TRACE, we gather four metrics about AOI and AOI contacts: population size, loneliness, contacts rate and contacts duration.

We define P_a as the set of avatars in a's AOI (excluding a) during an interval period T. AP is defined as the average AOI population for all the avatar in the virtual environment. When $|P_a| = 0$ an avatar is said to be alone, with L being the set of alone avatars. In TRACE, we register an AOI contact when an avatar enters in the AOI of another avatar. We represents with C_a the amount of AOI contacts experienced by an avatar a during an interval period T. The average contact rate CRA is the average number of new AOI contacts experienced by all avatars in the DVE during T, and it is defined as following:

$$CRA = \frac{1}{N} \sum_{n=1}^{N} \frac{C_n}{T} \tag{1}$$

Similarly, the average AOI contact duration CDA is the average of all the terminated contacts of all avatars during T. A terminated contact is registered by TRACE when an avatar exits from the AOI of another one. It is defined as following:

$$CDA = \frac{1}{N} \sum_{n=1}^{N} \frac{\sum_{z \in Z_n} z \times \Delta t}{|Z_n|} \tag{2}$$

where Z_n is the set of all terminated contacts of avatar n.

TRACE records the values for the above metrics during the generation of the traces. Such statistics, are stored in the same archive with the mobility trace itself for later use and comparison. In Section 5.6 we compare such metrics resulting from different mobility models.

4. SYMPATHY: A NOVEL MOBILITY MODEL

Normally, mobility models consider how to perform avatar movements between the source and destination points within the DVE, without considering how the presence of other avatars impact

DOI: 10.1002/cpe

on the movements. However, in many situation movements of avatars are driven by the presence of *other* avatars, not only considering their position but also the social role they play in the DVE.

Here we briefly present SYMPATHY, a mobility model that, in addition to hotspots and positions of avatars, also considers the relationship among avatars to simulate their environment. To this end, SYMPATHY models the movements as the consequence of attraction (and repulsion) forces among avatars, just like electric charges attract and repel in the real world. In order to define the social role within the DVE, SYMPATHY targets a specific class of online games called Multiplayer Online Battle Arena (MOBA), in which two factions battle to control an area of the DVE. In the following, we provide a description of a general MOBA game and the mobility model to generate traces accordingly.

4.0.1. Multiplayer Online Battle Arena Multiplayer Online Battle Arena (MOBA) is a genre of online games in which players control a single character in one of two factions. The objective is to destroy the opposing faction structures, usually following predetermined paths. Such genre recently received a lot of popularity as e-sport, with many world championships organized periodically [§].

A MOBA map is generally a squared area, in which avatars move mostly along predetermined paths that go thorough faction structures, which, in turn, represent hotspots. Figure 2 shows the map of two popular MOBA games *Heroes of Newerth* and *League of Legends*. Generally, in a MOBA game, avatars start weak and acquire power and abilities over time, by completing various objectives. This kind of advancements affects the strategy of the players with a consequence on the relationship among players and their movement.

By the analysis of a number of *Heroes of Newerth* games, we have noticed that a game can be divided in three main phases: the *beginning*, the *skirmish* and the *final battle*. These phases are characterized on how the players of different faction related to each other:

- *beginning*: in this phase each player tries to acquire skills and power as fast as possible, usually traveling alone or together with few components of its faction. In this phase the contact with players of the opposite faction is avoided, and battle among players are fast.
- *skirmish*: in this phase avatars coordinates in small groups to defeat opponent's structures at the hotspots or battle against other group of players.
- *final battle*: in the last phase, the majority of the avatars of both the teams, aggregate in large groups to achieve the final objective.

The behavior of the avatar changes during each phase, according to the relationship they have with the other avatars and hotspots. In the following, we describe how these relationship, under the form of attraction and repulsion, are the foundation of SYMPATHY.

4.0.2. The SYMPATHY *model* The core idea of SYMPATHY is to model the relationship among avatars by making an (inverted) analogy between avatars and electrically charged particles. The idea is to assign to a faction a positive charge, to the other faction a negative charge. In a neutral condition, if two avatars have the same charge they attract themselves; if the charge is different they repel themselves (note the analogy is inverted as with particle is exactly the opposite). Also, the attraction (repulsion) works inversely proportional to the distance: a closer avatar would attract more than a far one.

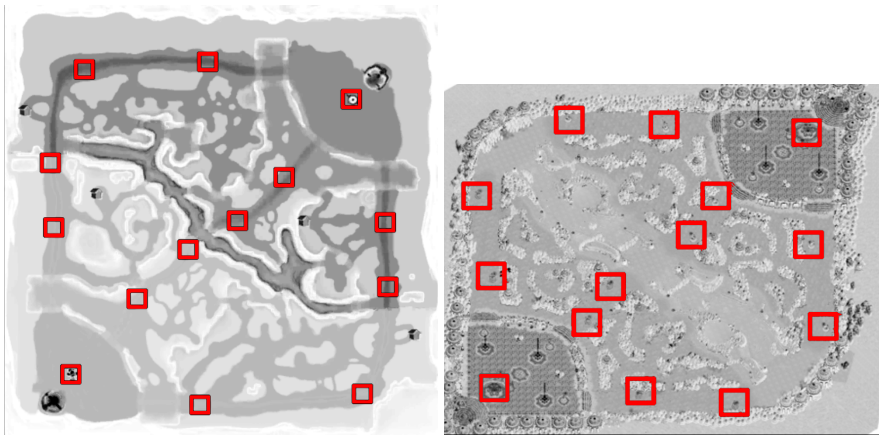
Given two electrical charge u and v, the electric vectorial force of u on v follows the equation (according to the Coulomb's law):

$$\overrightarrow{F}_{v,u} = k\frac{vu}{r^2}\widehat{r} \tag{3}$$

where k is the Coulomb's constant, r^2 is the distance between u and v and $\widehat{r} = r/|r|$ is the versor of the positional vector. In order to consider the effect of all the avatars, we exploit the superposition

[§]https://en.wikipedia.org/wiki/2016_League_of_Legends_World_Championship

DOI: 10.1002/cpe



(a) Heroes Of Newerth. (b) League Of Legends.

Figure 2. Examples of MOBA games map, hotspots are represented as squares

principle, which state that for point charges the force acting on a sigle charge is the vector addition of all the individual forces:

$$\overrightarrow{F} = \sum_{i=0}^{N} \overrightarrow{F}_i \tag{4}$$

where $\overrightarrow{F}_i$ is the i_{th} vector applied to the charge considered.

This model allows us to define a situation in which only the initial charge assignment affects the movement of avatars. However, over time, the relationship between players can change. For instance, a player can be attracted, instead than repelled, by a player of a different team for many reasons (e.g. they are fighting). We extend the model (Equation 3) to capture such variable nature of relationship among avatars over time. The improved model, which represents the force on v from u at time t is the following:

$$\overrightarrow{F}_{u,v} = \delta^t_{u,v} \frac{Q_u Q_v \alpha_{u,v}}{(r - r_{min})^2} \widehat{r} \tag{5}$$

The additional parameters are the following:

- the **aggregation coefficient**, which we define as $0 \leq Q_i \leq C$ for the generic avatar i. It represents the tendency of avatars to be in a group with others. In general, higher values of Q represent a tendency to be and move in groups, whereas lower values represents a solitary behavior In SYMPATHY we built proper values of Q by sampling players positions from a number of Heroes of Newerth game sessions. We then created a graph of players for each unit of time, and considered the distribution of the average *clustering coefficient* [38] of the players. The value of a generic Q is then computed by sampling the distribution of clustering coefficient and multiplied such value (which is between 0 and 1) by a constant C which depends on the specific scenario considered (in our evaluation $C = 10$).

- the **direction matrix**, which we define as δ and represents the direction of the force vector to apply. More in details, $\delta^t_{u,v}$ defines the direction for avatars u and v expressed at time t. It can have three values: to model *attraction* $\delta^t_{u,v} = -1$, such that u tends to move toward v; to model *repulsion* $\delta^t_{u,v} = 1$, such that u tends to move opposite to the direction of v; to model *stability* $\delta^t_{u,v} = 0$, such that the distance between u and v tends to be constant. This matrix helps in modeling the three phases of the MOBA games described above. Note, that in general the direction matrix can be asymmetric (i.e. $\delta_{u,v} != \delta_{v,u}$) . This models the situations in which an avatar a is chasing another avatar b (i.e. a is attracted by b and b has a repulsion effect to a).

DOI: 10.1002/cpe

Algorithm 2: *AMobilityModel.moveThreadBased()* Thread-Based Pseudo-code

Input : *map*: a Map representing the virtual world
t: the current time
avatarList: the avatars position at time $t-1$
executor : ThreadBased executor
Output: the position of the avatars at time t

```
List next = avatarList
foreach Avatar a∈avatarList do
    next(a) = executor.submit(AMobilityModel.move(t, map, a, avatarList))
end
foreach Avatar a∈avatarList do
    waitCompletion a
end
return next
```

Also, the direction matrix can change during the simulation to model the situation in which relationships between the entities change over time.

- the **preference matrix** that defines the personal preference among avatars, which we define with $\alpha_{u,v}$. In general terms, the preference matrix models leader avatars of online communities that are followed when moving in the area of the DVE. We construct the preference matrix by the analysis of game sessions, and then we consider the average distance $d_{i,j}$ between every pairs of avatars to populate the α in such way:

$$\alpha_{u,v}^{t} = \begin{cases} 0, & \text{if } d_{i,j} > d_{max} \\ 1, & \text{if } d_{i,j} < d_{min} \\ \dfrac{1}{d_{i,j}}, & \text{otherwise} \end{cases} \tag{6}$$

  In general, 0 represents no preference form u to v, while 1 represent a strong preference.

- the minimum **radius of collision** r_{min}. It prevents avatars to converge to the same point in the DVE due to a strong attraction by an avatar. Basically, the idea is that when two avatars are closer than r_{min} the attraction between them becomes repulsion.

In conclusion, each avatar u moves according to all the contributions of Equation 5 and the different phases of the MOBA game described above can be modeled tuning the parameters involved in the equation.

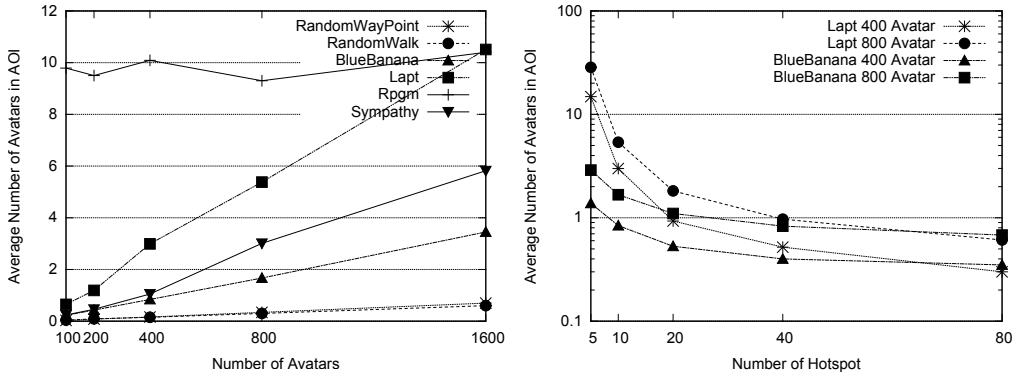
5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

This section first gives a brief overview of the overall structure of TRACE then presents a set of experimental results.

5.1. TRACE*: structure of the tool*

The assumption that each avatar moves independently, at time t, with respect to the other ones, can be exploited to define a concurrent architecture of TRACE. This assumption perfectly fits with the majority of the mobility models, in particular for all the models not requiring the positions of the other avatars, i.e. the ones without the *check-mark* in the column *Avatar* of Table III. For these models the support generates a thread for each avatar, which concurrently executes the movements of the avatar, at time t. For the models requiring the positions of the other avatars, i.e. RPGM and SYMPATHY, TRACE provides a read-only data structure containing the positions of the avatars at time $t-1$ (the *avatarList* in input of the function). With this solution, TRACE exploits concurrency fo the generation of the traces for all the models of Table III.

DOI: 10.1002/cpe



(a) Average number of avatars in AOI

(b) Average number of avatars in AOI with different number of hotspots

Figure 3. Evaluation of Optimizations

Algorithm 2 presents a high-level description of the function implementing the avatars' movement in TRACE. Note that the algorithm exploits a *ThreadBased executor*, which executes the submitted tasks. For each avatar, a task is created (Line 2), and the function *move*, which implements the mobility logic of the model, is called. For instance, this logic may be that of BlueBanana, which has been presented in Algorithm 2. Finally, the function waits for the completion of all the movements (Line 5) and the new positions of all the avatars is returned.

The level of parallelism can be tuned through a configuration variable. TRACE creates an executor service which handles all the threads submitted according to the configured concurrency level.

5.2. Experimental Results

We implemented TRACE in Java and we make the code publicly available ¶. For all the experiments, we considered a virtual environment composed by a squared region with side having 1500 points. Each avatar has a circular AoI, whose radius is 15 points. Each hotspot has a circular shape, whose radius is 100 points. The simulations ran on a machine equipped with Java 7, 128 Gb of RAM, an AMD Opteron(TM) Processor 6276 with 32 cores @1.4 Ghz. In the following, we present results showing some properties of the models implemented in TRACE. In particular, the avatars' crowding in the virtual world (Section 5.3) and the estimated bandwidth consumption to transmit objects of the virtual world (Section 5.4). We conclude our experiments with an evaluation of the computational time to generate a mobility model and the scalability of TRACE (Section 5.5).

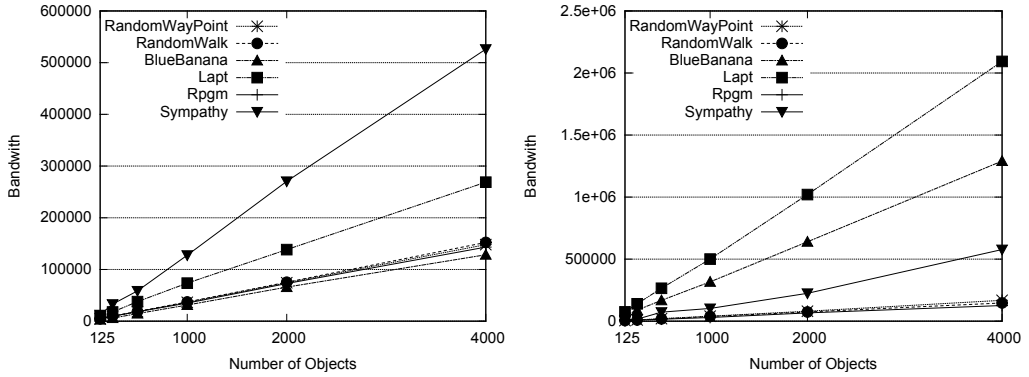
5.3. Evaluating the crowding generated

With the terms crowding we refer to the evaluation of the number of avatars present in each avatar's AoI. This metric assesses how much communication is required to keep updated the vision of the avatars with respect to the other players in the game.

Figure 3a shows the average number of avatars in the AoI of each avatar for all the models produced by TRACE. On the X axis is represented the number of avatars present in the virtual world, on the Y axis the average number of avatars in a AOI. We generate for each model a trace having a number of avatars in the range $[100, 1600]$. It is interesting to note that with RPGM we obtain similar results in all the configurations. This result is expected because we configure RPGM in order to keep the number of groups equals to $1/20$ of the number of avatars. The two models based on random movements are the ones having the less number of avatars in the AoI. Instead, LAPT is the model having the larger increase of crowding as the number of avatars grows, because all the avatars move

¶`https://github.com/hpclab/trace`

DOI: 10.1002/cpe



(a) Object placement: uniformly at random

(b) Object placement: higher probability in hotspots

Figure 4. Evaluation of Bandwidth consumption

only between hotspots. With BLUEBANANA this effect is mitigated because a percentage of the avatars is free to move outside the hotspots. Interestingly, SYMPATHY is one of the models having higher values. This model let the avatars to arrange in group, due to this such result is somehow expected.

For what concerns LAPT and BLUEBANANA, the models that take in consideration the hotspots, Figure 3b shows the impact of the number of hotspots using the same metric of the previous figure. Note the log scale on the Y axis. When the number of hotspots is kept low, LAPT is, in both the configurations, the model having a larger crowding factor. However, when the number of hotspots increases, the two models behave similarly.

5.4. Evaluating the bandwidth to transmit objects

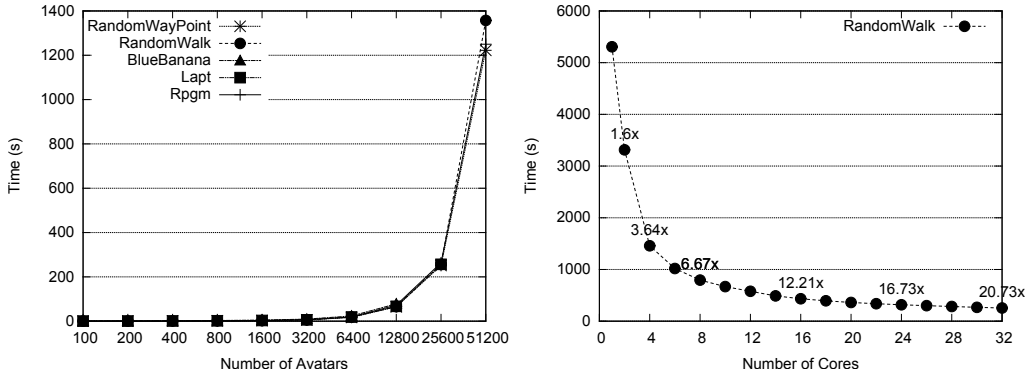
In this set of experiments, we evaluate the ability of TRACE to model the avatars and objects placement. In particular, when an object enters the AoI of an avatar, a transmission of the object to the avatar is required, resulting in a bandwidth consumption. We measure the total number of objects transmitted when increasing the total number of objects in the virtual world. We test the two methodologies to distribute the objects, respectively the uniformly at random in Figure 4a, and higher probability in the hotspots in Figure 4b. For the uniformly at random placement, all the models behave similarly and have a linear increase of the bandwidth with respect to the number of objects. SYMPATHY is the model requiring more bandwidth in all the configurations. Avatars form groups and explore the area all together, this may result in high bandwidth consumption with respect to models like LAPT that moves always on pre-determined paths.

When the objects are more present in the hotspots area, Figure 4b, the two models, LAPT and BLUEBANANA, as expected, require more bandwidth, because the avatars are more present in the hotspots area. Also, according to SYMPATHY the avatars sometimes are visting hotspots to simulate combat times. Due to this, although in less measure with respect the two previously descrbed model, also SYMPATHY have high bandwidth requirements.

5.5. Evaluating the computational time and scalability

Finally, we test the computational time required by TRACE to generate the traces. Figure 5a depicts the computational time when requesting a different number of avatars moving in the virtual world. As expected, the time increases when increasing the number of avatars but it is acceptable also with a large number of avatars, as well as $51\,200$ avatars. All the mobility models behaves similarly. Due to this, we perform the scalability of TRACE only with the RW model (we confirm that with other models the shape of the curve is identical). We are able to test our tool with a scenario having a number of cores in the range $[1, 32]$. We obtain a good scalability of TRACE. For instance, with 8

DOI: 10.1002/cpe



(a) Computational time with different number of avatars

(b) The scalability of TRACE

Figure 5. Evaluation of computational time

cores we obtain a speed-up of 6.67 to a maximum of 8 and with 12 cores a speed-up of 12.21 to a maximum of 16.

5.6. Traces Comparison

The trace analysis capabilities of TRACE are useful when it comes to compare different mobility models on a common ground. In this section we compare BLUEBANANA, LAPT, RPGM, SYMPATHY in a common scenario, consisting of 100 avatars moving in a 300x300 virtual world with 4 hotspots randomly placed. To perform the comparison, we used the metrics described in Section 3.3, and the results are depicted in Figure 6. Each trace has its own specific behavior coming from the mobility model considered, and it is detailed in the following.

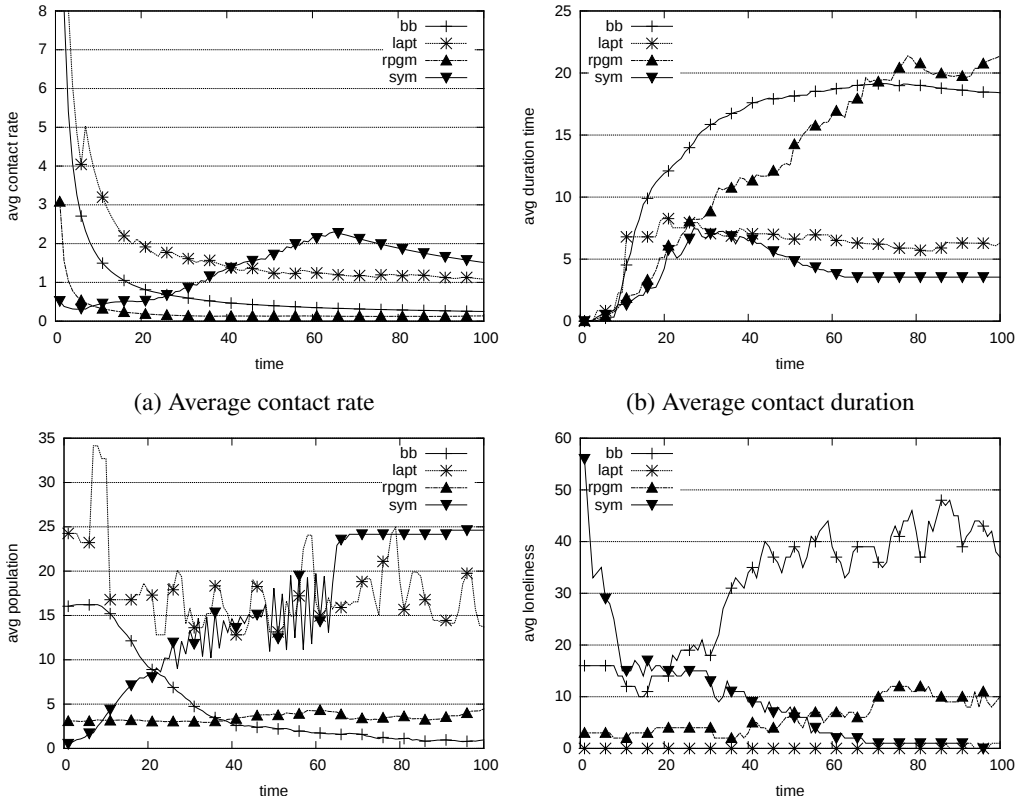
BLUEBANANA. The contact rate and AoI population of BLUEBANANA starts at high values and then converge to low values toward the end of the simulation. Rather, the contact duration and the number of lone avatars behave differently, starting at low values and converging to high values, 20 and 40 respectively. This suggests that avatars start very clustered before expand to the whole map. Toward convergence we can notice a relatively reduced mobility of the avatars, with small very tight groups (possibly composed by just a single avatar) having few contacts with avatars from other groups.

LAPT. It converges to medium values for contact rate (1 new avatar seen per unit of time) and contact duration (around 6). It also keeps a generally large AoI population which is very variable between 15 and 20 avatars during the whole simulation. Such data suggests a large group of uncoordinated avatars that have crowded AoI, in which other avatars exit and enters at a high rate. Also, the number of lone avatars is very small, suggesting that avatars have a tendency to visit the same locations over and over with few variance.

RPGM. RPGM converges to very low contact rate values ($<< 1$) and keeps a low stable population count (around 4) for the whole duration of the simulation. It has however high contact duration values, suggesting the presence of coordinated small-sized groups that move in a compact way around the map and do not meet each other frequently.

SYMPATHY. The reults of this model are very peculiar, and come directly from the definition of the mobility model. At the start, in the beginning phase, it shows many small groups (or lone avatars) having few interaction between each others. Then, during the skirmish phase, avatars starts to group together, as showed by the increasing AoI population, contact rate and contact duration, and the corresponding drop of lone avatars. In the third phase, final battle, there are large groups of avatars

DOI: 10.1002/cpe



(a) Average contact rate

(b) Average contact duration

(c) Average population count

(d) Average number of lone avatars

Figure 6. Evaluation of different traces

with coordinated movements, as it can be seen from the large population count and the drop of lone avatars to basically zero.

5.7. An Application of TRACE

In this section we show how TRACE can be effectively used in an application for the evaluation of the interest management in DVE. The interest management is the requirement that each client machines has its view of the interested area of the virtual environment up-to-date.

We consider an application that measures how two community discovery mechanism improve the area coverage [9]. Here, we show the applicability of TRACE in such scenario, for additional details on such application please refer to the original paper.

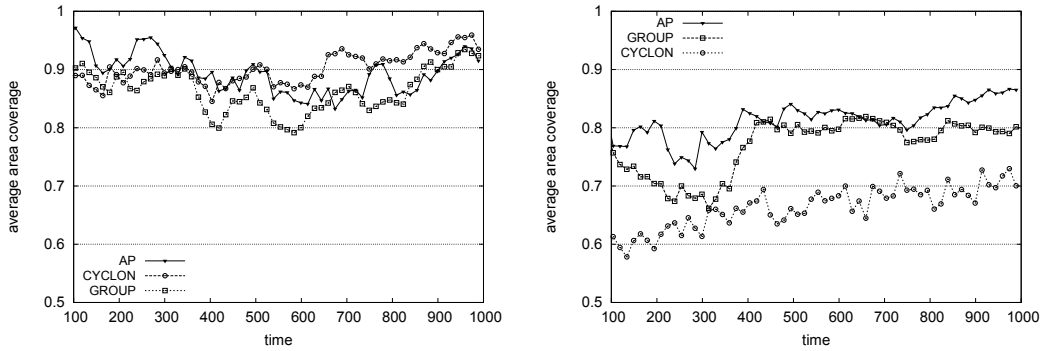
The application compares the impact of two popular community discovery algorithm Group [4] and Affinity Propagation [14] on the interest management. As a baseline, it is considered a basic version that just employs a random peer sampling.

To measure this impact it is used a metric called area coverage. An area coverage of 1 means that the considered avatar has its complete AoI up-to-date. The metric is averaged on all the avatars in the DVE.

Figure 7a and 7b show the result with respectively the BLUEBANANA and RWP mobility models. The results of the two models are different, and highlight interesting insight on the impact of communities on the interest management. When considering BLUEBANANA, all the versions behave similarly. When considering the RWP instead Affinity Propagation achieves the best results.

Such results highlight how TRACE can be an effective tool to easily provide mobility traces to evaluate an application. In such scenario the developers of the application can inject freely different mobility traces and study the differences when avatars move in the virtual world.

DOI: 10.1002/cpe



(a) Average Area Coverage with BLUEBANANA (b) Average Area Coverage with RWP

Figure 7. An application of TRACE: Evaluation of Interest Management in DVE

6. CONCLUSION

This paper provides a comprehensive description of the design and features of TRACE, an open source tool for the generation of mobility traces. TRACE is aimed at researchers and developers that look for a simple way to integrate mobility traces into their DVE solutions. The code of TRACE is publicly available for testing purpose at `https://github.com/hpclab/trace`.

The main aim of TRACE is to allow researchers focusing on the development of their solution rather than spend time to code and test custom mobility models and traces from scratch. To this end, the development of TRACE has followed (and will follow) the core design principles: efficiency, reusability and ease of use. In terms of efficiency, we exploit multi-core architectures to scale up to thousands of avatars, while keeping the memory footprint as minimal as possible. Regarding reusability, TRACE can manage both trace generation and analysis, as well as working as an helping tool to conduct trace analysis, which is useful to compare mobility traces and evaluate their impact on a DVE architecture. Finally, TRACE offers the possibility to implement a mobility model (also by combining multiple models) and create personalized mobility traces with few lines of code by extending the exposed API.

We think these principles are of paramount importance in order to exploit various traces when evaluating solutions targeting DVE, and to ease how such traces are collected, generated, and exploited. In conclusion, we believe TRACE to be an effective tool to facilitate the evaluation of DVEs architectures and to easily implement mobility models.

ACKNOWLEDGEMENT

The authors would like to thank Iacopo Peri for his thoughtful and consistent work on SYMPATHY.

REFERENCES

1. Blizzard entertainment, world of warcraft website. `https://worldofwarcraft.com`. Accessed: 09 Apr 2017.
2. Second life official website. `http://secondlife.com/`. Accessed: 09 Apr 2017.
3. F. Bai and A. Helmy. A survey of mobility models. *Wireless Adhoc Networks. University of Southern California, USA*, 206, 2004.
4. R. Baraglia, P. Dazzi, M. Mordacchini, L. Ricci, and L. Alessi. Group: A gossip based building community protocol. In *Smart Spaces and Next Generation Wired/Wireless Networking*, pages 496–507. Springer, 2011.
5. 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. *ACM SIGCOMM Computer Communication Review*, 38(4):389–400, 2008.
6. A. R. Bharambe, J. Pang, and S. Seshan. Colyseus: A distributed architecture for online multiplayer games. In *NSDI*, volume 6, pages 12–12, 2006.

DOI: 10.1002/cpe

7. E. Carlini. *Combining Peer-to-Peer and Cloud Computing for Large Scale Online Games*. PhD thesis, IMT Lucca, December 2012. arXiv preprint arXiv:1510.08940.
8. E. Carlini, M. Coppola, and L. Ricci. Evaluating compass routing based aoi-cast by mogs mobility models. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 328–335. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.
9. E. Carlini, P. Dazzi, M. Mordacchini, A. Lulli, and L. Ricci. Community discovery for interest management in dves: A case study. In *European Conference on Parallel Processing*, pages 273–285. Springer, 2015.
10. E. Carlini, A. Lulli, and L. Ricci. Trace: Generating traces from mobility models for distributed virtual environments. In *European Conference on Parallel Processing*, pages 272–283. Springer, Cham, 2016.
11. E. Carlini, L. Ricci, and M. Coppola. Reducing server load in mmog via p2p gossip. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, page 11. IEEE Press, 2012.
12. E. Carlini, L. Ricci, and M. Coppola. Flexible load distribution for hybrid distributed virtual environments. *Future Generation Computer Systems*, 29(6):1561–1572, 2013.
13. R. Diaconu and J. Keller. Kiwano: scaling virtual worlds. In *Winter Simulation Conference (WSC), 2016*, pages 1836–1847. IEEE, 2016.
14. B. J. Frey and D. Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.
15. C. Gross, M. Lehn, C. Münker, A. Buchmann, and R. Steinmetz. Towards a comparative performance evaluation of overlays for networked virtual environments. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pages 34–43. IEEE, 2011.
16. Y. Guo and A. Iosup. The game trace archive. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, page 4. IEEE Press, 2012.
17. X. Hong, M. Gerla, G. Pei, and C.-C. Chiang. A group mobility model for ad hoc wireless networks. In *Proc. of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 53–60. ACM, 1999.
18. S.-Y. Hu. Spatial publish subscribe. In *Proc. of IEEE Virtual Reality (IEEE VR) workshop, Massively Multiuser Virtual Environment (MMVE09)*, 2009.
19. S.-Y. Hu, H.-F. Chen, and T.-H. Chen. Von: a scalable peer-to-peer network for virtual environments. *Network, IEEE*, 20(4):22–31, 2006.
20. L. Itzel, F. Heger, G. Schiele, and C. Becker. The quest for meaningful mobility in massively multi-user virtual environments. In *Network and Systems Support for Games (NetGames), 2011 10th Annual Workshop on*, pages 1–2. IEEE, 2011.
21. H. Kavalionak, E. Carlini, L. Ricci, A. Montresor, and M. Coppola. Integrating peer-to-peer and cloud computing for massively multiuser online games. *Peer-to-Peer Networking and Applications*, 8(2):301–319, 2015.
22. A. Khelil, P. J. Marron, and K. Rothermel. Contact-based mobility metrics for delay-tolerant ad hoc networking. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 435–444. IEEE, 2005.
23. S. Kulkarni, S. Douglas, and D. Churchill. Badumna: A decentralised network engine for virtual environments. *Computer Networks*, 54(12):1953–1967, 2010.
24. K. Lee, S. Hong, S. J. Kim, I. Rhee, and S. Chong. Slaw: A new mobility model for human walks. In *INFOCOM 2009, IEEE*, pages 855–863. IEEE, 2009.
25. S. Legtchenko, S. Monnet, and G. Thomas. Blue banana: resilience to avatar mobility in distributed mmogs. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 171–180. IEEE, 2010.
26. H. Liang, R. N. De Silva, W. T. Ooi, and M. Motani. Avatar mobility in user-created networked virtual worlds: measurements, analysis, and implications. *Multimedia Tools and Applications*, 45(1-3):163–190, 2009.
27. J. L. Miller and J. Crowcroft. Avatar movement in world of warcraft battlegrounds. In *Proceedings of the 8th annual workshop on Network and systems support for games*, page 1. IEEE Press, 2009.
28. J. L. Miller and J. Crowcroft. Group movement in world of warcraft battlegrounds. *International Journal of Advanced Media and Communication*, 4(4):387–404, 2010.
29. V. Nae, R. Prodan, and T. Fahringer. Cost-efficient hosting and load balancing of massively multiplayer online games. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 9–16. IEEE, 2010.
30. L. Ricci and E. Carlini. Distributed virtual environments: From client server to cloud and p2p architectures. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 8–17. IEEE, 2012.
31. L. Ricci, E. Carlini, L. Genovali, and M. Coppola. Aoi-cast by compass routing in delaunay based dve overlays. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 135–142. IEEE, 2011.
32. 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 *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*, pages 247–256. IEEE, 2008.
33. S. Shen, N. Brouwers, A. Iosup, and D. Epema. Characterization of human mobility in networked virtual environments. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 13. ACM, 2014.
34. S. Shen and A. Iosup. Modeling avatar mobility of networked virtual environments. In *Proceedings of International Workshop on Massively Multiuser Virtual Environments*, pages 1–6. ACM, 2014.
35. M. Suznjevic, I. Stupar, and M. Matijasevic. A model and software architecture for mmorpg traffic generation based on player behavior. *Multimedia systems*, 19(3):231–253, 2013.
36. S. A. Tan, W. Lau, and A. Loh. Networked game mobility model for first-person-shooter games. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–9. ACM, 2005.
37. T. Triebel, M. Lehn, R. Rehner, B. Guthier, S. Kopf, and W. Effelsberg. Generation of synthetic workloads for multiplayer online gaming benchmarks. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, page 5. IEEE Press, 2012.

DOI: 10.1002/cpe

38. D. J. Watts and S. H. Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.
39. A. Yahyavi, K. Huguenin, J. Gascon-Samson, J. Kienzle, and B. Kemme. Watchmen: Scalable cheat-resistant support for distributed multi-player online games. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 134–144. IEEE, 2013.
40. 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*, pages 99–104. ACM, 2005.

DOI: 10.1002/cpe