Integrating Centralized and P2P Architectures to Support Interest Management in Massively Multiplayer On-line Games

Emanuele Carlini1∗, Laura Ricci1,2, Massimo Coppola1

1 Istituto A. Faedo, ISTI-CNR, Pisa
2 Department of Computer Science, University of Pisa

SUMMARY
A fundamental problem for the development of P2P Distributed Virtual Environments, like Massively Multiplayer On-line Games, is the definition of an overlay supporting interest management, i.e. determining all the entities of the virtual world that are relevant for a given player. To this end, this paper proposes a gossip-based approach that considers the coverage of the Area Of Interest of peers as the guiding principle for the definition of the P2P overlay and its maintenance. The resulting overlay provides a support for a best-effort resolution of interest management, mostly supported through communications on the P2P overlay, with minimal intervention of a centralized entity. The paper presents a set of extensive simulations based on realistic mobility traces. The experimental results show the effectiveness of gossiping for the construction and maintenance of a best-effort overlay for interest management. Copyright © 0000 John Wiley & Sons, Ltd.

KEY WORDS: Peer-to-Peer, MMOG, Gossip algorithms

1. INTRODUCTION
Distributed Virtual Environments (DVEs), like massively multiplayer games [1] or distributed training simulations, acquired enormous popularity in the last years from both commercial enterprises and research communities. The reason behind DVEs success is their ability to provide a shared sense of space to their users, whether they are players in a multiplayer game or soldiers in a military simulation. In particular, on-line gaming entertainment has acquired lots of popularity in the last years from both industry and research communities. This attention is justified by the economic growth of the field, in which Massively Multiplayer Online Games (MMOGs) have reached around 21 million of users worldwide in the 2012†. For this reason, in this work we focus on Multiplayer Online Games as they represent the killer application in the field of DVE.

To enable the engaging experience typical for these applications, information about the entities in the virtual environment has to be replicated in users machines. This issue is referred to as Interest Management (IM) [2] and it can be abstracted using a publish-subscribe model [3]. Publishers perform actions (e.g. they move) and subscribers should receive the result of these actions (i.e. their local replica should change accordingly). Through this paper we examine the movement of the avatars, which are the virtual representations of the users participating to the MMOG. However, our approach is general enough to be extended to other types of actions.

∗Correspondence to: Emanuele Carlini, Istituto A. Faedo, ISTI-CNR, Pisa, e-mail: emanuele.carlini@isti.cnr.it
†source: http://mmodata.net/
Whenever an avatar (or an object) changes its position it generates an event that should be propagated to all other users in the MMOG. IM poses well recognised scalability challenges, due to the number of participants and the fast-pace nature of the MMOGs. A plethora of approaches have been proposed to efficiently support IM (see related work in Section 6). One of the main strategies to address IM is to replicate only the entities in the visual/interaction area of players, which is called Area of Interest (AOI) and is typically a circular area whose center is the avatar position.

Many MMOG architectures [4, 5, 6] strive to resolve IM with Peer-To-Peer (P2P) principles, in order to cope with the limited scalability and flexibility of centralized solutions. Systems based on Distributed Hash Tables (DHTs) [7, 8, 9, 6] yield concrete advantages, as they offer a stable and reliable network for the distributed indexing of entities. However, in spite of optimizations like caching and pre-fetching, latency issues may still represent an issue. Compared to structured approaches, unstructured P2P solutions focus on building the overlay according to the spatial proximity of the peers [10, 11] in the virtual environment. In order to perform IM, each peer connects with a subset of its neighbours. In this way, neighbour peers may warn each other both about their movements and about new peers entering their AOI. These approaches naturally adapt to the rapid evolving scenarios of MMOGs and allow for large scalability, but introduce overhead in the complexity of the approach and make strong assumptions on the capability of peers. For different reasons, both structured and unstructured P2P overlays present weak points depending on some of their features. Several solutions strive to cope with these drawbacks, but even if they might work in principle, they introduce overhead in the IM mechanisms that may jeopardize the interactiveness of the application.

To avoid these drawbacks, we propose an hybrid solution based on a combination of a centralized server and a best-effort mechanism providing support for IM. This mechanism is based on a set of P2P epidemic (or gossip) protocols [12] to acquire the position of relevant entities in proximity of the players. Our approach to IM is designed to be lightweight, simple and scalable and to be able to reduce consistently the load on the central server.

The support for interest management is a component of the hybrid architecture for MMOG we have proposed in [13] which separates the management of the positional actions (i.e the actions that affects the position of the entities in the MMOG) from the state actions (i.e. the actions that modify the state of the entities). A more detailed description of the architecture is presented in Section 3.

In this paper we focus on the The Positional Action Manager (PAM, in short), the component devoted to managing the positional actions of the avatars. PAM exploits two gossip protocols to build an overlay in a completely distributed fashion. The first underlying protocol assures the network to remain connected, and provides a bootstrap point for the newly arrived users. The second protocol filters the user according their position in the MMOG, by continuously choosing the best set of users to connect with. This set can be chosen by mean of two different heuristics, one ”lighter” and less accurate, and the other one computationally intensive but more accurate. Then, each peer exploits each other knowledge to acquire information about close entities. At fixed intervals, the server provides fresh information to the users. This architecture reduces consistently the load on the central server. The MMOG operator can tune the PAM by defining the refreshing rate of the server. High rates reduce the cost but also limit the precision of the users view. On the other hand, low rates increase the cost, and also raise the precision of the results.

The best-effort nature of our approach pairs seamlessly with the hybrid MMOGs architecture [14, 15] which combines centralized and P2P solutions to create effective and economic architectures for MMOGs. For example, in the context of on-demand computing, our approach can be used to query servers only when necessary, thus to further reduce the economical cost of maintaining a MMOG platform. We have tested our protocol by a set of traces derived from Second Life, one of the most popular MMOG. The results we have obtained are encouraging, since on average up to 90% of entities in the AOI of a peer may be retrieved by exploiting our technique.

The paper is structured as follows. In Section 2 we give a background in the field of IM for MMOGs. A detailed description of the overall mechanism is presented in Section 3, whereas Section 4 discusses and evaluates the underlying gossip mechanism. Section 5 discusses a selection of our extensive experimental evaluation that shows the encouraging results obtained by our approach.
Section 6 presents a hierarchical classification of IM approaches, as well as a qualitative comparison with our solution. Finally, Section 7 concludes the paper.

2. BACKGROUND

A MMOG is a virtual world represented by a collection of entities, Avatars and Objects. Avatars represent the users in the virtual environment. Each user commands an avatar, that can travel across the virtual environment and interact with other entities. Objects (e.g. a door) are not directly controlled by the users, but Avatars can interact with them. A third type of entities are the Non Playing Characters (NPCs). These entities expose active behaviour controlled by the server. By comparison, objects have a passive behaviour, since they cannot decide to change their state autonomously.

Each entity is represented by the following elements: (i) an unique identifier (UID), (ii) a two-dimensional point, which represents the position of the entity in the virtual environment, (iii) when needed, the indication of the procedure to execute for AI-controlled entities (this procedure is sometimes referred to as think function [6]), and (iv) a collection of key-values pairs, which represents the attributes of the entity.

In order to illustrate some background concepts, let us assume that a central server stores all entity descriptors. In order to interact with the virtual environment, users connect to the server via a software agent, which we generally refer to as client. The client provides the representation of the virtual environment to the user. It also transforms the action of the user in messages sent to the server and receives back the modification of the virtual environment and updates. In other words, clients maintain a replica of the descriptors in their local memory that are periodically synchronized with the server. For the purpose of our work, here we underline a relevant difference in the events (or actions) that server can manage:

- **Avatars movement.** These actions are single-writer/multiple-reader, their effect is volatile and error tolerant. Since there is a single writer, these actions generate no conflicts. Further, positions are volatile, meaning that there is no need to keep them once the avatar has left. Also they are error tolerant, in the sense that a small error in consistency does not compromise the experience for the user.

- **State actions.** These actions are multiple-writers/multiple-readers, their effect is persistent and are not error tolerant. Since these actions are multiple writer, race conditions on entity descriptor may arise. In this case it is mandatory for the server to resolve possible conflicts. The effect of state actions is persistent, i.e. when the last writer leaves the environment, the descriptor must still be available for other possible writers. Since these actions operate on a discrete space (e.g. a door can be opened or closed) error are not allowed.

Normally, the server handles the events in an infinite loop of iterations. Every iteration has the same finite duration, and at each iteration, the server manages the flow of messages by resolving the possible conflicts on the entities descriptor and by broadcasting the new version of the state. In this case, it is possible that some clients need to revert the descriptors to a previous state, in order to be synchronized with the server.

**Interest Management** A common optimization in MMOGs architecture is to communicate to the clients only the minimal, but sufficient, set of relevant information. This operation is called Interest Management (IM, [2]). Exploiting the concept of IM drastically reduces the size of messages sent by the server to the clients. From an abstract point of view, IM can be modelled as a publish/subscribe service, in which users have role both of publisher and subscribers. In fact, users: (i) publish new entity descriptors by interacting with the virtual environment and (ii) subscribe to the relevant entities and receive updates when their descriptors change.

One of the most effective and straightforward strategies to define the set of a user’s relevant entities, is to consider the entities in the spatial proximity of the avatar. This subset is generally
modelled according to the so called focus-nimbus model [16]. In this model, the focus refers to the area of visualization of an avatar, while the nimbus is the area where an avatar can be seen. In other words, an avatar a is aware of an avatar b when the nimbus of a intersects with the focus of b. The shape and the size of the nimbus and focus are MMOG dependent. In the simplest (yet effective) case, both nimbus and focus are represented by the same circle whose center is the position of avatar in the virtual world. Generally, this region is called Area-of-Interest (AOI, [17]).

IM is applied in centralized systems as well, but it acquires even more importance in a case of a distributed MMOG infrastructure. Since relevant entities may be spread in different nodes, subscribing to a reduced set of entities also decreases the number of nodes to query when performing IM. In order to clarify this point, let us consider an example. Let us assume a virtual environment whose infrastructure is composed by multiple servers, which we refer to as $S_1$, $S_2$ and $S_3$. Each of the servers manages a contiguous non-overlapping area of the MMOG, called region. Let us also consider a generic client $C_1$, whose Avatar is in the region managed by $S_1$ but its AOI overlaps the additional two servers, $S_2$ $S_3$. The first step in order to perform IM, is to find out what are the entities in $C_1$’s AOI. This step is commonly called as Neighbour Discovery (ND). In order to optimize ND, servers are generally connected through spatial-based overlay (e.g. Voronoi-based) so that servers managing close regions in the MMOGs are connected to each others. Once ND is completed, $C_1$ subscribes (or is subscribed by the servers) to the entities in its AOI. We refer to this subscription as State Management (SM), since from now on, $C_1$ receives updates from the entities.

In a centralized architecture, the distinction between ND and SM is blurry, as they are executed at the same time and possibly by the same server. When this model is implemented in a classical distributed infrastructure, it presents several issues. Since each server manages a region, whenever an entity moves from one region to another, its descriptor must be transferred between the servers. This process is defined as ownership transfer [3]. Ownership transfer can create some problems to the users, since a migrating entity cannot be accessed. For that reason, it is important to design a MMOG infrastructure so to limit the amount and the duration of ownership transfer.

3. ARCHITECTURE

This section presents the distributed MMOG architecture (shown in Figure 1) that frames our solution for interest management. This architecture is composed by two main components The State Action Manager (SAM) and the positional action manager (PAM). Exploiting two different components yields concrete advantages. Separating the management of positions of entities from the management of associated data, allow us to reduce the effect of the ownership transfer. In addition, each component can be designed in isolation, allowing for an in-depth tuning of component’s characteristics.

SAM (originally presented in [13]) is the component that manages the state actions. SAM exploits a Distributed Hash Table, equipped with Virtual Servers to distribute the effort on management of the entities to multiple resources, including user-provided ones. SAM self-adapts to the load of the MMOG by releasing resources when the load is low, and acquiring additional resources when the current infrastructure has not enough capacity to manage the load. The operator can tune the SAM to decide the maximum amount of entities that can be managed by the user-provided resources. SAM recruits on-demand or user-assisted resources accordingly, always trying to minimize the cost and to avoid resources to be overloaded. In order to efficiently orchestrate resources, SAM is designed to adopt prediction mechanisms, so to take provisioning decision well in advance.

The description of PAM is the core of this paper (a preliminary version of PAM has been presented in [18]). It manages the positions of the entities by organizing a epidemic-based distributed overlay among players. The main goal of PAM is to assure a cost-effective tunable and up-to-date ND. To be up-to-date is a strict requirement for ND. Stale information on neighbours does not value anything in a complex and evolving system like a MMOG. On the other hand, cost effectiveness is a requirement necessary to make the component appealing to MMOG operators. The ability of tuning the trade-off between performance and economical cost can make the difference in a competitive market. To meet these requirements, PAM is composed by two integrated services:
• a backbone server which we refer to as PAM server, or only in this paper, generically as server
• a fully decentralized network, which we refer to as PAM overlay, or only in this paper with the generic term overlay.

Figure 2a shows the logical architecture of a PAM client. PAM clients maintain connections to both these services in order to receive information about their neighbours. Periodically, clients communicate their position to the server. On the other hand, periodically the server communicates to clients the list of their neighbours. The rate of these communications depends on the particular genre of the MMOG. Fast pace MMOGs require these communication to be less than 100 ms, whereas slow pace MMOGs may employ larger interval on the order of 500 ms [19]. In this context we are interested in the rate of the communication server-to-client (which we refer to as $T_s$), since it represents the major source of cost of the infrastructure. Indeed, this value can be tuned for more precise ND with higher costs, or, on the other hand, less precise ND but lower costs. Besides the server, clients also communicate with a custom overlay. The overlay is build such that to exploit the knowledge of close player in the virtual environment. More precisely, when a node $n$ has another node $m$ in its view, a connection between $n$ and $m$ exists in the overlay. However, this does not imply that the reverse connection exists. Nodes periodically query the overlay to learn about the entities in their AOI. In this context the overlay plays a fundamental role. If the overlay is effective, the MMOG operator can increase the $T_s$, so to reduce the economical cost, without sacrificing precision in ND. Note that the overlay can be paired with optimistic methods to maintain consistency about entities (e.g. Dead Reckoning [20]). Since the overlay is best-effort, Dead Reckoning would work for such entities not covered by the overlay.

From a player’s perspective, PAM and SAM work together to provide a MMOG service. Periodically the players send their movements (positional actions) to the PAM server, with a rate that depends on the particular virtual environment. The clients receive the positions of the entities in their AOI from either by the PAM-server or by the PAM-overlay. Regarding the state modification, clients send state actions to the SAM which is in charge to resolve possible conflict and computes the authoritative version of the state. In order to be notified of state updates of the entities in their AOI, clients periodically subscribe to the SAM.

3.1. PAM Server

Logically, the PAM server is composed by two asynchronous computation flows (typical of gossip-based protocols [12]):

• a passive thread that receives and stores the positions from the clients;
• an active thread that periodically informs clients about the content of their AOI.

These operations generate two different kinds of load on the server. First, they generate a computational load, as the server must maintain the connections, store the positions and resolve
spatial queries. Second, they generate bandwidth load, as the communications to the clients consume outgoing bandwidth. Therefore, the load imposed on the PAM-server is proportional to the number of players, and a single server might be not able to meet the target QoS when the number of players is large. Since the players have typically a seasonal behaviour [21], a statically provisioned set of servers may incur in over-provisioning [22]. Pay-per-use platforms (such as Cloud Computing) could be used to mitigate the over-provisioning. To properly exploit a cloud computing platform, the PAM server would dynamically acquire and release computational resources, in a way similar to the SAM [13]. However, due to the near real-time requirements of MMOG, the PAM server would execute a careful selection of cloud computing platforms according to the offered SLA (Service Level Agreement). In particular, platforms offering guarantee on bandwidth and computational capabilities in their SLA cloud be preferred. The core of this work is the PAM overlay, therefore how the PAM servers manage the dynamical provisioning of resources is left as future work.

3.2. PAM Overlay

The construction of the PAM overlay has been driven by protocols based on epidemic diffusion of information. These protocols (also known as as gossip protocols) are an effective building block for creating overlays in a pure distributed fashion. When exploiting a gossip-based protocol, each node maintains only a partial view of the network, consisting in a small set of neighbour nodes. Each node periodically interacts with nodes chosen among its neighbours. This simple, repeated local data exchange allows each node to achieve specific goals and results regarding the global network. More precisely, the behaviour of a node is defined by two separate logical flows, called active (see Figure 3) and passive (see Figure 4) threads.

```
while true do
  sleep T seconds;
  q ← selectPeer();
  sendBuffer ← ItemsToSend();
  send sendBuffer to q;
  receive receiveBuffer from q;
  view ← ItemsToKeep();
end
```

```
foreach Message m Received do
  extract sender p from m;
  sendBuffer ← ItemsToSend();
  send sendBuffer to p;
  view ← ItemsToKeep();
end
```

Figure 3. Active Thread

Figure 4. Passive Thread
The functions `selectPeer()`, `ItemsToSend()` and `ItemsToKeep()` define the goal and the characteristics of the particular gossip protocol. By changing these methods it is possible to achieve different goals or different overlay organisations.

Gossip protocols can be composed into a layered structure. Each layer has two main duties: (i) to maintain and exchange information and (ii) to feed the higher gossip levels with the proper information. For instance, an underlying random peer sampling protocol layer can be exploited in order to speed up the convergence toward the target overlay topology. The random peer sampling serves also in the initial gossip cycles, when the local view of the peer is empty and a peer needs to know a random sample of peers to bootstrap on the network. The layered approach has been used to build custom overlay, such as T-Man [23].

T-Man proposes a gossip-based probabilistic approach whose goal is to build, starting from an arbitrary initial peer configuration, a target overlay characterized by a set of well defined properties. These may be inferred by the profiles of the peers or directly characterize the topology of the target overlay. In the former case, for instance, a metrics based on the geographical location or on the semantic profile of the peer may be considered to define a proximity-aware target overlay.

The definition of a proper ranking function is a core element to build the target overlay. The ranking function is exploited to select the "best neighbours" according to the properties of the target topology. Hence, using only local gossip messages, the current topology gradually evolves towards the desired target structure with the help of the ranking function.

In our case, the ranking function should favour neighbours which may offer a larger number of entities in the interest set of a peer. Unlike most existing gossip-based overlay construction approaches, our goal is to build a continuing evolving overlay rather than predefined one. The view of a peer changes continuously in order to reflect the position updates of the peers in the virtual space. In our case, instead of evolving toward a predefined target topology, peer continuously gossip to each other to support the retrieval of new avatars and objects in their A0I.

3.3. Coverage Peer Sampling

Our technique to build an overlay supporting IM is based on the following reasoning. Let us consider a given peer $P$. At an arbitrary point in time it has in its local representation of the environment the replicas of the entities that belong to its AOI. When $P$ moves, its AOI changes accordingly. Hence, to maintain its local representation up-to-date, $P$ must discover the new entities belonging to the new AOI. In order to dynamically acquire this information, $P$ builds an overlay by considering a set of relevant neighbours.

The overlay creation poses two issues. First, $P$ needs to know the identifier of its candidate neighbours; second, $P$ needs a mechanism to discriminate among peers, in order to choose the more promising neighbours from the set of candidates.

The first issue is resolved by continuously refreshing candidate knowledge. This is obtained with a two-layer gossiping architecture, where each layer runs a gossip protocol (the structure of these layers is shown in Figure 2b). In the underlying layer, called random peer sampling, each client runs a random peer sampling protocol, which provides a subset of all the nodes in the system. This layer enables each peer to maintain a set of long range links that guarantees the connectivity of the overlay. In the second layer, called coverage peer sampling, a gossip protocol connects peers by exploiting a ranking function based on spatial AOI coverage. Since the selection of the neighbours is done according the proximity, entities are progressively discarded by the second layer gossip if they disappear from its AOI. The two gossip layers are independent, in the sense that layers execute their gossip cycle at their own rate.

The second issue is related to the AOI coverage offered by the neighbours of a peer. Each peer should choose the best configuration of its neighbours in order to optimize the number of entities which may be retrieved from them. At each iteration the peer adapts its overlay neighbours set by providing a partial order from multiple configurations of neighbour sets. Since avatars are continuously moving, a large part of the IM performances depends on the freshness of peers knowledge. In order to maintain the selection of the neighbours as fresh as possible, each entry in the view of the peers is marked with a time-stamp. Time-stamps provide an estimation on the
freshness of the entry. Our mechanism considers the age of the entries in two situations. First, before to rank the neighbour candidates, all the candidates whose age is greater than a certain threshold are not considered. Second, during the ranking, fresh configurations are favoured with respects to the stale ones. In general, the internal clock of the peers can be used as the source for the time stamp. However, for simulation purposes, we model the time as a discrete successions of iterations. The simulation starts at iteration zero for all the nodes, and for each gossip-cycle the count is increased by one. When an entry is created, the iteration count is used as time-stamp for such entry. The choice of the best neighbour set is realized by means of a ranking functions, which are discussed in detail in the next section.

4. AOI COVERAGE RANKING FUNCTIONS

In this section we explore in details the aspect of AOI coverage, which is the main principle driving our ranking functions. This aspect poses two distinct challenges: (i) to measure the amount of area covered by neighbours peers and (ii) to determine the best subset of neighbour peers that maximizes the AOI coverage. In the rest of this section we formalize these two problems and we provide a description of the adopted solutions.

Measuring Coverage

Definition 1 (AOI coverage)
Given a set of AOIs \( \mathcal{N} = \{N_1, ..., N_n\} \) and an AOI \( P \) such that \( P \notin \mathcal{N} \) we define the coverage of \( P \) given \( \mathcal{N} \), \( c(P, \mathcal{N}) \), as the area of \( P \) that is overlapped by the AOIs contained in \( \mathcal{N} \).

Computing \( c(P, \mathcal{N}) \) requires to compute all the unique intersections of AOIs in \( \mathcal{N} \) with \( P \)'s AOI and to evaluate their area. In trivial situations this is easy to compute. For example, Figure 5 depicts a simple scenario where \( \mathcal{N} = \{A, B\} \). In this case the coverage is just the sum of the intersections of \( A \) and \( B \) with \( P \), i.e. \( c(P, \mathcal{N}) = B \cap P + A \cap P \). However, in real situations, computing the AOI coverage is not a trivial problem. For instance, in the case depicted by Figure 6 we have that \( c(P, \mathcal{N}) = (P \cap B - P \cap B \cap A) + (P \cap B \cap A) + (P \cap A - P \cap B \cap A) \).

![Figure 5. Simple continuous \( c(P) \) with \( \mathcal{N} = \{A, B\} \).](image)

When many peers are close to each other, to compute the effective coverage may be prohibitively expensive in terms of computational effort. Practically, this happens for two reasons. First, the number of the intersections grows quadratically with the number of peers. Second, it might be computationally costly to evaluate the area of an intersection resulting from many AOIs. For these reasons, we approximate the continuous surface of the AOI as a grid of disjoint hexagonal tiles. In this way, instead of dealing with custom-shaped areas, we consider the tiles as the units to compute the coverage. This approximation reduces the complexity of the problem, since it makes easy to compute the area of each tile. Moreover, the amount of tiles is a parametric value and does not depends on the number of peers. Figure 6 shows an example on how to compute the coverage of a given AOI (P in the figure) considering a 7-tiles approximation. The number of tiles varies proportionally with the degree of the approximation. A high number of tiles leads to higher precision, in principle increasing the performance of our mechanism. In the experimental section, we show how the degree of approximation can affect the effectiveness of the ranking.
Figure 6. Continuous and approximate coverage with $\mathcal{N} = \{A, B\}$. In this case the AOI coverage is approximated to $5/7$.

The pseudo code of the function $\text{coverage()}$ that realizes the tile-based coverage approximation is presented in Algorithm 1. For each AOI $\in \mathcal{N}$ and for each tile, we check whether the AOI intersects with the tile. If it is, we check the counter associated to the tile. If the tile counter is zero, it means the tile is overlapped for the first time so we increase the $\text{covered\_tiles}$ counter. If the tile counter is greater than zero, we just increment it. Besides the number of the tiles covered, this function also counts the number of AOIs that cover each tile. It is easy to show that the complexity of the function is $O(n \times t)$, where $n$ is the cardinality of $\mathcal{N}$ and $t$ is the number of tiles.

**Algorithm 1: Coverage($P, \mathcal{N}$)**

- **Input**: $P$, the considered peer
- **Input**: $\mathcal{N}$, the set of neighbours AOIs
- **Output**: the approximated coverage given $\mathcal{N}$ and $P$

**Data**: $\text{covered\_tiles} \leftarrow 0$

1. **foreach** $\text{AOI} \in \mathcal{N}$ **do**
   2. **foreach** $\text{tile} \in \text{getTiles}(P)$ **do**
      3. **if** $\text{intersect}(\text{AOI}, \text{tile})$ **then**
         4. **if** $\text{tile.count} = 0$ **then**
            5. $\text{covered\_tiles} \leftarrow \text{covered\_tiles} + 1$
            6. **end**
            7. $\text{tile.count} \leftarrow \text{tile.count} + 1$
            8. **end**
      9. **end**
   10. **end**
11. **return** $\text{covered\_tiles}$;

**Maximizing AOI Coverage** The final aim of the PAM overlay is to discover the larger amount of objects in the AOI of the peer (possibly all of them). To this end it is necessary to keep links with the neighbours that maximizes the coverage. This indeed requires peers to make a choice, due to the bound imposed by the gossip view size. Hence, very often a peer needs to choose what is the best subset of peers to keep in its view. This subset is defined as follows:

Copyright © 0000 John Wiley & Sons, Ltd.
Concurrency Computat.: Pract. Exper. (0000)
Prepared using cpeauth.cls
DOI: 10.1002/cpe
Definition 2 (Maximum AOI coverage)
Given a set $\mathcal{N}$ of AOIs, $\mathcal{N} = N_1 \ldots N_n$ and a natural number $d \leq n$ and an AOI $P \notin \mathcal{N}$, we define $S_d = \{X \in \mathcal{P}(\mathcal{N}) : |X| = d\}$, find the set $M \in S_d$ that maximizes the coverage of $P$.

This problem is equivalent to the maximum coverage problem, which has been proven to be NP-hard [24]. A naive solution to this problem would be to enumerate the possible combinations of peers and for each of them compute the coverage. Unfortunately, this is highly impracticable since the combinatorial nature of the problem. Here we propose two heuristics algorithms (a score-based and a greedy-based) with different characteristics.

Score-based Heuristics  The rationale behind this heuristics algorithm is to assign a score to each tile. The tiles that intersect with few peers will receive a higher score than tiles intersected by a larger amount of peers. The idea is then to favour such peers that overlap high score tiles. The heuristics works as in the pseudo code in Algorithm 2.

First, we compute the coverage of the AOI by considering all the peers in $\mathcal{N}$. Each tile has a score that is the reciprocal of the number of intersected AOIs. Second, it computes the score for each AOI as the sum of the scores of each intersected tiles. Finally, it sorts the AOIs in descending order according to their score, and it chooses the first $d$ entries.

Algorithm 2: Score-based Heuristics

\begin{algorithm}
\begin{algorithmic}
\State Input : $P$, the considered peer
\State Input : $\mathcal{N}$, the set of neighbours AOIs
\State Input : $d$, the size of the returned set
\State Output: a subset of $\mathcal{N}$ with cardinality $d$
\State coverage($P, \mathcal{N}$);
\For{$AOI \in \mathcal{N}$ do}
\For{$tile \in$ getTiles($P$) do}
\If{intersect(AOI, tile) then}
\State AOI.score $\leftarrow$ AOI.score + $\frac{1}{tile.count}$;
\EndIf
\EndFor
\EndFor
\State sort AOIs in descending order according to score;
\State return the first $d$ AOIs;
\end{algorithmic}
\end{algorithm}

The complexity analysis of the score-based algorithm heuristics goes as following: (i) the coverage procedure, which we have already seen to be $O(nt)$, (ii) the computation of the score, that can be considered as $O(n)$, and (iii) the sorting, which is $O(n \log n)$.

Figure 7 shows a graphical execution of the score-based heuristic algorithm with a 7-tiles approximation and $\mathcal{N} = \{A, B, C\}$. For example, the central tile has a score of 0.3 since A and B and C intersect with it. If we consider $d = 2$, the heuristics chooses the combination $\{A, B\}$, which is also the combination that maximizes the coverage. However, the heuristics not always finds the optimum. Let us consider the example in Figure 8. In this case the score-based heuristic algorithm chooses the combination $\{ABC\}$ that covers 5 tiles instead of $\{ACE\}$ that covers 6 tiles.

Greedy Heuristics  The idea behind the greedy heuristics is to choose at each step the peer that yields the higher increment on the number of unique tiles covered. The pseudo-code of the greedy heuristic algorithm is represented at Algorithm 3. For each peer in the view, it is selected the AOI that maximizes the number of further covered tiles considering the already chosen AOIs. Note that: (i) an AOI can be selected only once as, upon selection, it is removed from the list of candidates, and (ii) to evaluate the number of tiles covered we use the function $\text{coverage()}$ described and evaluated in the previous section.
Figure 7. Graphical examples of the score-based heuristic considering a 7-tiles approximation, $d = 2$, and $N = \{A, B, C\}$.

**Algorithm 3: Greedy Heuristics**

*Input*: $P$, the considered peer  
*Input*: $N$, the set of neighbour peers  
*Input*: $d$, the size of the returned set  
*Output*: a subset of $N$ with cardinality $d$  
*Data*: $C \leftarrow \emptyset$

1. **while** $|C| < d$ **do**
   2. chosen $\leftarrow \emptyset$;
   3. max_score $\leftarrow 0$;
   4. **foreach** AOI $\in N$ **do**
      5. score $\leftarrow$ coverage($P$, $C \cup$ AOI);
      6. **if** score $>$ max_score **then**
         7. max_score $\leftarrow$ score;
         8. chosen $\leftarrow$ AOI;
   9. **end**
   10. remove chosen from $N$;
   11. add chosen to $C$;
13. **end**
14. **return** $C$;

The complexity analysis of the greedy heuristic algorithm goes as following. The outer cycle (line 1) is repeated $d$ times. The inner cycle (line 4) is repeated at maximum $|N| = n$ times. The function `coverage` has a complexity of $O(nt)$. Hence, the total complexity in time is $O(td \times n^2)$.

Figure 8 shows a graphical execution of the greedy heuristic algorithm considering a 7-tiles approximation, $d = 3$, and with $N = A, B, C, D, E$. At the first step, the heuristics chooses $C$, as it is the AOI that covers the most tiles. At the second step, $A$ is chosen so that the current combination becomes $\{CA\}$. At the third step $E$ is chosen, and the final combination is $\{CAE\}$. Note that at the second step, the heuristics could have chosen $E$. In such case the second combination was $\{CE\}$ that would have lead to the same results (i.e $\{CAE\}$).

To prove approximation guarantees of the greedy heuristics, we first have to introduce submodular function [25]. Consider $\Omega$ to be a finite set and an arbitrary function $f : 2^\Omega \rightarrow \mathbb{R}$, we can say $f$ is submodular if it satisfies the following property: the marginal gain of adding an element to a set $S$ is
Figure 8. Greedy and Score Heuristic considering a 7-tiles approximation, \( d = 3 \), and with \( N = A, B, C, D, E \)

at least as high as the marginal gain from adding the same element to a superset of \( S \). More formally a submodular function must satisfy

\[
f(X \cup x) - f(X) \geq f(Y \cup x) - f(Y)
\]

for all elements \( x \in \Omega \) and for all pairs \( X \subseteq Y \). Now, suppose \( f \) to be submodular, non-negative (i.e., takes only positive values) and monotone (i.e., adding an element to a set cannot cause \( f \) to decrease). Let also suppose that our aim is to find a set \( S \) of cardinality \( k \) such that \( f(S) \) is maximized. It has been proved in [25] that a greedy algorithm solves this problem with a worst-case approximation of \((1 - 1/e)\), where \( e \) is the base of the natural logarithm. In other words, if the optimum value is 100, the greedy algorithm is guaranteed to find a solution with a value of at least 63.

In order to apply this result to our greedy algorithm, \( c(P, N) \) must be submodular, non-negative and monotone. Non-negativity is immediate, since we measure an (approximation of) area. Monotonicity is also immediate, since adding an AOI to a set cannot change the number of tiles already counted. To prove submodularity, we show how it satisfies (1). Let us consider what happens when we add an arbitrary AOI \( x \) to a set \( X \) whose \( Y \) is a superset of: (i) \( x \) neither intersects with AOIs in \( X \) or AOIs in \( Y \). In this case the equality holds since the marginal gain for both sides of the equation is zero; (ii) \( x \) intersects only with AOI’s in \( X \). In this case we possibly have an increment on the left side, so the equality holds; (iii) \( x \) intersects only with AOI’s in \( Y \). In this case the left part of the equation is greater, since it considers all the area covered by \( x \), whereas the right part is incremented only of the part that is non overlapping, so the equation holds; (iv) \( x \) intersects with both \( X \) and \( Y \). The equation holds since for the left side it counts also the intersection of the AOI’s with the elements in \( Y \), that it would not count for the right side. Finally, since we have proved that our greedy algorithm is submodular, non-negative and monotone we can assert that in the worst case we obtain an approximation of \((1 - 1/e)\).

5. EXPERIMENTAL EVALUATION

This section presents the experimental evaluation of PAM. The simulations ran on a machine equipped with Java 7, 16GB of RAM, an Intel i5-2550 quad core @3,30 Ghz. Simulations have been run using epeerdemics [26], a thread-based gossiping simulation tool. In a single simulation run, each node executes a number of gossip iterations, and two consecutive iterations are separated...
by a fixed amount of time. This interval represents the gossip rate, i.e. how often a gossip iteration takes place. We fixed this value to 500 milliseconds. Each value reported in the graphs presented in this section was obtained by averaging the values of 20 independent simulation runs.

5.1. Virtual Environment and Mobility Model

For all the simulations we considered a virtual environment composed by a squared region of 5000 x 5000 points. The AOIs of the avatars have a radius of 100 points. The region has 5 circular fixed hotspots, whose radius is tuned so that the 20% of the total area of the region is a hotspot. The remaining 80% is considered as outland. Avatars move on the map according to realistic mobility traces that have been computed according to the mobility model presented by Legtchenko et al. [27], which simulates avatars movement in a commercial MMOG, Second Life [28]. In [29] we presented the implementation of this model, as well as a comparison with other mobility models. The model works according to the hotspots defined in the region. When an avatar reaches a hotspot, it explores the hotspot for a span of time and eventually it moves to another hotspot. This behaviour is defined by a Markov Chain characterized by three states: halted, exploring and travelling. When in halted state avatars stay still, whereas in the exploring state avatars explore a portion of the region close to their current position. Finally when in travelling state avatars move from one hotspot to another. This mobility model exposes a fair balance between the time spent by avatars in hotspots and outland. Furthermore, the path followed by avatars when moving between hotspots is not fixed, i.e. no predefined path connects two hotspots. Along with avatars, the region contains 1000 not moving objects. The 65% of the total number of the objects of the region lie in hotspot areas where their concentration follows a power law distribution, with a peak in the hotspot center. The remaining 35% of the objects stay in the outland area and they are distributed according to a uniform random distribution.

5.2. Evaluation Metric

The main metric to evaluate our approach is the difference between the local replica of the peer’s state against the authoritative server state. To measure this difference, we exploit a slightly modified version of the Jaccard similarity coefficient [30]. Let us consider \( C \) as the local replica of a peer and \( S \) as the remote replica of the server. The original Jaccard coefficient is computed as \( |S \cap C|/|S \cup C| \). However, this formulation either does not take the difference of the entities positions into account, or considers entities with a different position as distinct. In order to consider both the difference in position and the presence of the entities at the same time, we exploit the following formula to compute the Jaccard coefficient (in short JC):

\[
JC = \frac{\sum_{x_S,x_C \in S \cap C} 1 - \min\left(\frac{\text{dist}(x_S,x_C)}{d_{\text{MAX}}}, 1\right)}{|S \cup C|} \tag{2}
\]

where \( d_{\text{MAX}} \) is the diameter of the peer’s AOI. A peer with \( JC = 1 \) has its local replica perfectly synchronized with the state of the server while \( JC = 0 \) implies that the replica is completely out-of-sync with that of the server. Any value in between 0 and 1 gives a quantitative evaluation on the quality of the synchronization.

5.3. Number of Overlay Links

The aim of this section is to evaluate how the number of overlay links (i.e. the number of links maintained by the peers) affects the behaviour of the PAM. In the experiment we tried different amounts, ranging from 1% to 10% of the number of players. Naturally, a larger number of links increases the probability to find peers with useful and fresh data, which in turn yields to a better JC. Figure 9a shows that this behaviours holds with different number of players. A bigger overlay size also helps both the heuristics, as shown in Figure 9b, with a small advantage for the score-based algorithm. Note that in these tests we fixed the precision of the AOI approximation, which we explain later in this section.

Copyright © 0000 John Wiley & Sons, Ltd.
Concurrency Computat.: Pract. Exper. (0000)
Prepared using cpeauth.cls
DOI: 10.1002/cpe
5.4. AOI Approximation

The aim of this section is to evaluate how different levels of AOI approximation affect the effectiveness of PAM. To this purpose, we define as the precision of an AOI approximation the ratio between the radius of the AOI and the length of the diameter of the inscribed circle of the hexagon used as tile.

In our set of experiments we varied precision from 4 to 32. Surprisingly, higher values of precision does not lead to better results, rather they provides worse performances in terms of JC. This happens with different number of players (see Figure 10a), with both the algorithm strategies (see Figure 10b), and with different number of overlay link(see Figure 10c).

This behaviour can be explained by the fact that having more tiles requires more computation time for each peer to compute the ranking. By increasing the ranking time, the ranking itself may not be completed before the next gossip cycle is already started. In turn, this forces peers to deal with stale and inaccurate information, as they are prevented to communicate correctly with (a part of) their neighbours. This is in fact confirmed in Section 5.6, when we measure the actual CPU overhead to compute the ranking.

5.5. Number of Players

Figures 11a and 11b show how the number of players impact on the JC obtained by using the greedy and score algorithm. Generally, we can see that increasing the number of players has a positive effect on the JC. This behaviour was expected, as the PAM overlay was designed with the aim of exploiting local knowledge of the players. With more players around, there is simply more information to exploit. Figure 11b tells us also that the greedy algorithm slightly outperforms the
score one. Even this result was expected, it is interesting to notice that both algorithms benefit from the increases of players.

(a) Simulation with precision of 4 and using the greedy algorithm  
(b) Simulation with precision of 4 and a cache of 10%

Figure 11. JC with different number of players

5.6. Resource Consumption on Clients

The aim of this section is to evaluate, on clients, the following: (i) the impact of the number of players on the bandwidth, and (ii) the CPU overhead used to compute the ranking.

Regarding bandwidth, our evaluation focuses on the outgoing bandwidth, which is the typical bottleneck in standard asymmetric home connection. Naturally, the amount of bandwidth required depends on the amount of information exchanged by peers during iterations. In the coverage peer sampling, this information is composed by the peer descriptors (consisting of an unique id, the position and a time-stamp) contained in the peers’ cache. Hence, the dimension of the cache is the main parameter affecting the bandwidth. Figure 12a shows the average outgoing bandwidth (in KB per iteration) per peer per second to support the coverage peer sampling. The results are shown only for the score heuristics, as the bandwidth does not depends on the heuristic chosen. As expected, the bandwidth consumption increases with the dimension of the cache. Also, the bandwidth does not depend on the number of peers composing the network, but only on the size of the cache exchanged. In general, the bandwidth requirements to support the coverage peer sampling are relatively contained. For example, assuming an iteration every 250ms, with a cache dimension of 100 the bandwidth requirement is around 32 KB/s. Conversely, assuming a cache dimension of 20, which is a more typical scenario, the bandwidth requirement is less then 9 KB/s.

To measure the CPU overhead we counted the time spent by each peer in computing the ranking. This measurement is affected by the heuristic chosen, the dimension of the cache, and the precision degree. Figure 12b shows the average CPU overhead for the score and the greedy heuristics by varying the dimension of the cache. Figure 12c shows the same overhead by varying the degree of the precision. The experimental outcomes confirm the theoretical results discussed in the previous section. In particular, it is confirmed that the score heuristics scales good with the size of the cache and the degree of precision, whereas the performances of the greedy heuristics degrades. These measurements also show that the greedy heuristics takes a considerable amount of time to process caches larger than few peers, in fact making it unsuitable for a fast-paced MMOG applications.

5.7. Reducing Server Communication

The main purpose of the PAM is to be able to reduce the server outgoing bandwidth load, sacrificing none or few precision in the perception of the players. In this section we show empirical evidence of both these aspects.
We ran simulations by varying the interval of time ($T_s$) between two consecutive communications to the central server. For instance, with $T_s = 1$ the server communicates the authoritative version of the AOI to the players, every 4 simulation cycles.

Figure 13a shows the comparison of the JC between the greedy heuristic and the score heuristics. In general, we can observe how the reduction in the JC is limited even with high values of $T_s$. For instance, with $T_s = 1$ the average JC value for both the heuristics is around 0.9. From the application point of view, this means that the mechanism is able to fully support IM. As expected, further increments of $T_s$ imply a JC reduction. Note however, that even with the $T_s = 2.5$, the JC is still around 0.8. The effectiveness of the mechanism is further supported by the values of the JC when using only the server. In other words, increasing $T_s$ would be problematic if not supported by the overlay. For example, with $T_s = 1.5$, the JC with the support of the overlay is around 0.9, whereas is 0.65 using only the server.

To measure the reduction of outgoing bandwidth at the server, we have performed several tests considering networks with 200, 500, and 1000 players. Results are presented in Figure 13b. As expected, the amount of outgoing data transfer is greatly reduced by increasing $T_s$. With this reduction, the MMOG operator is able to evaluate alternative choices regarding the deployment of the IM server. For instance, let us consider an operator willing to deploy the IM server on a on-demand platform. With 1000 nodes, and $T_s = 0.25$ the bandwidth requirement is 3MB/s. Using the prices of a current commercial on-demand platform‡, the deployment would cost 30$ per day only considering bandwidth. With $T_s = 1$ the cost would be reduced to 10$ per day. This simple experiment shows how even a little reduction on the $T_s$ can result in a relevant saving for the MMOG operator.

6. RELATED WORK

In the last years numerous proposals have been presented to manage interest management in a P2P fashion. With respect to the literature, our approach is somehow unique: as far as we know it is one of the few approaches that exploits a best-effort gossip-based interest management resolution. The first part of this section presents a set of relevant works in the field, classified by the architecture, the P2P overlay, and the AOI management (see Table 1). Section 6.1 discusses a qualitative comparison between our approach and the state of the art.

Architectures. A possible classification for the architecture can be done by considering the difference in roles played by the nodes of the infrastructure. Hierarchical approaches consider some nodes having an enhanced role and importance with respect to regular nodes. For instance, [4] and

‡0.12$ per GB, Amazon EC2 prices, July 2012
[5] divide the virtual world into disjoint regions and pair a manager server to each region. Each manager is in charge to listen incoming events and broadcast them to the nodes of the region. We consider two types of hierarchical approaches: hybrid server and super-nodes.

Hybrid server approaches, such as [31, 32], combine public servers with P2P overlays. Such servers have a wide knowledge about the virtual world and typically work as backbones or backup. Rather, the peers of the P2P overlay relay only on local knowledge. By comparison to hybrid server approaches, in super-nodes solutions (such as [33, 34]) the role of backbone and backup is taken by a specific set of regular peers in the network. In order to cope with the unreliability that comes with peers, super-nodes approaches have to perform a careful node selection, organize failure recovery mechanisms and provide adaptive mechanisms for region sizing. Even if the hierarchical model may work well in practice, it might not scale well when the manager is assigned to a large region or in presence of crowded regions.

On the contrary, flat approaches do not perform any distinction between nodes, which have the same role and importance in the network. To this category belongs all the approaches that employs a single (or a combination of) P2P overlay, which are described in the following.

P2P overlay. One (or more) P2P overlay is constructed and maintained in order to support the interest management. This overlay can be either structured (such as Distributed Hash Tables) or unstructured (such as gossip-based or Voronoi networks).

Many solutions exploit DHTs to manage the state of the entities in the virtual environment [7, 8, 9, 6]. DHTs organize peers in a structured way, so that some properties, such as the length of routing path, hold even with large number of peers. In the context of a virtual environment, peers perform range-queries on the DHT to subscribe to the resources in their AOI in order to receive an update when their state changes. Usually, DHT-based approaches offer guarantees in terms of availability and embed failure recovery mechanisms, but suffer high latency when accessing and modify entities. To address this limitation, query-caching and pre-fetching mechanisms have been studied [6, 35], but they are ill paired with the dynamics of a MMOG. For these reasons, DHT are often used as a backup or a backbone mechanism, and queried only when necessary.

Compared with DHTs, unstructured overlays grant more emphasis to the dynamic grouping of MMOG by building the overlay according to the virtual proximity of the avatars. The most straightforward and easy-to-implement strategy is an overlay considering direct connections from all the closer peers [33, 36, 37]. This strategy yields a lower message latency, since nodes are one hop away from their neighbours. However, as the number of recipient nodes grows, this method may overload the bandwidth capability of the sources. To avoid source overloading, gossip-based [32, 38] and Voronoi-based [11, 36] approaches define an upper bound to the number of communicating neighbours and employ multi-hop communication schemas to disseminate information.
**AOI precision and shape.** The discrete AOI precision is used in those approaches that perform a static, grid-like partitioning of the virtual environment. For example, [7, 34] consider a static partitioning of the virtual world. In such approaches, the interest management is realized by considering the AOI of an avatar as the whole tile of the grid in which the avatar is located. Conversely, in continuous AOI precision, the AOIs do not correspond to a predefined region of the virtual world, but they are dynamically computed as entities move. The most recent proposals belong to this category. In some case, for example in [8], the size and shape of the AOI is adjusted dynamically according to the context, for example by shrinking the size of the AOI in a crowding scenario.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Architecture</th>
<th>P2P Overlay</th>
<th>AOI precision</th>
<th>AOI Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>our method</td>
<td>hybrid server</td>
<td>gossip</td>
<td>continuous</td>
<td>circle</td>
</tr>
<tr>
<td>SimMud [7]</td>
<td>flat</td>
<td>DHT</td>
<td>discrete</td>
<td>rectangle</td>
</tr>
<tr>
<td>MOPAR [34]</td>
<td>super-nodes</td>
<td>DHT &amp; unstr.</td>
<td>continuous</td>
<td>hexagon</td>
</tr>
<tr>
<td>P2P SecondLife [9]</td>
<td>flat</td>
<td>DHT (Kad)</td>
<td>discrete</td>
<td>rectangle</td>
</tr>
<tr>
<td>VON [36]</td>
<td>flat</td>
<td>voronoi</td>
<td>continuous</td>
<td>circle</td>
</tr>
<tr>
<td>Peer Clustering [31]</td>
<td>hybrid server</td>
<td>DHT</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HyperVerse [32]</td>
<td>hybrid server</td>
<td>gossip</td>
<td>continuous</td>
<td>circle</td>
</tr>
<tr>
<td>Hydra [31]</td>
<td>hybrid server</td>
<td>unstructured</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Colyseus [6]</td>
<td>flat</td>
<td>DHT</td>
<td>continuous</td>
<td>rectangle</td>
</tr>
<tr>
<td>Solipsis [33]</td>
<td>super-nodes</td>
<td>voronoi</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>pSense [38]</td>
<td>flat</td>
<td>gossip</td>
<td>continuous</td>
<td>circle</td>
</tr>
<tr>
<td>Message Exchange [37]</td>
<td>flat</td>
<td>unstructured</td>
<td>continuous</td>
<td>circle</td>
</tr>
<tr>
<td>Badumna [8]</td>
<td>dynamic approach</td>
<td>flat</td>
<td>discrete</td>
<td>square</td>
</tr>
<tr>
<td></td>
<td>gossip based</td>
<td>flat</td>
<td>continuous</td>
<td>circle</td>
</tr>
</tbody>
</table>

Table I. Classification of Interest Management approaches

### 6.1. Qualitative Comparison

Our proposal defines an interest management technique based on a hierarchical architecture. It includes a server and a P2P unstructured overlay where the peers communicate through a gossip protocol. To reduce the load on the server, each peer dynamically maintains connections with a subset of the peers in its AOI. Our P2P network is designed to work well in a crowded region, when the peers are likely to have theirs AOIs overlapped. Our proposal shares some traits and motivations with the approaches in [32], [38] and [8], but also some substantial differences. A trait characterizing our solution is that we dynamically build a specific overlay to support the interest management. Such overlay is built by exploiting the notion of coverage peer sampling. This mechanism allows a peer $P$ to retrieve, among its neighbours, those whose AOI maximize the overlap with its own AOI. $P$ establishes a connection only with these peers which, in turn, can notify it further entities of interest. pSense [38] maintains both a list of near nodes and a list of sensor nodes to manage the P2P overlay. The near nodes list contains peers that are within the vision range of the local node and supports interest management. The local node attempts to keep its list as accurate as possible. The sensor node list contains nodes that are outside the AOI of a peer, but close to its borders. The sensor nodes of a node notify it about the presence of new nodes entering its AOI so to avoid network partitioning. Sensor nodes must be distributed as evenly as possible around a node to guarantee uniform connections with the rest of the world. A localized multicast is exploited to spread position...
updates to the peers belonging to the AOI. With respect to [38] our approach reduces the number of connections established by each peer and, as a consequence, it is expectable a reduction in the bandwidth usage.

In [32], the authors propose a mechanism that integrates backbone servers with an unstructured P2P overlay to support interest management. The overlay is built by exploiting the concept of cluster, i.e. a region of the virtual environment where there is an high density of peers. Inside a cluster, clients exchange information about entities in a pure distributed fashion, using a torrent-like protocol. Outside the cluster, clients connect to the backbone server. With respect to [32], we do not explicitly recognize clusters in the virtual environments. Therefore, peers can benefit from situations when the density is high even if it is not possible to identify an entire cluster. In addition, since the cluster is seen as a black-box by the backbone server, failure of peers may lead to loss of data. Instead, in our approach, peers periodically communicate with the servers, ensuring the authoritative replica of data on the server to be up-to-date. However, the communication frequency with the server must be carefully tuned. Another possibility is to use an adaptive approach like in [8]. In any case, a low communication frequency may be useful to save both computational resources on server, and bandwidth on peers. However, gossip-based events delivery may be not fast enough for some particular type of events (such as firing a bullet). In such cases a low communication frequency with the server may lead to inconsistencies in the behaviours of avatars. Following these considerations, we believe that our approach might be not an ideal fit for fast-paced environments.

Badumna [8] supports interest management by switching between DHT, DHT with adaptive regions, and gossip according to the amount of interactions among players. At the start, the virtual world is divided into squared regions. These regions are then assigned to manager peers by using a DHT. When the number of interactions increase, peers are grouped in custom shaped super-regions, which are seen like single regions to the DHT. However, when the interactions further increase, peers enter in gossip mode. In gossip mode, peers communicate directly to each other, but still interact with the manager of the region at a reduced frequency. Unlike our approach, in Badumna the region manager is not a server but a regular peer. To limit the amount of load on regular peers, Badumna adapts automatically the frequency of communication with the region manager by taking into account the amount of iterations among peers.

7. CONCLUSION

This paper proposes a best-effort mechanism to support interest management in MMOG applications. The core of the proposal is represented by a gossip-based overlay construction, based on the novel concept of the coverage peer sampling. To keep the system fast, simple and lightweight, we analyzed two different algorithms (score and greedy) to implement the coverage peer sampling. From the analysis of the algorithms and from the experimental outcomes, we observed that the greedy algorithm yields more precise results but at the cost of a considerable computational complexity. Therefore, we concluded that the score-based algorithm, which is fast although less precise, works better in a fast-pace environment like a MMOG application. In addition, we also propose an estimate of the economical impact a MMOG operator obtain by applying our approach to a server-based architecture.

Our proposal can be further extended and studied. To this end, we plan to improve the precision of the result by considering additional information when ranking peers, such as movement forecasts and different neighbours selection functions. As to further validate our solution, we intend to test it with movement traces from different mobility models, and to compare it with the non best-effort works presents in literature. As a further improvement, we plan to extend the PAM server with a multi-server architecture. This would open to the proper exploitation of pay-per-use platforms which may enable a dynamic adjusting of the computational resources of the PAM server.
ACKNOWLEDGEMENTS

The authors acknowledge the support of Project FP7-257438, Contrail: Open Computing Infrastructures for Elastic Services. The authors thank the reviewers for the complete and helpful reviews. The authors would also thank Patrizio Dazzi, Rossano Venturini and Luca Colombini for their substantial help.

REFERENCES