

Mobile Online Gaming via Resource Sharing

Stefano Ferretti
sferrett@cs.unibo.it

Gabriele D'Angelo
g.dangelo@unibo.it

Department of Computer Science
University of Bologna
Mura A. Zamboni 7, I-40127 Bologna, Italy

ABSTRACT

Mobile gaming presents a number of main issues which remain open. These are concerned mainly with connectivity, computational capacities, memory and battery constraints. In this paper, we discuss the design of a fully distributed approach for the support of mobile Multiplayer Online Games (MOGs). In mobile environments, several features might be exploited to enable resource sharing among multiple devices / game consoles owned by different mobile users. We show the advantages of trading computing / networking facilities among mobile players. This operation mode opens a wide number of interesting sharing scenarios, thus promoting the deployment of novel mobile online games. In particular, once mobile nodes make their resource available for the community, it becomes possible to distribute the software modules that compose the game engine. This allows to distribute the workload for the game advancement management. We claim that resource sharing is in unison with the idea of ludic activity that is behind MOGs. Hence, such schemes can be profitably employed in these contexts.

General Terms

Algorithms, Performance, Theory

Keywords

Multiplayer Online Games, Peer-to-Peer, Mobile Architectures

1. INTRODUCTION

In the last years, Multiplayer Online Game (MOG) technologies have been widely studied. Several solutions have been identified which are concerned with many aspects of MOGs. An open aspect relates to the development of mobile games. Nowadays, portable game consoles exist which are quite powerful computing devices, equipped with networking technologies. For this reason, from a certain point of view the player does not expect great differences between

playing using a mobile game console and using a PC (apart from the obvious graphic limitations and the small screen of a mobile device). He/she expects to be able to generate inputs, perceive the evolution of the game in real-time, interact with other distributed players in the (more or less) same way when he/she uses a traditional PC/game console. However, the truth is that several technical constraints limit the gaming activities of mobile players.

Mobile devices are primarily constrained by the limited battery capacity. Every task, computation or communication has a power cost; hence these activities should be limited as much as possible in order to preserve the battery lifetime. Communications pass through wireless channels, which can be less reliable and offer lower data rates than traditional wired ones. This should be taken into consideration during the design of the game. In particular, different alternatives are possible depending whether the game is played among mobile players located in the same (limited) geographical area rather than when distributed players and nodes connected through the Internet must be reached. In the first case, in fact, a Mobile Ad-Hoc Network (MANET) can be built to optimize the game update distribution. In the second case, these forms of interaction are usually based on mobile devices that connect to their nearest access point to access the Internet. However, the proliferation of heterogeneous devices with different capabilities gives rise to new scenarios that promote the cooperation among individuals in order to guarantee the provision of "always on" services [8].

Resource sharing and optimization opens novel interesting scenarios for the deployment of mobile MOGs. There are two ways to optimize the digital resources usage. First, mobile users have many different portable devices in their pockets and suitcase, each of them with specific hardware and software characteristics. Quite often such devices are not enabled for seamless interaction with other devices belonging to the same owner. Instead, the use of the networking capabilities of a user mobile phone might, for instance, relieve the gaming console from performing long-range communications with other nodes participating to the game. Hence, the game console might communicate with the user's mobile phone through a short range networking technology (e.g. Bluetooth, ZigBee); in turn, the mobile phone relays messages generated by (or directed to) the game console by employing its long-range network technology (e.g. 3G, Wi-Fi).

Then, there is the opportunity of sharing resources among different people. The possibility for a user to exploit, in a Peer-to-Peer (P2P) and altruistic way, computing facilities

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DISIO Workshop 2012, March 19-23, Desenzano del Garda, Italy

Copyright © 2012 ICST 978-1-936968-47-3

DOI 10.4108/icst.simutools.2012.247720

ties owned by (known and trusted) neighbor players requires mechanisms for automatic negotiation [8]. In this case, a node might share its computation with its neighbor, for instance when it updates the game state due to novel received game events. Alternatively, a node might act as a relay for other ones during the distribution of game updates.

In a recent paper, we have identified mechanisms to promote resource sharing among distributed nodes in wireless communication environments [8]. In this paper, we describe the main possibilities arising from resource sharing for the deployment of novel mobile MOGs. Specifically, we review the main software components composing an online game, and discuss how and when these software modules can be distributed and/or replicated at mobile nodes.

In fact, certain software modules have to be executed at all nodes, such as those for managing inputs from the player, rendering the game state evolution and performing basic networking to enable communication with other nodes. Other software modules, instead, can be distributed (and replicated) at different nodes such as those for the game state management, the network overlay management (when the communication occurs through a MANET), and several sub-components of the game engine, such as those for simulating the physics of game objects, identifying collisions, etc. Usually, such modules are implemented as event-driven software components. This eases the distribution of these modules, since the game engine becomes a discrete-event based distributed system.

It is clear that when a node executes a task, additional communication is required to share the outputs of this module with other nodes. This has an additional cost, besides that required for executing the software module itself. Hence, in certain cases these software modules might be replicated to diminish the communication workload of the nodes with other ones and also to improve scalability and fault-tolerance, thus preventing that a single node becomes the bottleneck of the system. When certain modules are replicated, synchronization and consistency management techniques must be employed [11, 21].

We claim that resource sharing is in unison with the idea of ludic activity that is behind MOGs. Hence, such schemes can be profitably employed in these contexts.

The rest of this paper is organized as follows. Section 2 discusses the main architectural solutions which may be employed to support MOGs, with specific attention to games deployed for mobile devices. Section 3 discusses how effective mobile architectures for the support of mobile MOGs may be devised, which resort to resource sharing and to the distribution of software modules. Section 4 outlines some main aspects to consider when allocating and distributing software modules in a mobile MOG architecture. Finally, Section 5 provides some concluding remarks.

2. WIRELESS MOG ARCHITECTURES

This section presents an overview of the distributed software architectures that can be employed to build MOGs on wireless networks. Basically, a first distinction can be made between those gaming applications that involve mobile users placed in a localized geographical area only, rather than those games whose nodes need to communicate through the Internet.

When players are all confined in the same Wireless Local Area Network (WLAN), their mobile nodes might organize

in an ad-hoc manner, i.e. they form a MANET, and each node reaches another node using a multi-hop relays. Hence, all the communications remain within the MANET.

On the opposite case, nodes communicate through the Internet using some structured communication architecture such as, for instance, a 3G cellular networking technology (e.g. UMTS), rather than some structured Wi-Fi network. At the data-link layer, each node communicates with its access point directly, as in all traditional wireless Internet-based communications.

There is however a hybrid solution, according to which mobile nodes on the same geographical area communicate in an ad-hoc manner; but when needed, data can pass through the Internet, usually to reach some server and/or other distributed nodes not belonging to the MANET. In such a case, one or more (possibly all) nodes in the MANET must be able to send messages outside the local network. Hence, in this case some specific wireless communication technology must be exploited which is different to the network technology employed to interact in an ad-hoc fashion with other mobile neighbor nodes. In other words, some mobile node might exploit multiple network interfaces concurrently.

Despite the underlying communication architecture, at the application layer different solutions are possible, ranging from the client/server to the P2P scheme [13]. Our proposed solution adopts a hybrid approach. Next sections are devoted to describe this solution in detail.

Note that in this section, we consider each node is composed of a unique device. Hence, we do not take into account that a player might have actually different devices that might be coordinated. If this is the case, then the interaction of this set of devices might be properly configured, so that the mobile node forms a sort of “digital organism” able to exploit all its devices effectively [8].

2.1 The Client/Server Scheme

The client/server scheme is widely employed in traditional MOGs, where a fixed node is usually employed as the central node that maintains the game state and interacts with all other nodes in the network. It has been recognized that such a solution is quite simple to implement but, on the other hand, it raises several reliability and fault-tolerance issues.

In few words, client nodes execute the following two software components: i) the user interface that collects inputs from the user and to render the game state evolution; ii) a software module in charge of managing the network communication. The server has additional software modules to manage the game state evolution (we neglect additional software components that are used to perform offline operations, e.g. accounting).

Figure 1 shows such an architectural approach. While in common scenarios the server is in direct connection with all the clients (Figure 1(a)), when a mesh overlay structure is employed (like in a MANET) messages from certain client nodes might require multiple hops before reaching the server (Figure 1(b)). Note that in this case each node must run an “overlayManagement” software module in charge of managing the overlay and control the routing of messages passing through that node. This approach presents several disadvantages when employed over wireless networks. Indeed, the communication among the server and clients might be quite unstable. Nodes may move during their interactions;

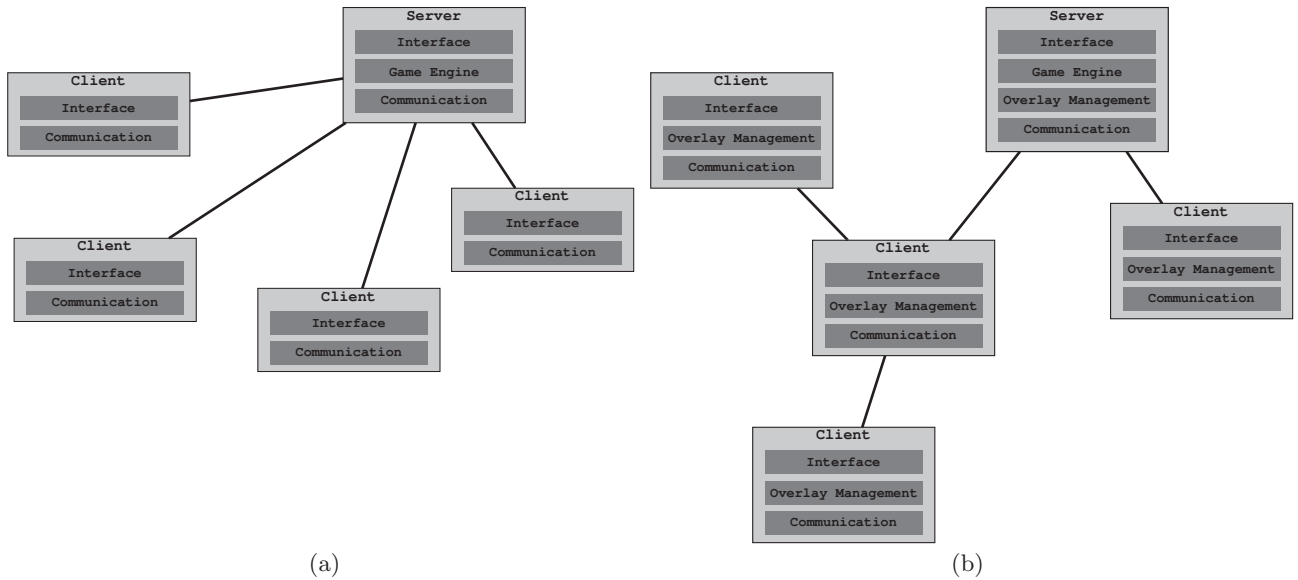


Figure 1: Client-Server Model: a) all clients connected to the server; b) with overlay

this would require reconfigurations of the overlay, due to the fact that some communication links may become unavailable. There are problems related to the fact that mobile nodes have restricted battery power. When a node fails, also the communications passing through it fails consequently. In substance, the overlay management may require an intense configuration and communication overhead.

2.2 The Peer-to-Peer Scheme

In a pure P2P system, each mobile node stores and manages its own copy of the game state. Hence, each peer executes all the software modules which in the client/server approach were executed on the server only (see Figure 2).

As concerns the overlay topology, a possibility is to organize nodes based on a fully connected network. Actually, this approach is impractical (or even impossible) in certain MANETs, since a node might not be able to reach all the nodes with a single-hop message transmission. The adoption of a mesh overlay network for disseminating game events represents a viable and more scalable choice. Of course, a careful management of the overlay is needed, so as to ensure that game events are delivered to all nodes in a timely fashion and that the game advances in real-time.

A possible technical solution is to resort to broadcast (or multicast) wireless communications in MANETs. The idea is that a single wireless message can reach multiple mobile nodes; this can be reasonably exploited in MOGs, since usually all game updates must be sent to all (or a subset of) peers. Such broadcast schemes can be employed with multi-hop communication protocols, in order to span the message through the entire overlay.

The clear advantage of a P2P scheme is that it removes the presence of a single server, which represents the bottleneck and single point of failure of the system. However, every node must execute all software modules, hence augmenting the computational work at each node (this has consequences on the battery consumption also). Moreover, since a P2P architecture is structured typically as a mesh overlay, churns

must be viably managed.

2.3 Game Software Modules

In the figures above, for the sake of a simpler presentation we referred to the game software modules by taking into consideration a coarse range of generic functionalities such as the “interface”, “game engine”, “overlay management”, “communication”. Actually, a more detailed set of software modules can be listed, based on the tasks associated. Based on their functionalities, some of them can be assigned to different nodes to be executed in a distributed architecture.

- **Input management.**

This software module includes all the functionalities needed to manage game events produced by the player, which causes a game state update.

- **Audio management and sound system.**

This module is responsible to manage all the audio sounds to be played, associated to particular game events occurring during the game. It is also responsible for the background audio, which is usually played during a game session. Mixing is required among audio played in background and sounds associated to specific game events.

- **Scene graph engine.**

Here, we include all the functionalities needed to render the game on the player screen. This includes management of (2D/3D) objects, skeletal animation, texturing and imaging, lighting and shading, rendering of the terrain, water, smoke, clouds and the like.

- **Physics system.**

This module includes point and rigid body dynamics, soft body dynamics, fluid dynamics and kinematics in general. In substance, this is a dynamics simulation component, which is responsible for managing and solving the simulated physical forces affecting the simulated game objects.

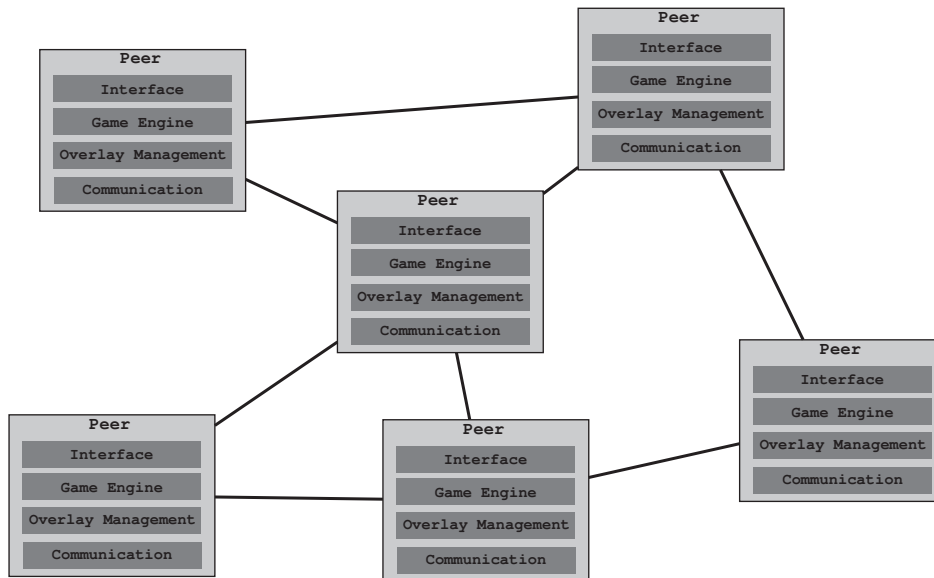


Figure 2: Peer-to-Peer Model

- **Collision detection.**

This is the module in charge for determining when two game objects collide during their movement. For the purpose of reducing the computation needs, usually simplified objects are employed; then, these simplified meshes (bounding boxes, spheres, convex hulls) are used for determining collisions. Besides these mentioned simplifications on the game objects' representations, such a task is computation demanding, nevertheless. Such calculations must be performed periodically, at a constant frame-rate. The higher the frame-rate the more accurate the model for determining collisions. Hence, this software modules should be executed on computationally efficient nodes.

- **Game state management.**

This module maintains the state of all the game objects (characters and other virtual objects) on the game map and, based on the events produced by all participants, updates it. It checks the validity of game events produced by players and works in strict collaboration with the collision detection module and the physics system. Indeed, in certain cases we might think that the two modules mentioned here above are included in this one, being their tasks employed for computing game advancements.

- **Virtual map and scene storing service.**

Usually, the information that describes the virtual map area is replicated on each host. However, especially when the game has a huge virtual map, then each node can maintain only the part of the virtual world where its character is located at that time; as soon as the character moves on another region, then the novel virtual area is downloaded on the terminal. If this is the case, then a server (or a set of servers) is responsible to maintain the complete virtual world; this is necessary to ensure that the node can download the virtual world description when it needs it. When a distributed P2P architecture is employed, a server node might still be

employed that maintains the whole virtual map, rather than distributing the map among different nodes. Of course, parts of the map might be replicated for the purposes of reliability and of an easier and more efficient data distribution.

- **Artificial intelligence.**

This software module is responsible for the management of virtual bots interacting with users in the virtual world. Some node executes the Artificial Intelligence (AI) module that decides the moves each bot performs during the game evolution. Usually, all these tasks are in charge to the server. When a decentralized solution is employed, the management of different virtual bots can be delegated to different nodes, such as in a mobile multi-agent system.

- **Finite state machine management.**

This module governs the evolution of the game as determined by users' actions. Based on the actual game state and on the events generated by users, the game can evolve based on certain rules. All this can be implemented through a finite state machine. (Actually, finite state machines can be employed also to specify the behavior of virtual bots; thus, they represent a possible tool employed to realize the AI of virtual players.)

- **Prediction schemes and dead reckoning.**

Dead reckoning is a technique employed in MOGs to reduce the effects of network induced delays and losses by applying prediction schemes [22]. Each node routinely uses dead reckoning to predict where an actor might be located at a given time, based on past information on its last known kinematic state. When correctly employed, it allows to avoid that each node sends game state updates that can be easily inferred from previous information. By resorting to such an approach, the use of the network is reduced and the

game advancement fastened. The quality of the prediction is thus quite important in order to ensure that all players perceive the game evolution in a consistent way.

- **Accounting and Score Management.**

These functionalities, concerned with the management of accounts and scores of players participating to the game, are prone to cheating. Hence, when executed on a P2P architecture, proper strategies must be taken into account so as to prevent that some malicious actor alters some information, or acquires data it is not allowed to access [6, 10].

- **Networking.**

Common communication capabilities are required to let mobile nodes to communicate. Depending on the game implementation both UDP or TCP transport level protocols can be employed. Indeed, while UDP is recognized as the typical choice for transporting data of real-time multimedia applications, some works suggest that some sort of tuned TCP represents an interesting alternative [15, 21]. While in typical situations the mobile node is configured to exploit a single network interface card, in general situations it might be the case when the mobile node is enabled to concurrently exploit multiple network interface cards. This would promote interesting novel communication scenarios; for instance, it would allow a node to interact with other nodes geographically located near it through some short range communication technology, while using at the same time long range communication technologies to reach other hosts on the Internet. Not only, sophisticated communication schemes can be employed to let the node to exploit different network interface cards to communicate with another host, in order to guarantee seamless interactions with it [14].

- **Overlay management.**

We already mentioned that in a MANET, nodes operate as both end hosts and routers, forwarding packets wirelessly towards other mobile nodes that may not be within the direct transmission range of each other [5]. Thus, routing strategies are needed which are able to adapt depending on the availability and position of nodes. Each node must thus maintain a table with its neighbors, in order to relay messages. Moreover, reconfiguration strategies are needed to adapt to the overlay changes.

3. ON THE OPTIMAL ORGANIZATION OF RESOURCES

In this section, we propose a distributed solution for the deployment of ubiquitous gaming applications. All devices available to each player are dynamically and adaptively configured depending on:

1. the devices themselves,
2. the environment in which they are deployed, and
3. the computational and communication capabilities of devices of other players.

The architecture must thus provide configuration protocols for the intra Personal Area Networking (PAN), to automatically organize all devices belonging to a single player. At the same time, it is necessary to identify algorithms and mechanisms for the simultaneous and adaptive use of different communication networks in an opportunistic fashion. In fact, the overall goal is to optimize the interactions across all players in the wireless overlay. Finally, we need mechanisms for the efficient distribution of software modules composing the game engine. This allows to create a distributed game engine whose execution spans through the whole mobile MOG architecture.

In essence, the idea is to resort to a hybrid architecture, where each software module can be executed on a specific node, depending on its computational and communication capacities. Hence, computation is distributed among different nodes. Similarly to a client-server approach, these services (e.g. computation of the game state evolution) must be made available to other nodes that do not have that software module locally active. Software modules might be replicated, so as to augment reliability and distribute the communication load with other nodes that require the service.

When needed, also the game engine can be divided into subcomponents, so as to distribute the computational load even further among players. For instance, while a node might be selected to perform collision detection, another node might be in charge of performing the AI of some virtual bots, and so on. (This specific example might lead to several security concerns, and in this case cheating prevention schemes would be required.) Each node must run the interaction and communication modules, so as to permit interactions with the user and other nodes, respectively.

Several heuristics and optimization strategies can be implemented to distribute and replicate all the software modules and the virtual world map. These schemes must take into account:

- the geographical location of players, and thus the overlay topology that may be built, based on players' position;
- the node capabilities in terms of computation capacity, communication, memory, status of the battery;
- the need to interact with nodes external to the MANET, i.e. Internet nodes. This depends on the specific game application.

To make this possible, we need to manage and optimize i) the interactions among the different devices that each single player has in his/her hands while playing and, ii) the interactions among different players in the overlay mesh. Then, it is possible to distribute all the software modules and create a smart game management architecture that may improve the quality of the gaming experience to all the mobile players.

3.1 Optimizing the Player's Devices

Full interaction among all digital devices held by each single player might promote the deployment of effective mobile MOGs [8]. This implies to optimize the use of available networking technologies, such as short-range communication technologies, e.g. Bluetooth, infrared, ZigBee, etc., for communicating with other players which are located near the considered node. The goal is to find the best configuration

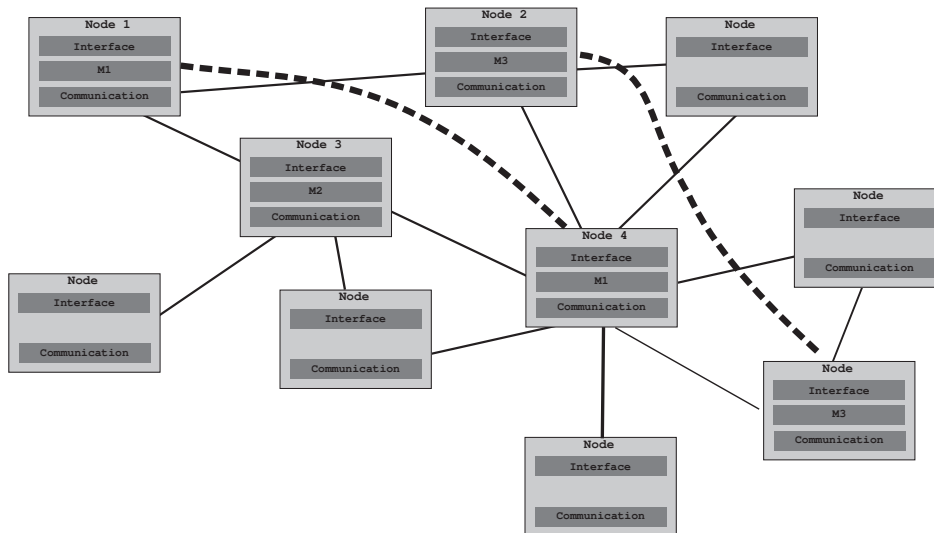


Figure 3: Distribution of Software Modules on the Mesh

for all the devices in use at each player. Specifically, based on the computational capacities of each device within the PAN, the battery levels, and the available network interfaces, devices must be configured so as to identify a primary computation entity, a primary gateway to send/receive data from the outside world, secondary network interfaces (e.g. short range ones) to allow communications with neighbor players.

According to our architecture, all devices exchange their profiles among each other, in order to enable a proper system configuration. Different alternatives exist to characterize profiles of devices, such as, for instance, CC/PP [1]. Such information is exploited to identify the coordinator, i.e. the device that acts as the resource manager of the player's devices. To accomplish this task, all devices' profiles must be distributed among the whole local device set, and some distributed algorithm must be executed to elect the coordinator. A similar approach must be employed to identify which device is to act as the primary gateway that manages communications with the outside world.

Upon a proper organization of the user's devices, that device set might be seen as a unique computation/communication node with a set of features which is composed by the aggregation of featured of single devices.

3.2 Optimizing Interactions Among Nodes

Peer players must be provided with a set of protocols to interact with others within their MANET. These protocols would allow a peer to opportunistically and dynamically adapt the interaction with its neighboring peers, by selecting the best communication protocol among the available ones (e.g. Always Best Connected, ABC) [12, 14]. The identification of the best available network may be based on different criteria such as bandwidth, connection cost, battery level and so on. Any of these criteria, alone or together with the others, can be used for assessing the best available network at any time.

4. DISTRIBUTION OF SOFTWARE MODULES OVER THE MESH

Figure 3 shows an example of a mobile MOG architecture,

where nodes are connected through a mesh overlay (small, continuous lines). Every node executes the software module interface, that refers to the capabilities of managing inputs from the user and outputs to be shown to the user; moreover, every node executes the communication module, which allows to interact with other nodes. As already mentioned, these two modules are mandatory to every node.

Then, other modules are distributed to be executed on certain nodes on the overlay only. Since the figure refers to an hypothetical example, we avoided to list specific names of modules composing the game engine. Hence, in the figure we used generic names $M1, M2, \dots$. Some of these modules are replicated (see larger, dashed lines in the figure). In this case, other nodes select to which node refer for that service. For instance, node 1 and node 4 both execute module $M1$. Hence, others select the one that requires less distance hops.

In substance, when configuring such an architecture, there are some main aspects to take into consideration, i.e. allocation of software modules, generation of the overlay mesh, distribution of nodes acting as clients for a given service executed at another node, synchronization of states managed at replicated services so as to ensure state consistency.

4.1 Modules Distribution

Here, we discuss which software modules can be distributed and or replicated, among those mentioned in the previous section.

- **Input management, Audio management and sound system, Scene graph engine.**

As already outlined above, and shown in Figure 3, these modules must be executed on each node.

- **Physics system, Collision detection, Game state management.**

These are main modules that can be distributed on some nodes. They can be replicated also, for the sake of scalability and fault tolerance. However, the outputs of each of these modules are important for other modules. In substance, these modules form a sort of core game engine. Hence, probably the best choice is

to execute them on the same nodes.

- **Virtual map and scene storing service.**
As already mentioned, this module (and mostly, the data composing the game virtual map) can be assigned to a third server rather than being distributed on different nodes.
- **Artificial intelligence.**
Each virtual bot and its AI can be executed on a given peer node, and possibly different bots might be managed at different peers.
- **Finite state machine management.**
Being one of the main functionalities of a server (in a client/server architecture), this module can be distributed.
- **Prediction schemes and dead reckoning.**
This module is client-specific, executed to hide communication latencies, hence it must be executed on every node.
- **Accounting and Score Management.**
It can be assigned to a distributed (trusted) node.
- **Networking.**
It must be executed on each node.
- **Overlay management.**
It can be distributed on nodes that would be in charge of deciding how to organize the overlay.

4.2 Allocation of Software Modules

When distributing software modules, several mechanisms can be employed to have a fair allocation over distributed nodes, taking into account the computation and communication capacities of the nodes. One possible solution is to employ simple heuristics. For instance, having an estimation of the workload of a software components, which depends on the number of users to be served, given the number of nodes it is possible to identify the number of replicated nodes acting as “servers” for that service. Then, such distribution can be performed by ranking nodes based on their computation/communication/battery capacities and on their geographical location (probably, it would be preferable to uniformly distributing the services on the overlay). This task should be repeated for each service.

Another option is to optimize the allocation by matching demands for executing different services and offers, using some kind of market-based approach in which requests are handled through ascending clock auctions [2, 16, 18, 24]. Such an approach can be performed by running a distributed algorithm to carry on the auction; then, the resource allocation problem can be treated as a classic optimization problem, which is used to compute the maximum number of allocations that can be matched.

It is clear that when several software modules are distributed over different nodes, there are several security concerns that must be considered. Cheating is a main problem in P2P MOGs; hence viable strategies must be enforced to prevent and detect cheats [10]. Moreover, some form of authentication and authorization must be considered in order to identify players that ask services to other distributed nodes.

4.3 Management of the Overlay Mesh

The task of generating and managing the overlay mesh can be performed by resorting to one among the plethora of proposals that manage mesh overlays in a MANET. Examples are works presented in [4, 5, 19, 25].

4.4 Associating Clients to Software Modules

Once software modules have been distributed and a mesh overlay has been built, nodes not running a given module must ask for updates to a node running the service. Also in this case, several options are possible, each one with its pros and cons. Each node might ask to its nearer node running the service. Thus, upon a novel update to be disseminated, that node would receive the update in the fewest number of hops. However, several updates from different services travel through a path; moreover, a given node might be overloaded with too many nodes to serve. These issues might influence the performance of this approach. Moreover, since an update must be multicast to several nodes, effective strategies might be employed that ensure that the mesh overlay is covered in a minimal number of hops.

In substance, the problem here refers to building a publish-subscribe scheme over a mobile overlay. Works that deal with this issues have been presented in [3, 9, 17, 23].

4.5 Synchronization

Replicating software modules at distributed nodes provide several advantages. Indeed, this approach augments the scalability of the system, its fault-tolerance and might improve the responsiveness of the interaction between the client node and that running the service, due to the fact that this node has to manage a lower number of client nodes.

However, when the replicated software module deals with the game state management, then synchronization algorithms must be executed between nodes running the service. This is important to guarantee that each node perceives a consistent evolution of the game state. There are different alternatives in this case, such as [7, 11, 20, 21].

5. CONCLUSIONS

Mobile MOG architectures may benefit from viable resource sharing strategies. In this paper, we have discussed a methodology to optimize the interactions of mobile players into dynamic and heterogeneous environments. The idea is to optimize the use and interaction of the devices available to each player, through dynamic and adaptive configuration strategies (optimization of the PAN). Then, interactions among players can be optimized using both the available communication infrastructures and P2P ad-hoc interactions. Based on these resource sharing mechanisms, software modules composing the game engine can be distributed (and replicated) across the whole gaming network. This would distribute the workload needed for the game advancement, hence resulting in an improved resource usage in a mobile environment.

As concerns the general deployment of the proposed scheme in a real distributed system, there are some open problems that require further investigation. Security issues are particularly important: for instance, cheating is a primary issue to deal with in this case. Moreover, authorization and authentication must be enforced to verify the identity of users that exploit resources of their neighbor players.

6. REFERENCES

- [1] CC/PP Information Page, <http://www.w3.org/Mobile/CCPP/>, December 2010.
- [2] L. M. Ausubel and P. Cramton. Auctioning many divisible goods. *Journal of the European Economic Association*, 2(2-3):480–493, 2004.
- [3] S. Baehni, C. S. Chhabra, and R. Guerraoui. Frugal event dissemination in a mobile environment. In *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, Middleware '05, pages 205–224, New York, NY, USA, 2005. Springer-Verlag New York, Inc.
- [4] L. Canourgues, J. Lephay, L. Soyer, and A.-L. Beylot. A scalable adaptation of the olsr protocol for large clustered mobile ad hoc networks. In P. Cuenca, C. Guerrero, R. Puigjaner, and B. Serra, editors, *Advances in Ad Hoc Networking*, volume 265 of *IFIP International Federation for Information Processing*, pages 97–108. Springer Boston, 2008.
- [5] M. C. Castro, A. J. Kassler, C.-F. Chiasserini, C. Casetti, and I. Korpeoglu. Peer-to-Peer Overlay in Mobile Ad-hoc Networks. In X. Shen, H. Yu, J. Buford, and M. Akon, editors, *Handbook of Peer-to-Peer Networking*, chapter 37, pages 1045–1080. Springer US, 2010.
- [6] C. Chambers, W.-c. Feng, W.-c. Feng, and D. Saha. Mitigating information exposure to cheaters in real-time strategy games. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '05, pages 7–12, New York, NY, USA, 2005. ACM.
- [7] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. *Multimedia Tools Appl.*, 23:7–30, May 2004.
- [8] G. D'Angelo, S. Ferretti, V. Ghini, and F. Panzieri. Mobile computing in digital ecosystems: Design issues and challenges. In IEEE, editor, *Proceedings of the 7th International Wireless Communications and Mobile Computing Conference (IWCMC 2011)*, July 2011.
- [9] A. Emmanuelle, A. K. Datta, M. Gradinariu, and G. Simon. Publish/subscribe scheme for mobile networks. In *Proc. of the second ACM international workshop on Principles of mobile computing*, POMC '02, pages 74–81. ACM, 2002.
- [10] S. Ferretti. Cheating detection through game time modeling: A better way to avoid time cheats in p2p mogs? *Multimedia Tools Appl.*, 37(3):339–363, 2008.
- [11] S. Ferretti. A synchronization protocol for supporting peer-to-peer multiplayer online games in overlay networks. In *DEBS '08: Proc. of the second international conference on Distributed event-based systems*, pages 83–94. ACM, 2008.
- [12] S. Ferretti and V. Ghini. A web 2.0, location-based architecture for a seamless discovery of points of interests. In *Proc. of the 2009 Fifth Advanced International Conference on Telecommunications*, pages 226–231, Washington, DC, USA, 2009. IEEE.
- [13] S. Ferretti and M. Roccetti. Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games. In *ACE '05: Proc. of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 405–412. ACM, 2005.
- [14] V. Ghini, S. Ferretti, and F. Panzieri. The "always best packet switching" architecture for sip-based mobile multimedia services. *Journal of Systems and Software*, 84(11):1827–1851, 2011.
- [15] C. Griwodz and P. Halvorsen. The fun of using tcp for an mmorg. In *Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '06, pages 1:1–1:7, New York, NY, USA, 2006. ACM.
- [16] J. Huang, Z. Han, M. Chiang, and H. V. Poor. Auction-based distributed resource allocation for cooperation transmission in wireless networks. In *Proceedings of the Global Communications Conference, GLOBECOM '07*, pages 4807–4812. IEEE, 2007.
- [17] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wirel. Netw.*, 10:643–652, November 2004.
- [18] T. Kelly. Generalized knapsack solvers for multi-unit combinatorial auctions: Analysis and application to computational resource allocation. In *Agent-Mediated Electronic Commerce VI, Theories for and Engineering of Distributed Mechanisms and Systems*, volume 3435 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2005.
- [19] Y. Li, S. Peng, and W. Chu. K-tree trunk and a distributed algorithm for effective overlay multicast on mobile ad hoc networks. In *Parallel Architectures, Algorithms, and Networks, 2008. I-SPAN 2008. International Symposium on*, pages 53–58, may 2008.
- [20] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. Local-lag and timewarp: providing consistency for replicated continuous applications. *Multimedia, IEEE Transactions on*, 6(1):47–57, feb. 2004.
- [21] C. E. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Roccetti. Interactivity-loss avoidance in event delivery synchronization for mirrored game architectures. *IEEE Transactions on Multimedia*, 8(4):874–879, 2006.
- [22] L. Pantel and L. C. Wolf. On the suitability of dead reckoning schemes for games. In *Proc. of the 1st workshop on Network and system support for games*, NetGames '02, pages 79–84. ACM, 2002.
- [23] C. G. Rezende, B. P. S. Rocha, and A. A. F. Loureiro. Publish/subscribe architecture for mobile ad hoc networks. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 1913–1917, New York, NY, USA, 2008. ACM.
- [24] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken. Using a Market Economy to Provision Compute Resources Across Planet-wide Clusters. In *Parallel and Distributed Processing Symposium*, 2009.
- [25] T. Wongsardsakul and K. Kanchanasut. A structured mesh overlay network for p2p applications on mobile ad hoc networks. In *Distributed Computing and Internet Technology*, volume 4882 of *Lecture Notes in Computer Science*, pages 67–72. Springer, 2007.