Peer-to-Peer Architecture for Real-Time Strategy MMOGs with Intelligent Cheater Detection

Marco Picone, Stefano Sebastio, Stefano Cagnoni, Michele Amoretti
Dept. of Information Engineering
University of Parma
Via Usberti 181/a
43124 Parma, Italy
Contact: michele.amoretti@unipr.it

ABSTRACT

Massively multi-player online games (MMOGs) are having increasing success, because they allow players to explore huge virtual worlds and to interact in many different ways, either cooperating or competing. To implement MMOGs that support several million users, a decentralized communication infrastructure is much better than a centralized one. Unfortunately, decentralization introduces many security problems, allowing dishonest players to apply cheating strategies, if appropriate countermeasures are not implemented. In this paper we illustrate our framework for real-time strategy MMOGs, called PATROL, focusing on the AI-based strategies it adopts to detect cheating attempts.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*connectionism and neural nets,knowledge acquisition*

Keywords

P2P games, cheating detection, neural networks

1. INTRODUCTION

Due to the significant advancements in artificial intelligence, computer graphics, and the availability of broadband links to home users, multi-player online games (MMOGs) have quickly become a profitable sector to vendors. Most research on MMOGs focuses on scalability and high speed, but other issues such as the existence of cheating have an equally large practical impact on game success. There are significant technical barriers to achieving all these properties at once, and no existing game does so.

Basically, there are two kinds of MMOGs: role-playing games (RPG) and real-time strategy (RTS) games. This work focuses on RTS games, in which participants position and move units and structures (generally speaking, *resources*) under their control to secure areas of the virtual world and/or destroy the assets of their opponents. In a typical RTS it is possible to create additional combat/exploration units and structures during the course of a game. New and more powerful resources may also be obtained by achieving control of special points on the map and/or by building certain types of structures devoted to this purpose. More specifically, the typical game of the RTS genre features resource gathering, base building, in-game technological development and indirect control of combat/exploration units.

In this paper we introduce the *Peer-to-peer Architecture for sTRategy games with Online inteLligent cheater detection (PATROL)*, which is a framework for the implementation of highly scalable and secure RTS MMOGs. Indeed, PATROL is characterized by a fully distributed structured overlay scheme, which guarantees fair and balanced workload distribution among peers. Moreover, cheating behaviors are detected by means of AI tools, such as neural networks, implemented by each node.

The paper is organized as follows. In section 2 we summarize some recent research work in the context of peer-to-peer RTS MMOGs, presenting cheating detection strategies. In section 3 we describe the PATROL architecture, with details on the P2P overlay scheme and cheating detection strategy we have adopted. In section 4 we illustrate an example RTS MMOG we are implementing, based on the proposed architecture. We focus on the performance of the module for intelligent cheating detection, presenting and discussing many experimental results. Finally, in section 5 we conclude the paper by specifying plans for extending our work further.

2. BACKGROUND

MMOG needs a messaging infrastructure for game actions and player communication. To this purpose, possible paradigms are Client-Server (CS), Peer-to-Peer (P2P), Client-Multi-Server (CMS), or Peer-to-Peer with Central Arbiter (PP-CA) [8].

Each solution has pros and cons, with respect to robustness, efficiency, scalability and security. In particular, facing malicious behaviors when the architecture of the game is decentralized (*e.g.* P2P), to support a large number of players, is particularly challenging. The following are the most common forms of cheating in RTS games:

- *speed cheating*: The player has the ability to move his combat units or to explore the virtual world, looking for resources to be recovered or hidden enemy troops,

DISIO 2010 March 15, Torremolinos, Malaga, Spain.
Copyright 2010 ICST, ISBN 78-963-9799-87-5.

faster than his competitors. This trick is based on the change of the timing of the player's client.

- *map hack*: The map of the virtual world is completely revealed to the player, for which he does not need to lose his time to explore it.

- *look-ahead cheat*: Submission of actions is modified in order to allow the cheater to be the last player to make the decision during simultaneous actions. During iterations players must specify the resources involved in the attack. The modified client allows the player to respond to the attack after discovering the opponent's move, which can be a significant advantage.

In [8], the authors propose a Mirrored-Arbiter (MA) architecture that combines the features of CMS and PP-CA. This architecture takes all the benefits of PP-CA, but also solves the main problems in PP-CA by using interest management techniques and multicast. Clients are divided into groups, each group being handled by an arbiter that maintains a global game region state and takes care of the consistency issue. When the arbiter receives an update from a client which conflicts with its game region state, it ignores the update and sends the correct region state to all clients in the group. The authors implemented a multiplayer game called "TankWar" to validate the design of the proposed MA architecture. In our opinion, such a scheme does not scale very well.

In [1], the authors present a Peer-to-Peer (P2P) MMOG design framework, Mediator, based on a super-peer network with multiple super-peer (Mediator) roles. In this framework, the functionalities of a traditional game server are distributed, capitalising on the potential of P2P networks, and enabling the MMOG to scale better in both communication and computation. Mediator integrates four elements: a reward scheme, distributed resource discovery, load management, and super-peer selection. The reward scheme differentiates a peer's contribution from their reputation, and pursues symmetrical reciprocity as well as discouraging misdemeanours. The authors suggest to adopt the EigenTrust reputation management algorithm [4] and the DCRC anti-free-riding algorithm [2] as possible implementations for the reward scheme. Unfortunately, such schemes are complex and bandwidth-consuming.

Using a *Distributed Hash Table (DHT)* for implementing the P2P approach is probably the best way to support a large number of players. This is the approach pursued in [3], adopting Pastry-based technologies to perform different tasks: finding an object by Pastry itself, accessing it synchronously using the Scribe extension, and handling the replica management by Past, by storing replicas to the current node's leaf set of the logical network. The DHT-based approach is adopted also in our PATROL framework.

3. DESCRIPTION OF THE FRAMEWORK

In order to increase security, the game infrastructure should properly manage the interaction events among nodes. In RTS games, the most frequent events are those for: i) moving resources, ii) receiving updates about the virtual world, and iii) submitting the attacks. In PATROL these events are managed through protocols that are appropriate for maintaining an adequate level of security.

PATROL adopts the Chord P2P overlay scheme [7] to support fair and robust information sharing among available players. Chord is a highly structured P2P architecture where all peers are assigned the same role and amount of work. It is based on the DHT approach for an efficient allocation and recovery of resources. The overlay network in PATROL also supports a distributed algorithm for cheater detection, based on feedbacks among peers and AI tools such as neural networks. This approach allows one to dynamically recognize malicious behaviors, collectively performed by peers without the need of specific and centralized control components. In the following we describe each aspect in details.

3.1 Chord

Chord [7] is probably the most known peer-to-peer protocol based on the Structured Model (SM), which uses DHTs as infrastructures for building large scale applications. Data are divided into *blocks*, each one identified by a unique *key* (a hash of the block's name) and described by a *value* (typically a pointer to the block's owner). Each peer is assigned a random ID in same space of data block keys, and it is responsible for storing key/value pairs for a limited subset of the entire key space.

Given a key (*i.e.* the identifier of a resource or a service), the Chord protocol maps the key onto a node (a host or a process identified by an IP address and a port number). Chord's consistent hash function assigns each node and key an m-bit identifier using a base hash function such as SHA-1. A node's identifier is chosen by hashing the node's IP address, while a key identifier is produced by hashing the key. The identifier length m must be large enough to minimize the probability of two nodes or keys hashing to the same identifier. Identifiers are ordered on an *identifier circle* modulo 2^m. Key k is assigned to the first node whose identifier is equal or follows the identifier of k. This node is called the *successor node* of key k. Figure 1 shows a Chord ring with $m = 4$.

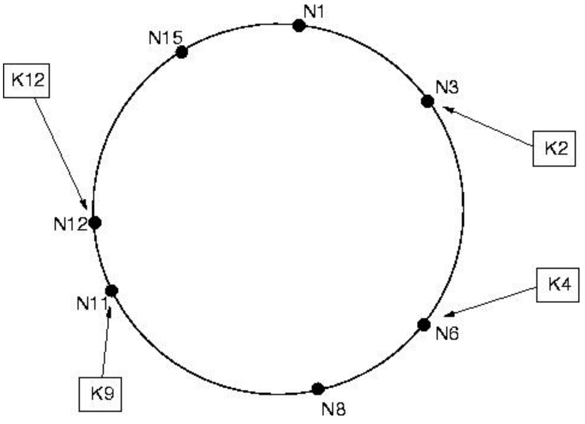

Figure 1: An identifier circle consisting of 7 nodes storing 4 keys. The successor of identifier 2 is the node with identifier 3, so $K2$ is located at $N3$. Similarly for the other 3 keys.

Chord's basic lookup algorithm, whose description we omit for space reasons, uses a number of messages which is linear in the number of nodes. To accelerate lookups, each node n could maintain a routing table with up to m entries, called the *finger table*. The i^{th} entry in the table at node n contains the identity of the first node s that follows n by at least 2^{i-1} on the identifier circle; i.e. $s = successor(n + 2^{i-1})$, where $1 \leq i \leq m$ and all the arithmetic is module 2^m. We call node s the i^{th} *finger* of node n, and denote it by $n.finger[i]$. A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node. Figure 2 illustrates the scalable lookup algorithm based on finger tables. In general, if node n searches for a key whose ID falls between n and its successor, node n finds the key in its successor; otherwise, n searches its finger table for the node n' whose ID most immediately precedes the one of the desired key, and then the basic algorithm is executed starting from n'. It is demonstrated that, with high probability, the number of nodes that must be contacted to find a successor in an N-node network is $O(\log N)$ [7].

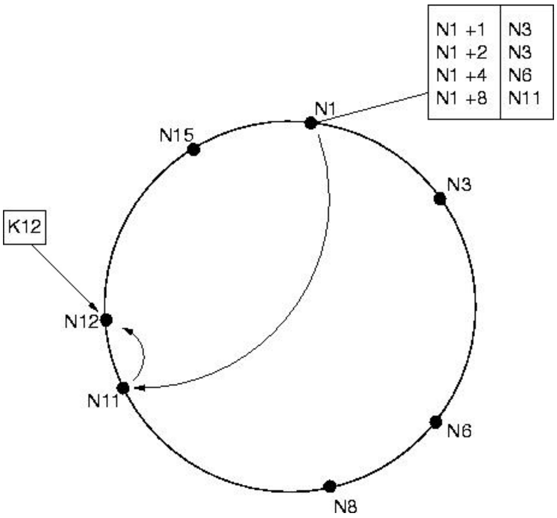

Figure 2: The finger table entries for node $N1$ and the path taken by a query from $N1$, searching for $K12$ using the scalable lookup algorithm.

3.2 Resource Distribution

Usually the distribution of resources (troops, ships, tanks, etc.) in P2P games is achieved by splitting the virtual world into sections and assigning them to available peers. In the case of RTS games, this solution may offer more opportunities for cheating. Indeed, it is possible for a peer to disclose or modify information about a particular area, whose maintenance is under its responsibility, even though none of its resources is in that area.

The PATROL architecture divides resources equally among peers, using the DHT to share information about whom is responsible for what (each peer is responsible for a subset of the key space). In a game, each existing resource has a position in the virtual world. Such a position is hashed, and the resulting key is assigned to the the peer whose key subset includes the resource key (figure 3). It is very unlikely that two resources that are placed in close locations in the virtual world have keys that are close in the key space (and vice versa). Moreover, Chord foresees data replication, in order to improve robustness against unexpected node departures.

3.3 Game Events

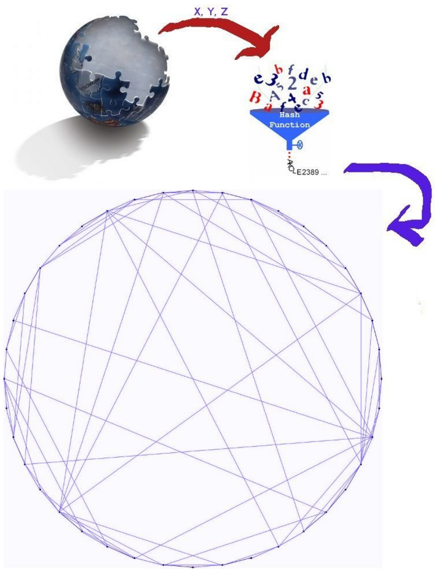

Figure 3: Resource distribution in PATROL (the Chord ring refers to a real network with 40 nodes).

The system uses a bootstrap server to support peers in joining the network (which includes authentication, as well as Chord initialization) and configuring themselves for entering a game. In this way the bootstrap server has control over the accounts of the players and consequently provides a basic level of security.

Periodically, each peer needs to update its view on the virtual world. To do so, it sends specific requests to peers that are responsible for the positions that are in view. Before responding to such a kind of request, peer j, that is responsible for position (x, y, z), checks his cache for updated information, and sends a request to verify the credentials for peer k. If everything is ok, it finally sends the response message (figure 4).

Before performing any action that involves a change of game state, players must submit a request to the responsible of the resource that is affected by the action. For example, suppose player (k) can select a resource to be displaced in the virtual world; to perform the action "displacing resource to position (x, y, z)" the peer must submit the request to the responsible of the key resulting from the hash of that position, *i.e.* $h(x, y, z)$ (peer j in figure 5).

The peer that must become the new responsible for the displaced resource of peer k searches for the manager of the resource's current position (declared by peer k). Such peer is discovered by means of the hash of the current position. Thus the old manager checks in its cache if it has the information on peer k and if it corresponds to what was declared to j.

If the check is successful, peer j can decide, according to game rules and considering the elapsed time between the changes of game state following the transition between the two positions, if it can accept the move and execute it, becoming the new responsible for the resource. If the position declared by k is not true, the request for resource displacement submitted by peer k is ignored and the state of the game remains the same.

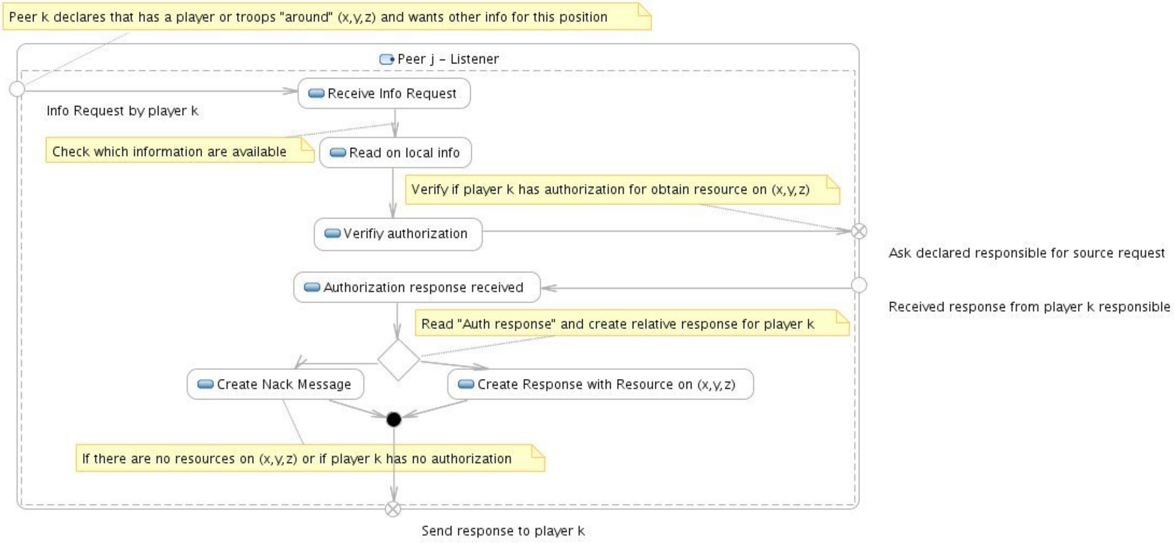
Peer k declares that has a player or troops "around" (x,y,z) and wants other info for this position
Peer j – Listener
Receive Info Request
Info Request by player k
Check which information are available
Read on local info
Verify if player k has authorization for obtain resource on (x,y,z)
Verifiy authorization
Ask declared responsible for source request
Authorization response received
Received response from player k responsible
Read "Auth response" and create relative response for player k
Create Nack Message
Create Response with Resource on (x,y,z)
If there are no resources on (x,y,z) or if player k has no authorization
Send response to player k

Figure 4: Request for updated information about the state of the virtual world, in correspondence with position (x, y, z).

Information about the virtual world may not be granted indiscriminately to any peer. Each peer has its own resources, which are placed in different positions of the virtual world, and has the right to receive information that refers to areas that are in the sight of such resources, according to the rules of the game.

3.4 Attack Submission

While troops can be moved asynchronously by each player, attacks must be either asynchronous or synchronous (*i.e.* with a turn-based approach). In case of synchronous attacks, knowing the decisions of other players before submitting his own move may be a considerable advantage for the player. But, of course, this would be unfair.

To avoid cheating, PATROL uses request hashing, a mechanism that is widely adopted in other P2P architectures and derives from distributed security systems. Players who submit their decisions have to send a hash of the message describing the attack concatenated with a *nonce*. The nonce is used to prevent a cheater from storing in a table all matches between hash values and attack decisions, revealing the decisions of honest players. The nonce is a use-once random value, chosen by the first player that submits a decision. The last player that submits its decision may send it manifestly.

At this point, all players that have previously sent the hash must send their nonce and attack decision manifestly. Thus, other players can re-calculate the hash and verify that it corresponds to what was previously declared.

The properties of hash functions guarantee that it is almost impossible for two different attacks to have the same hash, and therefore for a player to submit a different attack, with respect to the encrypted one.

3.5 Intelligent Cheater Detection

PATROL provides a good level of security for the overall state of the game. Anyway, the DHT does not prevent the game from offering cheaters (provided with hacked client) the possibility to alter the information under their responsibility. In a RTS game, a modified client that saves a history of recent attacks and their outcomes, may be asked to estimate the current level of resources available to other players, for making profit.

Using artificial intelligence techniques, a PATROL-based peer can detect anomalies in the behavior of other peers, compared to typical behavioral profiles, by means of temporal analysis of interaction events. Moreover, using the power of direct communication typical of P2P approaches, a peer may ask other peers their ”opinion” about a given peer in order to improve the evaluation process.

Peer x calculates the probability $P\{y|x\}$ that peer y is cheating. Then x sends a request to peers that have interacted with y, in order to match their probabilities and understand whether y is considered to be a cheater: $P\{y|i\}\ \forall i \neq x, y$. If the global probability exceeds a certain threshold, there is the option to contact other peers and the bootstrap server to promote a collective motion against the cheating player in order to ban him from future games. If all peers agree with the ”Ban Proposal”, peer y is gracefully disconnected from the Chord ring.

The artificial intelligence module analyzes all action events coming from an opponent. The opinions of the other players are requested only at the end of the local evaluation process, if the peer estimates a high probability of cheating. Of course, the peer must be careful since other peers may provide false reports related to their interactions with a given peer.

There are different strategies for a peer to learn from a sequence of events: sequence recognition, sequence playback, and temporal association. Among others, we focused on the following tools:

- *Multi Layer Perceptrons (MLPs)*
- *Time Delay Neural Networks (TDNNs)*
- *Back-Propagation Through Time (BPTT) learning algorithms*

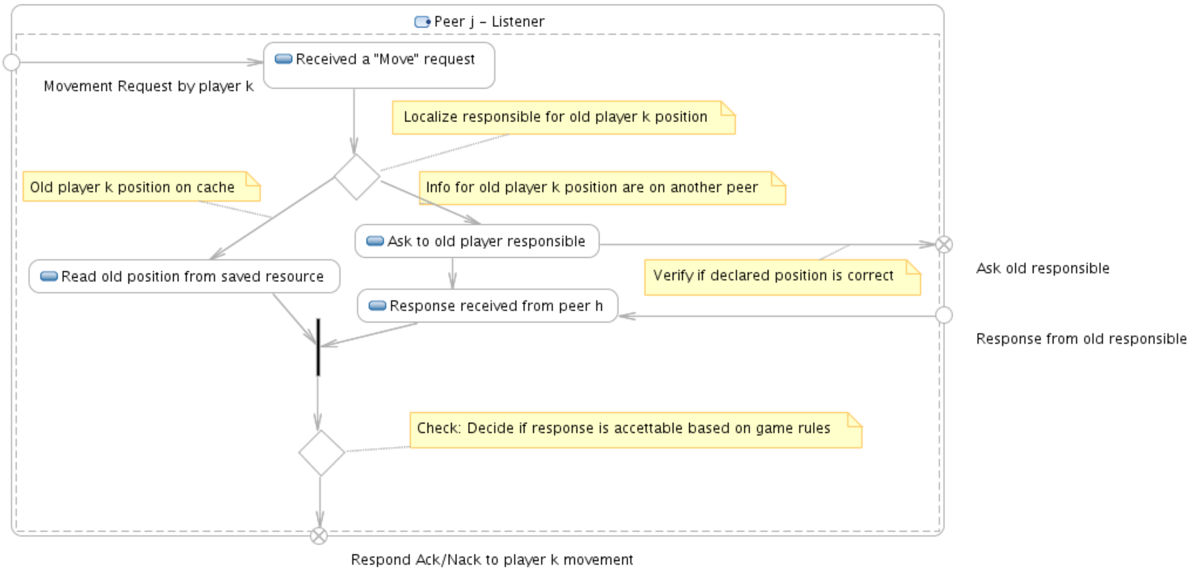
Peer j – Listener
Received a "Move" request
Movement Request by player k
Localize responsible for old player k position
Old player k position on cache
Info for old player k position are on another peer
Ask to old player responsible
Read old position from saved resource
Verify if declared position is correct
Ask old responsible
Response received from peer h
Response from old responsible
Check: Decide if response is accettable based on game rules
Respond Ack/Nack to player k movement

Figure 5: Request for moving a resource (*e.g.* a battalion) in the virtual world.

that seem to fit very well our needs.

MLPs are the traditional multilayer neural networks trained by the backpropagation algorithm. Weights are updated using the online algorithm, *i.e.* re-arranging the weights after each epoch, *i.e.* a learning iteration during which all examples are processed by the network (figure 6).

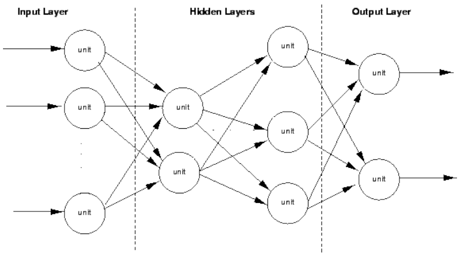

Figure 6: Example of Multi Layer Perceptron (MLP).

TDNNs are also called *Tapped Delay Lines* for the presence of a line on which they apply delays on signals. In these networks, memory is limited to the length of the delay line (figure 7).

Recurrent Neural Networks (RNNs) can be seen as a modification of the traditional feed-forward networks to allow temporal classification. RNNs add a context layer to the traditional structure of neural networks, in order to keep the information between different observations. In the example of figure 8, new inputs are provided to the network for each time point, and previous content is passed to the hidden Context Layer. The content of the context layer is then passed back to the hidden layer at the next time point. In this way the network does not consider only the input that receives but also the memory content. The BPTT algorithm converts the RNNs from a feedback system to a purely feedforward system by folding the network over time. Thus, if the network is to process a signal that is n time steps long, then n copies of the network are created and the feedback connections are modified so that they are feedforward (shared) connections from one network to the subsequent network.

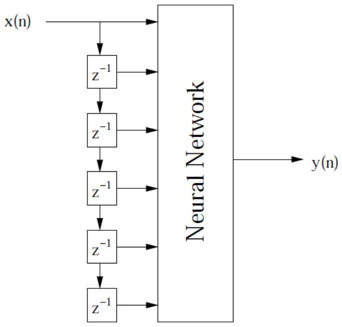

Figure 7: Example of Time Delay Neural Network (TDNN).

Future work will regard the use of keys and digital certificates to reduce perpetrations of coalitions of unfair players. At bootstrap the server provides, for every player subscribed to the game service, a certificate and a key that identifies him/her. The private key can be used to sign every message, including requests for ban of a player that is considered to be malicious. In this way, the server is able to detect suspicious ban requests (for example when they come always from the same player or group of players).

4. EXPERIMENTAL EVALUTION WITH A TEST GAME

We tested PATROL by implementing a simple RTS game, featuring a space scenario and based on exploration and attack of other planets owned by available players (figure 9).

The initial screen allows the user to select the bootstrap server for the game. The second screen shows all information

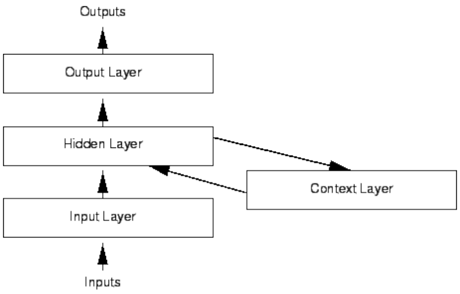

Figure 8: The working principle of Recurrent Neural Networks (RNNs).

about the player, like position of his planet and available resources (with their characteristics) and money. There is also the option for buying new resources. The third screen shows the (known) virtual world (a 2D space) and allows players to interact with it.

The game requires that each player logs in to the bootstrap server for authentication and gets a random position in the virtual world. The position assigned to the player describes the coordinates of the player's planet.

At startup, each player has only one type of resource: the mine. The mine is a resource that can evolve and provide economic resources. During the game, money deriving from the mine is available for buying advanced resources, starting from a minimum of 10 units. The player can buy resources for defense or attack, whose strength depends on the money spent to purchase them. Defense resources are static and can be used to face attacks from other players. Spacecrafts are dynamic resources that can be used to explore the virtual world and also to attack other player's planets.

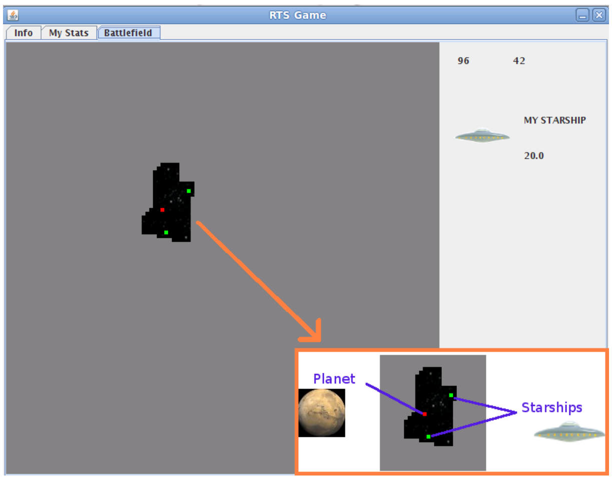

Figure 9: The RTS game implemented to test PATROL.

At the beginning of the game, the player knows only the space around his planet. Clicking on the planet, the player is allowed to select one of his spacecrafts for landing on the planet, and to select its destination point. During the spacecraft travel along the path, the player gets to know new parts of the wirtual world, and possibly resources of other players, placed along the trajectory.

When a spacecraft encounters another player's resource, it is possible to attack it. The attacked player can make a defense move only if he has defence resources. Each stage of a battle with other players is recorded by the client. The application generates a log that contains timestamp, involved resources and attacker for each attack faced. This log is used by the intelligent cheater detection system of PATROL.

4.1 Training Neural Networks

In order to provide a number of examples large enough for training the neural networks, we have generated some synthetic profiles. These profiles reproduce the behavior of a player that respects the rules, one that cheats with 20% probability and another one with 40%. In this context, a percentage x of cheating means that some attacks are executed using a set of resources that are more powerful ($+x\%$) than would be allowed according to the rules of the game. Different kinds of logs are generated to determine the best way to obtain a good behavior. We have analyzed the performance of networks with logs that consider a temporal window of three, six or nine moves. For networks that accept an input log with six or nine moves, the training set includes 100 examples, while for three moves it includes 300 examples.

It is necessary to clarify that the only type of cheating we have considered is the one in which a player obtains a lot of resources (*e.g.* troops, weapons), too quickly with respect to what could be expected according to game rules. Other forms of cheating and their impact will be analyzed in a future work, using a more complex test-game.

Figures 10 and 11 show four cheating and two honest profiles, respectively. Time is reported on the x axis, while the force of the attack (positive values) or defense (negative attacks) is shown on the y axis. During the game, a player can buy or build resources for defense and for attack. In a regular match, it should not be possible for a player to have a lot of attack resources and a lot of defense resources. Thus, a honest profile is more regular than a cheating one, which is characterized by peaks. For clarity, the profiles of players with 40% cheating rate are illustrated on a larger scale, since attacks and defences are very close to one to another.

In a first phase we performed learning with different neural network topologies, changing the learning rate with a step of 0.1. For each neural network, training has been performed with three different seeds to initialize the weights of the synapses.

After the training phase, networks have been evaluated on a validation set including samples that did not belong to the training set: the validation set. This set was used to evaluate the generalization ability of the network.

For the best network configurations we also varied the number of epochs to limit overfitting through an appropriate choice of the number of epochs.

We have selected the networks with best performance in terms of *Root Mean Squared Error (RMSE)*, and we have analyzed the behavior on another set of 100 examples including neither the training set nor the validation set. Such a final step allowed us to verify the generalization ability of the networks and rating the available networks in terms of the degrees of freedom given by the number of synapses.

Networks that analyzed the logs for six or nine seconds gave the best performance on the validation set (also with RMSE less than 0.01). It was subsequently decided not to

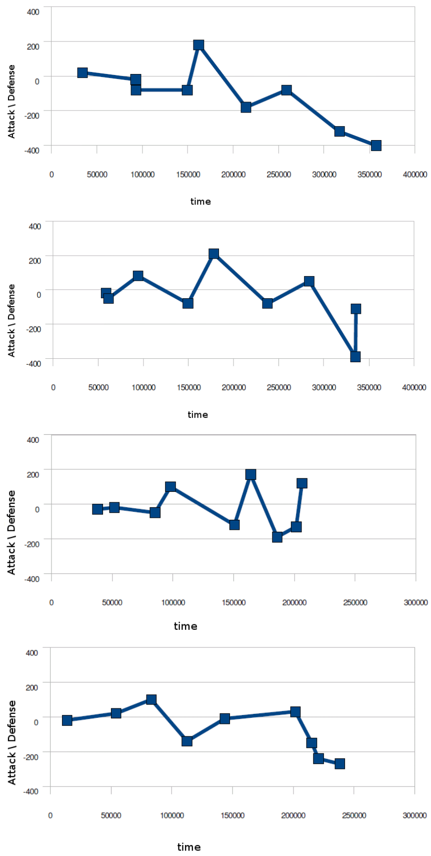

Figure 10: Examples of synthetic cheaters' profiles used to train the neural networks of the intelligent cheater detection system of PATROL.

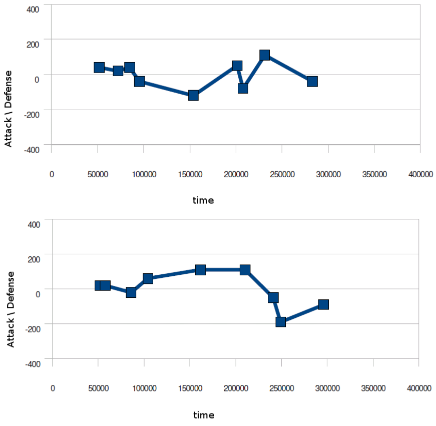

Figure 11: Examples of synthetic honest players' profiles used to train the neural networks of the intelligent cheater detection system of PATROL.

use this log window size, because it is too large for deciding and possibly ban the cheating player from the game in acceptable time.

Results on the test set are summarized in figure 12. On the horizontal axis we report the different types and topologies of the neural networks we tested. For each type of neural network we have tested several network topologies. The numbers in the description of the network indicate the neurons that are present on different layers. For example "MLP 6-6-3-1 (57)" indicates a network with 6 neurons in the input layer, 6 and 3 respectively in the first and second hidden layers and 1 neuron in the output layer. In parentheses we report the degrees of freedom of the considered network. The vertical axis shows the RMSE values.

The neural networks that are most suited for analysis over time (*i.e.* TDNNs and and those based on BPTT) have the best performance. For MLP networks, the best performance has been obtained with fewer degrees of freedom, *i.e.* less than one hundred, which avoids overfitting. TDNNs and BPTT-based networks have given stable results for variable degrees of freedom, more than MLPs. In particular, TDNNs have always yielded $RMSE < 0.45$ on the test set.

Figure 13 shows the classification performance of the $(6, 6, 3, 1)$ MLP neural network versus the actual player's behavior (a number larger than 1 indicates cheating - *e.g.* 1.2 means that the player has 20% cheating rate). We indicated the thresholds which separate the zones corresponding to the false-negative and false-positives.

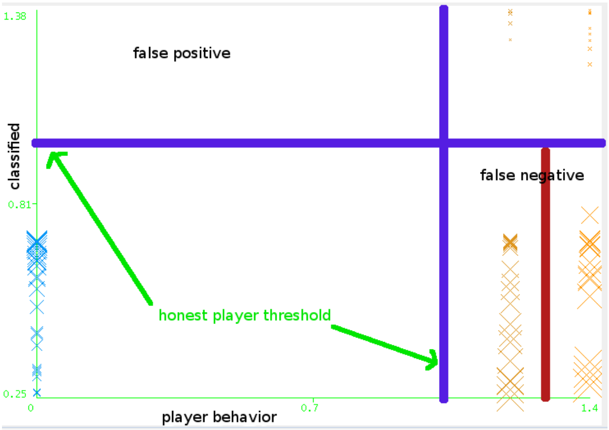

Figure 13: The classification performed by the $(6, 6, 3, 1)$ MLP neural network versus the actual player's behavior.

In general, trying to find the exact cheating ratio of a player is very difficult. It is better to set a threshold (*e.g.* 20%) and use the neural network to discriminate honest players from cheating players. With this approach, results are highly satisfactory and can be further strenghtened by matching them with those provided by other players.

5. CONCLUSIONS AND FUTURE WORK

In this work we illustrated a framework for creating peer-to-peer online RTS games, characterized by a high degree of robustness, efficiency and effectiveness against cheating behaviors.

The performance of the neural networks we adopted has been satisfactory. It is possible to envisage the use of other

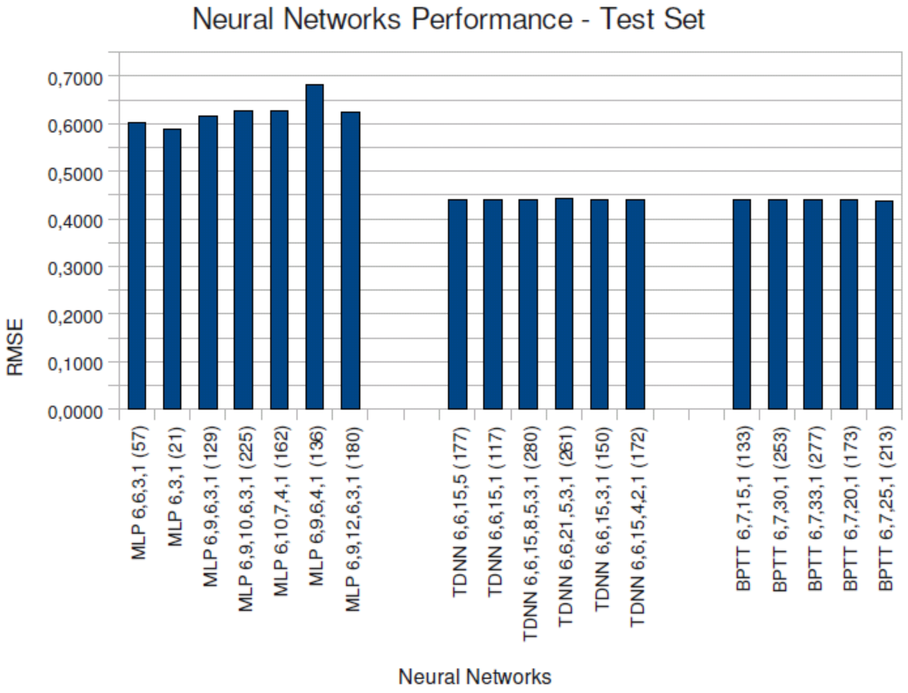

Figure 12: Resulting RMSE for the neural networks, experimented on the test set.

means of temporal analysis based on neural networks (such as *Real Time Recurrent Learning* and *Context Units* like Elman and Jordan nets) and on other techniques. It would also be possible to investigate the effects of adding a component capable of evaluating the *trust of peers* based on the past history of players.

6. REFERENCES

[1] L. Fan, H. Taylor and P. Trinder, *Mediator: A Design Framework for P2P MMOGs*, Proc. of NetGames'07, pp.43-48, Melbourne, Australia, September 2007.

[2] M. Gupta, P. Judge, M. Ammar, *A reputation system for peer-to-peer networks*. Proc. of the 13th ACM NOSSDAV workshop, pp.144-152, Monterey, CA, USA, 2003.

[3] T. Hampel, T. Boop and R. Hinn, *A Peer-to-Peer Architecture for Massive Multiplayer Online Games*, Proc. of the 5th ACM Workshop on Network & System Support for Games (NetGames '06), Singapore, October 2006.

[4] S. D. Kamvar, M. T. Schlosser and H. Garcia-Molina, *The EigenTrust Algorithm for Reputation Management in P2P Networks*, Proc. of the 12th Int'l Conf. World Wide Web, pp.640-651, Budapest, Hungary, May 2003.

[5] L. S. Liu, R. Zimmermann, B. Xiao and J. Christen. *PartyPeer: a P2P Massively Multiplayer Online Game*, pp.507-508, Proc. of ACM MM'06, Santa Barbara, California, USA, October 2006.

[6] A. El Rhalibi, M. Merabti and Y. Shen. *AoIM in Peer-to-Peer Multi-player Online Games*, Proc. of ACM ACE '06, Hollywood, California, USA, June 2006.

[7] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek and H. Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, Proc. of ACM SIGCOMM '01 Conference, pp.149-160, San Diego, USA, March 2001.

[8] L. Yang, P. Sutinrerk, *Mirrored Arbiter Architecture - A Network Architecture for Large Scale Multiplayer Games*, Summer Computer Simulation Conference (SCSC 2007), pp.709-716, San Diego, California, USA, July 2007.