AC/DC: an Algorithm for Cheating Detection by Cheating*

Stefano Ferretti
Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7, 40127 Bologna, Italy
sferrett@cs.unibo.it

Marco Roccetti
Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7, 40127 Bologna, Italy
roccetti@cs.unibo.it

ABSTRACT

Time cheats represent some of the most crucial issues in online gaming. Since they act on timing properties of generated game events, these malicious schemes are particularly difficult to thwart when distributed games are deployed over peer-to-peer architectures. Indeed, the absence of a global clock shared among peers enables cheaters to see into the future by waiting for events generated by other peers before generating its own ones (*lookahead cheat*). This may give an unfair advantage to the cheater. We consider a version of lookahead cheat generalized in the context of real-time (i.e., not round-based) games. To face this time cheat, we present **AC/DC**, an **A**lgorithm for **C**heating **D**etection by **C**heating. This algorithm enables to detect cheaters based on monitoring of network latencies. The basic idea is that of conducting against each suspected peer a sort of cheating counterattack, by delaying events before notifying them to the (hypothetic) cheater. This permits to detect whether that peer waits for these events before generating its own ones. Our claim is that an approach based on the monitoring of communication patterns among peers allows cheat detection without affecting the performances of the game.

Categories and Subject Descriptors

K.8.0 [**Computing Milieux**]: PERSONAL COMPUTING—*Games*

General Terms

Algorithms, Synchronization, Security

Keywords

Cheating, Online Games, Peer-to-Peer

*This work is financially supported by the Italian M.I.U.R. and R.E.R. under the Interlink, MOMA, DAMASCO and SWIMM initiatives.

NOSSDAV '06 Newport, Rhode Island USA

1. INTRODUCTION

The computer game industry is growing rapidly. Today, these digital forms of entertainment require complex software solutions able to ensure compelling experiences to users. Among the numerous technological novelties introduced in the gaming market, network playing is probably the most revolutionary one. From a user perspective, the Web can be seen as a whole game platform that enables one to discover and play with millions of unknown other players. However, several issues arise when geographically dispersed players interact together through the Internet. Indeed, the highly dynamic configuration of distributed game environments and the anonymity among players bring new security challenges for game developers that need to be addressed. Among these, cheating is certainly one of the most problematic.

Cheating is defined as any action that a player accomplishes to obtain an unfair advantage over other players. Cheaters try to alter game rules via deviant program behaviors [9]. There are several types of cheats in online games. Taxonomies have been proposed in [13, 18, 19, 20], where cheating schemes are organized depending whether cheaters try to violate game rules, authentication procedures, private information, timing constraints, etc. Our focus here is on those cheats that allow players to alter timing information of game events (*time cheats*). Time cheating is rampant in distributed games since players are placed at different geographical locations and typically connect to the game thanks to different system/networking technologies. This unfairness on players' technological capabilities is the first alibi for cheaters. Indeed, by simulating larger network response times, time cheats can be devised which allow cheaters to "see into the future", thus providing them with an unfair advantage with respect to honest players [6]. Moreover, games are particularly vulnerable to time cheats when they are hosted on peer-to-peer platforms which are, instead, one of the most promising game architectures for the support of these kinds of applications [3, 4, 7].

In this work, we face with a particular version of the well-known lookahead cheat, generalized in the context of real-time (i.e., not round-based) games. According to such malicious scheme, the cheater waits to see game events generated by other participants before generating its own game event. Then, the cheater mimics that such event has been generated prior to other ones. Schemes that prevent this specific time cheat have been presented in [1, 6, 13, 16]. The main characteristic common to these approaches is that the event delivery is supposed to be organized in rounds. Nodes are forced to exchange additional information (such as hash

values) before revealing their new generated moves. Unfortunately, these schemes result really expensive in terms of number of additional messages and introduced latencies. In fast paced real-time games, this represents an important limitation. Moreover, forcing the game to proceed in rounds limits the pace of the game evolution, thus jeopardizing responsiveness. Indeed, having a proof cheat protocol that affects the interactivity degree may be useless for a large class of fast paced games.

With this in view, we claim that, instead of avoiding cheats, more profitable solutions can be devised which are based on detection of cheaters. In essence, a peer may decide to control other suspected peers to assert with a certain confidence whether they are cheating or not. The algorithm presented in this work follows this approach. In particular, the proposed scheme works at the game communication level and monitors game event latencies during the game delivery. The scheme is based on the following fundamental assumption: the underlying network over which the game is deployed offers a best-effort service with unpredictable delay latencies and jitters but, on the long run, an average trend of network latencies may be observed. This assumption is in accordance with a plethora of works that model network traffic in different contexts of networks and applications such as, in the networked gaming literature, [2, 11, 15].

Our scheme is named **AC/DC**, i.e., an **A**lgorithm for **C**heating **D**etection by **C**heating. In substance, cheaters detection is accomplished by deliberately increasing transmission latencies of game events towards a given peer p, once p is suspected for cheating. This enables to notice whether p waits for these delayed game events before generating its own (cheated) game events. Put it in other words, a specific node in the system (i.e., the *leader* peer that controls others for cheating) performs a sort of cheat towards p, so as to detect if p is a cheater. Detection of cheaters is possible since increasing network latencies will be measured at the leader for subsequent cheated events from p, based on their associated timestamps.

According to our model, the game evolution is not forced to proceed in rounds. Moreover, even if game events are delivered to suspected cheaters with increasing latencies, all other nodes in the system are not directly affected by our cheating counterattack. Indeed, it could be the case that a cheating counterattack against a suspected cheater may lead to the introduction of additional delays whose effect may have some impacts on the general game dynamics. To counterbalance this problem, classic hiding approaches may be exploited, such as optimistic synchronization and dead-reckoning, to reduce the impact of additional delays. In conclusion, based on our approach, a real-time evolution can be guaranteed.

The remainder of this work is organized as follows. Section 2 describes our system model. Section 3 presents in detail AC/DC. Section 4 provides discussions on the devised scheme and on how peers may assume the role of leaders. Finally, Section 5 concludes the work.

2. SYSTEM MODEL

We use a framework for modelling games and related timing constraints based on the seminal work by Fujimoto [12]. As this model is quite general, our approach can be applied to a wide range of different gaming contexts.

We denote with Π the set of peers; p_i identifies a single peer, i.e., $p_i \in \Pi$. For the sake of a simpler notation, with Π_i we indicate all peers but p_i i.e., $\Pi_i = \Pi - \{p_i\}$. Similarly, notations such as $\Pi_{i,j}$ indicate all peers but p_i, p_j i.e., $\Pi_{i,j} = \Pi - \{p_i, p_j\}$. (Notations and symbols employed in the remainder of the work are summarized in Table 1.)

Table 1: Notations and Symbols

Symbol	Definition
Π	Set of peers
p_i	Single peer
Π_i	All peers but p_i
e_k^i	k-th event generated by p_i
WT_i	Wallclock time at p_i
ST_i	Simulation time at p_i
T_i^W	Mapping function from ST_i to WT_i
T_i^S	Mapping function from WT_i to ST_i
$drift_{ij}$	Drift between physical clocks at p_i and p_j
gap_{ij}	Time between the instants at which p_i and p_j start the game
UB	Upper bound on latencies in the system
$\delta_{ij}(e)$	Time to transmit e from p_i to p_j
δ_{ij}	Average latency from p_i to p_j
$WT_j^{rec}(e)$	Time of reception of e at p_j
λ_l	Additional delay employed during the cheating counterattack
γ	Value that accounts for a significant difference between two latency estimations
Λ	Upper bound on the increment of overall delay latencies during the cheating counterattack

Games are often viewed as specific simulations where characters interact each with other and evolve with the passing of time. Time is thus a primary characteristic to be modelled. We consider here a general model where the game advances in real-time using the notions of wallclock time and simulation time. Specifically, the so called *wallclock time* is the time that identifies when the game takes place. With WT_i, we refer to the wallclock time measured at p_i. *Simulation time*, instead, is the abstraction that is used to model when events have been produced within the virtual game timeline. With ST_i, we refer to the simulation time measured at p_i. Thus, once a game event e is generated at p_i, p_i associates to e a specific simulation time $ST_i(e)$, obtained from its wallclock time $WT_i(e)$ representing the instant of generation of e, measured at p_i.

We define a mapping function T_i^W that transforms a simulation time $s \in ST_i$ into the corresponding wallclock time $t \in WT_i$ at p_i i.e., $T_i^W(s) = t$. With T_i^S, instead, we represent T_i^W's inverse function. A typical equation to map wallclock times into simulation times is as follows:

$$T_i^S(t_{i,actual}) = T_i^S(t_{i,start}) + k(t_{i,actual} - t_{i,start}), \quad (1)$$

where k is a constant that determines the pace of game advancements in the simulated world, $t_{i,actual}$ represents the actual wallclock time at p_i, $t_{i,start}$ represents the wallclock time associated to the beginning of the game at p_i. The mapping $T_i^S(t_{i,start})$ returns a simulation time value, agreed and shared among all nodes, representing the time at which the beginning of the game plot takes place i.e., $T_i^S(t_{i,start}) = s_{start} \in ST$, $\forall p_i \in \Pi$. Using (1), the simulation time of a given game event e_k can be characterized as follows

$$ST_i(e_k) = s_{start} + k(WT_i(e_k) - t_{i,start}). \quad (2)$$

We assume that it is never the case that peers start the game at the same instant i.e., we have with high probability

that $t_{i,start} \neq t_{j,start}$, $\forall p_i, p_j \in \Pi$, $i \neq j$. We then assume that at the beginning of the game, each player notifies other peers with its own $t_{i,start}$. Based on this model, each peer p_i associates its starting wallclock time $t_{i,start}$ to the agreed constant starting simulation time i.e., $T_i^S(t_{i,start}) = s_{start}$.

Another assumption on which our approach is based is that at the beginning of the game a clock synchronization protocol is exploited, such as one inspired to those presented in literature, e.g., [8, 14, 17]. This allows us to obtain an initial estimation of the average network latencies among peers $p_i, p_j \in \Pi$ and of the drift among physical clocks at p_i and p_j (i.e., $drift_{ij}$). By convention, we suppose that $drift_{ij} > 0$, if p_j reaches a given wallclock time t^* *before* p_i (simply put, at a given time instant p_j has a wallclock time higher than p_i, see Figure 1).

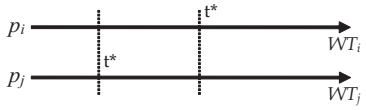

Figure 1: Drift among Physical Clocks of Peers

During the game evolution, each peer p_i maintains also a value gap_{ij} measuring the interval of time (real time) between the instants at which p_i and p_j, respectively, start the game (see Figure 2). A simple equation to measure gap_{ij}, based on the starting point of the beginning time instant (including $drift_{ij}$) is as follows

$$gap_{ij} = drift_{ij} + t_{i,start} - t_{j,start}. \quad (3)$$

In essence, gap_{ij} takes into account that a drift among clocks of p_i and p_j exists and that they started the game at different times. Clearly, $gap_{ij} = -gap_{ji}$ as well as $drift_{ij} = -drift_{ji}$. It is worth mentioning that methods may be devised to minimize the value of gap_{ij}. As an example, peers may simply agree to start the game at a certain point in time. Alternatively, a peer p_l may be set to broadcast a start message to other ones that begin the game as soon as they receive that message (while p_l sleeps for a properly set delay before starting the game).

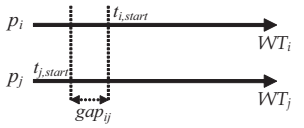

Figure 2: Gap among Peers

Sometimes, for the sake of a clearer notation we denote game events with subscripts (e.g., e_k) to characterize game events produced by the same player in different time instants, e.g., e_j, e_k, $j \neq k$. Moreover, prime notations (e.g., e^l) denote that the event has been generated by a specific peer (in this case p_l).

Game events are notified within messages. Without loss of generality, we assume that employed message transmission procedures for interprocess communication are reliable i.e., transmitted messages can experience different latencies and delay jitters but cannot be lost. We assume the existence of an upper bound UB on the latencies among peers in the system. UB is known by peers. Moreover, we assume to be in a fully connected peer-to-peer network. With $\delta_{ij}(e)$ we denote the time needed to transmit a game event e from p_i to p_j. With δ_{ij}, instead, we denote the average latency needed for the transmission of a non specified game event from p_i to p_j. We realistically assume that typically $\delta_{ij}(e) \simeq \delta_{ij}$. Further, $WT_j^{rec}(e)$ identifies the (wallclock) time of the reception of a given game event e at p_j, i.e., p_j receives e at $t = WT_j^{rec}(e)$.

As already mentioned, our approach requires the existence of a *leader* peer p_l which is in charge of controlling if some other peer nodes cheat. The leader must be a trusted player (p_l must not be a cheater). We discuss in Section 4.1 viable choices of a leader.

Finally, we assume that the generation of game events by a *honest peer* is independent of those generated by other peers. In other words, even if certain game events, generated by some peer, may influence the semantics of subsequent game events generated at other peers, in general, the pace of event generation at a given player is mainly influenced by autonomous player decisions. This assumption is supported by the typical use of techniques such as dead reckoning and/or optimistic synchronization, exploited to hide latencies on notifications and local losses of availability on updated information, thus providing players with the possibility of independently making the game to evolve [5, 10].

3. CHEATING DETECTION BY CHEATING

3.1 The Cheat

A cheater p_i typically waits for moves generated by other participants before generating an event. As soon as p_i receives these game events, he decides the best action to carry out, and notifies others with the new game event e, pretending that e has been generated concurrently with (or before) other ones. This kind of cheat is commonly referred to as *lookahead time* cheat [1, 6, 16].

To formalize this cheat in a real-time model, we denote with *W(e)* the set of those game events which p_i will be waiting before generating e. A first issue is concerned with determining which game events belong to *W(e)*. This clearly depends on the specific game. In round-based games, for example, *W(e)* corresponds to the set of all the game events generated by other players at that round. Different alternatives may be in use when the game proceeds in real-time. For example, *W(e)* may consist of the set of game events generated by other players within a given interval of simulated time Δs. Alternatively, other possible settings for *W(e)* exist and it is reasonable to assume that p_i knows which game events *W(e)* is comprised of. In the following, we will consider Δs as the simulation time value exploited to determine *W(e)*.

Upon generation of e, p_i may create a *cheated simulation time* obtained through the use of the following formula:

$$ST_i^c(e) = min\{ST_j(e_k) \mid e_k \in W(e), p_j \in \Pi_i\} - \omega, \quad (\omega \in ST, \omega > 0). \quad (4)$$

In essence, $ST_i^c(e)$ is chosen so that $ST_i^c(e) < ST_j(e_k)$, $\forall e_k \in W(e)$, i.e., p_i pretends that e has been generated before each other game event in *W(e)*. Finally, p_i calculates a *cheated wallclock time* $WT_i^c(e)$ in accord with $ST_i^c(e)$, obviously different from the real wallclock time of generation of e, $WT_i(e)$.

3.2 AC/DC: A Cheating Detection Scheme

To face this cheat, we present an Algorithm for Cheating Detection by Cheating (AC/DC), which is based on the idea

of deliberately increasing the transmission latency of events generated at p_l for p_i. The goal here is that of detecting whether p_i waits for game events generated by p_l, before generating its own messages. Thus, as soon as p_l decides to control p_i for possible cheating, it starts computing a new measure of the average latency for the transmission of game events from p_i to p_l. This value is obtained by averaging over time several measurements of $\delta_{il}(e_k^i)$,

$$\delta_{il}(e_k^i) = WT_l^{rec}(e_k^i) + drift_{li} - WT_i^c(e_k^i). \quad (5)$$

In (5), $WT_l^{rec}(e_k^i)$ represents the time of reception of e_k^i at p_l, $WT_i^c(e_k^i)$ is the (possibly cheated) wallclock time at which (p_i claims that) e_k^i has been generated, and $drift_{li}$ is the drift between physical clocks of the two peers. It is worth noticing that the use of an average computation of experienced latencies is probably a naive proposal. Alternative and more sophisticated approaches could be employed that exploit, for example, a low-pass filter to smooth the aleatory behavior of latencies caused by delay jitter.

Equipped with δ_{il}, p_l may begin a sort of cheating counterattack against p_i. In essence, for each game event e^l generated at p_l, p_l delays e^l for an additional amount of time (say λ_l) before delivering it to p_i. In other words, the time elapsed since the generation of e^l at p_l to its reception at p_i results equal to $\delta_{li}(e^l) + \lambda_l$.

Upon delivery of e^l to p_i, new latency values $\delta_{il}(e_k^i)$ are collected for a given amount of time whose average δ_{il}^* is contrasted with δ_{il}. Based on this comparison, the leader may take a decision. Let γ be a time value that accounts for a significant difference between δ_{il}^* and δ_{il}, i.e., the hypothesis that the two measured values are equal must be rejected. If $\delta_{il}^* > \delta_{il} + \gamma$ then p_l suspects p_i for cheating. Otherwise, the value of λ_l is progressively increased and the cheating counterattack mentioned above is iteratively repeated until a maximal value for $\delta_{li} + \lambda_l$ equal to Λ is exceeded, where

$$\Lambda = UB + T_l^W(\Delta s) + max\{gap_{jl}, p_j \in \Pi_l\}. \quad (6)$$

If the Λ value is reached while δ_{il}^* rests below $\delta_{il} + \gamma$ then p_i is not a cheater.

The rationale behind the choice of incrementing λ_l, till reaching the bound mentioned above, is as follows. The first term UB makes sure that no other peer p_j has higher latencies to reach p_i i.e., $\delta_{li} + \lambda_l > UB \geq \delta_{ji}$, $\forall p_j \in \Pi_{i,l}$. Furthermore, cases may arise where some game events e^j, subsequent to e^l but still within $W(e_k^i)$, can be generated by other peers in $\Pi_{i,l}$. With this in view, the second term $T_l^W(\Delta s)$ accounts for those events $e^j \in W(e_k^i)$ with simulation times higher than e^l but within a time interval of range Δs. The third and final term $max\{gap_{jl}, p_j \in \Pi_l\}$, instead, accounts for for those peers p_j with $gap_{jl} > 0$. Indeed, it may be the case that p_l and p_j generate events with same simulation times at different real times and p_j reaches these simulation times after p_l (see Figure 2). Summing up, the increment of λ_l guarantees that no event in $W(e_k^i)$ is received by p_i later than e^l. Thus, in the case that p_i is cheating, p_i will eventually stop, waiting for game events coming from p_l.

A general description of our AC/DC, executed by p_l, is presented in Figure 3 and behaves as discussed above. It is worth noticing that p_l assumes that network latencies are symmetric (i.e., $\delta_{li} = \delta_{il}$). Moreover, because of delay jitter, to obtain an accurate value of δ_{il}^* a properly tuned number of measurements must be set, based on the specific network conditions. Finally, the procedure $increase(\lambda_l)$ mentioned in Figure 3 is not specified here. There are different possible implementation choices such as, for example, $\lambda_l = \lambda_l * k, k > 1$, or a (less aggressive) linear growth of $\lambda_l = \lambda_l + k$, for some constant k.

```
Process p_l:
    p_i = peer to control;
    assume δ_li = δ_il; /*assumption of symmetry*/
    λ_l = init value; /*init value > 0*/
    while ((δ_li + λ_l ≤ Λ) ∧ (p_i is not suspected))
      set additional delaying time = λ_l;
      observe δ*_il of received game events;
      if (δ*_il > δ_il + γ)
        suspect p_i;
      else
        λ_l = increase(λ_l);
```

Figure 3: AC/DC Pseudo-Code

4. DISCUSSION

A first important issue relates to the choice of λ_l. One may argue that a direct use of a value for λ_l such that $\delta_{li} + \lambda_l > \Lambda$ could make the algorithm simpler, as no increments on λ_l would be needed during the check phase. However, an approach based on the progressive increment of λ_l may be of help to avoid a too high and unneeded growth of transmission latencies between p_l and p_i.

Note also that we are assuming that the increment of λ_l, bounded as discussed before, may lead new observed latencies from p_i to p_l to surpass a significance threshold, i.e., $\delta_{il}^* > \delta_{il} + \gamma$. In practice, γ should be properly tuned based on guarantees (in terms of latency and delay jitter) offered by the underlying network over which the game is deployed and on the needed significance level. In particular, the bound on the increment of λ_l enables to detect cheaters when γ is set so that $\gamma < \delta_{li} + \lambda_l - gap_{il}$ (see Appendix A). Conversely, if a value of γ is needed that is higher than this last term, higher values for Λ must be taken into account. Needless to say, the higher Λ the larger the introduced additional delays to catch a cheater. Finally, we do not suggest any specific statistical test to infer whether the two measures $\delta_{il}^*, \delta_{il}$ are different. Sophistication of the employed test should be determined on a case by case basis.

AC/DC makes p_l able to detect if p_i is cheating. Indeed, suppose that p_i is a cheater. Due to the progressive increment of λ_l, latencies experienced by p_i to receive game events generated by p_l will increase accordingly. Moreover, p_l lets λ_l to grow until the overall latency between p_i and p_l reaches Λ, if necessary. Thus, during the generation of cheated events e_k^i at p_i, eventually the time for receiving all events in $W(e_k^i)$ grows in accordance with $\delta_{li} + \lambda_l$. This confirms that p_i will generate successive events e_k^i after increasing waiting times and p_l will receive these events with increasing measured average delays $\delta_{li}(e_k^i)$. A detailed discussion on the correctness of AC/DC is reported in Appendix A.

When p_i is not a cheater, instead, the delays added by p_l before the notification of its generated game events will not influence the event generation activity at p_i. This claim is supported by our assumption that the event generation rate at a honest node is independent from other ones. Summing

up, during the cheating counterattack, transmission latencies of events from a honest peer p_i, $\delta_{ij}(e)$, will be approximately equal to the average latency δ_{ij}, i.e., $\delta_{ij}(e) \simeq \delta_{ij}$. Thus, p_l will not be able to conclude that p_i is cheating. On the other hand, p_i will observe progressively increasing latencies for events coming from p_l during the checking period. In point of this, however, we already observed that advanced techniques such as dead reckoning and/or optimistic synchronization schemes may be exploited at honest peers. These approaches guarantee a fast paced evolution of the game, even when augmented network latencies slow down the reception of "fresh" game state updates [5, 10].

Clearly, our cheating counterattack should be activated only when some peer is suspected by another one. In other words, methods are needed to suspect a peer for cheating. In practice, several heuristics that suggest when starting AC/DC against a possible cheater can be devised. It is not the aim of this work to list all possible causes for cheating suspicion. Probably, a combination of diverse factors should be taken into consideration. Examples are factors based on degradation of latencies between p_l and p_i, anomalies based on the observation (at p_l) that other peers "near" p_i have smaller latencies (than p_i), or, finally, a player presents particularly striking game skills i.e., he always wins, therefore he is very skilled or he is cheating.

4.1 On the Choice of a Leader

Our scheme assumes the presence of a leader in charge of controlling whether some other peer nodes cheat. Thus, viable choices for a leader election in a completely decentralized peer-to-peer architecture are needed. In general, the leader should be a peer trusted by all other peers. However, agreement among all peers on the trustworthiness of a single node is often hard to achieve. Furthermore, since the leader delays its generated game events before transmitting them, it results that peers cannot freely assume this role without informing others. Indeed, suppose there are two peers which concurrently elect themselves as leaders, and suppose they decide to control each other. In this case, both peers will delay transmission of game events towards the other one. Thus, both peers may erroneously suspect each other.

A possible solution consists in exploiting a token based scheme according to which, every time a process receives the token becomes the leader. When a leader, say p_k, suspects another node p_i, p_k passes the token to another peer $p_j, j \neq i, k$, informing it of its p_i's suspicion. Then, p_j will in turn control p_i. Once a majority of peers suspects p_i, then p_i is identified as cheater. This solution enables agreement among honest peers (on cheaters detection). Moreover, such a cooperative approach makes harder for the cheater to detect if some other node is monitoring his behavior. Thus, it results more difficult for the cheater to dynamically switch off its cheat as soon as he detects he is being examined.

5. CONCLUSIONS

From a game design point of view, the best-effort nature of the Internet may be interpreted as a system inadequacy for the support of fast paced online games. Probably, in a QoS guaranteed network, time cheats would be not possible. However if, on one side, Internet protocols present several limitations for the deployment of these kinds of distributed interactive applications, on the other hand, it is the possibility of playing over the Web that makes online games successful. Thus, until IPv4 will be used as network protocol for the Internet, distributed cheat proof solutions are required in the context of online gaming.

In this work, we have considered a version of the well-known lookahead cheat, adapted in the context of real-time games, and proposed a countermeasure to face with it. The provided model is very general and can be adopted in a wide range of different gaming scenarios. The idea at the basis of our approach is that a scheme that permits to detect cheaters with a certain probability represents a viable solution to face with cheating without affecting game performances. A general analysis has been provided to demonstrate the correctness of the devised approach. Extensions to this study include the introduction of asymmetric delays among peers in the system. This would enable to model variegated existing scenarios in residential networks such as ADSL or wireless networks, for example.

6. REFERENCES

[1] N.E. Baughman, B.N. Levine, "Cheat-proof Playout for Centralized and Distributed Online Games", in *Proc. of INFOCOM 2001*, Anchorage (USA), IEEE, April 2001, 104-113.

[2] M.S. Borella, "Source models for network game traffic", in *Computer Communications*, 23(4):403-410, February 2000.

[3] F.R. Cecin, R. Real, R. de Oliveira Jannone, C.F. Resin Geyer, M.G. Martins, J.L. Victoria Barbosa, "FreeMMG: A Scalable and Cheat-Resistant Distribution Model for Internet Games", in *Proc. of International Symposium on Distributed Simulation and Real-Time Applications*, Budapest (Hungary), IEEE, October 2004, 83-90.

[4] C. Chambers, W. Feng, D. Saha, "Mitigating information exposure to cheaters in real-time strategy games", in *Proc. of the International Workshop on Network and Operating Systems Support For Digital Audio and Video*, NOSSDAV '05, June 2005, ACM, New York (USA), 7-12.

[5] E. Cronin, B. Filstrup, S. Jamin, A.R. Kurc, "An efficient synchronization mechanism for mirrored game architectures", *Multimedia Tools and Applications*, 23(1):7-30, May 2004.

[6] E. Cronin, B. Filstrup, S. Jamin. "Cheat-proofing dead reckoned multiplayer games", In *Proc. of 2nd International Conference on Application and Development of Computer Games*, January 2003.

[7] B. Di Chen, M. Maheswaran, "A Fair Synchronization Protocol with Cheat Proofing for Decentralized Online Multiplayer Games, in *Proc. of Third IEEE International Symposium on Network Computing and Applications* (NCA'04), Cambridge (USA), IEEE, August 2004, 372-375.

[8] F. Cristian, "Probabilistic clock synchronization", *Distributed Computing*, 3(3):146-158, 1989.

[9] M. DeLap, B. Knutsson, H. Lu, O. Sokolsky, U. Sammapun, I. Lee, C. Tsarouchis, "Is runtime verification applicable to cheat detection?", in *Proc. of ACM SIGCOMM 2004 Workshops on Netgames '04: Network and System Support For Games*, Portland (USA), ACM, August 2004, 134-138.

[10] S. Ferretti, M. Roccetti, "Fast Delivery of Game Events with an Optimistic Synchronization Mechanism in Massive Multiplayer Online Games", in *Proc. of ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005)*, Valencia (Spain), ACM, June 2005, 405-412.

[11] J. Farber, "Network game traffic modelling", in *Proc. of the 1st Workshop on Network and system support for games*, Braunschweig (Germany), ACM, April 2002, 53–57.

[12] R. Fujimoto, "Parallel and Distribution Simulation Systems", John Wiley and Sons, Inc., 1999.

[13] C. GauthierDickey, D. Zappala, V. Lo, J. Marr, "Low

latency and cheat-proof event ordering for peer-to-peer games", in *Proc. of the 14th International Workshop on Network and Operating Systems Support For Digital Audio and Video* (NOSSDAV'04), Cork (Ireland), ACM, June 2004, 134-139.

[14] R. Gusella, S. Zatti, "The accuracy of clock synchronization achieved by tempo in Berkeley Unix 4.3BSD", in *IEEE Transactions of Software Engineering*, 15(7):47-53, July 1989.

[15] T. Henderson, S. Bhatti, "Modelling user behaviour in networked games", in *Proc. of the 9th ACM International Conference on Multimedia (ACM Multimedia)*, Ottawa (Canada), October 2001, 212-220.

[16] H. Lee, E. Kozlowski, S. Lenker, S. Jamin, "Synchronization and Cheat-Proofing Protocol for Real-Time Multiplayer Games", in *Proc. of the International Workshop on Entertainment Computing*, Makuari (Japan), May 2002.

[17] D.L. Mills, "Internet time synchronization: the Network Time Protocol", in *IEEE Transactions on Communications*, 39(10):1482-1493, October 1991.

[18] M. Pritchard, "How to hurt the hackers: the scoop on Internet Cheating and How You Can Combat It", in Gamasutra Web Site, July 2000, http://www.gamasutra.com/.

[19] J. Yan, H.J. Choi, "Security Issues in Online Games", in *The Electronic Library: International Journal for the application of technology in information environments*, Emerald, Vol. 20 No.2, 2002.

[20] J. Yan, B. Randell, "A Systematic Classification of Cheating in Online Games". in *Proc. of the 4th Workshop on Network and System Support for Games (NetGames'05)*, New York (USA), October 2005.

APPENDIX

A. ON THE CORRECTNESS OF AC/DC

We provide evidence of the general correctness of AC/DC. In particular, we show that by exploiting our proposed cheating counterattack scheme, p_l is able to detect if p_i is cheating.

Suppose p_i is a cheater. Then, denote with $\tau_{il}^c(e^l, e_k^i)$ the amount of time elapsed since the generation at p_l of its game event $e^l \in W(e_k^i)$, its notification to p_i, the time p_i waits to receive all events in $W(e_k^i)$, the subsequent (generation and) transmission of e_k^i (by p_i) and, finally, its reception at p_l. Before p_l starts AC/DC, τ_{il}^c depends on the time needed by p_i to receive events from Π_i and on the time needed to notify p_l with the newly cheated game event e_k^i.[1] At p_l, this amount of time can be measured as

$$\begin{aligned}\tau_{il}^c(e^l, e_k^i) &= max\Big\{\Big[(WT_j(e^j) + drift_{jl} - WT_l(e^l)) + \\ &+ \delta_{ji}(e^j)\Big], e^j, e^l \in W(e_k^i)\Big\} + \delta_{il}(e_k^i). \quad (7)\end{aligned}$$

In (7), the $max\{\ldots\}$ term accounts for the time needed to transmit all events in $W(e_k^i)$, starting such measurement from the time of generation of e^l.[2] The second term $\delta_{il}(e_k^i)$, instead, accounts for the latency needed for the transmission of the cheated event e_k^i (generated by p_i after having received all events in $W(e_k^i)$) to p_l. Generally, before p_l starts AC/DC, the average latencies δ_{li}, δ_{il} result smaller than $\tau_{il}^c()$ i.e., $\delta_{li} \leq \tau_{il}^c()$ and $\delta_{il} \leq \tau_{il}^c()$. This consideration is straightforward from (7).

Once p_l starts AC/DC, eventually the $\tau_{il}^c()$ value will be approximately equal to

$$\begin{aligned}\tau_{il}^c(e^l, e_k^i) &= \delta_{li}(e^l) + \lambda_l + \delta_{il}(e_k^i) \\ &\simeq \delta_{li} + \lambda_l + \delta_{il}, \\ &\simeq 2\delta_{li} + \lambda_l, \quad (8)\end{aligned}$$

as $\delta_{li}(e_k^i) + \lambda_l$ will result higher than every other event transmission delay for events in $W(e_k^i)$. In other words, once p_l starts AC/DC, $e^l \in W(e_k^i)$ is the last game event in $W(e_k^i)$ received by p_i before generating e_k^i. Thus, if p_i is a cheater, p_i will wait for the event $e^l \in W(e_k^i)$ generated by p_l. Thus, since p_l progressively increases λ_l, and Equation (8) is satisfied, eventually also $\tau_{il}^c()$ grows accordingly.

However, the cheater p_i alters timestamps of its generated events e_k^i. Based on these timestamps, the (cheated) latency $\delta_{il}^*(e_k^i)$ measured at p_l for the transmission of each cheated event e_k^i is

$$\delta_{il}^*(e_k^i) = WT_l^{rec}(e_k^i) + drift_{li} - WT_i^c(e_k^i). \quad (9)$$

$WT_i^c(e_k^i)$ is assigned to e_k^i by p_i, based on the cheated simulation time $ST_i^c(e_k^i)$, computed as specified in (4). Thus,

$$ST_i^c(e_k^i) < ST_j(e^j),\ \forall e^j \in W(e_k^i). \quad (10)$$

From (10), (2) and (3), it results that $\forall e^j \in W(e_k^i)$,

$$\begin{aligned}ST_i^c(e_k^i) &< ST_j(e^j), \\ WT_i^c(e_k^i) - t_{i,start} &< WT_j(e^j) - t_{i,start} + \\ &\quad - drift_{ij} + gap_{ij}, \\ WT_i^c(e_k^i) &< WT_j(e^j) + drift_{ji} + \\ &\quad gap_{ij}. \quad (11)\end{aligned}$$

As to e^l, generated by p_l,

$$e^l \in W(e_k^i) \Rightarrow WT_i^c(e_k^i) < WT_l(e^l) + drift_{li} + gap_{il}. \quad (12)$$

Thus, by substituting (12) in (9),

$$\begin{aligned}\delta_{il}^*(e_k^i) &= WT_l^{rec}(e_k^i) - (WT_i^c(e_k^i) - drift_{li}) \\ &> WT_l^{rec}(e_k^i) - WT_l(e^l) - gap_{il}. \quad (13)\end{aligned}$$

Now, since during AC/DC $\tau_{il}^c(e^l, e_k^i)$ eventually becomes equal to the time interval for the transmission of e^l from p_l to p_i and the time of reception of e_k^i at p_l, we have

$$\tau_{il}^c(e^l, e_k^i) = WT_l^{rec}(e_k^i) - WT_l(e^l). \quad (14)$$

Therefore, using (8), (13) and (14), eventually the following statement holds

$$\begin{aligned}\delta_{il}^*(e_k^i) &> WT_l^{rec}(e_k^i) - WT_l(e^l) - gap_{il} \\ &= \tau_{il}^c(e^l, e_k^i) - gap_{il} \\ &\simeq 2\delta_{li} + \lambda_l - gap_{il}. \quad (15)\end{aligned}$$

As discussed in Section 2, gap_{il} is a constant, short value. Hence, as λ_l grows (same for $\tau_{il}^c(e^l, e_k^i)$) during AC/DC, also $\delta_{il}^*(e_k^i)$ does it. In other words, during AC/DC, eventually timestamps associated to events generated by p_i will lead p_l to measure increasing network latencies. When γ may be set so that $\gamma < \delta_{li} + \lambda_l - gap_{il}$ (for some value of λ_l, bounded as already discussed), then p_l is able to detect if p_i is cheating.

[1] We assume that the processing time needed to analyze received events and generate e_k^i is negligible.

[2] For each event e^j in the considered set, the difference between the times of generation of e^j and the event generated by p_l is considered ($drift_{jl}$ is exploited to equalize physical clocks of the two peers). The time required for the transmission of the event e^j to p_i, i.e., $\delta_{ji}(e^j)$, is then added to this term. The higher value obtained among all events e^j is considered.