Challenges in Peer-to-Peer Gaming

Christoph Neumann
Thomson Research Lab, Paris, France
christoph.neumann@thomson.net

Nicolas Prigent
Thomson Security Research Lab, Rennes, France
nicolas.prigent@thomson.net

Matteo Varvello
Eurecom, Sophia-Antipolis, France
varvello@eurecom.fr

Kyoungwon Suh
University of Massachusetts, Amherst, US
kwsuh@cs.umass.edu

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
Authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

ABSTRACT

While multi-player online games are very successful, their fast deployment suffers from their server-based architecture. Indeed, servers both limit the scalability of the games and increase deployment costs. However, they make it easier to control the game (e.g. by preventing cheating and providing support for billing). Peer-to-peer, i.e. transfer of the game functions on each each player's machine, is an attractive communication model for online gaming. We investigate here the challenges of peer-to-peer gaming, hoping that this discussion will generate a broader interest in the research community.

1. INTRODUCTION

Today, millions of players are connected to several online gaming communities [1]. World of Warcraft[1] is an impressive example of online game success: more than $6,000,000$ players pay monthly fees to play in this Virtual World, $500,000$ players being connected at any time. These numbers are expected to grow in the future.

Most online games rely on a central server or a cluster of servers. Such architecture allows to easily handle tasks like players' access control, management of the states of the game, synchronization of players, as well as billing.

However, centralized architectures have also many drawbacks: (1) servers have to be physically deployed, operated and maintained; (2) they do not scale, and must be dimensioned according to the maximum number of expected players; (3) they limit service availability, the server being a single point of failure; (4) it is finally not optimal from a delay standpoint (choosing a central server with respect to delay is difficult as there always may be a player with higher delay.)

These drawbacks, the high cost of a centralized solution, as well as the players frustration in front of under-provisioned systems and long delays are strong incentives to investigate alternative decentralized peer-to-peer solutions. Indeed, a peer-to-peer architecture strongly reduces deployment investment: there is no need to physically deploy important resources, and all players can benefit from the shared resources (storage, CPU and bandwidth) making the architecture scalable. Moreover, peer-to-peer is by nature more robust as there is no single point of failure. Consequently, the architecture does not fail *a priori* if any of its members leaves the game (gracefully or because of a crash/disconnect). Last, a peer-to-peer network can organize itself very easily to optimize the delay among peers.

Aside these practical advantages of peer-to-peer architectures, we foresee that evolutions of online games may indeed require them to be based on peer-to-peer technologies: future virtual worlds could be build on top of infrastructure-less networks such as Delay Tolerant Networks. Users will play on their PC or on their television, but also on their PDAs, cellular phones, etc. Each user will have only an intermittent connectivity, or only connectivity to small sets of devices that are not necessarily those the players want to interact with. The devices used to play might therefore be very heterogeneous. Moreover, in addition to classical online games, "mixed reality games", where players physical location impacts the location of the avatar in the game [13], seem perfectly suited to run on infrastructure-less communication architectures. In both these contexts, the migration to peer-to-peer distributed gaming seems even more critical.

While peer-to-peer gaming architectures are promising, they yield a lot of challenging research topics. In particular, questions are raised about distributed storage, management and consistency of the game states, election of the peers to communicate with, optimization of delays, synchronization of players and protection against cheating. To our knowledge, little work has been done in this area (see related work, section 3), especially considering large scale games with possibly millions of players.

This paper proposes a discussion on peer-to-peer gaming challenges, hoping that it will encourage the research community to explore this topic further. We do not solve problems, but rather give hints on possible future research directions.

2. P2P GAMING CHALLENGES

We focus on the consequences of the game distribution among the peers in terms of (1) game state management, even in the presence of peer failures, (2) delay, (3) scalability

[1] http://www.worldofwarcraft.com/

and (4) cheating.

2.1 Game state management

A game is characterized by a set of states, these states being modified along the game course by events. Whatever the communication architecture, the system has to guarantee that *the game state is consistent* among all players, in other words that the information each player gets reflects all the relevant events that occurred in the past.

With a peer-to-peer gaming approach, an overlay is constructed in order to manage states. The overlay topology fundamentally influences the way the states are managed. Two approaches can be considered, static overlay and dynamic overlay.

With static overlays, peers are organized in a structured way, which is independent of the interests of each player. Each peer gets assigned objects, players or zones to manage. In other words, the overlay is defined beforehand and data is injected in this overlay, without rearranging it. Events and accesses to a state are routed through the overlay e.g. using a Distributed Hash Table (DHT) as in [12]. In dynamic overlays instead, peers dynamically construct the overlay according to their interests and interactions between players. These interests may be dynamic and change over time.

This discussion raises the question whether the overlay in the specific case of p2p-gaming should be interest-driven or not[2]? To answer this question we believe that it is necessary to divide game states and events in two types, local and global.

Global states (e.g. global clock in the virtual world) or events (e.g. an earthquake that affects the whole virtual world) are not related to a specific player or location and are *a priori* accessible at any time. By contrast, local states or events are either (1) physically bound to a player's avatar, e.g. its state or the one of an object he or she owns, (2) logically bound to a player or its avatar, e.g. several avatars of the same clique communicating together, or (3) bound to a specific location in the game, e.g. the state of an object at a given location or some local event such as rain on a limited location. Local states do not need to be consistent among all players. Other players who don't interact with it at this time can update it later.

Since a player is always interested in global states and events, static overlays are more adapted to handle them. By contrast, local states and events are better handled by dynamic overlays, since they can adapt to the dynamic nature of this information. Therefore the two type of overlays could be used in a complementary way.

Furthermore, other storage and access schemes could further complement them. For instance, critical information like access control policies could be stored and managed on centralized servers, while unmodifiable information could be stored on all peers.

Whatever the schemes chosen, handling information consistency becomes an issue with peer failure and churn, i.e. players that leave and join the game, or nodes that are disconnected or delayed for a short time because of network congestion. To address this problem, the classical approach is to replicate game state among several peers. However, there is trade-off to find between resistance to failure, churn, performances in terms of reactivity and bandwidth consumption.

Churn can almost be strongly mitigated if the game is deployed on Set-Top-Boxes or Residential Home Gateways that are controlled by some central authority (e.g. the content provider or the ISP) : these devices being "always on", they can participate in the game overlays even if the player is disconnected from the game or gracefully leaves it. In this case, the only cause of churn is a device or network failure.

2.2 Delays management

Delays must be kept beneath the human perceptible thresholds. Distributed Interactive Simulation (DIS) and the High Level Architecture (HLA) standards [11] specify delays beneath 200ms. Above that, playing becomes uncomfortable and there is unfairness if all players do not have the same delay.

Delays are caused by two factors: (1) the time to transport and process the information over the network, that we call *network delays*, and (2) the time to take into account all players actions and to synchronize them (e.g. by waiting for the slowest one) that we call *synchronization delays*.

Network delays are already often close from or even above the values tolerated by humans perception. In client-server architectures this delay is due to either the lack of CPU resources or bandwidth by the server and the RTT between each client and its server.

By using a overlay, players ideally communicate directly with each others without going through a central server. This removes the risk for the server to become a bottleneck and reduces the number of hops required for players communication. However, there is a risk to make some specific nodes in the overlay become themselves bottlenecks, for instance when a node having a low bandwidth manages game states a lot of other peers are interested in. This advocates for overlay building mechanisms that take into account available bandwidth of each peer, RTT between peers, and that try to let all players experience roughly the same delay so as to minimizes the effects of synchronization delays.

In both client-server and peer-to-peer technologies, techniques like "dead reckoning" can conceal players with higher delays by predicting and updating the states of their avatars according to the previous moves. Once the real actions have been received from a given player, the predicted state is corrected. However, dead reckoning may result in a degradation of playing experience [4]. Note also that dead-reckoning has some cheating issues as discussed in [2].

Regarding synchronization delays, play-out buffer can be used to make all players experience the "same" delay. A basic stop-and-wait protocol like Lockstep [2] waits for the slowest player to send its actions before updating the game state. Because it is often unacceptable to make a single player to slow down the entire game, other techniques like "Bucket-synchronization" [4] do not require all players to wait for the slowest one but periodically synchronize all the received actions. Unavailable information from a player can then be "replaced" by dead-reckoned state. The number of peers involved in the synchronization process can be optimized according to the players interests.

2.3 Scalability

Since each peer contributes in CPU, storage and bandwidth resources, most peer-to-peer systems are by nature

[2]Levine et al. [9] had a similar discussion on content-based multicast, i.e. what the best mapping of content to multicast groups according to receiver interests.

more scalable than server based architecture. However, experience shows that also peer-to-peer systems are limited in scalability (e.g. Gnutella). Applied to peer-to-peer gaming, this may be expressed as higher delays when the number of player increases. To push back these limits, and based on what is already done today for server-based solutions, we can envision to improve the scalability by breaking the game space down into smaller regions and treating each region as a separate game.

The regions can be statically predefined as in [15] or built ad-hoc because there are players at one spot in the game. This approach again raises the difference between non-interest-driven (fixed and predefined regions) and interest-driven (ad-hoc) overlay design.

The system can then use a dedicated overlay to manage the region, or elect a responsible peer acting as a server for that region: it handles information consistency, and might even stores all the information of that zone. A peer responsible for a zone is not necessarily in that zone. Instead, a metric that considers only stability in term of permanence in the game may help choosing the responsible peer.

However, it is difficult to anticipate scalability issues in peer-to-peer gaming without experimental deployment.

2.4 Cheating

Resistance to *cheating* is one of the biggest challenge in peer-to-peer gaming. We define *cheating* as an unauthorized interaction with the game system aimed at offering an advantage to the cheater.

We distinguish three categories of cheating[3] according to the threatened game property [10]: (1) Confidentiality, when the cheater obtains information that he or she is not supposed to, such as elements of the global state of the game (e.g., a place on the map), interactions between other players he or she is not involved in, or local information about another avatar's confidential characteristics; (2) Integrity, when the cheater modifies the state of the game or its fundamental laws, for instance by modifying the map, the state of an avatar, or the rules of physics; (3) Availability, when the cheater delays or switches off (parts of) the game such as a zone or a set of avatars, for instance when things go wrong for him or her.

To achieve her goals, a cheater can act on various targets. First, she can focus on the network and eavesdrop, inject, delay or drop messages. She may for instance passively examine the private communications that she is just supposed to forward. Peer-to-peer architectures are probably more vulnerable to this kind of attacks, since in server based architecture each player communicates only with the server and is not involved in message forwarding.

Second, a cheater may target the game states. She can for instance use the application data she stores (that in the peer-to-peer case may not be only related to her) to discover where specific entities are hidden in the virtual world, or modify the state of the virtual world or of avatars. While client-server solutions can benefit from a trusted server to manage the game states and ensure its integrity and confidentiality if necessary, game states are distributed among the peers in peer-to-peer architectures. Consequently the cheater potentially controls the information stored on its peer.

Finally, a cheater may target the game application itself, or even the host system on which it runs. For instance, she may reverse-engineer the game application to obtain more precise information on how it works. She can also modify the rules by changing the code of the program, or by tweaking the pseudo-random number generator of the host system it runs on. Here again peer-to-peer architectures are probably more vulnerable to these cheats because they are made of a set of *a priori* untrusted peers and cannot rely on a trusted server to implement the security critical primitives.

This list of target is not exhaustive. Particularly, we have not considered the cheating methods that use other communications channels, such as the one for which the cheater uses its phone to call other players and disclose them information while he is not supposed to be able to communicate with them in the game. Using once again the terminology of the security field, these cheating methods can be related to "side-channel attacks".

To address cheating, two complementary approaches can be considered: cheating-resistant systems and cheating-evident systems. Cheating-resistant systems' goal is to prevent cheating. Among the relevant services are (1) player authentication, (2) accountability (that allows to impute the actions of a player to him or her after he or she played), (3) confidentiality of information and communications, (4) application integrity and (5) tamper-proof devices to prevent a cheater from interfering with the game application and its data. By contrast, cheating-evident systems' goal is to reactively detect cheaters and punish them (e.g. ban them or degrade their experience).

Cheating prevention and detection could be relatively simple in a centralized server-based architecture, especially because a reasonably controlled and thus trustworthy server is available. A peer-to-peer environment yields more challenges. First, it may be very complicated to apply cheating resistance extensively to a whole peer-to-peer architecture, that is mostly made of uncontrolled hosts. Moreover, it is unclear where to place the mechanisms to be used: on all the devices that take part to the game, or only on a subset of them? In this latter case how is this subset selected? Aside from this, in the case of a fully decentralized approach, the management of trust between the devices and between the players is a complex issue. How can we be sure that a device or a player acts faithfully? How is trust established and maintained? How is slandering handled? All these questions are close from the one that are yet only partially solved in security proposals for peer-to-peer and ad hoc networks.

Another intuitive way to prevent cheating, that takes advantage from the feature of distribution of peer-to-peer architectures, would be to make a peer to handle a region only if its player's avatar is not present in it. However, this solution remains hazardous, because a player may nevertheless want to cheat and influence what happens in this region, due to the side effects it involves (e.g. making avatars of its team win, making an enemy avatar eliminated etc.). Thus, such a proposal would also require to authorize a peer to handle a region only if the events that happen on it have no interaction with its player's interests. This may be a very difficult problem, because it is very complex to define precisely each player's interests, especially when handling the side effects.

[3]In this paper, and by contrast with the field of security, we do not consider the case of vandals, that are not cheaters because their actions offer them no advantage in the game.

3. RELATED WORK

MiMaze [4, 5] was one of the first attempts to design a fully distributed online game. It relies on IP Multicast: each player has an entire replication of the virtual world on his machine (as opposed to having just a partial view), and multicasts its actions to all others players. This technique can be considered as peer-to-peer, since each player communicates its actions directly to the other players.

More recent papers proposed fully-distributed peer-to-peer gaming solutions, that either rely on structured overlays (i.e. DHT) or an unstructured peer-to-peer architecture. All take locality into account, and build the storage and communication system accordingly. More precisely, Knutsson et al. [8] present a peer-to-peer solution for massively multi-player games based on the structured overlay Pastry [12]. Scribe [3] is used as a multicast application layer build on top of Pastry. The approach divides the virtual world into *fixed and predefined* regions and elects (with the help of the DHT) one peer as "coordinator" for each region. The coordinator maintains the state of the region he is responsible for (which is not necessarily the region where he is playing) and multicasts (using Scribe) updates to all peers in the same region. The paper also discusses how the coordinator should be replicated in order to be robust to peer failures. However, [6] argues that the communication overhead induced by the multicasting of state information to the peers of a region in [8] is too high.

In a similar mindset, [15] proposes to split the virtual world into hexagonal cells, each cell being mapped to a determined node (using a DHT). Using a master and slave node mechanism for each cell, this approach further allows players to extend their view to neighboring cells and not being bound by the border of a cell.

Regarding cheat-proof distributed gaming, Yan et al. [14] have proposed a classification of cheating in online games. This classification, that is made of 15 categories, is built upon existing attacks and may not be able to integrate new kinds of attacks that may appear, especially in p2p-gaming.

Baughman et al. [2] propose the cheating-prevention gaming protocols "Lockstep" and "Asynchronous Synchronization" (AS). Lockstep is a stop and wait protocol with commitment. Each player sends a hash of its next move to all other players. Once everybody has sent her commitment, all players reveal their real moves. However, this approach does not scale. AS relaxes the synchronization of Lockstep, and only imposes synchronization between players that "see" each other in the game. The problem with lockstep and AS is that they assume homogeneous players with respect to their network and client resources. This assumption is relaxed by the follow-up work, Ghost [7].

4. CONCLUSIONS

Based on the hypothesis that centralized architectures will not scale for online gaming as more and more players are involved in a single session, this paper presented research challenges related to the use of peer-to-peer technologies. While they are pervasively used for all kind of applications involving numerous users, we showed that a lot of work is still required to apply them to online-gaming. Particularly, we identified the following axes of research: first, new overlay managements are necessary to fulfill both the strong requirements of delay and information consistency that multiplayer gaming exhibits; second, the community needs to check experimentally that these new overlay managements are really scalable; finally, ways to handle cheating in such uncontrolled networks are such to be found. In summary, the main challenge seems to be how to manage the overlays as there are many candidate parameters to optimize concurrently, that will most probably make the optimisation problem NP-hard.

5. ACKNOWLEDGMENTS

We would like to acknowledge Christophe Diot and Eric Diehl for their helpful comments and Moritz Steiner for pointing to some interesting references.

6. REFERENCES

[1] *An Analysis of MMOG Subscription Growth web site.* http://www.mmogchart.com/.

[2] N. E. Baughman, M. Liberatore, and B. N. Levine. Cheat-proof playout for centralized and peer-to-peer gaming. *IEEE/ACM Transactions on Networking*. To appear.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), 2002.

[4] L. Gautier and C. Diot. Design and evaluation of mimaze, a multi-player game in the internet. In *Proceedings of IEEE Multimedia Systems Conference*, June 1998.

[5] L. Gautier, C. Diot, and J. Kurose. End-to-end transmission control mechanisms for multiparty interactive applications on the internet. In *Proceedings of IEEE INFOCOM'99*, Mar. 1999.

[6] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04*, pages 116–120, New York, NY, USA, September 2004.

[7] A. S. John and B. N. Levine. Supporting p2p gaming when players have heterogeneous resources. In *Proceedings of NOSSDAV '05*, June 2005.

[8] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceedings of IEEE INFOCOM'04*, March 2004.

[9] B. N. Levine, J. Crowcroft, C. Diot, J. Garcia-Luna-Aceves, and J. F. Kurose. Consideration of receiver interest for ip multicast delivery. In *Proceedings of IEEE INFOCOM'00*, Mar. 2000.

[10] D. of Defense. *Trusted Computer System Evaluation Criteria (The Orange Book)*, Dec. 1985. DoD 5200.28-STD.

[11] M. Pullen, M. Myjak, and C. Bouwens. *Limitations of Internet protocol suite for distributed simulation in the large mulitcast environment*, Feb. 1999. Request For Comments 2502.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[13] B. Wietrzyk and M. Radenkovic. Enabling rapid and cost-effective creation of massive pervasive games in vey unstable environments. In *IEEE/IFIP WONS 2007*, January 2007.

[14] J. Yan and B. Randell. A systematic classification of cheating in online games. In *Proceedings of the 4th Workshop on NetGames'05*, October 2005.

[15] A. P. Yu and S. T. Vuong. Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *Proceedings of NOSSDAV '05*, pages 99–104, june 2005.