VFC4FPS - Vector-Field Consistency for a First Person Shooter Game

Bruno Loureiro[1], Luis Veiga[2], and Paulo Ferreira[2]

[1] IST/Technical University of Lisbon
[2] INESC-ID/IST/Technical University of Lisbon
Distributed Systems Group
`bruno.loureiro@ist.utl.pt, [luis.veiga, paulo.ferreira]@inesc-id.pt`

Abstract. Multiplayer online games are increasingly more popular. Keeping the game state updated and consistent among all players in soft real-time is critical. Sending the complete game state to all players does not scale with the number of players. Increasing the game scalability can be done by reducing its network traffic, and one way to reduce network traffic is by exploiting the player's sensory limits. However, current solutions typically use an all or nothing approach, where a player only receives updates of objects inside his sensory zone.
In this work we use the Vector-Field Consistency model to offer progressive consistency decay. We use multiple zones, each with a set of consistency requirements, which decay with the increasing distance to a point of interest (called pivot).
We have modified the game Cube 2: Sauerbraten (a first person shooter game) to use the VFC model. In addition, by observing that players have a limited view of the virtual world, we modified the original VFC model to add the concept of Field of View.
The performance results obtained show a network traffic reduction of at least half of the original traffic, without harming consistency and playability.

Keywords: Multiplayer Online Games, Optimistic Consistency, Interest Management, Spatial locality

1 Introduction

In recent years, the popularity of online multiplayer games has been growing rapidly. Among the reasons for such growth is the increasing availability of broadband internet. A type of game that emerged with these new possibilities is the massively online multiplayer game (MMOG). MMOGs are characterized by high numbers of simultaneous players sharing a huge persistent virtual world.

A popular category of online games that do not fit in the size of MMOGs are the First Person Shooters (FPS). FPS have fast paced interactivity with emphasis on dexterity and reaction times of players. Compared to the hundreds or thousands of simultaneous players that participate in a MMOG, the number of players in a FPS is around the dozens (typically between 16 and 32). FPS differ from MMOGs by allowing servers to be hosted by players on their personal computers. However, the number of players supported by these servers is mainly limited by the available bandwidth. Game servers can also be housed on more powerful servers with more bandwidth, normally provided by communities of players.

In order to provide good performance, each player has local copies (replicas) of the game state of other players (player's positions, shots, etc.). Maintaining these replicas consistent in soft real time to ensure good playability without using too much bandwidth is the main difficulty in implementing an online game. Associated with the maintenance of game consistency is scalability, since lower communication efficiency means support for less players.

One solution for dealing with scalability is to use an scalable architecture. There are two main types of such an architecture: Client-Server and Peer-to-Peer (P2P)[5][4]. In client-server architectures there is a central server that receives state updates from clients and propagates these updates to other clients. In P2P architectures there is no central server, the server functions are divided among all the clients involved, which communicate directly with each other. Both architectures have hybrid variations.

In the case of consistency management, the technique called Interest Management(IM)[3][15] allows for a reduction of the amount of state updates required. This is achieved through the exploration of the sensory limits of the players. Each player has an area of interest (AoI) and is only interested in the state updates regarding objects within that area. The AoI can be based on regions[7][16][9], auras[3] or line of sight visibility[3][10]. This way a player is only interested in state updates of objects within the same region, that lie within a surrounding area or who they can see.

Another technique, that reduces the frequency of messages, is Dead Reckoning[15][12]. This technique reduces the frequency of player's position update messages through motion prediction, taking into account past movements. It works well for reduced intervals between messages[12].

Finally, at the communication layer, we have a set of techniques [5][15][6] that focus on compression, aggregation and multicast of packets in order to make communication as efficient as possible.

This work aims to adapt an existing IM technique and apply it to a real game. The main goal is to increase the scalability by reducing the network traffic, maintaining the playability and consistency. For this purpose we use the Vector-Field Consistency (VFC)[14] as a basis of our solution. VFC, in contrast to other IM techniques does not apply an all or nothing filter. VFC allows the consistency to decay gradually as the distance from a point of interest increases. A secondary goal is the development of our solution in the form of a library, thus being able to separate game logic from the details of consistency management.

Implementing the solution with a real game requires the game to have the source code available, i.e., the game has to be open source or a commercial game for which the source code was made available. We chose a First Person Shooter

(FPS) because there are a lot of games of this kind with these conditions. The FPS chosen was the Cube 2: Sauerbraten[3].

A key challenge of this work is linked to the nature of FPSs. These games offer mainly maps (virtual worlds) of small size while providing a wide range of vision. In order to increase the impact of the solution, the FPS characteristics will be taken into account. This is reflected with the inclusion of the field of view concept in VFC, allowing to define different consistency requirements for objects, as they are inside or outside the field of view of the player.

This document is organized as follows. In section 2 we describe the related word, focusing on the various existing solutions to reduce bandwidth usage. Section 3 describes the system's architecture and Section 4 presents the most important implementation details. In Section 5 we describe the evaluation and the obtained results. We finalize with Section 6, where we summarize this work and present ideas for future work.

2 Related Work

In a FPS, players compete in a virtual world through the internet. Each player controls an avatar (a digital representation of the player). Avatars normally compete against each other. Each avatar has an associated state, characterized by attributes such as position, health level, weapons, ammunition and armour. This state is changed through interaction with other avatars, objects and through his own movement.

Since the local state of each player consists of many remote objects, a naive solution to keep them updated would be asking all players for their updated status in each frame. This solution is impractical because the communication latency is higher than the frame processing time (typically 16 ms for 60 frames per second).

In practice, games keep replicas of each client's remote objects. Games then use the local state of replicas for processing frames. Replicas may not reflect the actual state and are updated periodically based on a consistency model. Due to the fast paced nature of FPS games, players tolerate latencies up to 100ms[11].

2.1 Consistency

Replication means that copies (replicas) of information are stored on several computers. Performance is obtained due to local access of information being faster than accessing the same information remotely. However, in order to keep local replicas consistent, replication mechanisms are needed. The main difficulty is managing changes to the same replica by different clients at the same time, and how quickly that consistency is maintained in order to guarantee game playability.

Replication can be managed in two ways: pessimistic and optimistic replication[13]. Pessimistic replication locks access to the replicas during an update.

[3] Cube 2: Sauerbraten, http://sauerbraten.org/

This keeps all replicas consistent between each other. Optimistic replication, on the other hand, allows replicas to diverge between clients. In order to guarantee game playability there must be a way to limit how much the replicas are allowed to diverge (e.g. a maximum time limit[1]). In TACT[17], besides keeping the divergence within a time limit, we can use the number of local updates to decide when to propagate the updates.

The system described in Donnybrook[10] was thought for low bandwidth environments. It uses a P2P architecture. It aggressively explores the limited perception of the virtual world by an avatar. Each player has a bandwidth limit which is distributed between the objects in which the avatar has more attention.

In RING[8], the system precomputes visibility between rooms to decide which avatars are visible for each player and exchanges updates only between visible avatars. This system is good for environments with high occlusion.

A^3[2] is an algorithm that combines a circular aura with a field of view. They do this to avoid inconsistencies when an avatar turns around abruptly. The frequency with which an avatar receives updates from others is reduced linearly with an increasing distance.

Vector-Field Consistency[14] introduces the concept of multiple concentric circular zones with decreasingly consistency requirements. The VFC is an optimistic consistency model that allows replicated objects to diverge in a limited way.

The levels of consistency associated with an avatar are specified by three-dimensional vectors. Each vector κ is connected to one consistency zone. Zones are defined around a pivot. A pivot can be an avatar or other object. Objects within the same zone are subjected to the same consistency level defined by the vector κ. This vectors specify the limits of which a replica is allowed to diverge. The three dimensions of the vector are time, sequence and value.

- Time: defines the maximum time (in seconds) that a replica is allowed to diverge.
- Sequence: defines the maximum number of updates that a replica can ignore.
- Value: defines the maximum difference between the content of an object and is measured in percentage.

When one of this dimensions is exceeded, the replica needs a new update.

3 Architecture

We use the Vector-Field Consistency model as the basis of our work given its flexibility and progressive consistency decay. However, we improved VFC to better support FPS. As a matter of fact, the original VFC consistency zones provide the same consistency level for all the 360^o area surrounding an avatar. However, the visibility of an avatar is limited to a field of view that is only part of such a 360^o area.

Thus, we introduced, the notion of Field of View which helps strengthen the consistency level in the avatar's field of view, while simultaneously decreases the consistency level for objects "behind" the avatar. Due to this adaptation of the VFC to the context of FPS, we named our system VFC4FPS (Vector-Field Consistency for First Person Shooters).

The game to which the VFC4FPS was applied to was the Cube 2: Sauerbraten. The reasons for this choice are the following: being open source, implemented in C++, popular, having large maps and an in-game map editor.

3.1 VFC Architecture

VFC was designed to be used as a library, thus relieving the game programmer from the communication details. Through the API provided by the library, the programmer can parametrize the consistency requirements. VFC uses a Client-server architecture.

Clients keep replicas of all shared objects (secondary object pool). The server keeps the main replicas (main object pool). A client is free to read the replicas, but when a update is made to a replica it must be propagated to the server. This propagation is not immediate. Both the client and the server send replica updates in a periodic fashion (rounds).

The server contains a consistency management block, which is responsible for deciding which updates are sent to each player at each round. This is done by a VFC function named round-triggered. Using each client view (set of consistency parameters), it decides if an object should be sent to the client.

In order to support the round-triggered function, VFC uses data structures to store the number of updates that each object has received since the last message sent to the player. It also stores the last time an object was sent and the value that the object had. This data is necessary to see if any of the three parameters of the κ vector was exceeded (in which case an update is needed).

3.2 VFC4FPS

As already said, VFC4FPS introduces the field of view (FoV) as a new parameter for consistency purposes. This is done by adding an array that contains the values for the angles that constitute the fields of view. Thus, w.r.t. original VFC, the array that contained the κ vectors gains a new dimension, and is now indexed by zone and FoV in order to obtain the κ vector. This means, for each zone, there is one κ vector for each FoV.

Another modification made by the VFC4FPS is as follows: change the units of the time dimension from seconds to milliseconds due to the high frequency with which updates are sent.

In Table 1 we can see one view with three zones, three FoVs and the corresponding κ vectors.

VFC4FPS maintains the basic architecture of VFC. However, one of the main changes was in the round-triggered function. This function was modified to use the orientation of the pivot to determine in which FoV an object is located. This is done in addition to the zone definition. With these two aspects considered, it

FoV \ Zone	300	600	∞
100°	[30, ∞, 0]	[60, ∞, 3%]	[90, ∞, 5%]
140°	[60, ∞, 6%]	[90, ∞, 9%]	[130, ∞, 13%]
360°	[90, ∞, 9%]	[200, ∞, 20%]	[300, ∞, 30%]

Table 1. Normal view.

is possible to obtain the κ vector with which the consistency state of an object is then compared.

Another change was the introduction of a compression module after the serialization stage in order to reduce the size of the updates.

Cube 2: Sauerbraten has two types of communication: object updates and events. Object updates are periodic and contain the avatar's state. Events are immediate and are sent as messages that change the global state of the game.

In Cube 2: Sauerbraten all shared objects are avatars. Guns, ammo, armour and health are controlled by events. Object updates contains all the state and can be mapped to the VFC4FPS.

Events, however, cannot be immediately mapped to the VFC4FPS model. This is due to the fact that events can not have their frequency reduced; at most they can be filtered. But then, there would be clients whose global state would differ from other clients.

4 Implementation

VFC4FPS was developed in a library form. The library was implemented in C# in the .Net platform from Microsoft[4]. Cube 2: Sauerbraten is implemented in C++. The use of the VFC4FPS functionality is done via the library's API. In order to simplify the implementation and leverage the interoperability offered by the .Net platform, the game's C++ code was compiled as managed C++, thus allowing to directly interface the game and library.

4.1 Interfaces for Shared Objects

Not all the objects in a game have the same characteristics. For this reason VFC4FPS offers an hierarchy of interfaces that objects can implement. The ISharedObject interface is useful for objects without spatial locality, i.e., objects that represent the overall state of the game. Next in the hierarchy there is the IPositionableObject interface. This interface applies to objects that have spatial locality. Finally, there is the IOrientableObject interface which adds the fields that define the orientation of the object in the form of a normalized vector. This interface is applicable to objects that have a limited view of the virtual world.

[4] http://www.microsoft.com/net/

4.2 Communication

The communication between VFC4FPS clients and the server is done through .Net remoting. Events are managed by the original Cube 2 system. Object updates are managed by the VFC4FPS. Objects are serialized in binary form and sent using the UDP protocol. This option is motivated by the fact that the .Net remoting protocol generates too much network traffic. In addition, given that a serialized object is considerably bigger than the game's original objects, we use two methods of compression to reduce the object size.

4.3 Serialization and Compression

A serialized object contains meta-data which takes up much space. However, only a subset of bytes, those of the fields, are the only ones that change. In order do eliminate the redundant information we use a delta compression system. After that we still had objects bigger than the original game. We then implemented a bitmap based compression to further reduce the object's size.

An instance of the CubeOriObj class in its serialized form has a small region of bytes that corresponds to the size of the fields of the class. Using introspection we can determine the total size of the fields. With this size, we calculate the offset where these bytes start. These bytes constitutes the delta of the class. This method was devised from empiric observation, and can have unexpected results when applied to other circumstances. One limitation is the use of variable fields, such as strings and arrays. However, in the context of the VFC4FPS and Cube 2, we observed that this approach works correctly.

The bitmap compression has the goal of eliminating useless bytes, in this case, null bytes. This method of compressing an array of bytes uses a bitmap to mark if the position in the array has a null value or not. Since each bit corresponds to a position in the array, a byte can index eight positions in the array. In short, this algorithm creates a bitmap of the array and marks each bit with zero or one depending on the byte value. This produces a bitmap and an array without null bytes. These two are concatenated and a byte is added with the total size of the bitmap, necessary for decompression.

5 Evaluation

To evaluate the network traffic in a real scenario, it would take a great number of players and infrastructure. Instead, our assessments were made through the use of avatars controlled by the artificial intelligence (bots) of Cube 2: Sauerbraten. However, in order to run multiple bots per computer it was needed to disable the graphical processing. Due to dependences with the game logic, it was not possible to completely disable de graphical output. Instead we disabled textures, sounds, light maps and some models. This way we obtained a lightweight bot controlled client.

Zone / FoV	600	1200	∞
35°	[30, ∞, 0]	[60, ∞, 3%]	[90, ∞, 5%]
75°	[60, ∞, 6%]	[90, ∞, 9%]	[130, ∞, 13%]
360°	[90, ∞, 9%]	[200, ∞, 20%]	[300, ∞, 30%]

Table 2. Zoom view.

Zone / FoV	300	600	∞
360°	[30, ∞, 0]	[60, ∞, 3%]	[90, ∞, 5%]

Table 3. VFC view.

5.1 Views

The performance of VFC4FPS is mostly associated to the views specified. Measurements were made using three different views: normal view, zoom view and view with only one FoV (VFC view). The normal view (Table 1) contains the appropriate parameters to the normal view of the avatar. In the case of Cube 2: Sauerbraten, an avatar has a 100° field of view. Besides the 100°, in this view we see a field of view of 140°; this is to avoid momentary inconsistencies when doing a quick spin. These 140° represent 20° either side of the normal field of view.

The zoom view (Table 2) contains the parameters used in the case where the avatar has the zoom activated. In this case, the field of view is reduced from 100° to 35°. The zones's radius were doubled in relation to the normal view. Again, the field of view has 20° to each side, which results in the 75° angle. The *kappa* vectors remain the same as in the normal view.

The VFC view (Table 3) is equal to the normal view, but contains only one field of view that covers all 360° around the avatar. This view is used to provide a comparison between the VFC4FPS and the original VFC, and prove that the inclusion of FoVs leads to better performance.

5.2 Traffic Evaluation

To measure the reduction of traffic generated, we resorted to the use of 48 bots on different maps. The choice of the number of bots is linked to the normal number of players normally seen in online servers (which is around 24 players). Since the aim is to reduce traffic by half, it was decided to double the number of "players" (bots) to determine if the gains were close to the traffic generated by 24 players.

All measurements were made in game mode ffa (free for all, with various types of weapons) in order to have more events of shooting and better demonstrate the impact of the event filter. Measurements were recorded only from the time

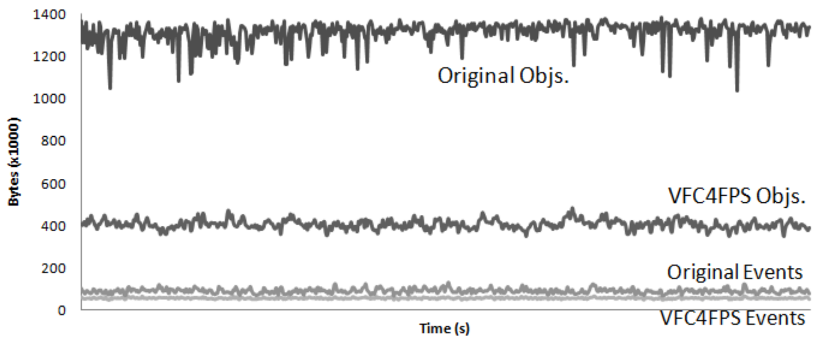

Fig. 1. Server outbound traffic during 10 minutes in the flagstone map using the normal view.

when all 48 bots were connected to the server. The duration of each test was 10 minutes. This value was chosen to obtain stable average values.

A server has inbound and outbound traffic. The inbound traffic is low and constant. The highest traffic is outbound. Thus, we only present results regarding the server's outgoing traffic.

Figure 1 presents the results using the normal view: the rate of outgoing traffic in bytes per second for the two communications channels for the two solutions (Original and VFC4FPS). As can be seen, the traffic rates are stable over time. We can observe a reduction of more than half on the network traffic for object updates. In addition, we can also observe the low traffic generated by events. The small difference between the original events and the VFC4FPS events is due to the filtering applied.

Using the zoom view, the results are very similar to the normal view, proving that increasing the radius of the zones along with the reduction of field of view did not impair the performance of VFC4FPS.

In order to justify the VFC4FPS changes, we did the same measurements for the VFC view.

Table 4 presents the average values for the ratio between the original game network traffic and the VFC4FPS traffic for different views and maps. There is a trend for the ratio to increase with the size of the maps. This is justified by the possibility of avatars to be more dispersed than in small maps. Numerically, we can conclude that the reduction achieved through the normal view and the zoom view is equal. We can also conclude that without the inclusion of field of view, the reduction is lower.

5.3 Qualitative Evaluation

In order to evaluate if our use of VFC4FPS did not harm the game playability, we conducted a test with real players. Each one plays two versions of the game:

Map		**Views**		
Name	**Dimension**	**Normal**	**Zoom**	**VFC**
aard3c	250x250	1,7		
academy	250x250	1,5		
aqueducts	875x750	1,7		
arabic	875x750	2		
akroseum	1000x1000	2,2		
dust2	1000x1000	2,5		
campo	1000x1000	1,7		
venice	1000x1000	2,5		
wdcd	1000x1000	2,6	2,6	1,6
redemption	1250x500	2,3		
core_transfer	1250x750	2,7		
face-capture	1500x250	2,5		
damnation	1500x500	2,1		
shipwreck	1500x1250	3		
flagstone	1500x1500	3	3	1,8
hallo	1500x1500	2,7		
ph-capture	1500x1500	2,7		
river_c	1500x1500	2,9		
mach2	1750x750	2,9		
urban_c	2250x1750	2,8	2,7	1,7

Table 4. Average ratios after 10 minutes.

the original game, and the game using VFC4FPS. This test was done in a blind fashion, i.e. players did not know which version was which.

Tests were made one player at a time, against 48 bots. Tests were conducted in the urban_c map in the instagib mode (where one shot kills). Each player started by playing 5 minutes in one version to practice. After that, he play for 10 minutes in the same version, after which it was asked for him to switch versions and play for another 10 minutes. The starting version (the practice one) was alternated between players. When the player ended playing both versions, he answered a short questionnaire. The main question was if he noticed differences between the two versions, mainly related to the opponent's movement.

Tests were conducted with 16 volunteers. Their average age was 25 years. Only 3 of the volunteers answered that they encountered differences between the two versions, one of which favoured the VFC4FPS version. This means 14 favourable answers towards VFC4FPS. Most players had some experience with FPS and didn't saw any differences. Most players used the zoom function, and, despite the higher vision range, that did not affect their perception between the two versions.

6 Conclusions

First Person Shooters have a fast paced action where the precision of the opponents movements is very important. When played online, communication is required to be very frequent to keep the players updated. This translates into a large network traffic, especially outbound traffic in the server in a Client-Server architecture. The high use of bandwidth is the main limitation of the number of players supported.

In this paper we present VFC4FPS, a library to manage the consistency of FPS. The VFC4FPS is based on the VFC model, which allows a progressive and controlled consistency decay. Our system exploits the context of the limited view of the avatars in a FPS and adds to the VFC the concept of fields of view.

VFC4FPS was applied to the open source game Cube 2: Sauerbraten with the aim of reducing, at least by half, the use of bandwidth on the server, thus offering support to a greater number of players. We evaluated the performance of Cube 2: Sauerbraten with VFC4FPS in several maps, and the results show that the goal of reducing by half the network traffic has been reached and in many cases exceeded. This is was achieved without harming the gameplay compared to the original Cube 2: Sauerbraten.

References

1. R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Trans. Database Syst.*, 15(3):359–384, 1990.
2. C. E. Bezerra, F. R. Cecin, and C. F. R. Geyer. A3: A novel interest management algorithm for distributed simulations of mmogs. In *DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 35–42, Washington, DC, USA, 2008. IEEE Computer Society.
3. J.-S. Boulanger, J. Kienzle, and C. Verbrugge. Comparing interest management algorithms for massively multiplayer games. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 6, New York, NY, USA, 2006. ACM.
4. E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 67–73, New York, NY, USA, 2002. ACM.
5. J. Dyck. A survey of application-layer networking techniques for real-time distributed groupware. Technical report.
6. J. Dyck, C. Gutwin, T. C. N. Graham, and D. Pinelle. Beyond the lan: techniques from network games for improving groupware performance. In *GROUP '07: Proceedings of the 2007 international ACM conference on Supporting group work*, pages 291–300, New York, NY, USA, 2007. ACM.
7. S. Fiedler, M. Wallner, and M. Weber. A communication architecture for massive multiplayer games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 14–22, New York, NY, USA, 2002. ACM.

8. T. A. Funkhouser. Ring: a client-server system for multi-user virtual environments. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 85–ff., New York, NY, USA, 1995. ACM.
9. R. Krishna Balan, M. Ebling, P. Castro, and A. Misra. Matrix: adaptive middleware for distributed multiplayer games. In *Middleware '05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, pages 390–400, New York, NY, USA, 2005. Springer-Verlag New York, Inc.
10. J. Pang. Scaling peer-to-peer games in low-bandwidth environments. In *In Proc. 6th Intl. Workshop on Peer-to-Peer Systems IPTPS*, 2007.
11. L. Pantel and L. C. Wolf. On the impact of delay on real-time multiplayer games. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29, New York, NY, USA, 2002. ACM.
12. L. Pantel and L. C. Wolf. On the suitability of dead reckoning schemes for games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 79–84, New York, NY, USA, 2002. ACM.
13. Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.
14. N. Santos, L. Veiga, and P. Ferreira. Vector-field consistency for ad-hoc gaming. In *Middleware '07: Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, pages 80–100, New York, NY, USA, 2007. Springer-Verlag New York, Inc.
15. J. Smed, T. Kaukoranta, and H. Hakonen. A review on networking and multiplayer computer games. In *Multiplayer Computer Games, Proc. Int. Conf. on Application and Development of Computer Games in the 21st century*, pages 1–5, 2002.
16. S. Xiang-bin, W. Yue, L. Qiang, D. Ling, and L. Fang. An interest management mechanism based on n-tree. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD '08. Ninth ACIS International Conference on*, pages 917–922, Aug. 2008.
17. H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, 20(3):239–282, 2002.