MULTIPLAYER GAME CHEATING PREVENTION WITH PIPELINED LOCKSTEP PROTOCOL

Ho Lee, Eric Kozlowski, Scott Lenker, Sugih Jamin
Electrical Engineering and Computer Science Department
University of Michigan, Ann Arbor
{leehz, ekozlow, slenker, jamin}@eecs.umich.edu

Abstract Multiplayer games are becoming more and more popular. One of the reasons is that they let geographically distant players play together. We first look at a synchronization protocol and a protocol for prevention of common cheating techniques in real-time multiplayer games. We then propose an optimization to the current cheat-proofing techniques which takes advantage of synchronization delay inherent in current implementation of multiplayer games. Previous work has studied synchronization techniques [1] and cheat-proofing techniques [2] in isolation, without exploring their interaction. Our optimization has the same guarantee as current cheat-proofing techniques and can increase interaction rate between players in real-time multiplayer games.

1. Multiplayer Games

Systems can use a centralized or distributed approach for communication in multiplayer games. A centralized approach, where all game clients use a single server, provides simplicity in synchronization because all of the communication takes place through a central point. Many current multiplayer games use the approach where one player configures a machine as a server, and other players connect to it.

To increase the scalability of multiplayer games, a distributed approach can be used. The distributed approach requires each client to independently compute the game state. To make sure players correctly compute the same game state, the system needs synchronization support. The distributed approach provides more opportunities for players to cheat since there is no central authority controlling the game play.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35660-0_65
R. Nakatsu et al. (eds.), *Entertainment Computing*

Our work in this paper focuses on the distributed approach to multiplayer games.

As distributed architectures for multiplayer games become more popular, much work is being done in the areas of synchronization, cheat-proofing and scalability. Below we look at techniques currently used in each of these areas. Then we propose an optimization to enhance the interactivity of the current cheat-proofing techniques.

1.1. Network Delay and Bucket Synchronization

The distributed approach requires each client to independently compute the game state. However, due to network delay, players may have different views of the game. For example, in a first person shooter game, player A, at time t, shoots at coordinate (2, 3) which is the location of player B according to her view. But just right before time t, player B moves from (2,3) to (3,3). Due to network delay, player A does not see player B's move until after time t. The view of player A is that her shot was successful. However, the view of player B is player A missed her shot. This leads to an inconsistent game state.

The above example illustrates the need for systems to have synchronization support to ensure that players' views are consistent. Consistency means that actions by all players are processed at the same time and that all sessions display their game states simultaneously. Currently, the main synchronization scheme used is bucket synchronization [1].

Bucket synchronization breaks time up into segments, and represents each segment with a bucket. As other players' moves packets are received, in the form of network packets, they are inserted into their associated bucket according to their timestamps. Moves occur at the game's virtual time and are delayed for a set time before being processed. This allows for packets to be sent across the network and placed into appropriate buckets along with all other players' actions issued for that time segment. The length of each segment is dependent on the frame-rate the game can handle. Once the current time reaches that of a bucket, all of the packets are taken out of the bucket and processed into the game state.

1.2. Cheating and Cheating Techniques

The distributed approach provides more opportunities for players to cheat since there is no central authority controlling the game play. Baughman and Levine presented two common methods of cheating for real-time multiplayer games: lookahead cheat and suppress-correct cheat [2]. The lookahead cheat takes advantage of the interaction resolution required

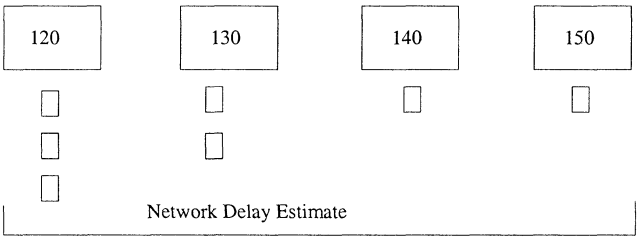

Figure 1. Bucket synchronization representation with time segment length of 10ms.

at each discrete unit of time (per round, or per frame). A player can cheat by waiting until all other players have sent their decisions and then calculate its move based on the moves received. In games that use bucket synchronization, lookahead cheat can be in the form of modifying packets' timestamps. Under the bucket synchronization scheme, a received packet does not update the game state immediately, but is instead put into a bucket until the bucket's time arrived. A cheating player can claim that an event has happened earlier than it actually has by putting an earlier timestamp in the packet. For example, a cheating player, on seeing a lethal shot at bucket time t, can immediately send out a packet that claims she has moved to another location at bucket time $t-1$. In this way, she can prevent other players from successfully killing her.

Suppress-correct cheat takes advantage of dead-reckoning, which is a technique used to accommodate packet loss and network transmission delay by allowing a game client to estimate the state of other players based on their recent vectors when their packets are not received on time. For example, when a position packet is not received on time, the position of the corresponding player can be estimated by its velocity, calculated based on previous moves, and the time since the last move. A cheater cheating through suppress-correct cheat can purposefully not send packets to reveal its current state, and then uses knowledge of the current states of other players to construct an update packet for a future move, which provides the player with an advantage. Suppress-correct cheat is possible only when dead-reckoning is used. The protocols presented in this paper and in [2] do not allow for dead-reckoning.

1.3. Cheat-proofing Protocols and Their Limitation

Baughman and Levine proposed two cheat-proofing protocols: the lockstep (LS) protocol and its optimization, the asynchronous synchronization (AS) protocol [2].

In distributed games without any cheat-proofing guarantees, players send their moves to other players without any specific constraints. This makes it difficult to prevent cheating. Under the LS protocol, each player first sends a cryptographically secure one-way hash of her decision as a commitment and waits for other players' commitments. After receiving other players' commitments, each player then reveals her decision in plain text. Players can verify if other players' moves are valid by calculating a hash from the plain text and then comparing the result with the commitment. As a result of this requirement, dead reckoning is not allowed in the LS protocol. Since the LS protocol involves sending two packets for each move and waiting for other players' commitments, it requires longer latency for each move.[1] To increase the event generation rate, the AS protocol was proposed and a notion of Zone of Control (ZoC)[2] was used. A player's ZoC is defined as the area of the game in which the player can possibly be affected by another player in the next turn, and therefore resolution with other players' decisions is required [4]. Under the AS protocol, a pair of players communicate with each other using the LS protocol only when they are within each other's ZoC. When they are outside of each other's ZoC, they could send plain texts to each other.

The major limitation of the LS protocol is that the interaction between players is limited by the worst network delay between players. The event generation rate is not faster than 1/(worst one-way network delay). Even with the AS protocol, the interaction between players is still limited by the slowest player within the ZoC. This is because the event generation rate of each client is not faster than 1/(worst one-way network delay between players in each other's ZoC). Our proposed optimization reduces the dependency of the event generation rate on network delay.

[1] Baughman and Levine observed that a player who has received all other players' hashes but has not sent out her own hash can skip sending her hash and just send the plain text [2].

[2] The authors of paper [2] used the term Sphere of Influence to mean ZoC. We decided to adopt the terminology ZoC used in wargaming [4] to differentiate it from LoS used in Section 3.2.

2. Pipelined Lockstep Protocol

We have observed that for games that do not require interaction resolution at each discrete unit of turn, the sending of a commitment may be independent from the sending of the plain texts corresponding to previous commitments. This means that, as long as there is no conflict between the local player's new decision and other players' pending plain texts, the local player can send the hash of this new decision ahead of her pending plain texts waiting to be sent. We call this optimization Pipelined Lockstep (PLS) Protocol. Let us take a general first person shooter game as an example. In such a game, we observed two possible cases that could cause an irresolvable interaction problem.[3] The first case is that no two players can occupy the same location at the same time. Therefore, if there are 2 players, 7 units apart, running the LS protocol, then we can allow each player to send out at most 3 hashes and have 3 pending plain texts without having to worry about any possibility in causing this case of irresolvable interaction problem. Another case occurs when a player successfully shoots another player. For example, one player moves to location (2,3) at bucket time t, and at the same time another player shoots at location (2,3). The first player is supposed to die and leave the game after this event is displayed at bucket time t. However, due to network and synchronization delays, the first player does not see the event immediately and may send future moves to the second player. After both players have processed the packets corresponding to bucket time t, they see the future moves of the first player. Both players can resolve this problem easily by ignoring all the packets generated by the first player after bucket time t. The PLS protocol can eliminate the possibility of this irresolvable interaction problem because the packets have not been displayed yet.

Thus the PLS protocol takes advantage of the presentation delay that bucket synchronization adds to the display of the packets. This presentation delay helps prevent certain potential irresolvable interaction problems as we have mentioned above. Baughman and Levine used the assumption that packets are displayed immediately once they are received or generated [2]. However, we have observed that this constraint can be loosened for a large number of real-time multiplayer games, such as Age of Empires [3].

[3]Baughman and Levine defined *irresolvable interaction* as a problem that results when dead reckoning is used and interactions are either determined unfairly by a server or potentially incorrectly by a distributed host [2].

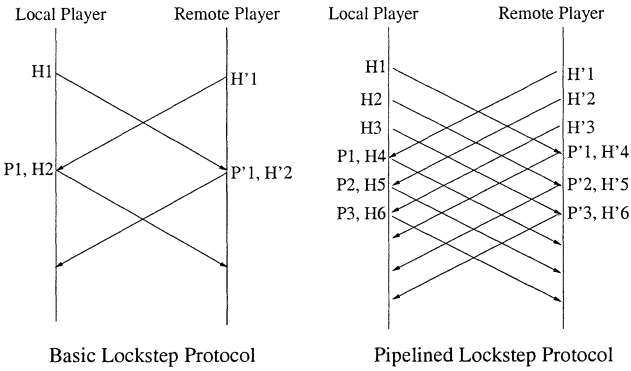

Figure 2. Two diagrams showing the difference between the basic lockstep protocol and the pipelined lockstep protocol. In this diagram, the pipe size is 3. (For these two diagrams, we assume that there are only two players involved in the lockstep protocol and that there is no conlict between these turns.)

The PLS protocol is a generalized version of the LS protocol. How fast players can interact with each other depends on the size of the pipe between them. For games that require interaction resolution at each discrete unit of turn, the pipe size is one, which is the basic LS protocol.

The PLS protocol has the same assumptions as the LS protocol: there is a reliable communication channel between all players; players have knowledge of the existence of all other players; each player can authenticate messages from each other player. Under these assumptions, the PLS protocol is safe [2] and it provides liveness [2], as does the basic LS protocol. The PLS protocol is safe, meaning that no players can receive the state of another player before the game rules permit. The PLS protocol is live, meaning that the timestamps of the packets each player resolves advance monotonically with wall-clock time. The PLS protocol can be used along with the definition of ZoC without any conflict. When the PLS protocol is used along with ZoC, the amount of traffic can be cut down, events can be generated at a faster rate and at the same time the same guarantee as the basic LS protocol is provided.

Figure 2 shows the case for only two players. When there are more then two players running the PLS protocol, each player should make sure that a plain text is sent only after the receipts of all hashes of the same turn from all players. This is to make sure that players cannot cheat through cooperation with each other. For example, if player A sends her plain text to player B before the receipt of player C's hash, player B, on receiving the plain text could send it to player C and give player C advantage.

Sometimes packets may not be received before their bucket time due to long network delay. Since the PLS protocol does not allow dead reckoning, the way to handle late packets is to freeze the bucket until the late packets arrive. Bucket synchronization and the LS protocol do not affect the correctness of each other. With the support of the LS protocol or the PLS protocol, we can prevent lookahead cheating or cheating through timestamp modification that could happen in games that use bucket synchronization.

3. Interest Management to Improve Scalability

To scale games to hundreds of players, clients must minimize bandwidth usage. Because of this, interest management has been proposed. Interest management means that players only need to receive information that is useful for their decision making. The goal of interest management is to reduce the amount of traffic that each client sends and receives.

There are mainly two types of interest management: multiple multicast group, and ZoC that depends on proximity to other players. Figure 3 gives a general view of interest management.

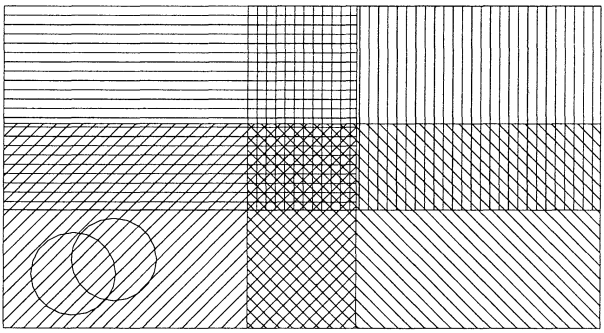

Figure 3. A diagram showing how a game board can be broken down. The large outer rectangle is the global multicast group. Then the board is broken down into four other multicast groups. Players are a member of one or more sub-groups if they are near a boundary. The circles represent two playes who have entered each others' Zone of Control which extends for a small radius out from each of them.

3.1. Multiple Multicast Groups

Let us take a first person shooter game to explain the use of multiple multicast groups. A global multicast group is used for players to communicate with global information such as heartbeats, join messages, and death messages. The game board can be broken up into overlapping

multicast groups to provide more efficient communication. Each multicast group overlaps the others so that players can detect other players near the boundary. Each multicast group can be used for players within the same group to send position information. Players that are not in the same multicast group do not need to know about the positions of the other players.

3.2. Zone of Control and Line of Sight

A player's Zone of Control (ZoC) is defined as the area of the game in which a player can possibly be affected by another player in the next turn, and therefore resolution with other players' decisions is required. A player's Line of Sight (LoS) is defined as the area of the game that the player can see. In a first person shooter game, the ZoC could be defined as the maximum shooting distance. LoS includes ZoC and has to be greater than ZoC so that players can know other players are approaching. To cut down the amount of traffic, players that are not within each other's ZoC may not need to send their shooting packets to one another.

maximum area

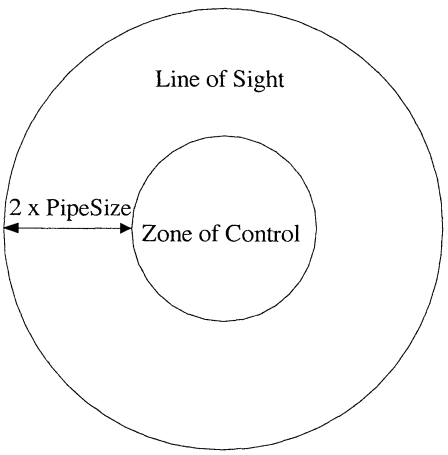

Figure 4. A diagram showing how the Line of Sight is defined under the Pipelined Lockstep Protocol

For a 2D game board, it is useful to define ZoC and LoS as two concentric circles, the inner circle being ZoC and the outer circle being LoS. For players outside the LoS circle, no PLS protocol is required. For players inside the ZoC circle, the PLS protocol is mandatory. The area between the two circles serves as a buffer zone in which players can

become aware of each other's presence and initiates PLS, as described below.

The definition of ZoC does not depend on the pipe size of the PLS protocol. However, the radius difference of LoS to ZoC should be twice the pipe size of the PLS protocol. The pipe size represents the number of packets on the way to other players. A player should start running the PLS protocol when she enters another player's LoS to make sure that when she enters the player's ZoC, both players process packets that have been negotiated through the PLS protocol. The reason the radii difference is twice the pipe size is that, in the worst case, the two players approach each other.

When ZoC and LoS are used together with multiple multicast groups, the amount of overlap between adjacent multicast groups must be at least equal to the radius of LoS. This is to ensure that a player can see approaching players and be able to start running the PLS protocol when the other players enter her LoS.

4. Conclusion

The current cheat-proofing techniques have the drawback of their dependency on network delay. We have looked into the interaction of these cheat-proofing techniques with bucket synchronization and proposed an optimization, the PLS protocol, which allows players to interact at a faster rate, independent of network delay. Our proposal allows faster interaction between players while at the same time provides the same guarantee as the current synchronization and cheat-proofing techniques. Our proposal can be applied to commercial games.

The use of interest management is useful in reducing the bandwidth usage and thus increasing the scalability of multiplayer games. However, when the PLS protocol is supported, the definition of LoS changes according to the pipe size in the PLS protocol.

References

[1] Gautier, L. and Diot, C. *A Distributed Architecture for Multiplayer Interactive Applications on the Internet. IEEE Network*, July/August 1999, pp. 6-15.

[2] Baughman, N. and Levine, B. *Cheat-Proof Playout for Centralized and Distributed Online Games. Proc. IEEE Infocom*, April 2001.

[3] Bettner, P. and Terrano, M. *'1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. Proc. of the Game Developers Conference*, 2001, http://www.gdconf.com/archives/proceedings/2001/terrano_1500arch.doc

[4] Dunnigan, J.F. *Wargames Handbook: How to Play and Design Commercial and Professional Wargames.* 3rd ed., Writers Club Press, Dec. 2000.