<table>
<thead>
<tr>
<th>項目</th>
<th>内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Minimization of latency in cheat-proof real-time gaming by trusting time-stamp servers</td>
</tr>
<tr>
<td>Author(s)</td>
<td>MOGAKI, Shunsuke / KAMADA, Masaru / YONEKURA, Tatsuhiro</td>
</tr>
<tr>
<td>Citation</td>
<td>Proceedings of international conference on cyber worlds 2007, 1: 343-346</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2007</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10109/912">http://hdl.handle.net/10109/912</a></td>
</tr>
</tbody>
</table>

このリポジトリに収録されているコンテンツの著作権は、それぞれの著作権者に帰属します。引用、転載、複製等される場合は、著作権法を遵守してください。
Minimization of Latency in Cheat-Proof Real-Time Gaming by Trusting Time-Stamp Servers

Shunsuke Mogaki Masaru Kamada Tatsuhiro Yonekura
Department of Computer and Information Sciences
Ibaraki University
Hitachi, Ibaraki 316-8511, Japan
07nm721n@hcs.ibaraki.ac.jp, {kamada, yone}@mx.ibaraki.ac.jp

Abstract

Communication latency with the lockstep protocol is minimized under the assumptions that 1) time-stamp servers are available near each player that issue serially numbered time stamps and that 2) there is no communication break down between the player and the time-stamp server. Those assumptions are relatively moderate since the time-stamping service is becoming the fundamental infrastructure of the cyberworlds. Each player sends its own action to the other player and also sends its hash to the nearest time-stamp server. The time-stamp server sends back to the player the signed hash with time and a serial number involved. The signature is an undeniable evidence of the action. The actions are checked if they are compatible with the hashes and the signed hashes are checked if they have the correct time and the serial numbers are contiguous. This verification can be done as a batch after the game is finished. While the original lockstep protocol suffers from the latency as much as the round-trip time of packets and the other existing improved protocols alleviate the latency at the cost of lower security, the proposed protocol has the latency as little as the packet travelling time from one player to another and has the same security as the original under the assumption that we trust the time-stamp servers.

Keywords: real-time network gaming, cheat-proofing, time-stamping service.

1 Introduction

Cyberworlds enable people to play a game even if they are remote in the real world. But the network latency and the possible packet loss between the remote players give an excuse to allow for cheating.

The bit commitment protocol [1] is an established cryptographic technique that provides a perfectly fair environment for static games like coin-tossing. The tosser and teller conceal a bit representing face or tail by a hash function and exchange the hash values, called commitments, as undeniable evidences. After receiving the evidence from the counterpart, they unveil what the concealed bit is. The players verify the unveiled information against the evidences. This protocol is completely cheat-free and has been applied to mental poker games [2], where the players play cards in turn.

The lockstep synchronization [3] is the first application of bit commitment to real-time games. A lockstep is simply a round of the turns in which the players exchange their commitments and unveil their actions. The actions are made in synchronization to the game clock. A problem with this protocol is that a player can stop the game clock to gain time to think for the next action by withholding his commitment as long as he likes. Injection of fair random noises to the actions [4] [5] has been shown to be an effective method to deter this kind of cheat. But those protocols are basically not quite real-time.

The advanced protocols [6] [7] are based on a fundamental assumption that the players take actions in synchronization to the real-world clock. They have been devised to alleviate the problem that a lockstep takes the round-trip time of communications between the players. The pipelined lockstep protocol [6] implements a pipelined exchange of commitments and actions. The larger pipeline size makes the players feel the latency shorter. At the same time, the larger pipeline size makes it easier to cheat since there is no means to verify if a player acts really in synchronization to the real-world clock ticks. The sliding pipeline protocol [7] assumes that every player can measure the traveling time of the commitments from the other player in order for adaptive pipeline size. This assumption on the players allows for another means of cheat. A player can report earlier departure time of commitments.

In order to force the players to act really synchronized
to the real-world clock, it seems to be the only way that we prepare distributed and synchronized supervisors near the players. The fair synchronization protocol [8] is based on this kind of supervisors called pulsers. Each player sends its action to the nearest pulser. The pulser sends the action in an encrypted form to the other player, who can decrypt it after receiving the key from the nearest pulser. In this way, the pulsers control the times when the actions are decided and the times when they are disclosed to the other players. The distributed pulsers, however, constitute a trusted third party that the players have to believe in.

It is essential for fair real-time gaming to keep verifiable the times when the players commit actions. For that purpose, a time-stamping service suffices, which is a common service in the cyberworld [9]. Like the pulsers, we may distribute time-stamp servers so that every player has a time-stamp server in its neighborhood.

In this paper, a cheat-proof protocol for real-time gaming is proposed that shortens the latency without any loss of security under the assumptions that 1) time-stamp servers are available near each player that issue serially numbered time stamps and that 2) there is no communication break down between the player and the time-stamp server.

2 Review of Existing Protocols

2.1 Lockstep Protocol

For simplicity, the following protocols are summarized in the two-player setting although they also work for three players or more.

In synchronization to the game clock $i$, $(i = 1, 2, 3, \cdots)$, each player sends a commitment of his action and wait for the one from the counterpart before unveiling the action. The detailed procedure is illustrated in Fig. 1 and performed as follows:

0. The players agree on a common hash function $H$.
1. Players A and B decide their actions $A_i$ and $B_i$ respectively at time $t_i$ of the game clock, respectively, and exchange the hashes $H(A_i)$ and $H(B_i)$.
2. On receiving the hash from the other player, the players unveil their actions $A_i$ and $B_i$ to each other.

The above steps 1 and 2 are iterated during the game session.

A lockstep takes the round-trip time between the players. So the game clock can tick only slowly by a time interval longer than $2l$ for any $i$, where $l$ denotes the packet travelling time from a player to another.

Figure 1. Lockstep Protocol

2.2 Pipelined Lockstep Protocol

The real-time axis is divided into uniform frames $[t_{i}, t_{i+1})$, $(t_{i} = i\Delta t, i = 1, 2, 3, \cdots)$. Players are assumed to take an action at the time $t_i$. The locksteps are pipelined in order to make the frame interval $\Delta t = t_{i+1} - t_i$ shorter than the network latency $l$. The detailed procedure is illustrated in Fig. 2 and performed as follows:

1. Given the pipeline size $p$, the players agree on the frame interval $\Delta t$ such that $\Delta t \geq \frac{l}{p}$ on a common hash function $H$.
2. For $i = 1, 2, \cdots, p$, players A and B decide their action $A_i$ and $B_i$ respectively at time $t_i$ and send their hashes $H(A_i)$ and $H(B_i)$ to each other.
3. For $i > p$, player A decides the action $A_i$ and send the new hash $H(A_i)$ along with the unveiled action $A_{i-p}$ as soon as A receives the hash $H(B_{i-p})$ from player B. Player B also performs the same.

Figure 2. Pipelined Lockstep Protocol (for the case $p = 2$)

The frame interval is $\frac{1}{p}$ shorter than the original lockstep protocol. In this sense, players may feel the latency shorter. Techniques like dead reckoning may compensate better the
effect due to latency with the faster frame rate although the unveiled action arrives by the same latency $2l$.

This protocol lacks a means to verify if a player is really sending the hash and the unveiled action at time $t_i$. If player A is malicious and B is honest, A may receive $B_{i-p}$ and $H(A_i)$ at $t_i$, but B may not receive $A_{i-p}$ and $H(A_i)$ until $t_{i+1}$. Then A takes advantage of the information $B_{i-2p+1}, B_{i-2p+2}, \ldots, B_{i-p}$ to decide his action $A_i$ while B does not know $A_{i-2p+1}, A_{i-2p+2}, \ldots, A_{i-p}$. In this situation, player B must wish to appeal. But B cannot give any evidences for the late arrival of $B_{i-2p+1}, B_{i-2p+2}, \ldots, B_{i-p}$. Player A may blame a varying network condition to insist his innocence. That is a new kind of cheat that the pipelining brought into the original lockstep protocol.

2.3 Sliding Pipeline Protocol

Cheat by the intensional delay may be detected if we know the network latency. The sliding pipeline protocol incorporates a means to estimate the generally variable network latency.

The real-time axis is divided into nonuniform frames $[t_i, t_{i+1})$, $(\Delta_i = t_{i+1} - t_i, i = 1, 2, 3, \ldots)$. By exchanging several packets, player A and B estimate the network latency as $l^A_i$ and $l^B_i$, respectively. Then they exchange the estimates to decide the initial latency $l_i = \max(l^A_i, l^B_i)$.

The detailed procedure is illustrated in Fig. 3 and performed as follows:

1. Given the initial pipeline size $p_1$, the players agree on the frame interval $\Delta t_i$ such that $\Delta t_i \geq \frac{l_i}{p_i}$ on a common hash function $H$.

2. For $i = 1, 2, \ldots, p_1$, players A and B decide their action $A_i$ and $B_i$ respectively at time $t_i$ and send their hashes $H(A_i)$ and $H(B_i)$ to each other.

3. For $i > p_1$, player A decides the action $A_i$ and send the new hash $H(A_i)$ along with the unveiled action $A_{i-p_i}$ and new estimated network latency $l^A_{i-p_i}$ as soon as A receives the hash $H(B_{i-p_i})$ from player B. Player B also performs the same. The updated latency will be $l_{i-p_i} = \max(l^A_{i-p_i}, l^B_{i-p_i})$, which will be given to players A and B. Once the latency is updated, the pipeline size will be changed to $p_i$ according to $l_{i-p_i}$ and the frame interval $\Delta t_i$ such that $\Delta t_i \geq \frac{l_{i-p_i}}{p_i}$.

The frame interval is $\frac{\Delta t_i}{p_i}$ shorter than the original lockstep protocol. Dead reckoning may perform better if $p_i$ is large.

By the estimated latency $l_i$, the players may be able to detect possibly intensional delay of communications from the counterpart. But the latency is estimated by trusting the departure time of packets which can be tampered by the sender. So the sliding technique does not provide an essential remedy for cheat with respect to time.

2.4 Fair Synchronization Protocol

The fair synchronization protocol defines distributed supervisors named pulsers that control the times when the actions are decided and the times when they are disclosed to the other players. Each player is supervised and supported by a pulser in its neighborhood so that the latency is minimized.

The communications are made in a pipeline to shorten the frame interval. The pulsers decide a variable pipeline size $p_i$ by the network latency so that all the players may receive actions made at $t_i$ by the time $t_{i+p_i}$.

The detailed procedure is illustrated in Fig. 4 and performed as follows:

1. Players A and B decide their actions $A_i$ and $B_i$ at $t_i$. Player A send $A_i$ to the nearest pulser. Player B also does the same.

2. A’s pulser encrypts $A_i$ into $E_i(A_i)$ and send this $E_i(A_i)$ to player B. B’s pulser also sends $E_i(B_i)$ to A.

3. Player A decrypt $E_i(B_i)$ to know $B_i$ after receiving the decryption key $D_i$ from A’s pulser at $t_{i+p_i}$. Player B does the same.

The minimum frame interval is the round trip time of packets between player and pulser, which is very small because pulsers exist near each player. A major drawback is that the pulsers constitute a trusted third party which all the players have to believe in.
3 Proposed Protocol

3.1 Assumptions

The preceding two pipelined protocols share the same assumption that the players decide their actions in synchronization with the real-world clock tick. But they lack a means to verify if the players really respect the assumption. It is rather moderate to trust a third party in some senses than this strong assumption on the honesty of players.

It is a brave step made by the fair synchronization protocol to assume ubiquitously distributed supervisors. But the supervisors named pulsers to support the fair synchronization protocol collect more information than necessary. We have to believe in the pulsers.

It is essential for fair real-time gaming to supervise the players only with respect to the times when they commit actions. For that purpose, a time-stamping service suffices, which is a common service in the cyberworld. Like the pulsers, we may distribute time-stamp servers so that every player has a time-stamp server or more in its neighborhood.

In accordance with the RFC3161 protocol [10], the time business accreditation center, for example in Japan, authorizes time-stamping companies. Time-stamp servers are synchronized to the national time authority, which checks from time to time if they are really synchronized. The maximum allowable error is 10msec.

We shall trust the time stamp servers only with respect to their honesty in time stamping. Then we can force the players to decide the action in synchronization to the real-world clock tick \( t_i = i \Delta t \). More precisely, we shall assume the following.

1. Time-stamp servers are available near each player that issue signed hashes with time and serial numbers involved.

2. There is no communication break down between the player and the nearest time-stamp server. The packet traveling time from a player to the nearest time-stamp server is so small as \( \varepsilon \).

3.2 Procedure

The proposed protocol is a modification of the original lockstep protocol in a different sense other than the pipelined and sliding pipeline protocols. In the lockstep protocol, each player sends a commitment of the action to the other player. In our protocol, each player sends the commitment to the nearest time-stamp server. At the same time, the player sends its own action to the other player. The time-stamp server sends back to the player the signed hash that hash contains signing time and serial number \( i \) of the lockstep.

The signature is an undeniable evidence of the action. The players exchange all the signed hashes to verify the actions made in the game.

The detailed procedure is illustrated in Fig. 5 and performed as follows:

1. Players A and B decide their action \( A_i \) and \( B_i \) respectively at time \( t_i \) and send it to the other player and they send their hashes \( H(A_i) \) and \( H(B_i) \) to the nearest time-stamp servers.

2. The time-stamp server sends back to the player the signed hash \( T(H(A_i), t_i^A, i) \) and \( T(H(B_i), t_i^B, i) \) containing the signing time \( t_i^A \) and the serial number \( i \), where \( t_i^A \) and \( t_i^B \) represents the time-stamp servers received \( H(A_i) \) and \( H(B_i) \), respectively.

The above steps 1 and 2 are iterated concurrently during the game.
3.3 Latency and Frame Interval

In the proposed protocol, each player sends its own action directly to the other player. So the latency is minimized to the limit, the packet travelling time from one player to another.

The minimum frame interval is round-trip time \(2\varepsilon\) of packets between player and time-stamp server, which is very small under our assumption that time-stamp servers exist near each player.

3.4 Security

The players exchange all the signed hashes for the verification of their actions made in the game by the counterpart. This verification can be done as a batch after the end of the game. It can also be made by another process running concurrently after the main gaming process if the players wish to detect cheats sooner.

Player A verifies \(B_i\) against \(H(B_i)\) and verifies the signature \(T(H(B_i), t_i^B, i)\) against \(H(B_i), t_i^B\) and \(i\). If \(t_i^B \gg t_i^B + \varepsilon\), A can accuse B of the late commitment.

It is possible that player A commits an action correctly but delays intentionally its shipping. In that case, player B has to decide \(B_{i+n}\) (for large \(n\)) without knowing \(A_i\). Player B is encouraged to take a countermeasure by delaying its packets, too. Then the game may be blind shooting. But the players stay in an equal condition.

4 Conclusion

Employing the time-stamp servers as an infrastructure that we trust in cyberworlds, we made a cheat-proof protocol for real-time gaming that has the shortest latency without any loss of security compared to the original lockstep protocol.

Acknowledgment

This work was partially supported by the JSPS Grant-In-Aid no.18300027 and Osamu Miyamoto foundation of the Ibaraki University VBL.

References


