Time-Stamp Service makes Real-Time Gaming Cheat-Free

Shunsuke Mogaki
Ibaraki University
4-12-1, Nakanarusawa
Hitachi, Ibaraki 316-8511,
Japan
+81-294-38-5142
07nm721n@hcs.ibaraki.ac.jp

Masaru Kamada
Ibaraki University
4-12-1, Nakanarusawa
Hitachi, Ibaraki 316-8511,
Japan
+81-294-38-5142
kamada@mx.ibaraki.ac.jp

Tatsuhiro Yonekura
Ibaraki University
4-12-1, Nakanarusawa
Hitachi, Ibaraki 316-8511,
Japan
+81-294-38-5142
yone@mx.ibaraki.ac.jp

Shusuke Okamoto
Seikei University
Musashino, Tokyo 180-8633
Japan
okam@st.seikei.ac.jp

Yasuhiro Ohtaki
Ibaraki University
4-12-1, Nakanarusawa
Hitachi, Ibaraki 316-8511,
Japan
+81-294-38-5142
y.ohtaki@mx.ibaraki.ac.jp

Mamun Bin Ibne Reaz
Department of Electrical and
Computer Engineering,
International Islamic University
Malaysia, Jalan Gombak,
53100 Kuala Lumpur,
Malaysia
mamun.reaz@iiu.edu.my

ABSTRACT

Assuming time-stamp servers that we can trust exist everywhere in the Internet, we propose a cheat-proof protocol for real-time gaming that has the minimum latency. The assumptions are: 1) Time-stamp servers are available near each player that issue serially numbered time stamps. 2) There is no communication break down between the player and the nearest time-stamp server. By this protocol, 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 if the serial numbers are contiguous. This verification can be done as a batch after the game is finished. The latency in this protocol is only the packet traveling time from a player to another.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol verification*

General Terms

Security, Legal Aspects

NetGames'07, September 19-20, 2007, Melbourne, Australia.

Keywords

real-time network gaming, cheat-proofing, time-stamp service

1. INTRODUCTION

The time-stamp service is becoming a fundamental infrastructure of the Internet. It guarantees that the data existed at the time of stamping and that the data have not been tampered since then. Without trusting the time-stamp servers, it is not possible to have such a fundamental service as electronic notary over the Internet [1]. In Japan, for example, the time business accreditation center [2] 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.

The existing cheat-proof protocols for netgames have been made on the peer-to-peer setting and do not assume any trusted third parties. It is even considered as a scientific "cheat" and thus prohibited by people working on cryptographic protocols to assume a trusted third party. The lockstep protocol [3] is the most successful cryptographic protocol to realize a fair environment for gaming without trusting any third parties. It is a pity that the lockstep is not tied to the real world clock but to the game clock. The game clock can be stopped by any player. So gaming by the lockstep protocol is not quite real time.

In attempt to make a real time version of the lockstep protocol, several authors have proposed its modifications under the assumptions that the players decide their actions in synchronization to the real world clock and that the network latency is known. But players can easily cheat each other by reporting a longer latency than the real one.

It may be the time when we conclude that one can never make a cheat-proof protocol for real time gaming without assuming some trusted time keepers. In this paper, a cheat-proof protocol for real-time gaming is proposed that has the minimum latency under the minimum assumption that time-stamp servers are everywhere in the Internet and that

we trust them with respect to their job in time-stamping.

2. PROPOSED PROTOCOL

2.1 Assumptions

The time stamp servers will be trusted in order to force the players take actions synchronized to the real world clock $t_i = i\Delta t$, $(i = 1,2,3,\cdots)$. More precisely, following assumption will be taken:

1. Time-stamp servers are available near each player that issue signed hashes, which includes data, time and serial numbers.

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 will be as small as ε.

2.2 Procedure

In the proposed protocol, each player sends their own action to the nearest time-stamp server and the other player simultaneously. The time-stamp server sends back the signed hash to the respective player, which contains data, signing time and serial number i. 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. 1 and performed as follows:

1. Player A and B decide their action A_i and B_i respectively at time t_i. At the same time, they send their hashes $H(A_i)$ and $H(B_i)$ to the nearest time-stamp servers.

2. The time-stamp server sends back the signed hash $T(H(A_i), t_i^A, i)$ and $T(H(B_i), t_i^B, i)$ to the player A and B respectively, which contains data, the signing time t_i^A and the serial number i. t_i^A and t_i^B represents the time-stamp server's received time of $H(A_i)$ and $H(B_i)$, respectively.

Steps 1 and 2 are iterated concurrently during the game.

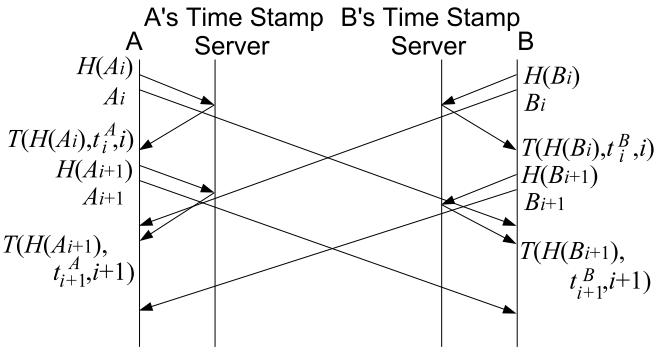

Figure 1: Proposed Protocol

2.3 Security

The actions are hashed so that the time stamp servers obtain no information about the actions. Players' privacy is protected in this sense.

A time stamp server may give retroactive stamps or sign several possible actions with the same serial number in favor of a specific player. In either case, the proposed protocol collapses. So does the electronic notary service. That is the justification of placing our trust on the time stamp service.

The players exchange their signed hashes for the verification of actions made during the game to identify possible cheats. This verification can also be done as a batch after the end of gaming. Player A verifies B_i against $H(B_i)$ and also verifies the signature $T(H(B_i), t_i^B, i)$ against $H(B_i)$, t_i^B and i. If $t_i^B \gg t_i + \varepsilon$, A can accuse B for the cheat of delayed action.

It is possible that player A commits an action correctly but delays intensionally its shipping. In that case, player B has to decide B_{i+n}(for large n) without knowing A_i. If player B receives A_i later than expected, Player B is encouraged to take a countermeasure by delaying the disclosure of its action, too. Then the game may be blind shooting. But the players stay in an equal condition. Besides, including this countermeasure in the protocol, we can deter players to attempt the intentional delay because a cheater does not gain any advantages while the gaming becomes uncomfortable.

2.4 Latency and Frame Rate

In the proposed protocol, each player sends his own action directly to the other player. Therefore, the latency is only the packet traveling time l from one player to another. No one can make this shorter so that the proposed protocol has the minimum latency.

The maximum frame rate can be as fast as $\frac{1}{2\varepsilon}$, the reciprocal of the round-trip time 2ε of packets between a player and the nearest time-stamp server. This ε is very small under the assumption that time-stamp servers exist near each player.

3. COMPARISON

The following is a comparison made on the latency, frame rate and security of the proposed protocol with typical cheat-proof protocols, such as lockstep protocol, pipelined lockstep protocol [4] and sliding pipeline protocol [5].

3.1 Lockstep Protocol

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. It should be noted that the game clock is not tied to the real world clock.

The detailed procedure is illustrated in Fig. 1 and performed as follows:

1. The players agree on a common hash function H.

2. Players A and B decide their actions A_i and B_i at time i of the game clock respectively, and exchange the hashes $H(A_i)$ and $H(B_i)$ as the commitment of actions.

3. Upon receiving the hash from the counterpart, the players unveil their actions A_i and B_i to each other.

Steps 2 and 3 are iterated during the game session.

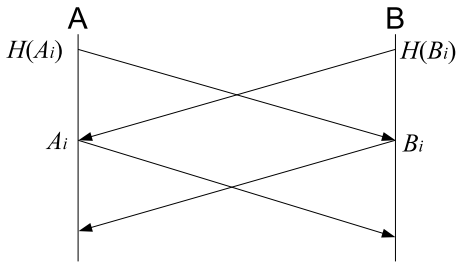

Figure 2: Lockstep Protocol

In the lockstep protocol, it takes the round trip time $2l$ of the packets between the players for an action of a player to reach the other. On the other hand, the proposed protocol brings the action only by the latency l. This protocol works at the frame rate $\frac{1}{2l}$ while the proposed protocol can achieve the frame rate as fast as $\frac{1}{2\varepsilon}$.

The actions taken by the players are unknown to each other until the actions are committed. Therefore this protocol is completely cheat-free which is also valid for the proposed protocol as well. But the game clock remains stopped until players complete exchanging their commitments or actions. A malicious player can gain time to think for the next action as long as he likes simply by withholding its decision.

3.2 Pipelined Lockstep Protocol

That is true of the case we interchange A and B. In the pipelined lockstep protocol, players are assumed to take an action at the time $t_i = i\Delta t$ of the real world clock. 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 Δt 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.

In the pipelined lockstep protocol, the latency remains the same as the lockstep protocol. But players do not wait for commitments from the other player like the lockstep protocol. The frame rate is $2p$-times faster than the original lockstep protocol. Using the dead reckoning techniques may give a faster game experience even though the latency is as large as $2l$. The maximum frame rate is $\frac{p}{l}$ which is much slower that $\frac{1}{2\varepsilon}$ of the proposed protocol.

If player A is malicious and B is honest, A will receive the unveiled action B_{i-p} and new hash $H(B_i)$ at t_i but B will not receive A_{i-p} and $H(A_i)$ until t_{i+p}. Therefore, A can take the advantage of the information $B_{i-2p+1}, B_{i-2p+2}, \cdots, B_{i-p}$ to decide his next action A_i while B is unaware about the information $A_{i-2p+1}, A_{i-2p+2}, \cdots, A_{i-p}$. In this situation, player B has right to appeal but unable to put any evidences for the late arrival of B_{i-2p+1}, B_{i-2p+2}, $\cdots$,B_{i-p}. Thus, player A can able to insist his innocence by blaming the varying network condition. The above explanation proves that the pipelined lockstep protocol is not a cheat-free protocol.

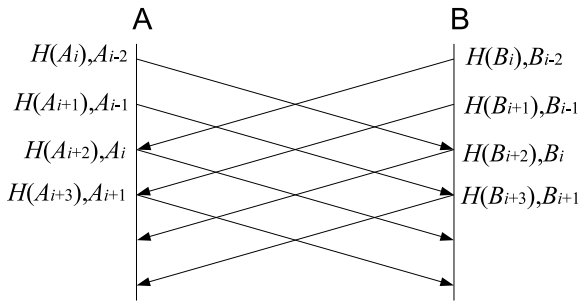

Figure 3: Pipelined Lockstep Protocol (for the case $p = 2$)

3.3 Sliding Pipeline Protocol

Cheat by the intentional delay may be detected if the network latency is known. The sliding pipeline protocol incorporates a means to estimate the variable network latency.

The players are assumed to take an action at nonuniform instances t_i, where the frame interval $\Delta t_i = t_{i+1} - t_i$ is variable. By exchanging several packets, player A and B estimate the network latency as l_i^A and l_i^B, respectively. Then they exchange the estimated network latency to decide the initial latency $l_i = \max(l_i^A, l_i^b)$.

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 Δt_1 such that $\Delta t_1 \geq \frac{l_1}{p_1}$ on a common hash function H.

2. For $i = 1,2,\cdots,p_1$, 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 $l_{i-p_i}^A$ 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 $l_{i-p_i} = \max(l_{i-p_i}^A, l_{i-p_i}^B)$, 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 Δt_i such that $\Delta t_i \geq \frac{l_{i-p_i}}{p_i}$.

In the sliding pipeline protocol, the latency remains the same as the lockstep protocol. But the frame rate is $2p_i$-times faster than the original lockstep protocol. This protocol may cope with the variable network congestion. The dead reckoning techniques will give a faster game experience if p_i is larger. The maximum frame rate is $\frac{p_i}{l_{i-p}}$. which may be still slower than $\frac{1}{2\varepsilon}$.

By the estimated latency l_i, the players may be able to detect possible intentional delay. But the network latency is

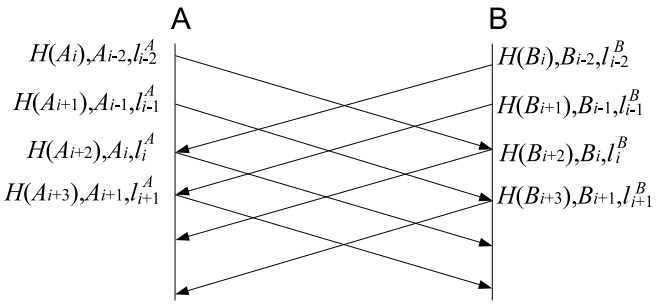

Figure 4: Sliding Pipeline Protocol (for the case $p = 2$)

estimated by trusting the departure time of packets, which can be tampered by the sender. Therefore, the sliding technique does not provide an essential remedy for cheat with respect to time. The above explanation proves that the sliding pipeline protocol is not a cheat-free protocol.

4. CONCLUSION

Employing the time-stamp servers as an infrastructure, the proposed protocol proves itself a cheat-free protocol for real-time gaming, which achieves the minimum latency without any loss of security.

5. ACKNOWLEDGMENT

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

6. REFERENCES

[1] C. Adams, P. Cain, D. Pinkas and R. Zuccherato. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), *RFC 3161*, August 2001.

[2] Time business accrediation center. Accrediation program for time-stamping services. http://www.dekyo.or.jp/tb/english/index.html. 2007.

[3] N. E. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. *Proc. IEEE INFOCOM 2001*, U.S.A, pages 104–113, April 2001.

[4] H. Lee, S. Lenker, E. Kozlowski, and S. Jamin. Synchronization and cheat-proofing protocol for multiplayer games. *Playing with the Future: Development and Direction in Computer Gaming*, April 2002. http://les1.man.ac.uk/cric/gamez/abstracts/lee.html

[5] E. Cronin, B. Filstrup, and S. Jamin. Cheat-proofing dead reckoned multiplayer games. *Proc. Int. Conf. Appl. Devel. Compu. Games(ADCOG) 2003*, Hong Kong, January 2003.