Online Game Protocol for P2P Using Byzantine Agreement

Daisuke Wada, Junichiro Kitagawa and Hiromi Kobayashi
Graduate School of Engineering, Tokai University, 1117, Hiratsuka, 259-1292, Japan
7adrm030@mail.tokai-u.jp, koba@tokai.ac.jp

Abstract

Peer to peer (P2P) online game systems have garnered attention recently. This paper presents an online game protocol based on the lockstep protocol and the Byzantine agreement algorithm. The former prevents dishonest actions called time-cheats in a peer to peer (P2P) network. The latter disables Byzantine cheats, with actions resembling the Byzantine fault. Using this protocol, a game can be continued if a player performs a cheat action.

1. Introduction

Nowadays, many online games have been developed along with the progress of the Internet. These games usually use a central server to manage games. However, the maximum permissible number of accesses limits the number of clients. In addition, game development is delayed because of the capacity of transactions that must be processed by a server. Dishonest actions are facilitated by delayed communication among players in an online game using time-cheats such as the lookahead cheat and the suppress-correct cheat [1–3]. The lookahead cheat is a cheat that gains advantage by being the last player to make what are ostensibly simultaneous decisions. The suppress-correct cheat is a cheat by which a cheater intentionally renders the communication speed or transaction speed as slower to put an opponent at a disadvantage. The protocol described herein uses distributed processing by completely sharing the load of a central server's processing to clients (i.e. peers) to solve this problem [1–6]. We use pure peer to peer (P2P) systems in which every peer communicates without a central server, thereby avoiding heavy traffic on the central server. A mechanism to eliminate dishonest actions must be included in a pure P2P online game system because of the lack of a central server for managing a game. We propose an online game protocol to prevent the suppress-correct cheat in pure P2P systems. This protocol consists of the combination of the lockstep protocol [1–3] and the interactive consensus algorithm [8] based on the Byzantine agreement algorithm [7–10]. The lockstep protocol is a fundamental protocol to prevent the lookahead cheat using synchronization of plays in every peer. The Byzantine consensus algorithm is a fault tolerant algorithm against the Byzantine fault that is the most generalized fault for a distributed system. Players can continue an online game even if one player performs a dishonest act.

2. Description of Protocol

We first describe the lockstep protocol and the Byzantine agreement algorithm included in our protocol.

2.1. Lockstep Protocol

The lockstep protocol is a stop-and-wait type protocol with a decision commitment step. This protocol makes it possible to prevent the lookahead cheat and the suppress-correct cheat using synchronization of peer's actions, called lockstep synchronization, shown below. Each player first sends a one-way hash of action data to all other players. Each player replies to the first confirmation message as a commitment to all other players when receiving hashed data from all other players. After receiving the confirmation messages from all other players, each player sends action data to all other players. Each player replies to the second confirmation message as a commitment to all other players and makes hashes of the received action data applying the hash function that was used in the first phase when receiving the action data from all other players. Subsequently, each player compares these hashes with the corresponding hash that was received in the first phase. If the compared hashes are equivalent, the comparison proves that no cheat, such as the lookahead cheat and the suppress-correct cheat, was conducted in that turn. Consequently, each player can proceed to the next turn of this game. The lookahead cheat and the suppress-correct cheat are prevented using the two-phase protocol described above. This procedure is depicted as a sequential diagram in Fig. 1. A two-player game is assumed for Fig. 1. Each player first makes a hash, represented as H1 and H2, from action data represented as Data1 and Data2 using makeHash( ) in the first phase. Subsequently, players exchange these hash messages mutually using Send( ). After receiving H1 and H2, each player respectively sends back a confirmation message. Next, players send Data1 and Data2 to each other using Send( ) in the second phase. After receiving Data1 and Data2, each player sends back a confirmation message and makes a hash statement represented as H2' and H1' using makeHash( ) from Data2 and Data1 that was received in the second phase. Each player compares H1 and H1', H2 and H2'. Equivalent hashes given first and second verify that no cheat such as the lookahead cheat or suppress-correct cheat was conducted.

2.2. Byzantine Agreement Algorithm

The Byzantine agreement (or, consensus) algorithm is a fault tolerant algorithm that deals with the most generalized fault called a Byzantine fault in distributed systems. A player sends an inconsistency message to other players simultaneously in the case of the Byzantine fault. Honest players can make agreements in spite of the existence of the Byzantine faulty player using the Byzantine agreement algorithm. We apply this algorithm for distributed online game by substituting a dishonest player, called a Byzantine cheat, for a Byzantine faulty peer. Let n be the number of players and m be the number of faulty (i.e. dishonest) players then $n \geq 3m+1$ for making an agreement using the Byzantine agreement algorithm. We show an example of this algorithm called the oral message algorithm (OM) as follows, where OM is a recursive algorithm. The value of majority $(v_1, ..., v_{n-1})$ is the majority value among v_i if it exists, otherwise it takes the value *retreat* in this algorithm [7].

Algorithm OM(0)

(1) The commander sends a chosen value to every lieutenant.

(2) Each lieutenant uses the value received from the commander, or uses the value *retreat* if receiving no value.

Algorithm OM(*m*), *m*>0

(1) The commander sends a chosen value to every lieutenant.

(2) For each i, let v_i be the value that Lieutenant i receives from the commander, or be *retreat* if receiving no value. Lieutenant i acts as the commander in Algorithm OM(m-1) to send the value v_i to each of the n-2 other lieutenants.

(3) For each i and for each j≠i, let v_i be the value that Lieutenant i received from Lieutenant j in step (2) (using Algorithm OM(m-1)), or else *retreat* if receiving no such value. Lieutenant i uses the value majority $(v_1, ..., v_{n-1})$.

The remaining players make it possible to find an agreement using majority function if one of the lieutenants (children peers) is a dishonest player.

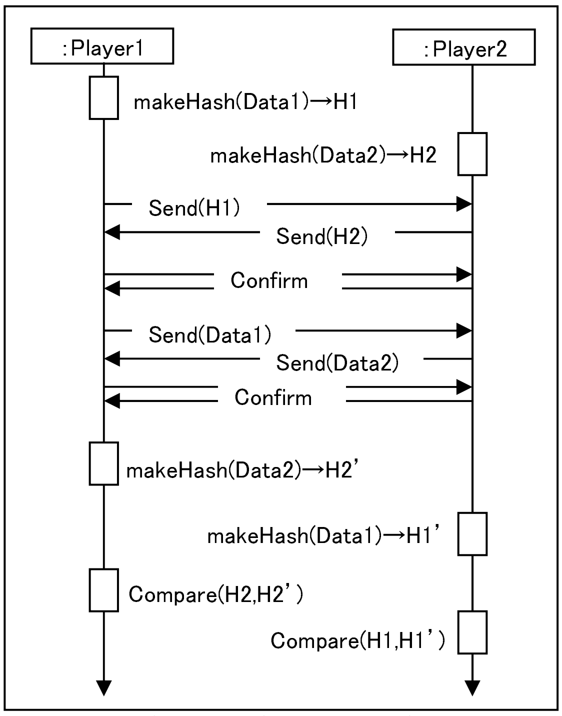


Fig. 1. Lockstep protocol.

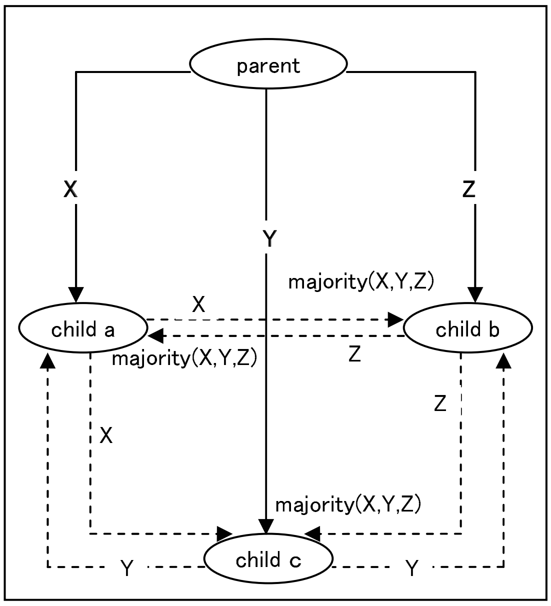


Fig. 2. Byzantine agreement algorithm.
(the behavior in OM(1))

Even if the commander (parent peer) is a dishonest player who sends different values to other players lieutenants (children peers), children peers make it possible to reach agreement using the majority function. The number of allowances of dishonest players is m, such that $n \geq 3m+1$. Figure 2 portrays the behavior of OM(1) for n=4 and m=1 in the case in which a commander (parent peer) is a dishonest peer to send respective values X, Y, and Z to other lieutenants (children peers). Every lieutenant makes an agreement using the same majority value using majority (X, Y, Z).

3. Game Model

3.1. Assumption of an Online Game

We assume an online game on the pure P2P distributed systems. In this game, every player advances play with exchanged status in a certain interval time. As an example, a kind of Real-Time Strategy (RTS) game is assumed. The features of this game are that players give instructions to characters such as soldiers to defeat an enemy. Every player takes a sequence of actions in accordance with the real time progress of a game. This feature differs from a strategy game having distinct alternate turns of players. Each player must send status to other players based on an action and situation to implement the RTS game on pure P2P systems. Every player takes actions according to the status. We propose a protocol based on the lockstep protocol and the Byzantine agreement algorithm. The former is to prevent time-cheats such as the lookahead cheat and the suppress-correct cheat caused by synchronization delay or dead reckoning. The latter is to continue a game if a dishonest player sends different status to other players for gaining an advantage or confusing the game. We deal with the following dishonest actions in our protocol.

(1) Timing cheats include the suppress-correct cheat and the lookahead cheat: One gains an advantage by being the last player, in cases of simultaneous actions, to make a decision based on look actions of other players.

(2) Byzantine cheat, by which transmissions of different data are used to gain advantage or confuse a game by sending different data to other players.

3.2. Protocol

We show our protocol as presented in Fig. 3. Preliminary to a game, participant peers (i.e. players) must be fixed and a parent peer must be decided. As an example, a parent peer can be made by ascendant order of peer's node ID number. This parent peer is tentative, so the parent peer changes with the advance of a game. Here, $n \geq 3m+1$, where the number of participant peers is n and the number of dishonest peers is m. In the first phase, a parent peer makes a hash message of its action data and sends this hashed action data to every child peer. After receiving hashed data from a parent peer, each child peer sends this hashed data to other children. Each child peer decides its value using majority when receiving the hashed data from all other children.

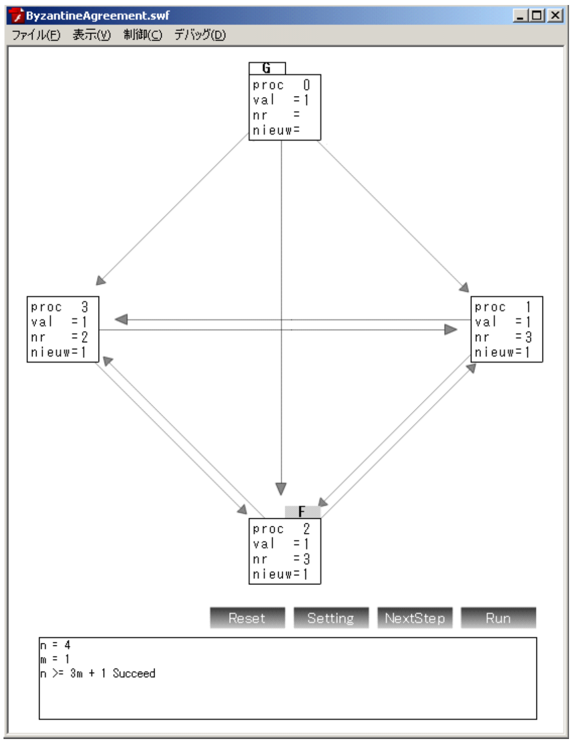
```
m: the number of maximum tolerant dishonest peers
$x_i$: action data at peer i in a game ( i ∈{1,⋯,n} )
$x_{def}$: default value of an action data representing *no move*
$H(x_i)$: hash value of $x_i$

*while* ( end of a game )
  *@each peer i*
    make $H(x_i)$
    send $H(x_i)$ using OM(m, $H(x_i)$)
    wait ack message from all children peers

    send $x_i$ using OM(m, $x_i$)
    wait ack message from all children peers

    *for* ( j = 1 to n , where i<>j )
      make $H'(x_j)$ from received action data $x_j$
      compare( $H(x_j)$, $H'(x_j)$ )
      *if* ( $H(x_j)$ = = $H'(x_j)$ ) use $x_j$ as an action data of peer j
      *else* $x_j = x_{def}$   /* $x_j$ is disregarded */
      *endif*
    *endfor*
*endwhile*   /* go on to the next step of this game */

OM(0, x)   /* Byzantine general agreement */
  *@parent peer*    send x to every child peer
  *@each child peer i*, where i = 1 to n
    *if* ( no value received ) $v_i = x_{def}$
    *else*    $v_i$ = received value from a parent peer as x
    *endif*
    send ack message to a parent peer
OM(m, x)
  *@parent peer*    send x to every child peer
  *@each child peer i*, where i = 1 to n
    *if* ( no value received ) $v_i = x_{def}$
    *else*   $v_i$ = received value sent from a parent peer as x
    *endif*
    send $v_i$ to each other child peer
              as if to be a parent peer using OM(m−1, $v_i$)
  *@each child peer i*, where i = 1 to n and i<>j
    *if* ( no value received ) $v_j = x_{def}$
    *else*   $v_j$ = received value from child peer j
                            in OM(m−1,$v_i$) shown above
    *endif*
    x = *majority*($v_1$, … , $v_{n-1}$)
    send ack message to a parent peer
```

Fig. 3. Proposed protocol

This child peer then maintains a log as a dishonest action if a hashed datum is not equivalent to other hashed data in this decision by majority. Next, a parent peer shifts to the next peer and the same procedure is repeated until each participant peer has become a parent peer. The procedure described above is designated as the interactive consensus algorithm composed of multiple Byzantine agreement algorithms. Every peer becomes a parent peer orderly in the interactive consensus algorithm.
However, the protocol mentioned above becomes more efficient when the process of OM at each peer conducts in parallel and the received data from other peers are vectorized at each peer [7].
In the second phase, a parent peer sends its action data to every child peer. After receiving action data from a parent peer, each child peer sends this action data to the other children. Each child peer decides its value using the majority when the action data are received from all other children. This child peer maintains a log as a dishonest action if an action datum is not equivalent to other action data in this decision by a majority. Next, a parent peer shifts to the next peer and the same procedure is repeated until each participant peer has become a parent peer.
In the final phase, each peer makes a hash message for use in the first phase from other peers and compares the hashed data with hashed data received in the first phase. Time cheats are therefore shown. An action of a dishonest peer is neglected by other honest peers if compared data show a discrepancy. Therefore, a dishonest peer might be at a disadvantage. In addition, a dishonest peer acting according to the Byzantine cheat can be eliminated from a game using logs mentioned above as follows. *Peer i* sends warning message w_{ij} to every other peer if a disagreement values coming from *peer j*. When *peer k* has received w_{ij}, peer k writes it to a log and makes a decision by a majority, where let $w_{ij}=1$ if *peer k* has received w_{ij} at least once. Each peer makes a decision from a majority of warning messages to exclude *peer j* from the game.

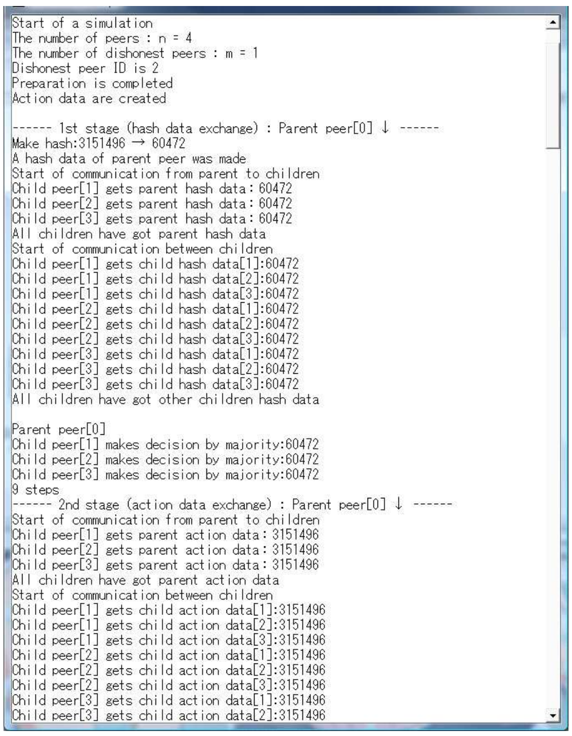
4. Simulation and Evaluation

We developed two simulation programs using to verify the protocol described above. Figure 4 is a screenshot for checking the Byzantine agreement part of a simulation using Flash and ActionScript.
The number of peers is n=4 and the number of dishonest peers m=1. G denotes general(commander) and F denotes dishonest player. *proc, val, nr*, and *nieuw* represent peer identification number, received value from G, the number of received 1s in each majority, and the majority value of each node, respectively. We use binary value {0, 1} for simplicity in this simulation. Figure 5 shows the beginning of a total simulation process of our



```
n = 4
m = 1
n >= 3m + 1 Succeed
```

Fig. 4. Byzantine simulation screenshot
(n = 4, m = 1)



```
Start of a simulation
The number of peers : n = 4
The number of dishonest peers : m = 1
Dishonest peer ID is 2
Preparation is completed
Action data are created

------ 1st stage (hash data exchange) : Parent peer[0] ↓ ------
Make hash:3151496 → 60472
A hash data of parent peer was made
Start of communication from parent to children
Child peer[1] gets parent hash data: 60472
Child peer[2] gets parent hash data: 60472
Child peer[3] gets parent hash data: 60472
All children have got parent hash data
Start of communication between children
Child peer[1] gets child hash data[1]:60472
Child peer[1] gets child hash data[2]:60472
Child peer[1] gets child hash data[3]:60472
Child peer[2] gets child hash data[1]:60472
Child peer[2] gets child hash data[2]:60472
Child peer[2] gets child hash data[3]:60472
Child peer[3] gets child hash data[1]:60472
Child peer[3] gets child hash data[2]:60472
Child peer[3] gets child hash data[3]:60472
All children have got other children hash data

Parent peer[0]
Child peer[1] makes decision by majority:60472
Child peer[2] makes decision by majority:60472
Child peer[3] makes decision by majority:60472
9 steps
------ 2nd stage (action data exchange) : Parent peer[0] ↓ ------
Start of communication from parent to children
Child peer[1] gets parent action data: 3151496
Child peer[2] gets parent action data: 3151496
Child peer[3] gets parent action data: 3151496
All children have got parent action data
Start of communication between children
Child peer[1] gets child action data[1]:3151496
Child peer[1] gets child action data[2]:3151496
Child peer[1] gets child action data[3]:3151496
Child peer[2] gets child action data[1]:3151496
Child peer[2] gets child action data[2]:3151496
Child peer[2] gets child action data[3]:3151496
Child peer[3] gets child action data[1]:3151496
Child peer[3] gets child action data[2]:3151496
```

Fig. 5. Simulation process of our protocol

TABLE 1 The number of peers and messages.

peers	messages
4	72
25	28800
46	186300
67	583704
88	1332144

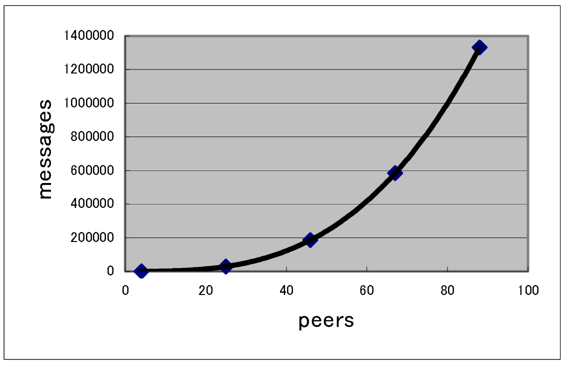


Fig. 6. The number of peers vs. messages.

protocol presented in Fig. 3. This simulation program is made using Java.

Next, we evaluate the number of messages in this protocol. Theoretically, the number of messages between children peers in the Byzantine agreement algorithm is $(n-1)\cdot(n-2)\cdots(n-k-1)$, $n \geq 3m+1$, where the number of peers is n and the number of Byzantine faulty peers is m. However, the number of peers is n=10 at most (n=4 in most RTS games). Therefore, the number of dishonest peers might be m=1 in RTS game at present. The number of messages between children peers is $(n-1)\cdot(n-2)$ in this case. Furthermore, the number of messages from a parent peer to children peers is n-1. Considering the interactive consensus problem in which every peer becomes a parent peer, the above total number of messages is

$n\cdot((n-1)+(n-1)\cdot(n-2)) = n\cdot(n-1)^2$.

Moreover, considering the lockstep protocol, the number of messages is $2n(n-1)^2 = O(n^3)$.

Table 1 and Figure 6 show numbers of our simulation results with the number of messages against the number of peers in an RTS game.

The curved line in Fig. 6 expresses the third order that is fit to the result of this simulation. Therefore, this simulation result is suitable to the theoretical result described above.

5. Related Work

The lockstep protocol was proposed in [1-2]. It requires that all players advance their game clock synchronously. Cronin et al. [2] have presented the pipelined lockstep protocol for efficiency of throughput. In addition, they have proposed the adaptive pipeline protocol in order to mask the jitter and provide a constant frame rate to the players. These protocols are efficient against several time-cheats. Baughman and Levine [1] have addressed the problem of dead reckoning that compensates for variable communication latency and loss across a network. They have proposed a new synchronization technique with guarantee fair playout called asynchronous synchronization. This technique relaxes the lockstep synchronization problem by decentralizing the clock of a game. In addition, they have proposed the player spheres of influence in order to improve advancement of a game by restricting the region of synchronization. The above work is the improvement of the lockstep protocol.

Blum [11], Reyneri and Karnin [12] have presented the problem and the protocol of two parties who wish to agree on the result of a coin toss through telephone under the circumstance that these two parties do not trust each other. The notion of their protocol resembles the lockstep protocol; however, the number of players is limited to two and time-cheats are not considered explicitly. A protocol of two dishonest players playing a fair game of poker without using any card was presented in the work of Shamir, Rivest and Adleman [13]. They use cryptography in order to solve this problem. Their work deal with a game in fully-distributed environment; however, their protocol based on one to one communication and time-cheats are not considered, too. Yoshimoto et al. [4] have proposed a secure protocol using cryptography for peer to peer online game. Their method is based on one to one communication, too. The characteristic of their method is that each player has a game simulator to verify the data of opponent players. Secure multi-party computation was investigated by Yao [14].

In contrast to these works, our protocol does not use cryptography because the objective of above work is different from our protocol.

Many studies [15-20] have been conducted to reduce the amount of messages for the scalability of the number of peers as to the Byzantine agreement algorithm. However, we concentrate our attention to combine the Byzantine agreement algorithm with the lockstep protocol in this paper. We consider to reduce the amount of messages using more scalable algorithm mentioned above at the next step of this study. However, the maximum number of peers is n=10 in most games; therefore, the amount of messages is not increased extremely at present.

6. Discussion and Concluding Remarks

We present an online game protocol that acts in a pure P2P computer network. This protocol enables us to prevent time-cheats including the suppress-correct cheat and the lookahead cheat, and a dishonest action based on the Byzantine fault. A game can be continued without a disadvantage to honest players if a player conducts cheat actions in this protocol.

Discussion might be needed about the following issues. First, how can we play a game over upper bound of dishonest peers? The number of peers n and the upper bound number of dishonest peers m are set at the beginning of a game when using the Byzantine agreement algorithm. If the number of dishonest peers is greater than this upper bound: $m_{\max} = \lfloor (n-1)/3 \rfloor$ (e.g., m_{max}=1 at n=4, m_{max}=2 at n=7), an agreement is not completed in this algorithm. However, do we intend to play a game under such circumstance as existence of relative numerous dishonest players compared with n ? Actually, the number of peers is n=10 at most (n=4 in most RTS games) at present.

We add the following another solution of this issue. Besides the players, other peers are added for making a decision by majority in the Byzantine agreement algorithm. As an example, when the number of players is 6 in a game, the upper bound of m is 1; however, if another peer is added in this algorithm, the upper bound of m becomes 2; if four peers are added, the upper bound of m becomes 3. One of the purposes in this study is how to avoid heavy traffic of a central server under playing multiple online games. Our solution is to use total peer to peer environment. The traffic is increased by using the lockstep protocol and the Byzantine agreement algorithm for preventing dishonest actions of peers. Especially, the traffic is increased with the number of dishonest peers in the Byzantine algorithm. However, the number of dishonest peers might be m=1 in RTS game at present. Therefore, the traffic is not increased extremely. The next issue is that participant peers must be fixed in our protocol; in contrast, participant peers join freely in peer to peer environment. However, the players must be fixed in a certain duration time in order to hold most games. The third issue is how long time lag is permitted in this protocol. We consider that a peer who responds no answer in a certain period of time might be regarded as a dishonest peer in an actual game. The default value representing no action might be taken in an action game. This problem occurs not only in a peer to peer environment but in a central server environment. More discussion is needed about this issue.

We have been developing a prototype of an action game system to implement our protocol. We use JXTA(ver.2.3.7) [21-22] as a platform of peer to peer environment and Java to develop an application. Further work is necessary to examine how to give a hash function that is suitable to this protocol to every peer and how to reduce the number of messages in this protocol. Additionally, we must study ways to impose penalties on dishonest peers in a game.

References

[1] N.E. Baughman and B.N. Levine, "Cheat-proof playout for centralized and distributed online games," *Proc. IEEE INFOCOM2001*, pp.104-113,2001.

[2] E. Cronin, B. Filstrup and S. Jamin, "Cheat-proofing dead reckoned multiplayer games," *Proc. 2nd Int'l conference on ADCOG*, 2003.

[3] C. GauthierDickey, D. Zappala ,V. Lo and J. Marr, "Low latency and cheat-proof event ordering for peer-to-peer games," *Proc. NOSSDAV'04*, 2004.

[4] H. Yoshimoto, R. Shigetomi and H. Imai, "How to protect peer-to-peer online games from cheats," *Proc. IEEE CIG2005*, 2005.

[5] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems,* Cambridge University Press, 2008.

[6] S. Ghosh, *Distributed Systems: An Algorithmic Approach*, Chapman & Hall/CRC, 2007.

[7] L. Lamport, R. Shostak and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, 4, 3, pp.382-401, 1982.

[8] M. Pease, R. Shostak and L. Lamport, "Reaching agreement in the presence of faults," *JACM*, 27, 2, pp.228-234, 1980.

[9] M. Barborak, M. Malek and A. Dahbura, "The Consensus problem in fault-tolerant computing," *ACM Computing Surveys*, 25,2, pp.171-220, 1993.

[10] G. Tel, *Introduction to Distributed Algorithms: Second Edition*, Cambridge University Press, 2000.

[11] M. Blum, " Coin flipping by telephone," *Proc. IEEE Workshop on Communications Security (Crypto-81),* pp.23-26, 1981.

[12] J. M. Reyneri and E. D. Karnin, "Coin flipping by telephone," *IEEE Trans. Information Theory,* IT-30,5, pp.775-776, 1984.

[13] A. Shamir, R. L. Rivest and L. M. Adleman, " Mental poker", *Technical Report LCS/TR-125*, MIT, April 1979.

[14] A. C. Yao, " How to generate and exchange secrets," *Proc. IEEE Symp. on Foundation of Computer Science,* pp.162-167, 1986.

[15] B. Baum-Waidner, "Byzantine agreement with a minimum number of messages both in the faultless and worst case, *Proc. 23rd Int'l Symposium on Fault-Tolerant Computing (FTCS23)*, pp.554-563, 1993.

[16] P. Berman and J. Gray, "Randomized distributed agreement revisited," *Proc. 23rd Int'l Symposium on Fault-Tolerant Computing (FTSC23)*, pp.412-419, 1993.

[17] P. Berman and J. Gray, "Efficient distributed consensus with n=(3+epsilon)t processors," *Proc. 5th Int'l Workshop on Distributed Algorithms*," no.129-142, 1991.

[18] G. Bracha, "An asynchronous [(n-1/3)]-resilient consensus protocol,'' *Proc. ACM Conference on the Principles of Distributed Computing (PODC'84)*, pp.154-162, 1984.

[19] R. Canetti and T. Rabin, "Fast asynchronous Byzantine agreement with optimal resilience," *Proc. ACM Symposium on Theory of Computing (STOC'93)*, pp.42-51, 1993.

[20] C. S. Lewis and J. Saia, "Scalable Byzantine Agreement," *NIPS2003 Workshop on Robust Communication Dynamics in Computer Networks*, 2003.

[21] B. J. Wilson, " *JXTA*," New Riders Publishing, 2002.

[22] JXTA Homepage:http://www.jxta.org/