JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 8, NO. 6, JUNE 2005(pp. 798-806)

A History Retransmission Algorithm for Online Arcade Video Games

Seong-Hoo Kim†, Kyoo-Seok Park††

ABSTRACT

In this paper, we suggest a game system that can support network modules for multi-platform based video games, and built a system that can convert from a single-user game to multi-user game. In this system, we bring in an initial delay buffering scheme on clients to handle any periods of latency occurring from the load fluctuation in a network, when a real-time game is played, and shows that stable play for a game is achieved as the result of the scheme. This paper also presents a retransmission algorithm based on the history of game commands to handle drawbacks of UDP mechanism. And, we evaluate the network delay and packet loss using the simulation tool NS2, and shows the case of 0.3 second buffer delay is the most suitable for recovery.

Keywords: Network Game, Arcade Video Game, Synchronization, Mutiplayer, Real-time Game, Latency, History Packet

1. INTRODUCTION

The Current video games and arcade games are for the most part played on various emulators supported by personal computers as a single-user mode. Gaming industries try to provide online mode playing of single-user mode games, but only limited number of games are served in the form of online mode[1,2].

For complete network games, have to be transmitted accurate game data and must be synchronized games, then it can view synchronizing and can play games. Current game synchronization mechanism includes the AI compensation(Dead Reckoning) algorithm based on packet delay and loss and a speedup method through packet skip. Which scheme is actually applied to a game is wholly decided by the characteristics of the game[4,7,10,13].

In order to secure stable game play, some methods to buffer game commands, as well as retransmission algorithm against erroneous packets, are required. Furthermore, any scheme complementing packet loss and error for UDP communication should also be adopted because most of network games are played on preferred UDP communication to support multiple players[4,6,11].

This paper introduces an initial delay buffering scheme for clients to handle any latency time occurring from the load fluctuation in a network when real-time game is played, and suggests a history of game commands based retransmission algorithm to handle drawbacks of UDP mechanism, such as packet losses and errors.

The focus of suggest system is not the issue of retransmission existing packet units but rather focuses on the packets game command data which are found in each packet. Lost game command data can be transmitted, but in the most recent game data, included in the transmissions. It does not involve any increase in costs to the packet.

※ Corresponding Author : Seong-Hoo Kim, Address : (631-701) 449 Wolyong-Dong, Masan, Kyungnam, Korea. TEL : +82-55-249-2650, E-mail : arrayiv@csc.ac.kr
Receipt date : Feb. 25, 2005, Approval date : June. 29, 2005
† Adjunct Professor, Dept. of Computer Engineering, Kyungnam University, Masan, Korea.
†† Professor, Dept. of Computer Engineering, Kyungnam University, Masan, Korea. (E-mail : kspark@kyungnam.ac.kr)
※ This research has been funded by the Kyungnam University, Masan, Korea.

2. CONCEPT OF GAME PLAYOUT IN THE SUGGESTED SYSTEM

Fig. 1 shows a concept of how a network game is played. Network games are synchronized by 2-byte status registers of which contents are decided by keyboards or joysticks and delivered to an input port of the other player.

The main objects and assumptions of the suggested system are as follows.

- Video games generally require better performance than common PC games, and should support and implement 60 frames per second in case of a 3D game.

- Almost current video games, such as PS2, X-Box, etc., support a multi-play mode for simultaneous 2 or 4 players, and can easily support plays on a network when a game engine is accessible through I/O synchronization if game modules and APIs are applied to.

- When a certain solution is applied to PS2 or X-Box level, or the solution plays a role of a library, up to several thousands of video arcade games can be played on a network through simple I/O controls even though source codes are not modified.

- Video arcade games should provide the same views between remote 2 players. It means that coordinates of 3D object movement which is used in common PC network game engines need not be remotely shared. When this notion is actually applied, a lot of synchronization packets can be reduced and the potential following the decrease in the packets are expected once input signals of remote joysticks from 2 remote systems are controlled in the way that those signals are input from a local system and I/O is synchronized based on this kind of signal input.

- The UDP protocol is used to raise speed. A retransmission algorithm should handle any packet loss.

- Views can be synchronized when input signals from remote players' joysticks are transmitted. And time stamp mechanism and mutual synchronization numbers are used to secure stable and consistent synchronization as well as properly handle latency time in a network. Operational variables in a network, such as jitter and delay, can be controlled by buffer management schemes in order to achieve stable game play.

- Dynamic buffering values are set to improve speed. When any packet does not arrive at the predefined time for network failures, some methods should be used to prevent slow display picture, or sometimes stop of picture.

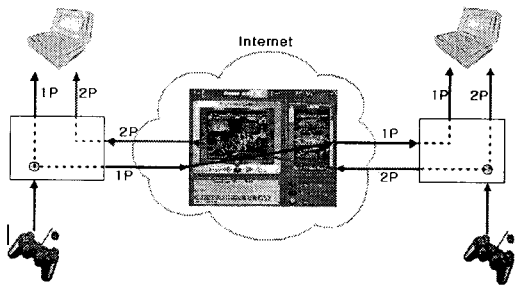


Fig.1. Concept of P2P network game playout.

3. SUGGESTED SYSTEM

This paper represents how to add network modules to non-network game engines and support real-time network game to be played seamlessly. Data of asynchronous commands should be processed synchronously to allow network games to be played. And, data of commands among network players should be executable independent of characteristics of a network. Furthermore, network games should be played even in a worst case due to network loads.

Consequently, the suggested system in this paper is implemented in the way that it can support stable playout of a real-time network game which consists of modules for handling I/O mapping of game engines, packet loss and buffer management.

3.1 I/O Mapping Processing of a Game Engine

Fig. 2 shows a game emulator having input

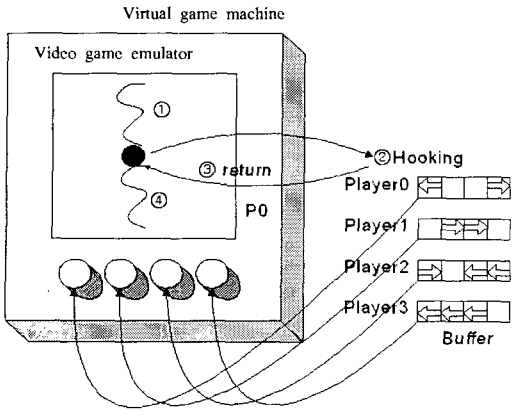


Fig. 2. Processing diagram of commands from players in input buffers.

command data and transmitted input data stored in a buffer with an interval of Time Stamp. The contents in a buffer, which are status values of keyboards and joypads, are sequentially stored with Sync numbers. If input from keyboards or joypads are not applied, commands are presumed not to happen. But, sequential command data is required to keep synchronization going on. The values in the buffer are status values of hardware buttons of a game emulator, and used to make any required operations.

In Fig. 2, 30 frames to 60 frames per second are displayed when game is played. If an interrupt is hooked with another interrupt when display pictures are scanned, frame playback is stopped and command data from each player stored in a buffer is mapped with assigned joypads. Once an interrupt is returned, command data stored in a buffer is applied to status values of hardware buttons by the game emulator and the game is resumed to go on.

3.2 Buffer Management

Each game client suggested in this paper controls its own buffer, which is provided to secure stable game play. If network jitter or packet loss or error under UDP communication occurs, a game can produce a fatal result. In order to prevent this kind of problem, a checking process should be performed to find out whether or not any packet loss occurs in a transmitted stream. Once any packet loss is detected, buffers are rearranged by the lost packet through a request for history retransmission against the lost packet. If a jitter is occurred, then it can adjust the buffer size to level-2 maximum for adapt the load alteration. If the command data received from a network exceeds the number of data processed and played in a game engine, there is a possibility of an overrun.

When the condition expressed in the Equation (1) and (2) occurs for a certain unit time temporarily, the increased or decreased volume in the remaining command data can be processed within the capacity of a buffer at game client side. But, the condition lasts for a long time out of the capacity of a buffer, any overrun or starvation is expected to occur.

$$\sum^{n} Input\ Command\ i\ >\ \sum^{n} Playout\ Command\ i \qquad (1)$$

If the number of command data received from a network is less than that of the data processed and played in a game engine, there is a possibility of starvation.

$$\sum^{n} Input\ Command\ i\ <\ \sum^{n} Playout\ Command\ i \qquad (2)$$

In Fig. 3, the backward threshold is a warning section for starvation so that before actual starvation each player can be notified in this time period. The forward threshold is a time period in which each player can be notified of an overrun. Players receiving these notification signals stop temporarily and resume after buffering delay time.

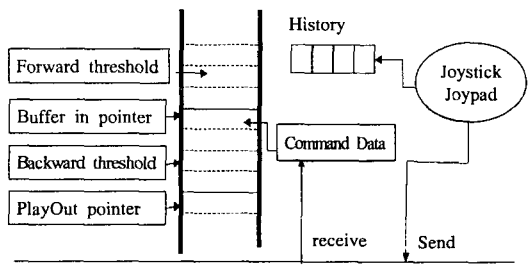


Fig. 3. Asynchronous view.

Buffers are provided as many as the number of game clients playing the game. History buffers store input signals from keyboards and joysticks, which are used when a request for retransmission is received.

For the buffer, an initial value is set to *Ts*. After the *Ts* time input from user's key is processed as if it is a data stream, and proper values for time stamp is calculated. And, a certain time is add to the transmission time that has slower packet transmission time than the other player, then additional initial buffer value is calculated according to defined formulas.

Until 0.1 sec. to 0.3 sec., even though input from user's key is not reflected just in time on a display screen, there is no problem for a game to proceed. Actually, players can not experience any latency during this period. A buffer gathers command data during the initial delay of 0.3 sec, and after a delay of an input to output connection for 0.3 sec, the game keeps going on. *Ts*

The value of time stamp is decided as follows.

(RTT_{max} : Maximum of Round Trip Time,
RTT_{min} : Minimum of Round Trip Time,
FPS : Frame Per Second)

$$Ts = Max((RTT_{max}/2 - RTT_{min}/2), 1/FPS) \quad (3)$$

3.3 Determination of Buffer Size

The buffer size for the system suggested in this paper is set aiming at supporting stable game play as well as optimal synchronization after a game is started. The size can vary according to the number of players and circumstances of network.

The buffer size is first decided at the beginning of a game by the frame rates of each player, round-trip time and jitter allowance. Afterward, the size is re-calculated once any starvation or overrun occurs or any player leaves the game during play.

The initial buffer size is decided as follows. ($B_{de\ lay}$: buffer initial delay)

$$\Delta B_{size} = B_{de\ lay} + RTT_{max} * TS \quad (4)$$

When synchronization is set up again for the above reasons, the buffer size is calculated by the following equation.

$$\Delta B_{size} = \frac{B_{de\ lay} - RTT_{max}/2}{RTT_{max})/2 - RTT_{min}/2} \quad (5)$$

3.4 Threshold Section of a Buffer Value

The buffer for game clients has forward and backward threshold section. The backward threshold notifies each player of an expected abnormal condition before command data stored in a buffer is used in all, and requests a processing delay in order to make the buffer filled with data again. (Δ^j_{max}: maximum latency of arrival command, Δ^j_{max}: maximum jitter of arrival command)

For the above role, thresholds should have time allowance to delay some buffer processing or synchronize buffers again before all resources are consumed.

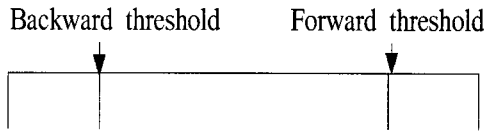


Fig. 4. Threshold section.

3.5 Retransmission Algorithm

Fig. 5 shows conceptual flow of the retransmission algorithm.

For a transmitted packet, the current sequence number is compared with the previous number. When the result of the comparison shows that the packet is not sequential, the retransmission algorithm is activated following a request for retransmission and according to the extent of time stamp. If time stamp is included in the time period in which user's key is pressed down and a packet does not arrive at a destination, the previous command data is compared with the next command

data. When the result of the comparison shows discrepancy between them, retransmission is requested. Otherwise, the data is duplicated for further use.

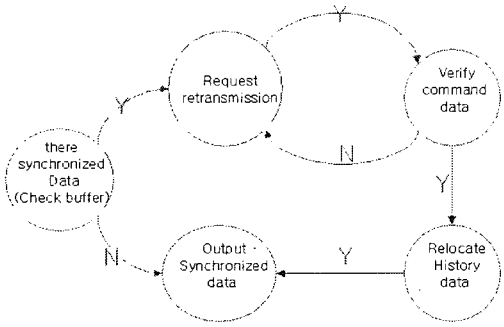


Fig. 5. Retransmission algorithm.

3.6 I/O Based History Retransmission

Fig. 6 represents conceptual flow of an I/O based history retransmission algorithm. The transmitted packet is filtered out first, and the synchronization sequence number and command data is separated before a request for retransmission. The request for packet retransmission is decided on the basis of time stamp. Once the request is generated, a sender transmits a requesting message for retransmission and its game data integrated altogether into control signals. Upon receipt of the requesting packet, a receiver send out the requested game data included into the data which is originally intended to be transmitted.

Fig. 7 shows how the receiving process of the history retransmission algorithm is performed. When a packet is arrived, the message type is classified by packet filtering. In case of the analyzed packet is retransmitted one by retransmission algorithm, the whole content of the packet is stored into a buffer. When the classified packet is normal one, current synchronization number and the next number are marked by way of precaution against packet loss or error using the Estimate sync Table. If the currently identified synchronization number differs form an expected number, 'R' is marked in the Retransmit Table, which means that a request for retransmission is going to be included in the normal data subject to being transmitted. The History Transmit Table keeps the history of synchronization number to which senders requested until now.

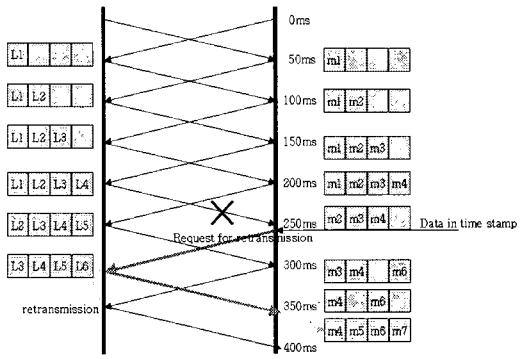


Fig. 6. I/O based history retransmission flow.

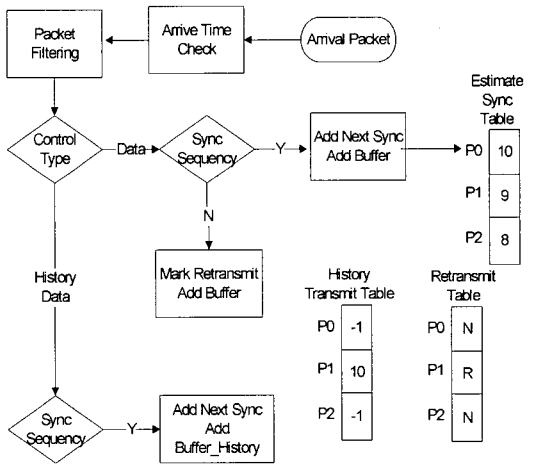


Fig. 7. Receiving process of the history retransmission algorithm.

Fig. 8 shows how the sending process of the history retransmission algorithm is performed. The process operates with an interval of Time Stamp. The data type of the packet to be sent out is decided by the Retransmit Table and History Transmit Table. In addition, the size of the history data is set referring to the History Transmit Table. In case unidentified or un-received synchronization data number is detected according to the contents of the Retransmit Table, a requesting message for retransmission is included into a packet.

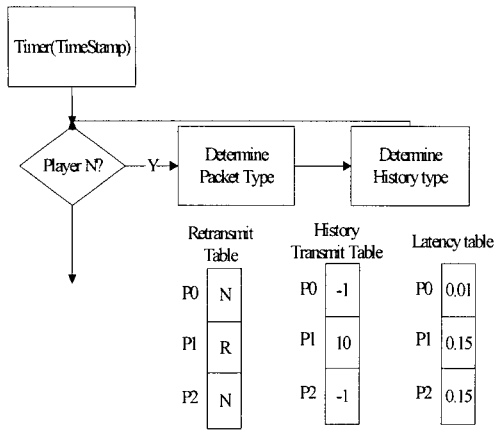


Fig. 8. Sending process of the history retransmission algorithm.

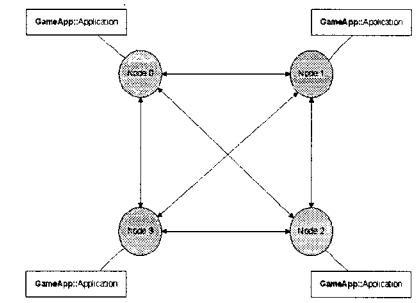
4. ANALYSIS OF THE EFFICIENCY OF THE SUGGESTED ALGORITHM

The environment of simulation is shown in Fig. 9(a,b). With NS2 simulation, the efficiency of the suggested algorithm is analyzed. Fig. 10 shows a buffer status having contents with error ratios of 1% and 5%, respectively. The buffer contents are filled during an initial delay of 0.3 sec. and then they are consumed along with playout. From the simulation result, it can be inferred that history based retransmission algorithm can keep a buffer stable even though packet errors occur.

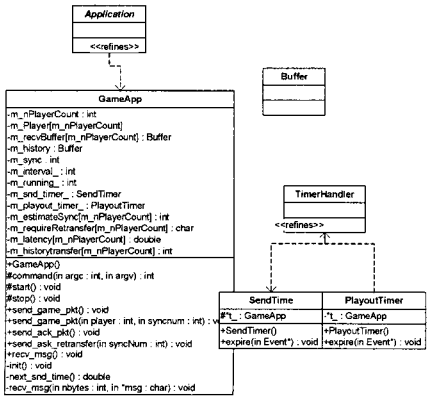
Fig. 11 shows the change of requested retransmission packet size with error ratios of 1% and 5%, respectively. From the simulation result, it can be inferred that the request increases several bytes only in data volume.

Fig. 12 shows the number of requested retransmission with error ratios of 1% and 5%, respectively.

Fig. 13 shows the number of requested packets between the suggested algorithm and other retransmission request with error ratios of 1% and 5%, respectively. Under the suggested algorithm, packets are generated in a uniform manner. On the other hand, in the case of general retransmission,



(a) Components of Game Application for each node



(b) Object component of Game Application.

Fig. 9 The environment of simulation.

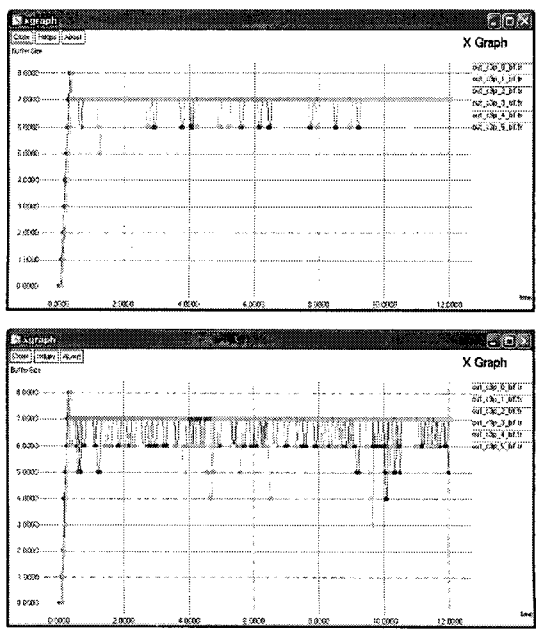


Fig. 10. Buffer contents with error ratios of 1% and 5%, respectively.

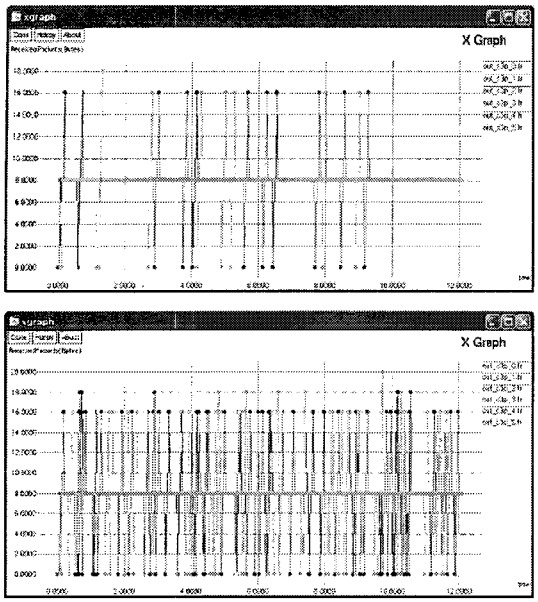


Fig. 11 Retransmission packet size.

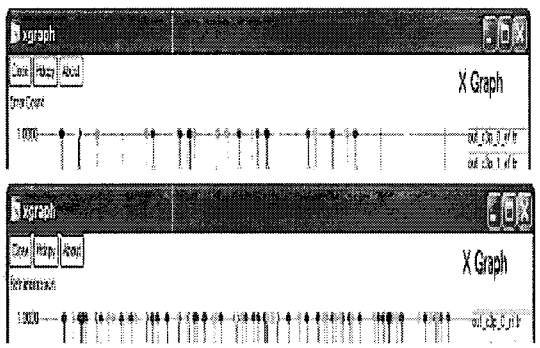


Fig. 12 Number of requested retransmission.

it shows increase in packet numbers.

Fig. 14 shows the number of requested total packets between the suggested algorithm and other retransmission request with error ratios of 5% and 10%, respectively.

Fig. 15 shows to analyze the process of resynchronization with setting up 0.2 sec. in initial delay and with packet error ratios of 0.8%. In case of short time of intial delay, it can be recognized happening to resynchronization even if the error ratio is low.

Fig. 16 shows the process of resynchronization for simulation analysis make change one node from

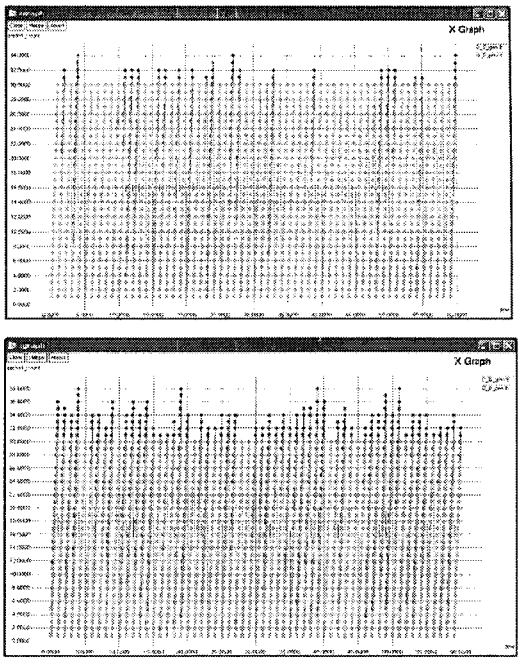


Fig. 13 Number of requested packets.

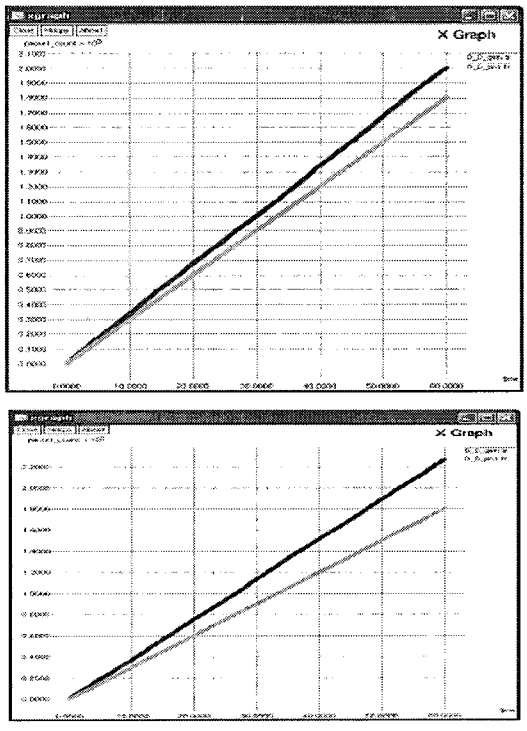


Fig. 14 Number of requested total packets.

30ms to 200ms of delay and the other node from 100ms to 200ms of delay with packet error ratios

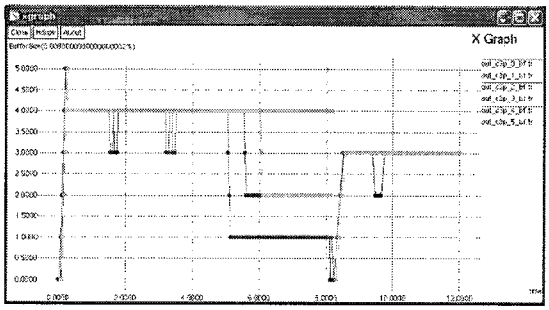


Fig. 15. Process of resyncronization I. (initial delay=0.2sec, packet error ratio=0.8%)

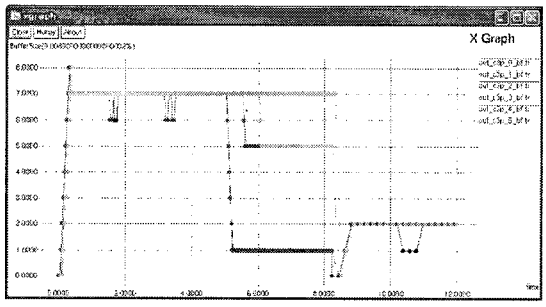


Fig. 16. Process of resyncronization II. (increase delay=200ms)

of 0.8%.

We tried to analyze the process of resynchronization in Fig.15 and Fig.16. Although network traffic can happen, it was tested the whole time of resynchronization, and the buffer can be changed with downsizing of buffer.

Here, we can see the case of 0.3 sec. buffer delay is the most suitable for recovery. And even if the packet losses are occurred continuously, retransmission packet will be increased resynchronization.

5. CONCLUSION

The suggested algorithm in this paper uses synchronization numbers for frames to check and find out any packet loss, and bring in the Time-stamp mechanism, to detect consecutive packet losses. With this algorithm, along with game play, packets of game commands are transmitted with an interval of the time stamp determined by taking latency time into account to keep the same and synchronized game views. A history of game commands based retransmission algorithm to handle lost packets is designed and implemented as an application protocol. Under the algorithm, control codes required for game play and data of game commands are altogether sent and received with a request for retransmission. It means that a lost packet is retransmitted, if requested against packet loss, with requested game command data so that overhead to retransmit game commands can be eliminated.

If the proportion of packet error is 1%~5% with 0.3sec in a buffer initial delay, it does play with stable situation. In other case, if the proportion of packet error is over 5%, the only game data are added on it. It means that even if the packet losses are occurred continuously, retransmission packet will not be increased to resynchronization.

Authors have a plan to evaluate the suggested algorithm in a mobile environment, and develop an algorithm which can relay the retransmission to each client from a pivot client showing better network performance as a further research.

6. REFERENCES

[1] Wu-chang Feng and Wu-chi Feng, "On the Geographic Distribution of On-line Game Servers and Players," *NetGames*, pp. 173-179, May, 2003.

[2] Francis Chang, Wu-chang Feng, and Wu-chi Feng, "Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server," *IMW2002*, pp. 151-156, Nov, 2002.

[3] Yutaka Ishibashi and Shuji Tasaka, "Causality and Media Synchronization Control For Network Multimedia Games," *NetGames*, pp. 42-51, 2003.

[4] E. Cronin, B. Filstrup, A.R. Kurc, and S.Jamin "An Efficient Synchronization Mechanism for Mirrored Game Architecture," *NetGames*, pp. 67-73, May, 2002.

[5] G. Armitage, "Sensitivity of Quake3 players

to network latency," *ACM SIGCOMM Internet Measurement Workshop 2001*, Berkeley, CA, USA, Nov. 2001.

[6] Matthew K. H. Leung, and John C. S. Lui, "Adaptive Proportional Delay Differentiated Services: Characterization and Performance Evaluation," *IEEE/ACM TRANSACTIONS ON NETWORKING*, VOL. 9, NO. 6, DECEMBER 2001, pp. 801-817.

[7] Yang, J. and Lee, D., "Scalable Prediction Based Concurrency Control for Distributed Virtual Environments," *Proc. IEEE Virtual Reality 2000*, New Brunswick, March, 2000.

[8] N. Baughman and B. Levine., "Cheat-proof playout for centralized and distributed online games," In *Proc. Infocom 2001*, April 2001.

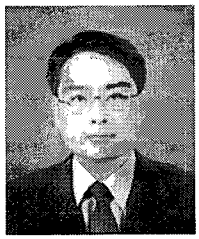
[9] J. Färber, "Network Game Traffic Modelling," *Proceedings of NetGames 2002*, pp. 74-78, Braunschweig (BRD), 16-17 April 2002.

[10] Diot, C. and Gautier, L., "A Distributed Architecture for Multiplayer Applications on the Internet," *IEEE Networks Magazine*, Vol. 13, N. 4, July-August, 1999.

[11] L.Gautier and C.Diot, "End-to-end Transmission Control Machanisms for Multipartyr Interactive Application on the Internet," *IEEE INFOCOM'99*, pp. 1470-1479. March 1999.

[12] Katherine Guo and Sarit Mukherjee, "A Fair Message Exchanges Framework for Distributed Multi-Player Games," *NetGames 2003* pp. 29-41, May, 1998.

[13] Cai, W., Lee, and Francis B.S., "Auto-Adaptive Dead Reckoning Algorithm for Distributed Interactive Simulation," *Proceedings of Thirteen Workshop on Parallel and Distributed Simulation*, pp. 82-89, 1999.



Seong-Hoo Kim

He received his M.S and Ph.D degrees from the department of Computer science, Kyungnam University, Korea in 1997 and 2005. He was a associate professor in school of Computer science at Chang-Shin College from 1998 and 2004. At present, he works for YeSoft/Director and he was a Adjunct Professor at Kyungnam University. his research field has been Network Game and Mobile & Ubiquitous Multimedia.

Kyoo-Seok Park

He is a professor of Computer engineering at Kyungnam University, Korea. He is the honorary president of Korea Multimedia society now. His research focuses on Distributed system, specifically applied to internet computing, Security system and Multimedia system. M.S and Ph.D in Computer science from the Chung-Ang University, Korea.