A Challenge for Reusing Multiplayer Online Games without Modifying Binaries *

Yugo Kaneda† Hitomi Takahashi† Masato Saito† Hiroto Aida† Hideyuki Tokuda†,‡

†Graduate School of Media and Governance
‡Faculty of Environmental Information
Keio University
5322 Endo, Fujisawa, Kanagawa 252-8520, Japan
{ichiriki, hitomi, masato, haru, hxt}@ht.sfc.keio.ac.jp

ABSTRACT

In this paper, we advocate the problem of reusing Multiplayer Online Game (MOG) in Client-Server (C/S) architecture. The problem is that MOG services cannot continue to be provided because of high maintenance cost for operating game servers. Additionally, it is caused by the decreasing service users who getting tired of the game and game providers who have faced difficulties in collecting charge of the game service. It is important that every user can play MOGs at any time whether game servers of game providers run them or not.

We describe a challenging method for solving the problem. Our solution provides a middleware which is inserted under the game applications to switch network architecture from C/S to Peer-to-Peer (P2P). By exploiting this method, the network architecture of MOGs can be easily changed from C/S to P2P under some restrictions without modifying binaries.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Management, Design, Experimentation

Keywords

Multiplayer Online Games, Reuse, Client-Server Architecture, Peer-to-Peer Architecture, Middleware

1. INTRODUCTION

Multiplayer Online Gaming (MOG) has been coming into wide use along with the popularization of various broadband networks and the technical advancement of gaming devices. Quake [14], Age of Empires [18], and Counter-Strike [24] are the examples of the most popular game titles. In these network games, we can share the same virtual gaming space and communicate with other players to play the game at the same time through the local area network or the Internet.

When we refer to the network architecture of MOG, it is divided into C/S architecture and P2P architecture. In C/S architecture, every player who wants to share the same gaming space connects to a game server which calculates the player's interaction and transition of the whole game state, and they update their game states based on the results which are sent by the game server. Most commercial online games adopt C/S architecture. It is clear that game management is easy for game providers because they can manage gaming environments centrally and distribute update programs to all users at once. However, players cannot use gaming service when the game server is down.

On the other hand, P2P architecture has no central management node for handling players and calculation of game states. This architecture forms distributed network by connecting players directly and has no single point of failure such as game server. However it is difficult for this architecture to deliver game packets fairly with each other, to synchronize each game state, and to construct a centralized accounting system.

In this paper, we mention the reuse problem which signify that some MOGs, that are not able to play, will increase in the near future. Additionally, it is necessary for game providers and game users to reuse the MOG's game applications which take C/S architecture. An important key is that game providers distribute such MOGs as P2P architectural applications. Our middleware supports to construct P2P overlay network of MOGs using C/S game applications.

The following sections are constructed into eight sections. We have explained background and architecture of MOGs in section 1. Section 2 describes the reuse problem of MOG which we advocate. Section 3 introduces our solution approach with middleware. Additionally, section 4 and section 5 show its design and detailed mechanism, and section 6 shows implementation. After focusing on our system, we refer other related work in section 8. Finally, we describe our conclusion and future work in section 9.

*This research has been conducted as part of the Ubila Project supported by the Japanese Ministry of Internal Affairs and Communications.

NetGames'05, October 10–11, 2005, Hawthorne, New York, USA.

2. REUSE PROBLEM IN CLIENT-SERVER ARCHITECTURE

While C/S architecture become increasingly popular in commercial MOGs, it has some critical drawbacks at the time when game servers are halted by reducing its game users. We indicate two concrete weak points in the follows.

- Difficulty for continuing services for a few users

  There is a serious problem that it is difficult to continue hosting a gaming service when the game becomes unpopular because of getting tired by service users. One of the reason is that game providers must continue paying for maintaining game servers in despite of decreasing users.

  Additionally, when the game server stops as the end of the game service, it is not clear whether there are any organizations or any individuals who would support to continue providing such game services in the future. Thus, users who play such unpopular network games are forced to stop enjoying the game services.

- Restricted access of personal record data

  Most MOGs save personal record data in server-side binaries. These record data includes important factors for each user to resume interrupted game: avatar's status, personal belongings, and funding.

  This approach is useful for preventing the point that malicious user accesses such data. However it causes that users cannot access personal record data for resuming situation of game world while game servers are down. Furthermore, when the support for official game servers is terminated, no user can download each record data from servers.

While unpopular games are increasing, MOGs also have a important aspect of entertainment software. It signifies that users enjoy every game regardless of the age. In fact, there are many reprint versions of old games in present platforms and there are many successful commercial cases to resell such games at a low price. This case similarly happens to MOGs and we define this as **"reuse of MOG."**

It is efficient for game providers to be supplied with reuse architecture for MOGs because they can increase the chance to sell such games. When we begin reusing of MOG with C/S architecture, it is also a bottleneck that we must consider the cost of mantaining game servers.

2.1 General Approach

It is one of the solutions that game providers distribute binaries or sources codes of server applications for game users. Althought it seems to be the best solution, it remains a problem that each user who provides game servers must operate them. Furthermore, when users who provides service stop their game servers, it is not possible for the users to play games on the server to use own personal record data of game playing.

Another approach is to exchange network architecture of such games from C/S to P2P for a reason that we do not need the cost to maintain game servers. It implies that we must reprogram network parts of its game software. However, for general users, it is overly complicated and impossible. In a similar way, it is highly expensive for game providers to reconstruct the software.

3. PEER BOOSTER

In this section, we explain our novel approach for easily switching network architecture of game applications. Our solution is to develop a middleware which constructs Peer-to-Peer network infrastructure under the game applications. We call this middleware **PeerBooster**.

3.1 Overview of PeerBooster

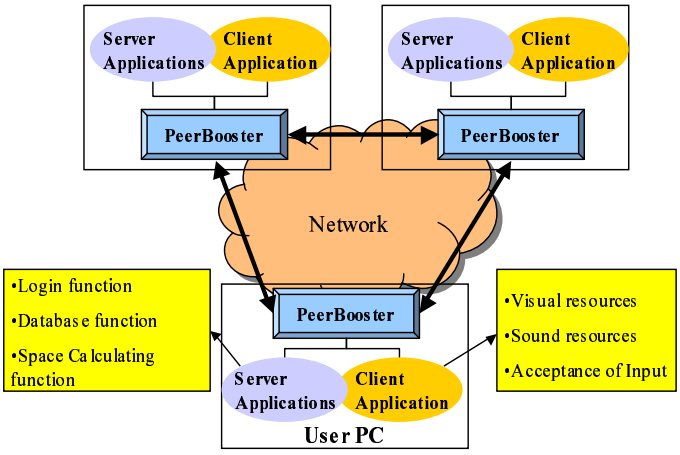

Figure 1: System image of PeerBooster

We show the system image of PeerBooster in Figure 1. Each user's terminal contains both client and server applications. PeerBooster is independent from game applications and it captures all messages generated by such applications. Basically, game users connect the server application in their terminal. Afterward, PeerBooster captures the game packets which are forwarded to server applications, and the packets are delivered to all terminals which join the same game session. Each client application receives update information from the local server application.

3.2 Assumptions of PeerBooster

As for an important point, when PeerBooster is first operated, PeerBooster should be able to understand detailed information of game application to distinguish network packets instead of not modifying the application code. We describe such assumptions as follows.

- The server binaries are distributed by its game provider.

  PeerBooster needs server binary for emulating the C/S communication of the Internet in the local client terminals. Though this seems to be difficult, there are many activities to distribute or emulate server functions nowadays. For example, id-software has positively opened the binary to the public. Futhermore, some open source projects exist for implementing the secreted server applications [4] [1].

- Network and application specific protocols are indicated.

  PeerBooster needs to recognize network protocol and application specific protocol of MOGs for selecting from

game packets in the network and duplicating emulation packets which is sent to other user's terminal. This restriction indicates that the session initization, game communication, and session end protocol should be clearly and well opened.

- The network port numbers which game applications use do not overlap.

  One of the important restriction is that there is no overlap of port numbers between client and server applications. Most MOGs use specific ports for communicating between client and server applications. In general, game packets are delivered by IP address and the port number when communicating with C/S architecture. However, when the client and the server application are started in the same terminal, they cannot use same ports. It is necessary to judge the delivery of the game packets by forwarding them to different ports on the inside of the terminal.

- The network port which related packets pass through is opened.

  Recently, most computers which connect to the Internet are protected by firewall or use Network Address Translation (NAT) [23]. These technologies often hinder that overlay nodes construct direct connections with each other. For constructing overlay network using PeerBooster, we assume that the ports which are related to PeerBooster or game sessions are opened.

- The games only include static game resources.

  Some MOG contains the factors which user cannot control directly. For example, it signifies Non Player Character (NPC) or items are generated in the game fields at random. These are automatically and randomly operated in the server applications.

  In the section 4, we show that PeerBooster synchronizes state of client applications by synchronizing state of servers which are included in each user's terminals. Currently, PeerBooster supposes that the state agreement of the server among users is achieved by synchronizing the client packets. We must synchronize random factors of servers when server applications include these NPC or items, which are randomly controlled. For such reason, we define that PeerBooster focus on the MOGs which include no random resources.

4. DESIGN OF PEER BOOSTER

Detailed design of PeerBooster is described in this section. PeerBooster is composed of five main modules: Topology Construction Module (TM), Login Module (LM), Game Message Handling Module (GM), Synchronization Module (SM), and Zone Management Module (ZM). In addition, two data tables, which are Internal Server Table and Member Table, corporate with them.

First, we describe the entire design with system configuration chart of PeerBooster. Second, the functions of each module and data tables are explained in the following subsections.

4.1 System Architecture

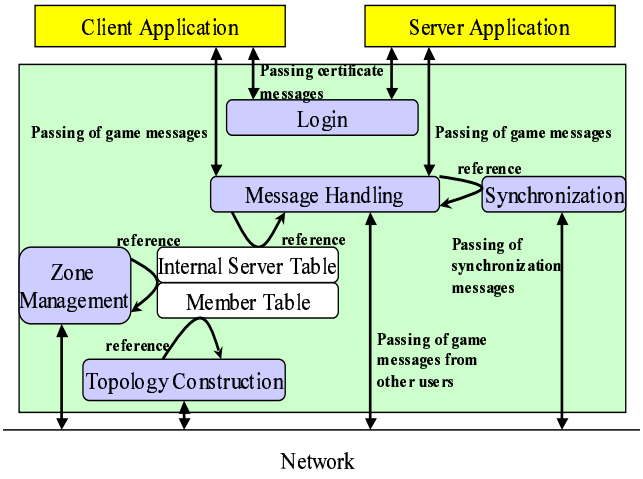

Figure 2: Architecture of PeerBooster

We show the system architecture of PeerBooster in Figure 2. First of all, Topology Construction Module constructs overlay network for preparing game sessions when PeerBooster starts.

PeerBooster operates on each user terminal before the game applications begin as a assumption. First, Topology Construction Module negotiates session among users who join the same game. This module also manages synchronization of game database for matching each first state. Second, PeerBooster begin to capture the game application packets. At the same time, Login Module operates duplicating and converting user's certification packets. These packets are used to pass the application specific certification of server applications which other user's terminal includes.

Afterward, Message Handling Module supports sending game messages correctly to other users to synchronize game states of each server. In C/S architecture, a single client application cannot maintain multiple session for servers. Then, when each client sends their updates to the users, it needs for them to change the packets which depend on information of each server session. This module converts a part of messages to match each game session by using recorded information in PeerBooster. Each server which is included in each user's terminal receives all client updates which is converted as acceptable game messages for other user's servers.

- Topology Construction Module (TM)

  TM realizes constructing network among users who join same game session. Most MOG applications use a preset data for starting game which includes skin, status or some personalized parameters of avatars. Each server application need to match the state of these data for providing the same states for clients. As a specific function, TM duplicates game setting or database files of every server application to synchronize the first state of each user.

  In addition, application-level paths among users are optimized by topology optimization function of TM. There are effective methods and systems of optimizing application-level paths [13] [15]. We build them in

our middleware by encapsuling and tagging outside of game packets. The following shows the main components for setting TM in Table 1.

Table 1: Components of Topology Construction Module

Component	Content
Routing Algorithm	Routing algorithm used in P2P overlay network
Game Database Path	Database paths for synchrozing the states of game users in the same session

- Login Module (LM)

  The function of this module is to pass certification of each game applications. Most of MOGs have an application specific protoccol for initiating game sessions to certificate each client. In paticular, we frequently need to input user-ID and password phrase for login of game servers.

  GauthierDickey et al. advocate some problems of cheating method in MOGs at protocol and network level [10]. Protocol level cheating indicates that a malicious attacker fudge on game play with taking advance of application protocol's weak point.

  In the protocol level cheat, there is Inconsistency Cheat which a malicious player sends different update packets to other players or game server by changing content of packets by making bad use of the application protocol information.

  We focus on this cheating method, then our middleware reproduce certification packets and camouflage the authentication on each game session. Typically, CM not only store local information such as user-ID and password phrase but also store the identification values used by challenge and response protocol.

- Message Handling Module (MM)

  This module operates delivering game messages among applications.

  The architecture of MM is shown in Figure 3. The part enclosed by a big rectangle is MM. MM cooperates with other modules for delivering of messages fairly and masquerading of game sessions.

  When MM receives game messages from other users through the overlay network, it reaches Message Receiver which manages timing to deliver messages for own internal server application. Then, the synchronization timing is decided by referencing Synchronization Module that checks current step of game processing for checking sequence number of game messages among users. Message Receiver forwards game messages to MM after waiting the time notified by Synchronization Module

- Synchronization Module (SM)

  One of the important issues is synchronization of each user's state for sending client messages to other users in a fair way. In the P2P overlay environment, it is

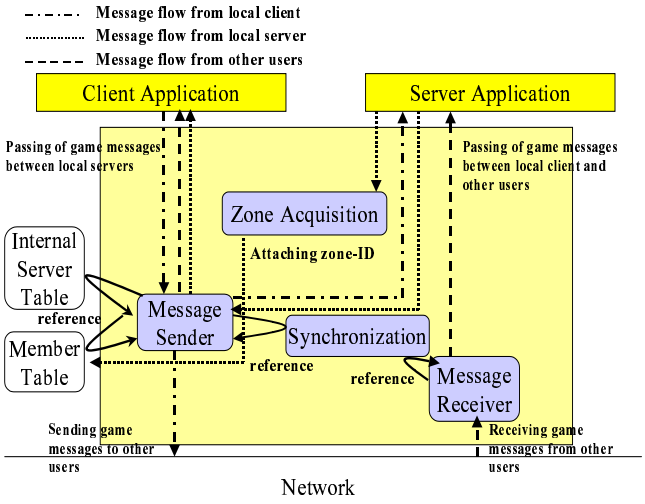

Figure 3: Architecture of Message Handling Module

difficult to synchronize the states among users because there are various values of delay and jitter by the heterogeneous network links.

Lin et al. present synchronization architecture for C/S network gaming applications [16]. This architecture takes server proxy between game server and client application. By using proxy, it realizes fair message exchanging for adding a little delay to send each client updates to server application. We assume that each terminal have synchronized clocks using Network Time Protocol (NTP) [19] or Global Positioning Systems (GPS) [12], then we adopts this algorithm. More detailed description is referred this in section 5.

- Zone Management Module (ZM)

  ZM is used by the game category such as MMORPG. Generally, the game world of MMORPG are divided by zones which are significant space in games. For example, game world is composed by fictitious countries. In an application architecture, a binary contains information of a single world. When an avatar which exists in a single world moves to another world, client application switches the connection of port numbers or binary process to change the world.

  Under this situation, all avatars need not receive all client update message [22]. Let us assume that two users, who exist in different worlds, need not take notice of instant messages that are generated with each other.

  Our architecture focuses on the division of game worlds based on each map data, and avatars only connected same map exchange all client messages to reduce unrelated traffic. For distinguishing each game zone, ZM is attached simple information of map context or zone-ID which is generated from binary name and binary size by using hash algorithm. Additionally, simple contexts of messages are leveraged to cooperate with SM. The detail how these factors are integrated is also discussed in section 5.

The following explanation is description of the databases which supply needful information for each module.

- Internal Server Table (IST)

  IST are referred so that MM searches for the destination to send game messages. We show the main components of the IST's rule description in Table 2.

Table 2: Components of Internal Server Table

Component	Content	Remarks
Client port number	Port number used by client process	
Client binary name	Binary name used by client process	
Address of C-server	Address in process of server where game space is calculated	The port number and the binary name are described by the pair.
Port of C-server	Port number used by calculation server	The port number and the binary name are described by the pair.
Binary name of C-server	Binary name used by calculation server	The port number and the binary name are described by the pair.

  Our analysis defines that the server binaries of some MOGs in C/S architecture are divided by its functions. In particular, the main divisions of these functions are two categories: login server (L-server) and calculation server (C-server). At first, L-server takes charge of certification steps of each client at first. After certification process, C-server manages accepting sessions for starting games.

  The components of IST are described based on previous definitions. These components contain network information of local client and local server application that should forward updates to local client.

- Member Table (MT)

  In the P2P infrastructure of PeerBooster, the network information of each user is registered in MT. PeerBooster converts game packets by using these information for matching each P2P session.

  When PeerBooster constructs P2P overlay network, it challenges to connect lobby server which manages negotiation of sessions. The information of lobby server is described in the setting file, and MT tries to connect its service. Furthermore, Message Sender, which is included MM, refers to this table for converting and sending game messages to other terminals.

  Table 3 shows the component description of MT.

Table 3: Components of Member Table

Component	Content
Lobby server address	Lobby server address for searching users which play same game
Member address	Address for constructing P2P overlay by users directly
Member port number	Port number for constructing P2P overlay by users directly

  At first, MT needs the description of lobby server which has global address. If there are no information about lobby server, MT must contain specific information about members which directly construct overlay network.

5. DETAILED DISCUSSION

Some detailed processes of PeerBooster are discussed in this section. We focus on three important aspects of system: Consistency Scheme, Zone Management Scheme, and Security Scheme.

5.1 Consistency Scheme

Real-time P2P game architecture involves a significant problem that it is difficult to maintain consistency of peers at fair update timing. This problem is caused by random fluctuating network delay of heterogeneous network paths among peers. While there is no perfect solution to keep on consistency for all time of game play at present, we suppose that interval time of each packet, which server applications receive from each client, fall within the estimated range based on actual network analysis [6]. We can optimize this threshold based on network characteristics of each game. Additionally, we define each server application generates constant update packets toward client applications.

Our consistency scheme utilizes proxy and buffering technique with SM [16]. Figure 4 indicates a simple inconsistency case by two client's action messages. C_1, C_2 signify two clients, and S signifies a server. M_1, M_2 are action messages generated by C_1, C_2 at t_1 and t_2 ($t_1 > t_2$) respectively.

Let remind that each terminal keeps a global clock by NTP [19] or GPS [12]. Client messages are marked with timestamp at the point that they are sent out for other terminals.

SM operates as proxy system. When receiving messages, SM buffers them and monitors until passing Wait For Consistency time (WFC-time). WFC-time is deadline to guarantee delivering client messages to server application in sequential order, and this threshold is regulated between measured average delay in current group and comfortable response time that users do not feel stressed by delay of feedback to screen [20] [21].

First, M_1 arrives at SM. SM involves counter to number user terminals. SM buffers this message in sending queue for server. Second, although M_2 arrives at SM, it is generated before M_1. Then SM reorders these messages by chronological order in the queue. After passing deadline of each message, they are delivered to its server application.

When inconsistency is occurred at SM by exceeding deadline, we cannot guarantee consistent processing of its server application. However the probability of inconsistency can

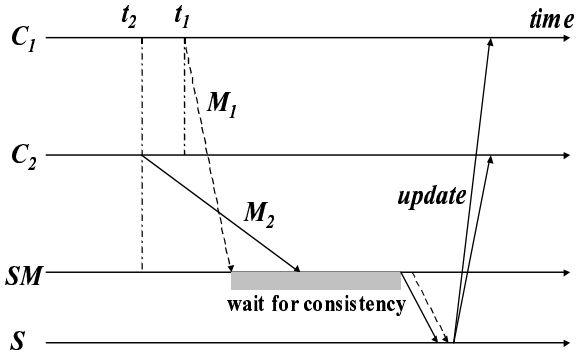

Figure 4: consistency scheme to resequence out of order messages

be reduced to utilize filtering based game contexts by appending each message to context information. As a simple example, if a server receives duplicated messages about update of position or advertisement of status from same client in the wrong order after deadline of previous update message, they do not affect any critical state changes of game world. Therefore SM does not need to deliver them to server applications and these messages can be dropped.

Moreover, fairness is another important problem. It indicates that all users can only commit own action at the same frequency and fair order. An advanced architecture for fairly delivering messages is proposed, and it can realize fairness without synchronization by a global clock [11]. Though the capability whether it can adapt to game or not depends on some implementation of message update in individual game and it drops messages of incorrect order without judging context of messages, we believe to apply its improved method.

5.2 Zone Management Scheme

In the large distributed virtual environment such as MMOG, the basic reduction technique of communication messages is achieved by filtering messages by distance between peers. Our architecture cannot take this method because it requires reimplementation of source code or complex filter description which is related to context of game messages, nevertheless it is dramatically effective to reduce overhead of useless traffic.

We adopt to data-centric filtering approach called Map and Context based Packet Filter (MCP-Filter) to filter messages for communication with ZM. Game messages among peers are filtered by preset information and verifying messages. In a actual case of MMORPG [4], port number and IP address are important indicators of evaluating filter by reason that clients change game maps by switching connection ports.

At first, we define the correspondent information both zone name and key value which are shared in a group. Zone name is related to each data of game map, and key value shows trigger to examine game messages such as port number, or IP address.

When ZM detects transitions of each client's trigger, ZM stores them and starts to check context of game messages. After that, ZM drops output messages of the client which exists in the same terminal if the context of messages fulfill the condition of filtering and these messages does not be related to client's zone.

Note that MCP-Filter does not drop all messages which are not related to each map among all users. The reason is that PeerBooster needs immediately to reproduce map state when own client switches game maps. Then MCP-Filter works closely with filtering based on message context.

5.3 Security Scheme

For fairness of game playing, it is necessary that game processing must be guarded by cheat preventing mechanism from malicious attacks. From previous research, protogcol-level cheating can be divided into five main categories [10]: Fixed-Delay Cheat, Timestamp Cheat, Suppressed Update Cheat, Inconsistency Cheat, and Collusion Cheat.

In distributed overlay environment, Lookahead cheating, which corresponds to Fixed-Delay cheat, is one of the critical problems [5]. This problem shows that it is possible for the user who transmits updates the latest can decide their action after seeing the state of all other players at any time.

Some advanced algorithms have already been proposed in some research for solving this problem. One of the most famous approaches is lockstep synchronization [5]. Lockstep is a cheating prevention method that each user cannot proceed simulation clock until being announced a cryptographically secure one-way hash of other user's decision as a commitment. After announcing all user's commitment with each other, users send their decisions in plaintext.

This method prevents cheating to add fixed delay. Furthermore it allows that no user can commit one's different update without transmitting the valid update preliminarily. However, there is another problem that this method causes high delay because it locks the each step of game processing. Although asynchronous approach, which is another method to relax this restriction, is proposed in its paper, it is not acceptable with considering the comparison of frame rate of current MOGs.

NEO protocol is another solution to prevent protocol-level cheating and to realize low latency delivering in P2P architecture [10]. This architecture adopts voting protocol for omitting strict acknowledgement among each user. They accept their commitment when voting rate is greater than a threshold.

We do not adopt NEO protocol to PeerBooster because it drops commitments when voting rate is smaller than a threshold. We consider that this method signifies game developers need prescreen imcoming packets by the impact for game play. Our architecture aims to simply construct overlay network without describing detailed rules of game applications. Therefore PeerBooster needs to guarantee every transmission of game messages in place of modifying game application code though this method is advanced solution.

Therefore, it is our solution for our middleware to adopt Sliding Pipeline protocol (SP) in Synchronization Module [8]. SP is the improved method for decreasing commitment delay by omitting some acknowledgements of each step in Lockstep synchronization. While it cannot realize perfect prevention about protocol level cheating, it performs the better

frame rate than Lockstep method by adjusting the size of adaptive pipeline.

6. IMPLEMENTATION OF PEER BOOSTER

Our prototype of PeerBooster has been implemented on Windows 2000 and XP platforms by using C language. It includes Login Module and Message Handling Module.

Furthermore, XML which we have used as expat library [3] was adopted as a rule description language. Table 4 are an example of the XML rule description in Login Module.

Table 4: Setting description for Login Module

```
<certificationforwarding>
  <client>
    <port>10000</port>
  </client>
  <certification>
    <server>
      <ip>133.27.XX.XX</ip>
      <port>10010</port>
      <process>cserver.exe</process>
    </server>
  <certification>
  <database>
    <server>
      <ip>133.27.XX.XX</ip>
      <port>10020</port>
      <process>dserver.exe</process>
    </server>
  </database>
</certificationforwarding>
```

Client tag signifies information about local client application. Detailed information of every server binary is enclosed by *certification* tag and *database* tag which show login and database server information. In addition, *ip*, *port*, and *process* tags include information of IP address, port number, and process name for transmitting messages to server application respectively.

We also show an description example of IST in Table 6.

Table 6: Setting description for Internal Server Table

```
<internalforwarding>
  <client>
    <port>20000</port>
  </client>
  <calculation>
    <origin>
      <ip>133.27.XX.XX</ip>
      <port>20010</port>
    </origin>
    <forward>
      <ip>133.27.XX.XX</ip>
      <port>20010</port>
      <process>gamezone1.exe</process>
    </forward>
  </calculation>
</internalforwarding>
```

This setting also divided two main parts: *client* tagging part and *calculation* tagging part. *Client* tag shows description about client session information for playing games. As for *server* tag, it is necessary to describe inside information, which is in the forwarding origin and the forwarding destination of a server in pair.

For capturing all packets, we have used WinPcap library [9] and Network Driver Interface Specification Intermediate (NDIS IM) driver [7]. WinPcap is the major library, which is compatible driver with other UNIX platforms, then we can lightly transplant PeerBooster to such platforms. Moreover, NDIS IM driver realizes dropping and forwarding game packets at kernel level before their packets reach the game applications.

7. EXPERIMENTAL EVALUATION

We have evaluated the prototype of PeerBooster in aspect of local performance on the termimal. Our experiments have measured the two processing overheads which are to forward messages on the inside of its terminal and to transmit messages to other terminals. For measuring exactly in the terminal, Pentium Time Stamp Counter has been used in all evaluations. Furthermore, we used Quake2 application to generate game packets [14].

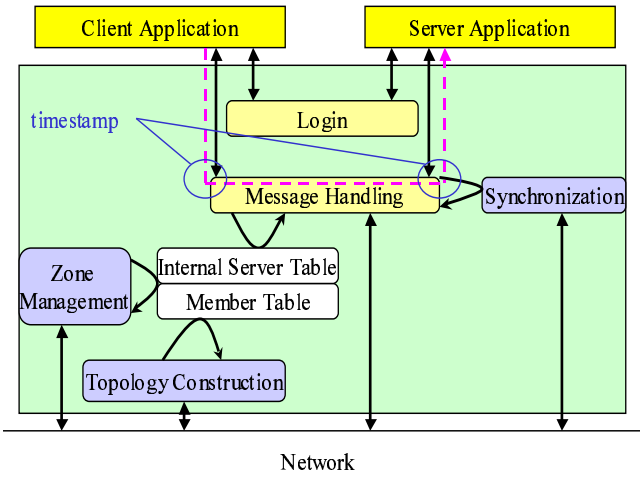

Figure 5: Average overhead for sending messages to other terminal

Figure 5 shows the measured points in the evaluation of inside forwarding. Inside forwarding signify the transit time from local client application to local server application in PeerBooster. We have pushed time stamps between the point when PeerBooster captured messages from the client and the point when PeerBooster outputted messages to the server. The overhead was measured 300 times.

Moreover, we show the result of this experiment in Figure 6. The experimental result shows that the average processing time is 68.9 microseconds and all processing times fall within the range from 60 to 80 microseconds. In Figure 6, the plots stratify by two layers. We consider that the reason of this interval is because of scheduling in Windows XP.

Most First Person Shooting games which includes Counterstrike and Quake2 process refreshing one frame by about 16 milliseconds at a maximum when these games update dis-

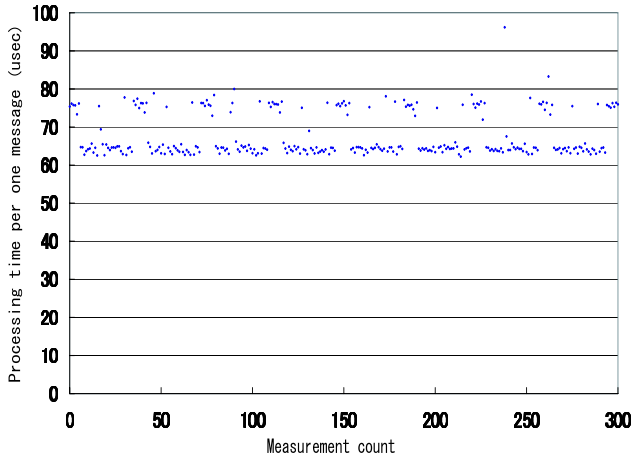

Figure 6: Measurement of overhead for forwarding messages to local server application

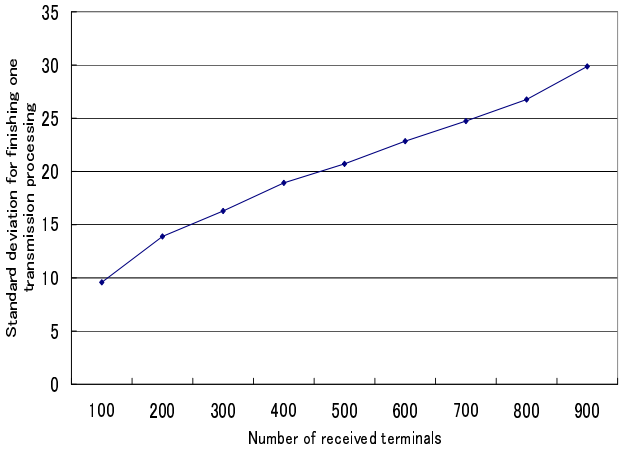

Figure 8: Standard deviation of overhead for sending messages to other teminal

play as 60 fps. Although it is necessary to examine overhead precisely through the network, the mean overhead is tolerable for playing games by comparison of processing time of frames.

In the following, we refer to the experiment of transmitting to other terminals. In this case, two desktop computers are connected by single hub. In addition, PeerBooster and the game application have been started with one computer. The computer in which PeerBooster runs under has duplicated and sent game messages from local client to other user's terminal such as explanation of section 4. For investigating transmission overhead by increasing number of terminals, we have changed processing frequency of transmission. The locations of pushing time stamp are the identical as previous experiment.

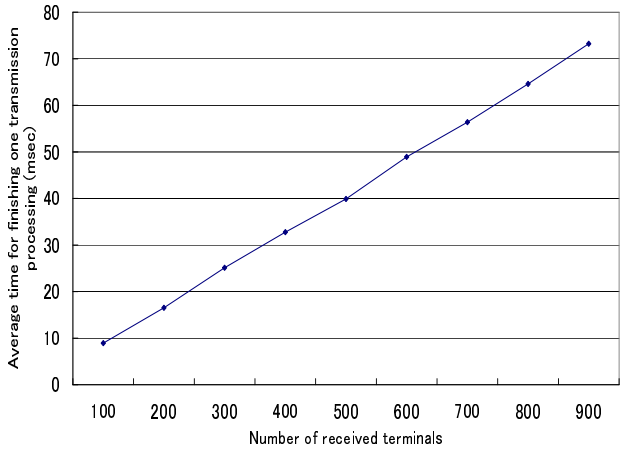

Figure 7: Average overhead for sending messages to other terminal

We show the average processing time and standard deviation of one transmission in Figure 7 and Figure 8. It is clear that the overhead linearly increases by increasing transmission processing. A permissible delay of the real-time multiplayer game is displayed from an existing research as 200 milliseconds or less [20]. In a previous analysis of MOG, packet intervals of Counter-strike which is closely similar to Quake2 concentrate the range 10 to 300 milliseconds [6]. Therefore we consider that the permissible delay for processing transmission of game packets is less 40 milliseconds and we predict that the maximum number of nodes which can connect in overlay network is less 400 nodes from our result.

8. RELATED WORK

The solution of reuse problem can be tried by using various approaches. We divide it into two classes: emulation tool and development support middleware. In the following subsection, we describe that these are not enough to solve reuse problem though there are some advantages in aspect of developing or constructing C/S architecture of MOGs.

8.1 Server Emulation Tool

Server emulation tools are one of the simplest approarches [1] [4]. The emulation realizes the server function by outputting the packet that the game server should put out. This tool cannot change the network architecture of game applications and it remains that the server applications are single point of failure. In addition, there exists an important problem that the game databases of MOG may not be available when the services are down.

8.2 Development Support Middleware

Most game development companies are using the Development Support Middleware to spend time in the programming of the nucleus of the game without spending time in the network code [17] [2] [25]. Game providers can construct network architecture of MOGs easily to program it using middleware. However it is necessary to modify the programs of the network part from the source code by this method. The modification is a complex work for most developers because the game applications is composed of the source code that reaches by millions of lines.

9. CONCLUSION AND FUTURE WORK

In this paper, we have advocated reuse problem of MOGs which indicates some MOGs cannot be played because of

difficulty of maintaining game servers in C/S architecture. For the purpose of solving this problem, our idea is to reuse C/S game applications as P2P manner. Furthermore our middleware called PeerBooster has been designed and implemented to support its construction. The advantage of PeerBooster is that game developers and users are able to switch the network architecture of MOGs from C/S to P2P by duplicating game sessions without modifying game binaries.

There are still some future work for implementing all modules. Investigating more efficient synchronization architecture is also needed. In addition, we need to validate scalability of PeerBooster architecture, and effectiveness of rule descriptions.

10. ACKNOWLEDGEMENTS

We are indebted to the anonymous reviewers of NETGAMES 2005 for helpful suggestions. Furthermore Tadashi Yanagihara gave us the useful report and information for my research, and I greatly appreciate Mika Minematsu for supervising my total research work of undergraduate years.

11. REFERENCES

[1] *bnetd: a free Battle.net server: http://www.chiark.greenend.org.uk/ owend/free/bnetd.html.*

[2] *Massiv - Massively Multiplayer Online Game Middleware: http://massiv.objectweb.org/.*

[3] *The Expat XML Parser: http://expat.sourceforge.net/.*

[4] *Project:Athena - Ragnarok Online server emulation tool*, 2004.

[5] Nathaniel E. Baughman and Brian Neil Levine. Cheat-proof playout for centralized and distributed online games. In *INFOCOM*, pages 104–113, 2001.

[6] Mark Claypool, David LaPoint, and Josh Winslow. Network Analysis of Counter-strike and Starcraft. In *Proceedings of the 22nd IEEE International Performance, Computing and Communications Conference(IPCCC)*, Phoenix, Arizona, USA, 2003.

[7] Microsoft Corporation. *NDIS - Network Driver Interface Specification: http://www.microsoft.com/whdc/ device/network/ndis/default.mspx.*

[8] Eric Cronin, Burton Filstrup, and Sugih Jamin. Cheat-proofing dead reckoned multiplayer games. In *Proc. of 2^{nd} International Conference on Application and Development of Computer Games*, Jan 2003.

[9] Loris Degioanni, Mario Baldi, Fulvio Risso, and Gianluca Varenni. Profiling and optimization of software-based network-analysis applications. In *SBAC-PAD*, pages 226–234, 2003.

[10] Chris GauthierDickey, Daniel Zappala, Virginia Lo, and James Marr. Low latency and cheat-proof event ordering for peer-to-peer games. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 134–139. ACM Press, 2004.

[11] Katherine Guo, Sarit Mukherjee, Sampath Rangarajan, and Sanjoy Paul. A fair message exchange framework for distributed multi-player games. In *NETGAMES '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 29–41, New York, NY, USA, 2003. ACM Press.

[12] B. Hoffmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning System: Theory and Practice, Fourth Edition. Springer-Verlag.*, 1997.

[13] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *ACM SIGCOMM 2001*, San Diago, CA, August. ACM.

[14] id software. *Quake: http://www.idsoftware.com/games/quake/.*

[15] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. of USENIX OSDI 2000*, pages 197–212.

[16] Yow-Jian Lin, Katherine Guo, and Sanjoy Paul. Sync-ms: Synchronized messaging service for real-time multi-player distributed games. In *ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols*, pages 155–164, Washington, DC, USA, 2002. IEEE Computer Society.

[17] Microsoft Corporation. *DirectPlay: http://www.microsoft.com/windows/directx/default.aspx.*

[18] Microsoft Corporation. *AGE OF EMPIRE: http://www.microsoft.com/games/empires/*, 1997.

[19] David L. Mills. Internet time synchronization: The network time protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press.* 1994.

[20] Lothar Pantel and Lars C. Wolf. On the impact of delay on real-time multiplayer games. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29, New York, NY, USA, 2002. ACM Press.

[21] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in warcraft iii. In *NETGAMES '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 3–14, New York, NY, USA, 2003. ACM Press.

[22] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. A Review on Networking and Multiplayer Computer Games. In *Technical Report 454, Turku Centre for Computer Science*, 2002.

[23] P. Srisuresh and M. Holdrege. *IP Network Address Translator (NAT) Terminology and Considerations. Request For Comments (Informational)*, 1999.

[24] VALVE SOFTWARE. *Half Life:Counter Strike: http://www.counter-strike.net/*, 2002.

[25] ZONA inc. *Terazona: http://www.zona.net/.*