ANGEL: A Hierarchical Path Optimization Middleware for Real-Time Multiplayer Gaming in Wired and Wireless Networks

Yugo Kaneda† Mika Minematsu‡ Masato Saito‡ Hiroto Aida‡ Hideyuki Tokuda†,‡

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

Abstract

In this paper, *we propose a novel middleware for supporting real-time distributed multiplayer network games, called "Advanced middleware for Network Games utilizing Eligible Leaders (ANGEL)."* ANGEL optimizes the gaming network based on autonomous node hierarchization. ANGEL hierarchizes game players' nodes based on their message loss ratio and network latency, and selects the most appropriate path for exchanging game messages.

We have implemented a prototype of ANGEL on Linux and evaluated it in both wired and wireless ad-hoc environments by using the sample real-time application embedding ANGEL. We show that ANGEL improves network latency, network jitter, and message loss of the paths among gaming nodes. In wired and wireless ad-hoc environments, we demonstrate to maintain interactivity of real-time distributed multiplayer network gaming such as First-Person Shooting(FPS) dynamically.

1 Introduction

Online Multiplayer Gaming comes into wide use along with the popularization of various broadband networks and the technical advantages of gaming devices. Counter-Strike [11] or WarCraft [1]are the examples of the most popular game titles. In these network games, we can share the same virtual gaming space and communicate to play the game with other players who join the same game at the same time through the Internet.

When we refer to the network architecture of online multiplayer games, it is divided into two main categories which are Client-Server architecture (C/S) and Peer-to-Peer architecture (P2P). In C/S, every player who wants to share the same gaming space connects to game server which calculates player's interaction and transition of the whole game state, and players update their game states based on results which are sent by the game server. Most commercial online games adopt C/S. It is clear that game management is easy for game providers because they can manage gaming environments centrally. However, it is difficult to continue hosting gaming service for the reason that they must maintain game servers. In addition, players cannot use gaming service when the game server is down.

On the other hand, P2P 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. In the near future, more widespread use of this architecture is expected in network gaming because this architecture also has an affinity to wireless ad-hoc network that do not rely on a pre-existing infrastructure.

Though P2P has merits for applying real-time distributed gaming, we find it hard to create these games because there are technological problems. One of the most important problems is synchronization of game states. If we do not maintain the same game states with each other at the same time, we cannot play game fairly and impartially. However, it is not easy for both wired and wireless networks to synchronize with each other accurately since we must allow for network conditions and mobility of players.

The goal of ANGEL is to optimize wired and wireless real-time distributed gaming network based on hierarchization and maintain the best interactive state of game players. Our approach exploits autonomous node hierarchization that hierarchizes nodes by measuring network latency, network jitter, and message loss ratio, which we define as *link state*, after constructing the complete network graph. After hierarchizing nodes, master node, which manages gaming network, is elected automatically and calculates the most appropriate path.

We optimize both wired and wireless gaming networks by using the middleware which includes these functions. Seven sections compose this paper. In Section 2, we describe some backgrounds of network gaming. We refer to the problem of wired and wireless gaming environments in Section 3. Section 4 shows the architecture design of our approach and its implementation. Furthermore, we describe the experimental evaluation in Section 5. Section 6 shows some related work and we conclude our work and give a brief outlook for future work in Section 7.

2 Background Briefing

In this section, we explain short background information of network gaming in the Internet. Moreover, we introduce wireless ad-hoc gaming environments, which we call "Mobile Ad-hoc Group Gaming (MAGG)." MAGG will gain popularity by contribution of high-end mobile wireless gaming devices in the near future.

2.1 Network Architecture

In ordinary network games, the ways of exchanging game messages among players are divided into two main categories.

1. **Time-based synchronization (TS):** Synchronization activities performed by each node sending its game state *periodically* to other nodes.

2. **Event-based synchronization (ES):** Synchronization activities performed in response to *external events* such as user input.

Also, in P2P network games there are two ways to connect each player at the beginning of a game.

1. **Hybrid configuration (HYBRID) :** When players start a game, a server node waits to be connected by other nodes. After all nodes are connected to the server, the server sends players' information to all nodes and players start playing game after connecting with each other.

2. **Pure configuration (PURE) :** Each node attempts to find other nodes by sending search packets periodically. When the sender finds a node that responds to search packets, it acquires information of other nodes through that node.

ES is well used and will be suitable for most of future distributed multiplayer gaming such as FPS in the Internet. The reason is that players cannot always reserve constant time to arrive game messages due to the heterogeneity of condition in network, and TS needs high overhead to copy all game state information of each player rapidly and periodically. The way of finding nodes used is **HYBRID** in FPS games. Additionally, players play such games for a short period and when they are playing a round game, other player who wants to play the same round cannot join the game.

2.2 Mobile Ad-hoc Group Gaming

In the near future, we will be able to not only play network games in the Internet but also play wireless ad-hoc network games by using mobile wireless gaming devices such as *"Play Station Portable* [10]." Players gather at small area where they can see each other, and they retain mobile gaming devices attached with a wireless device. A player starts as a server and waits to be connected from other players. Other players connect to the server, and they obtain information of players who connect directly. Additionally, they connect with each other and start to play game immediately. We assume to construct a simple wireless complete graph network to play such games by applying **HYBRID** and these games are also designed by **ES**. The number of players will be not over 50, and we do not assume massively multiplayer gaming. We show an example of MAGG in Figure 1.

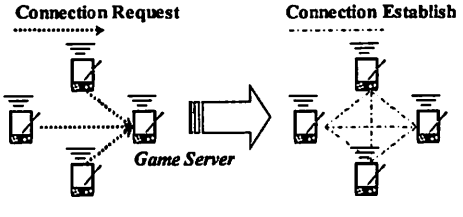

Figure 1: An example of communications in MAGG

3 The Problem of Synchronization in Network Gaming

When we make consideration about architecture of network games, synchronizing game states will be one of the major problems. In the Internet, this is caused by network latency and jitter [2]. Commercial network games use dead reckoning technique [9], which predicts movement of player's characters by using direction, speed, and acceleration to avoid inconsistency of game states among players. However, this technique does not necessarily have a beneficial effect on all real-time network games because operations of some games do not include factors of predictable movement. In distributed environment, latency becomes more serious for the reason that players, which have heterogeneous link states, receive game messages from others at different time.

In a similar way, in MAGG, it is not easy to synchronize with each player well since we must allow mobility of players. Each player does not always stay at the same place. The physical locations of players will be gradually changed. We can assume the case of playing in the house. At the beginning, there are players near the same place. However, there may be a player who moves to the back of an obstacle. For example, a player moves to be seated in a chair, which is the back of an obstacle from a player, to be relaxed. Due to the characteristic of wireless technology such as 802.11b [7], it is difficult between wireless devices to communicate through large obstacles and impossible to maintain the direct connection between such players. In MAGG, if there is a player who cannot communicate with others, game state of each player becomes asynchronized.

4 Design and Implementation of ANGEL

For supporting synchronization of game states, our approach is to incorporate a layer of autonomous path optimization middleware in game applications. We name it ANGEL. We realize multi-hop communication on application-layer by using ANGEL after players construct the complete graph network to optimize paths in wired and wireless gaming networks.

4.1

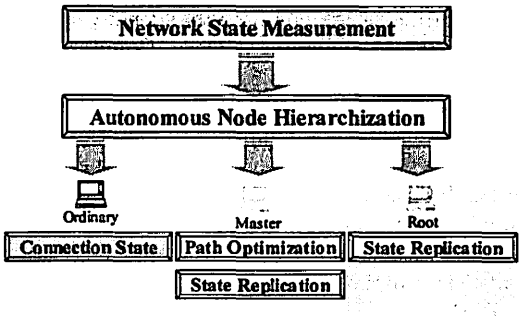

Figure 2: System architecture of ANGEL

We show the basic architecture of ANGEL in Figure 2. At the beginning, nodes form a graph network by connecting with HYBRID. After that, each node attempts to measure message loss ratio, network latency, and latency variance of direct links. Based on these information, nodes that join the game are divided into three categories: ordinary node (ON), master node (MN), and root node (RN). ON measures link state and reports to MN. MN constructs information of network topology based on link state information from ONs. MN attempts to check their link states periodically. If MN finds a node that can acquire a better network environment via an other node or a node that will break away from the gaming network, MN sends message for ON to change path. Additionally, ANGEL adapts hybrid synchronization architecture since it synchronizes to send game state from MN to

ONs with TS if application programmers need to guarantee periodical synchronization among players.

ANGEL consists of five main functions. We explain each function in the following.

- **Network State Measurement Function**

  Each node measures link state by sending small packets after connecting with each other. Based on this, each node exchanges the average network latency, maximum network latency, message loss ratio, and latency variance from itself to others with each other. We define these information as *link state information*.

- **Autonomous Node Hierarchization Function**

  After exchanging *link state information*, the node that retained minimum value of average network latency starts running as master node automatically. When there are nodes that have the same value, MN is elected by comparing maximum network latency. MN sends alive packets that tell the existence of it. When MN leaves the gaming network, each node detects it from the lack of alive packets and starts to elect MN among them by the same process again.

  For the scalable management, approximately 15 % of the nodes are elected as MNs. When there exist multiple MNs, RN is selected the same way to elect MN. RN acts the leader among MNs.

- **Connection State Report Function**

  After electing MN, other nodes act as ONs and measure link states. ONs keep link state information in their data table and send them periodically to MN.

- **Path Optimization Function**

  MN constructs application-layer routing tree based on link state information from ONs. In addition, MN attempts to find a path which has less message loss ratio, less network latency, and less latency variance. If MN finds such a path, MN orders ON to change the path.

- **State Replication Function (Optional)**

  MN sends its own game state to ONs to synchronize periodically. MN transports game state at the optimized time since they are selected based on the best link state.

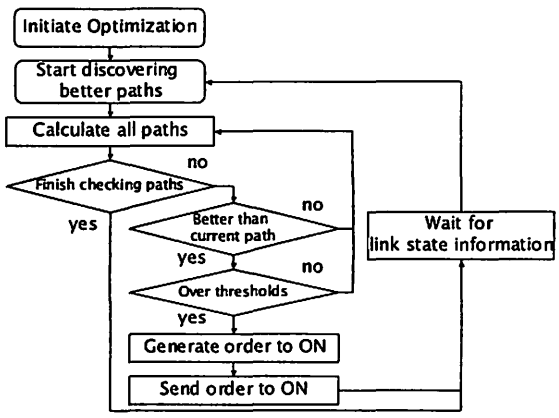

Figure 3: Path optimization performed by MN

We mention how MN calculates and determines to optimize paths in detail. The process is illustrated in Figure 3. *Path Optimization Function* is executed after hierarchizing nodes. This function continues to compare the current paths and tree of all paths that are constructed by link state information from ONs. For calculation of paths, we use Dijkstra algorithm. Additionally, we set up thresholds to optimize paths to prevent switching paths at short intervals. If MN finds a better path, MN compares the path and thresholds. When found path has acceptable values to optimize, MN generates order message for ON to change path and sends it to ON. These realize effective optimization without changing paths drastically. We have also defined suitable thresholds for both wired and wireless networks because both environments possess different characteristics. The details are discussed in the following subsection.

4.2 Implementation of ANGEL

We have implemented a prototype of ANGEL on application-layer using UDP on Linux kernel 2.4.20. This prototype includes *Connection State Report Function* and *Path Optimization Function*. It also provides the function of game server for certificating if game applications do not equip with certification function. Game applications use ANGEL API. We need only to write these functions in game applications. ANGEL sends game messages after applying message type to game message to determine which packet is ordinary game message or system message for ANGEL.

ANGEL works as following. A measurement packet includes packet sequence number. When other nodes receive this packet, they send an ack packet back to the sender. After the sender receives this packet, it calculates the round trip time (RTT) and keeps the value in its data table. Sender repeats this 5 times, and it calculates the average network latency, average message loss ratio, and standard variation of network latency. Then, their information are copied in the send buffer as the same way to create a game message and sent to MN. The 5 times is defined by experimental approach on the basis of overhead of system and adaptation to mobility.

ANGEL checks ANGEL header of all packets when MN receives packets from ONs. All these packets include the message type number in ANGEL header. Then, ANGEL finds a packet which includes the header of ordinary game message, and the game message is copied in the game receive buffer, which is used for game applications to pick up game messages from ANGEL, after deleting the ANGEL header.

There is the major difference between MN and ON when they receive system packet of ANGEL. An ON receives message from MN with address of destination node which orders whether to use the link or not. Based on this order, the ON changes the path. Furthermore, an ON also receives the message to forward to other node. When the ON receives this message, the ON sets the forwarding flag in data table and forwards the game messages when the ON receives it from the node. MN is the main node for processing path optimization of real-time distributed gaming network. *Path Optimization Function* is executed at MN after collecting link state information from ONs in a given amount of time.

4.2.1 Providing Simple API in ANGEL

To develop distributed real-time game application easily is one of the important problems. ANGEL provides simple API for trying to solve this problem. APIs are suitable for ES gaming architecture [2]. Application programmers do not need to concern about the timing to deliver game

messages. We show these examples of API and we post the pseudo application code using ANGEL-API in Figure 4.

- **ANGEL_SEND_MSG**

  This function supports that ANGEL sends player's inputs to other players. After using this function, ANGEL sends messages to other players with appropriate paths.

- **ANGEL_RECEIVE_MSG**

  This function supports to snoop the network buffer and receive game messages automatically. When game messages arrive at the destination host, ANGEL deletes its header and copies the message to game buffer.

- **ANGEL_DELETE_HDR**

  This function deletes ANGEL's headers from received messages.

```
BUFFER game_buffer;

eventDrivenFunction{
  if(communicationEventOccured(keyEvent) =valid){
    ANGEL_SEND_MSG(send_message);
  }
  else if(messageReceivedOccured(read_buffer) =valid){
    ANGEL_DELETE_HDR(read_buffer, game_buffer);
    gameUpdating(game_buffer);
  }
}
```

Figure 4: A pseudo code of game application with ANGEL-API

4.2.2 Implementation for Wired Network

As we referred in the previous subsection, it is important to set suitable thresholds in *Path Optimization Function* for stable path changing. First, we show the thresholds in Table 1 in wired network. Most of the interactive problems are caused by network latency and jitter [4] in wired network gaming. Based on stress of human interaction toleration, we set the latency threshold to 200 msec and standard variance of jitter to 65.

4.2.3 Implementation for Wireless Network

We have implemented a technique for adapting to mobility of nodes when MN optimizes paths in wireless network. If MN finds links which have high jitter variance, we define the links as moving nodes and mark such links on data table to detect moving node and avoid highly mobile nodes quickly. A link is marked when standard variance of jitter is over 60. If MN finds such links over jitter threshold, MN sends order to change path to the destination node. At the same time, to avoid switching path frequently, we set up the threshold of message loss ratio to optimize paths. A path exchange occurs when message loss ratio from ON exceeds 30 %.

4.2.4 Support at Application-layer

The reason why we realize application-layer multi-hop and not routing-layer consists of three main reasons. First, it is difficult for players to adopt ad-hoc routing protocol in mobile computing devices. Its process needs acknowledgements of operating system that game applications are running on. Second, we find it impossible to adopt most of these routing-layer protocols for gaming devices because they are composed of embedded operating system or architecture. Third, if we specialize routing-layer protocol in optimization for gaming applications, it may cause overhead for other applications from differences of routing policy. For gaming applications, it is important to deliver packets for synchronizing game states. However, file transfer applications need bandwidth for transferring a large amount of data. Most of routing-layer protocols cannot perceive routing policy of applications automatically.

Table 1: Thresholds for wired and wireless environments

factor / environment	wired	wireless
latency (msec)	200	200
message loss (%)	50	30
jitter variance	65	60

4.3 Sample Real-Time Application

We have created a sample real-time application running on ANGEL for evaluation. It is a simple ES application with 2D view. Characters that correspond with each player move to the location of mouse pointer. The application program code is independent from the network protocol stack. We programmed two types using this same application program code. One which we call sample P2P version does not include ANGEL and use direct application-layer multicast (DALM), and another which we call ANGEL version includes ANGEL network APIs. DALM is implemented such that each node connects to other nodes directly with unicast using UDP. For fair evaluation, all characters always act as the same move pattern on every time. We show the screenshot in Figure 5.

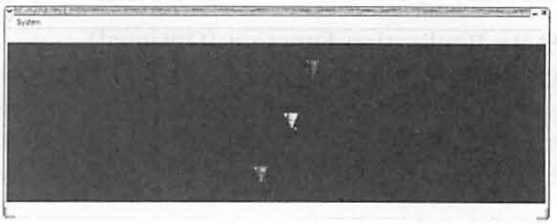

Figure 5: A screenshot of sample real-time application

5 Performance Evaluation

We focus on three points to evaluate ANGEL, which are whether ANGEL optimizes paths of players or not, how fast ANGEL realizes it, and how much overhead is expected when used it in wired and wireless networks. To investigate them, we have conducted two experiments.

5.1 Wired Environment

First, we prepared artificial wired network environment by using Dummynet [8] for simulating the Internet. It is illustrated in Figure 6. We used three notebook PCs and a desktop PC. These notebook PCs are connected to the different network segment through the desktop PC with each other and we generated three round-trip latencies that are

approximately 200 msec, 40 msec, and 40 msec on desktop PC using Dummynet. Then we ran two types of sample real-time game applications that are described in Section 4.5 on each notebook PC.

In this evaluation, a node started as server also runs as MN. At first, other players try to connect to it to join a game. After connecting each other, ONs measure their link states, then the MN detects link of high network latency and network jitter, and orders ONs to change to the better path from reporting link states of ONs. To compare performance of them, we evaluate the traffic overhead, percentage of message loss ratio, and convergence time for changing path in MN of ANGEL.

We have changed the interval that each node sends small packets. This is heuristic approach with allowing for reducing of overhead to measure link states. We have done the evaluation 15 times for each interval and have defined each session time as 10 minutes which is based on the model of playing time in FPS game [3].

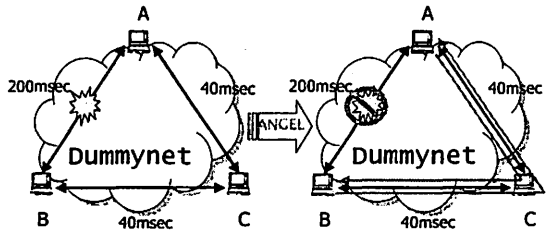

Figure 6: The experimental environment for wired gaming network

5.2 Performance Analysis in Wired Environment

We show transition of convergence time in Figure 7 and the overhead of ANGEL in Figure 8. In this evaluation, convergence time is defined as the time from start of communication between players to the time that path changing is ordered by MN after collecting link state information from ONs. Furthermore, we define traffic overhead as sent data among three nodes and investigated them to measure overhead of sample P2P and ANGEL. When we define the interval 100 msec, ANGEL optimizes wired network at approximately 1 second after connecting with each other. It is tolerance level because commercial FPS keeps playing game session from 15 to 30 minutes at average [3], then players can play most of session time with optimized paths. In addition, we can understand that there is trade-off between convergence time and overhead.

Additionally, the maximum overhead in this evaluation is 11 %, which was compared with sample P2P. When we adopt ANGEL to commercial network games, this overhead will become lower since most of the commercial network games send and receive more traffic than this application.

5.3 Wireless Environment

For the wireless evaluation, we experimented in the wireless environment shown in Figure 9. We used same note PCs attached with 802.11b wireless card and each node is set ad-hoc mode. Then a node that acts as MN starts up a round game, and other nodes connect to the node.

The definitions of experimentation number and session time are same with wired evaluation, and we have changed the interval that each node sends small packets from 100 msec to 1000 msec. We decided that message loss is always

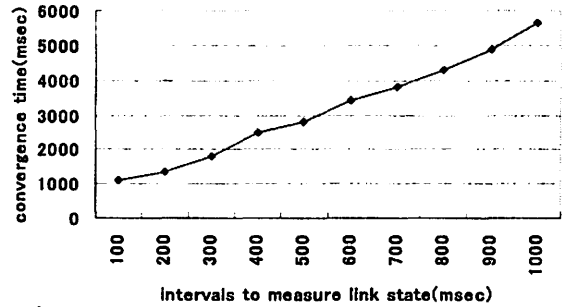

Figure 7: Convergence time of path optimization in wired environment

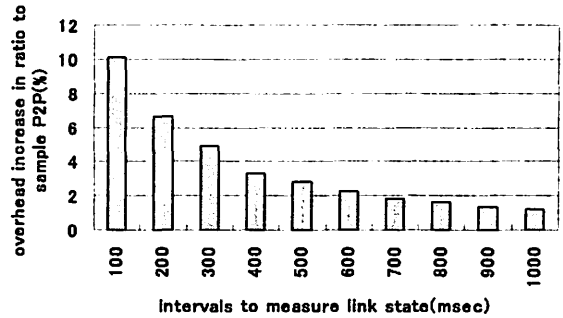

Figure 8: ANGEL's traffic overhead in wired environment

caused in wireless ad-hoc environment and it is important to think of overhead to measure link state and to detect message loss from our study. We also evaluated traffic overhead, the percentage of message loss ratio, and convergence time for changing path in MN between sample P2P and ANGEL.

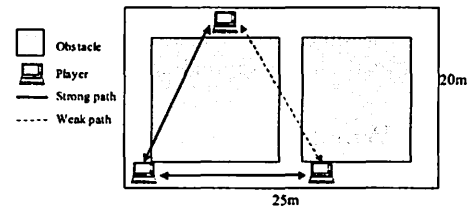

Figure 9: The experimental environment for MAGG

5.4 Performance Analysis in Wireless Environment

The results of experiments are illustrated in Figures 10 and 11. When nodes measure link states at 200 msec, path optimization is occurred at minimum time of 1 second. It is sufficient span to detect link states and optimize paths because we assume that game players move around by using wireless gaming device including ANGEL at average walking speed.

At this time, ANGEL increases the traffic overhead by 10 %. In commercial network games, each player sends game packets approximately 2800 bytes every second. The percentage of ANGEL's traffic overhead also becomes lower when we adopt ANGEL to commercial network games for the reason that each player sends game packets approximately 1900 bytes every second in our sample application. These figures also demonstrate that there is trade-off between convergence time and overhead, and interval of 800 msec is the best and its overhead is 1.7 % which is tolerable for mobile game applications and the overhead of network I/O.

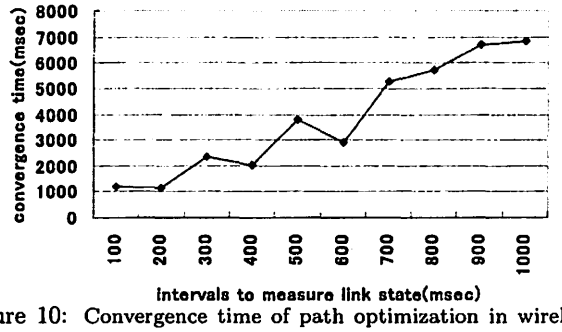

Figure 10: Convergence time of path optimization in wireless environment

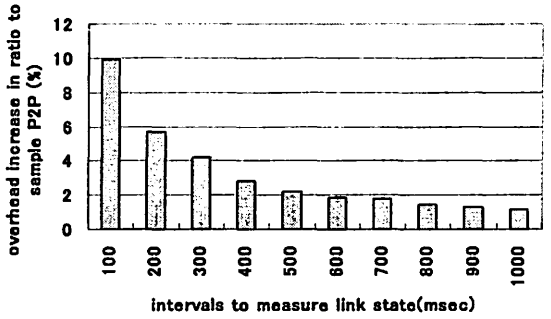

Figure 11: ANGEL's traffic overhead in wireless environment

We also show the difference of message loss ratio between sample P2P and ANGEL at this experimental environment. We have calculated message loss ratio from difference between the number of sent messages and received messages between nodes. In the case we use sample P2P, there is approximately 40 % of average message loss ratio between 2 nodes by existence of weak connection. By using ANGEL, which detects such links with measurement and changes to the better path, it becomes approximately 5 %.

6 Related Work

V. Ramakrishna et al. [12] have proposed the similar middleware for optimizing real-time multiplayer gaming network. This futuristic point is that it can be adaptable without reprogramming game applications and realizes packet aggregation for decreasing overhead. However, players cannot communicate without root node because all of game messages pass through it. Furthermore, there is no reference about mechanism to elect appropriate root. ANGEL includes function to elect MN and hierarchizes nodes automatically and it does not need all communication through MN.

There are also some previous research of optimizing application-level multicast for real time applications in the Internet. Narada [6, 5] is distributed protocol for optimizing application-layer multicast of real-time applications. Each node constructs source-based routing tree by exchanging routing information with directly connected nodes. In addition, this protocol optimizes application-layer path based on bandwidth in wired network. However, in wired network gaming, it is not sufficient enough to optimize paths based on bandwidth since the problem of maintaining state synchronization is caused by latency and jitter in many cases. Compared to Narada, ANGEL utilizes also network latency and jitter variance for wired network. Furthermore, we use powerless devices such as PDA in MAGG. It is hard for all of group players to calculate optimized paths for the reason that game applications spend much machine power for creating realistic 3D graphics or sounds. Most of nodes that play game do not need to spend processing power for network optimization in ANGEL since it is a simple architecture which relies on MN for calculation.

7 Conclusions and Future Work

In this paper, we have advocated growing expectations of real-time distributed network games and the problem of synchronization of game states. Furthermore, we have proposed our middleware approach that is called ANGEL for optimizing application-layer paths and supporting delivery of game messages rapidly. ANGEL is also designed from the viewpoint of application programmers, game players, and embedded architecture's gaming devices. ANGEL makes it easy to create real-time distributed multiplayer games by providing simple APIs.

We have shown the results of ANGEL's overhead and convergence time of path optimization are tolerable from evaluations. Additionally, we found there is trade-off between path convergence time and traffic overhead.

There are some future work to worthwhile endeavoring to find better intervals of sending small packets and link state information packets, and to check the optimized path on the basis of overhead and path convergence time by using realistic scenario. We must also refer the scalability of ANGEL.

We will realize the way to elect MN based on node machine power. Lastly, we have a plan to implement simulator of ANGEL to show the effectiveness.

Acknowledgement

This work has been conducted in Ubila Project by Ministry of Internal Affairs and Communications(MIC).

References

[1] Blizzard Entertainment. *Warcraft: http://www.blizzard.com/wow/.*
[2] J. Blow. A LOOK AT LATENCY IN NETWORKED GAMES. *Game Developer Magagine*, 1998.
[3] F. Chang and W. chang Feng. Modeling Player Session Times of On-line Games. In *Proceedings of ACM SIG NetGames*, 2003.
[4] J. Farber. Network Game Traffic Modelling. In *Proceedings of ACM SIG NetGames*, 2002.
[5] Y. hua Chu, S. G.Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture. In *Proceedings of ACM SIGCOMM*, 2001.
[6] Y. hua Chu, S. G.Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM SIGMETRICS*, 2000.
[7] IEEE 802.11 Standard (IEEE Computer Society LAN MAN Standards Committee). *Wireless LAN MAC and PHY specifications: Higher speed Physical Layer (PHY) extension in the 2.4 GHz band*, 1999.
[8] L. Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. In *Proceedings of ACM Computer Communication Review*, 1997.
[9] J. Smed, T. Kaukoranta, and H. Hakonen. A Review on Networking and Multiplayer Computer Games. In *Technical Report 454, Turku Centre for Computer Science*, 2002.
[10] Sony Computer Entertainment Inc. *Play Station Portable: http://www.us.playstation.com/pressreleases.aspx?id=207*, 2004.
[11] VALVE SOFTWARE. *Half Life:Counter Strike: http://www.counter-strike.net/*, 2002.
[12] V.Ramakrishna, M. Robinson, K. Eustice, and P. Reiher. An Active Self-Optimizing Multiplayer Gaming Architecture. In *Proceedings of Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS'03)*, 2003.